SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

ON THE RELATION BETWEEN THE UPWIND-DIFFERENCING SCHEMES OF GODUNOV, ENGQUIST–OSHER AND ROE*

BRAM VAN LEER†

Abstract. The upwind-differencing first-order schemes of Godunov, Engquist–Osher and Roe are discussed on the basis of the inviscid Burgers equations. The differences between the schemes are interpreted as differences between the approximate Riemann solutions on which their numerical flux-functions are based. Special attention is given to the proper formulation of these schemes when a source term is present. Second-order two-step schemes, based on the numerical flux-functions of the first-order schemes are also described. The schemes are compared in a numerical experiment and recommendations on their use are included.

Key words. upwind differencing, approximate Riemann solution, conservation laws

1. Introduction.

Upwind differencing, while trivial for a diagonalized hyperbolic system, is difficult to achieve when the difference scheme has to be written in conservation form. The oldest and most complicated version is due to Godunov [1], [2]; the increasing popularity of upwind differencing recently has led to a variety of simpler implementation techniques. A review of these is given by Harten, Lax and van Leer [3].

Among the recent additions to the family of upwind conservative schemes the method of Engquist and Osher [4], [5] is closest to the original Godunov scheme, while the method of Roe [6], [7] offers the greatest simplification. In the present paper the differences between these schemes are discussed on the basis of the inviscid Burgers equation. Part of the discussion covers known, but not well-known, aspects of the schemes, thus rendering the paper to some extent a review paper. For maximum clarity the presentation leans heavily on geometrical insights.

The first-order accurate schemes are explained in §§ 2, 3 and 4 for the homogeneous equation; § 6 describes how to include a source term and § 7 how to achieve second-order accuracy in a two-step format. Their rendition of a stationary shock and a transonic expansion is discussed in § 5 and illustrated in the numerical experiments of § 8. Section 9 rounds off with recommendations regarding the application of these schemes to single conservation laws and systems of conservation laws.

2. Godunov's method.

Godunov's [1], [2] method for integrating a hyperbolic system of conservation laws

$$u_t + [f(u)]_x = 0 \tag{1}$$

is a scheme in conservation form:

$$(u_i^{n+1} - u_i^n)/\Delta t + \{F(u_i^n, u_{i+1}^n) - F(u_{i-1}^n, u_i^n)\}/\Delta x = 0; \tag{2}$$

here u_i^n represents the average value at time $t^n = n\Delta t$ in the computational zone centered on $x_i = i\Delta x$. The numerical flux-function $F_G(u_i^n, u_{i+1}^n)$ in the Godunov scheme

* Received by the editors June 5, 1981, and in revised form June 2, 1982. This work was supported under NASA contract NAS1-14472 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, Virginia 23665.

† Leiden State University, Leiden, The Netherlands. Present address: Department of Mathematics and Computer Science, Delft University of Technology, 2600 AJ Delft, the Netherlands.

is taken to be the flux value arising at $x_{i+1/2}$ in the exact solution of the initial-value problem with a piecewise uniform initial distribution

$$u^n(x) = u_i^n, \qquad x_i - \Delta x/2 < x < x_i + \Delta x/2. \tag{3.1}$$

That is, if

$$u(x,t) = v(x/t; u_L, u_R) \tag{3.2}$$

is the (weak) similarity solution of the Riemann problem with initial values

$$u = \begin{cases} u_L, & x < 0, \\ u_R, & x > 0, \end{cases} \tag{3.3}$$

then

$$F_G(u_i^n, u_{i+1}^n) = f[v(0; u_i^n, u_{i+1}^n)]. \tag{4}$$

For Burgers' equation in the inviscid limit,

$$u_t + (\tfrac{1}{2}u^2)_x = 0, \tag{5}$$

the similarity solution to the initial-value problem (3.2) is

$$\begin{array}{c} u_L \leqq u_R \\ \text{(expansion)} \end{array} \begin{cases} v = u_L, & x/t \leqq u_L, \\ v = x/t, & u_L < x/t < u_R, \\ v = u_R, & x/t \geqq u_R, \end{cases} \tag{6.1}$$

$$\begin{array}{c} u_L > u_R \\ \text{(shock)} \end{array} \begin{cases} v = u_L, & x/t < u_S = \tfrac{1}{2}(u_L + u_R), \\ v = u_R, & x/t > u_S. \end{cases} \tag{6.2}$$

For numerical reasons it is useful to distinguish in the formula for $F_G(u_L, u_R)$ three cases:

(i) *fully supersonic/subsonic*

$$F_G(u_L, u_R) = \begin{cases} \tfrac{1}{2}u_L^2, & u_L, u_R > 0, \\ \tfrac{1}{2}u_R^2, & u_L, u_R < 0; \end{cases} \tag{7.1}$$

(ii) *transonic expansion*

$$F_G(u_L, u_R) = 0, \qquad u_L \leqq 0 \leqq u_R; \tag{7.2}$$

(iii) *transonic shock*

$$F_G(u_L, u_R) = \begin{cases} \tfrac{1}{2}u_L^2, & u_L > u_S \geqq 0 \geqq u_R, \\ \tfrac{1}{2}u_R^2, & u_L \geqq 0 > u_S > u_R. \end{cases} \tag{7.3}$$

These cases are illustrated in Figs. 1, 2 and 3. The reference to a sonic speed $(=0)$ arises from the use of (5) in transonic aerodynamics.

In order to combine the formulas (7) in one compact algorithm, we follow Engquist and Osher by introducing

$$u^+ \equiv \max(u, 0), \tag{8.1}$$

$$u^- \equiv \min(u, 0), \tag{8.2}$$

$$u = u^+ + u^-, \tag{8.3}$$

$$|u| = u^+ - u^-; \tag{8.4}$$

we then have

$$F_G(u_L, u_R) = \max[\tfrac{1}{2}(u_L^+)^2, \tfrac{1}{2}(u_R^-)^2]. \tag{9}$$

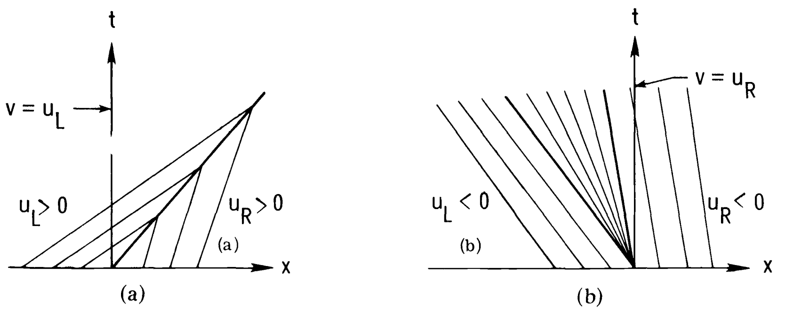


FIG. 1. *Riemann solution*: (x, t) *diagrams for case* (i), *with* $u_L > u_R > 0$ (a); $u_L < u_R < 0$ (b).

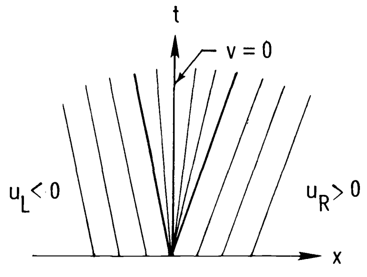


FIG. 2. (x, t) *diagram for case* (ii).

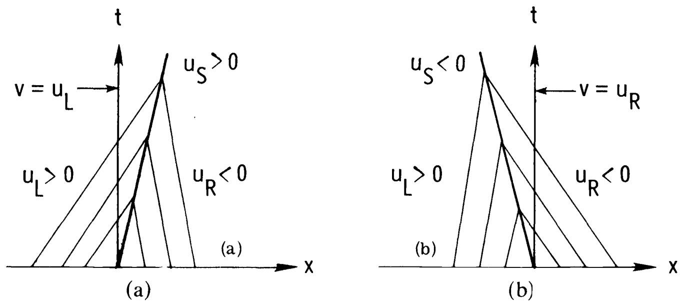


FIG. 3. (x, t) *diagram for case* (iii), *with* $u_S > 0$ (a); $u_S < 0$ (b).

3. The Engquist–Osher scheme. The Engquist–Osher [4], [5] scheme for integrating (1) also has the conservation form (2) and tacitly assumes the initial-value distribution to be (3.1); its numerical flux-function is

$$F_{EO}(u_L, u_R) = f(u_L) + \int_{u_L}^{u_R} A^-(u)\,du = f(u_R) - \int_{u_L}^{u_R} A^+(u)\,du \tag{10}$$
$$= \frac{1}{2}[f(u_L) + f(u_R)] - \frac{1}{2}\int_{u_L}^{u_R} |A(u)|\,du,$$

where the matrices $A^+(u)$, $A^-(u)$ and $|A(u)|$ are related to

$$A(u) \equiv df/du \tag{11}$$

by an extension of (8). For precise definitions and a description of the integration path used for the integrals in (10), see [5] or the review [3].

For Burgers' equation (5) the above recipe boils down to

$$F_{EO}(u_L, u_R) = \frac{1}{2}u_L^2 + \int_{u_L}^{u_R} u^- \, du = \frac{1}{2}u_R^2 - \int_{u_L}^{u_R} u^+ \, du \tag{12}$$
$$= \frac{1}{2}\left(\frac{1}{2}u_L^2 + \frac{1}{2}u_R^2\right) - \frac{1}{2}\int_{u_L}^{u_R} |u| \, du.$$

The integrals over u in (12) are defined in phase space, without reference to any mapping onto the (x, t)-plane. We may, however, introduce a mapping in the style of (3.2), namely,

$$u(x, t) = w(x/t; u_L, u_R), \qquad \min(u_L, u_R) \leqq u \leqq \max(u_L, u_R) \tag{13}$$

and consider $w(x/t; u_L, u_R)$ an approximation to the exact solution $v(x/t; u_L, u_R)$ of the Riemann problem (3.3). It follows that w is the following function of x/t:

$$w = \begin{cases} u_L, & x/t \leqq u_L, \\ x/t, & \min(u_L, u_R) < x/t < \max(u_L, u_R), \\ u_R, & x/t \geqq u_R. \end{cases} \tag{14}$$

In relating $F_{EO}(u_L, u_R)$ to $w(0; u_L, u_R)$ after the example of (4), caution is required, since, for $u_L > u_R$, w becomes multivalued in the domain $u_L \geqq x/t \geqq u_R$. Specifically, we have three branches

$$w^{(1)} = u_L, \quad w^{(2)} = x/t, \quad w^{(3)} = u_R, \qquad u_L \geqq x/t \geqq u_R. \tag{15}$$

The picture associated with this case is that of an overturned centered compression wave or folded characteristic field (see Figs. 4 and 5); in the exact Rieman solution (6) such a wave would be replaced by a shock discontinuity.

The proper formula for $F_{EO}(u_L, u_R)$ in terms of $w(x/t; u_L, u_R)$, equivalent to (12), is

$$F_{EO}(u_L, u_R) = \sum_k (-1)^{k-1} \frac{1}{2}\{w^{(k)}(0; u_L, u_R)\}^2, \tag{16}$$

where the sum is taken over all branches present at $x/t = 0$.

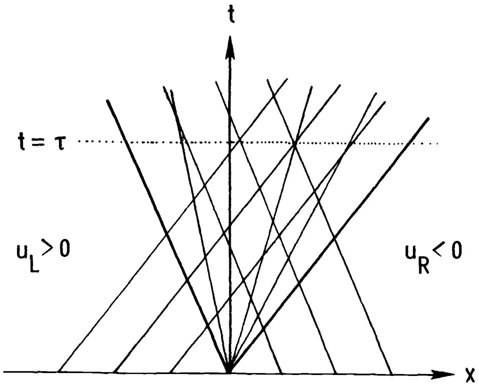


FIG. 4. *Approximate Riemann solution in the Engquist–Osher scheme*: (x, t) *diagram for case* (iii). *The multivalued solution at* $t = \tau$ *is displayed in Fig.* 5.

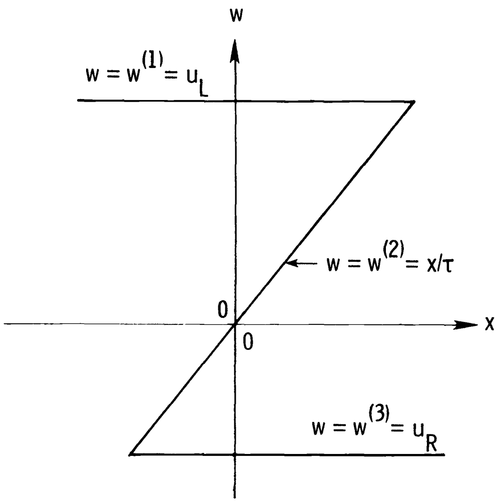


FIG. 5. *Approximate Riemann solution at $t=\tau$.*

In the cases (i), (ii) and (iii), distinguished earlier in (7) for $F_G(u_L, u_R)$, (12) or (16) yields

$$(17)\quad \begin{array}{lll} \text{(i)} & F_{EO}(u_L,u_R)=F_G(u_L,u_R)=\begin{cases}\frac{1}{2}u_L^2, \\ \frac{1}{2}u_R^2,\end{cases} & \begin{array}{l} u_L, u_R>0, \\ u_L, u_R<0,\end{array} \\ \text{(ii)} & F_{EO}(u_L,u_R)=F_G(u_L,u_R)=0, & u_L\leqq 0\leqq u_R, \\ \text{(iii)} & F_{EO}(u_L,u_R)=\frac{1}{2}u_L^2+\frac{1}{2}u_R^2\neq F_G(u_L,u_R), & u_L\geqq 0\geqq u_R. \end{array}$$

As indicated in [4], these formulas combine into

$$(18)\qquad F_{EO}(u_L,u_R)=\tfrac{1}{2}(u_L^+)^2+\tfrac{1}{2}(u_R^-)^2.$$

The difference between the Godunov and Engquist–Osher schemes lies entirely in the treatment of a transonic compression (iii). The latter scheme is the simpler one, since it does away with one test, namely, a test for the sign of u_S or for the maximum of $\frac{1}{2}(u_L^+)^2$ and $\frac{1}{2}(u_R^-)^2$.

4. Roe's method. Roe's [6], [7] method for integrating (1) again has the conservation form (2) and employs the initial-value distribution (3.1). Its numerical flux-function is defined as

$$(19)\qquad F_R(u_i^n, u_{i+1}^n)=f[w(0; u_i^n, u_{i+1}^n)],$$

where $w(x/t; u_L, u_R)$ is the exact solution of the Riemann problem (3.3) for the locally linearized system

$$(20.1)\qquad w_t+A(u_L,u_R)w_x=0.$$

The approximate Jacobian $A(u_L, u_R)$ is constructed such as to satisfy the discrete version of (11)

$$(20.2)\qquad f(u_R)-f(u_L)=A(u_L,u_R)(u_R-u_L).$$

For a scalar equation like (5), (20.2) uniquely determines $A(u_L, u_R)$:

$$A(u_L, u_R) = \tfrac{1}{2}(u_L + u_R) = u_S; \tag{21}$$

thus, the Riemann problem for the linear equation (20.1) with (21) has the similarity solution

$$w(x/t, u_L, u_R) = \begin{cases} u_L, & x/t < u_S, \\ u_R, & x/t > u_S. \end{cases} \tag{22}$$

In all cases (i), (ii) and (iii) this yields

$$F_R(u_L, u_R) = \begin{cases} \tfrac{1}{2}u_L^2, & u_S > 0, \\ \tfrac{1}{2}u_R^2, & u_S < 0. \end{cases} \tag{23}$$

The difference with Godunov's method lies in the treatment of an expansion: where the exact Riemann solution, used in Godunov's method, would include an expansion fan, Roe's method puts in a so-called expansion shock (see Fig. 6).

The numerical flux-function (23) deviates from the Godunov flux (7) only in case (ii), when the expansion is transonic. The computational simplification is, again, the elimination of one test.

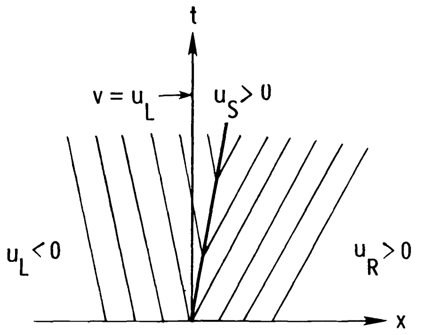


FIG. 6. *Approximate Riemann solution in Roe's scheme:* (x, t) *diagram for case* (iii), *with* $u_S > 0$, *showing the expansion shock.*

Rewriting (23) as an approximation to the Engquist–Osher flux (12),

$$F_R(u_L, u_R) = \tfrac{1}{2}(\tfrac{1}{2}u_L^2 + \tfrac{1}{2}u_R^2) - \tfrac{1}{2}|\tfrac{1}{2}(u_L + u_R)|(u_R - u_L), \tag{24}$$

we see that, for Burgers' equation, Roe's scheme is identical to the scheme of Murman and Cole [8], used in transonic aerodynamics. The latter is known to occasionally yield numerical results that include a (physically inadmissible) expansion shock. This is a direct consequence of the admission of an expansion shock in the underlying approximate Riemann solution.

Roe [14] has proposed a modification in the numerical flux-function for a transonic expansion that not only does away with expansion shocks, but may be an improvement over the flux-function in the Godunov and Engquist–Osher schemes. This is discussed at the end of the next section.

The problem of preventing inadmissible discontinuities in the use of upwind schemes has been addressed in general by Harten and Lax [15] and for gas dynamics by Colella [16].

5. Steady-shock and sonic-point representation. The fluxes in the schemes of Godunov and Engquist–Osher differ only on meshes where the data constitute a transonic compression; therefore, numerical results from these schemes differ only if a transonic shock, in particular, a stationary shock, is present. Likewise, the results of Roe's scheme will differ from those of Godunov's scheme only if a transonic expansion is present.

For Burgers' equation (5), Godunov's scheme admits the following stationary discrete representation of a stationary shock connecting the states $u_L > 0$ and $u_R = -u_L < 0$:

$$\begin{aligned} u_i &= u_L, & i &\leqq -1, \\ u_0 &= u_M, & u_L &\geqq u_M \geqq u_R, \\ u_i &= -u_L, & i &\geqq 1. \end{aligned} \tag{25}$$

This is the only stationary distribution admitted by the scheme.

The interior value u_M indicates the subgrid position x_S of the shock; by conservation we have (see Fig. 7a),

$$u_L(x_S + \tfrac{1}{2}\Delta x) + u_R(\tfrac{1}{2}\Delta x - x_S) = u_M \Delta x \tag{26.1}$$

or

$$x_S = \tfrac{1}{2}\Delta x\, u_M / u_L. \tag{26.2}$$

A shock standing exactly on the boundary of a zone will be represented without any interior value.

The Engquist–Osher scheme admits discrete steady shock-profiles with, in general, two interior states:

$$\begin{aligned} u_i &= u_L, & i &\leqq -1, \\ u_0 &= u_M, & u_L &\geqq u_M \geqq 0, \\ u_i &= u_N, & 0 &\geqq u_N \geqq -u_L, \\ u_i &= -u_L, & i &\geqq 2, \end{aligned} \tag{27.1}$$

where u_M and u_N are constrained by

$$\tfrac{1}{2}u_M^2 + \tfrac{1}{2}u_N^2 = \tfrac{1}{2}u_L^2. \tag{27.2}$$

Again, this is the only stationary distribution admitted by the scheme.

The pair of interior states indicates the subgrid position of the shock; by conservation we have (see Fig. 7b):

$$u_L(x_S + \tfrac{1}{2}\Delta x) + u_R(\tfrac{3}{2}\Delta x - x_S) = (u_M + u_N)\Delta x \tag{28.1}$$

or

$$x_S = \tfrac{1}{2}\Delta x(u_M + u_N + u_L)/u_L. \tag{28.2}$$

A shock standing exactly in the middle of a zone will be represented with only one interior value.

The representation of a stationary shock by Roe's scheme is the same as by Godunov's scheme, at least for Burgers' equation. However, Roe's scheme also admits as a stationary solution the expansion shock

$$\begin{aligned} u_i &= u_L < 0, & i &\leqq -1, \\ u_i &= -u_L > 0, & i &\geqq 0. \end{aligned} \tag{29.1}$$

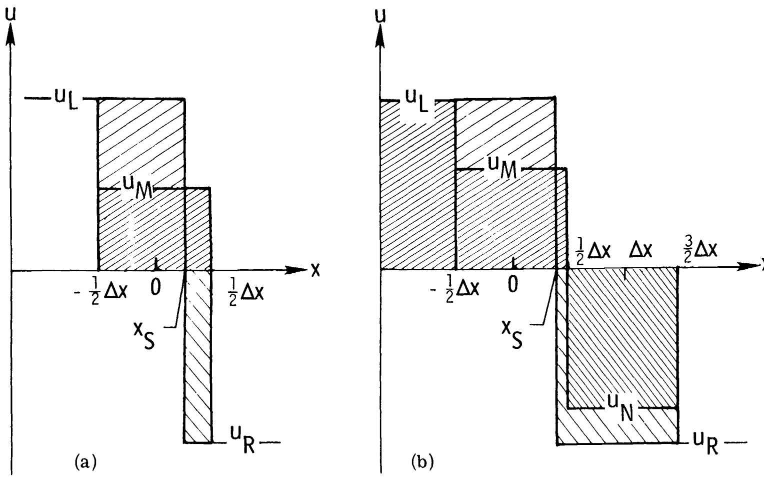


FIG. 7. *Fitting a shock discontinuity to stationary discrete shocks yielded by the schemes of Godunov or Roe* (a) *and Engquist–Osher* (b). *The numerical values u_M and u_N represent the average value of u in the intervals* $(-\frac{1}{2}\Delta x, \frac{1}{2}\Delta x)$ *and* $(\frac{1}{2}\Delta x, \frac{3}{2}\Delta x)$. *A subgrid distribution representing a shock transition from u_L to u_R at x_S must have the same integral as the numerical solution. In case* (a) *the integrals from* $-\frac{1}{2}\Delta x$ *to* $\frac{1}{2}\Delta x$ *are compared, in case* (b) *from* $-\frac{1}{2}\Delta x$ *to* $\frac{3}{2}\Delta x$.

A transonic shock profile, steady or not, obtained with any of the upwind schemes, has the property that the interior zones cannot influence the exterior solution (see Fig. 8a). Inversely, in a transonic expansion computed with the Roe–Murman–Cole scheme, the zone with the value closest to the sonic value cannot be influenced by the rest of the grid (see Fig. 8b). Once established by transient waves, the value in this zone changes no more, and it fixes the amplitude of the steady expansion shock that will remain.

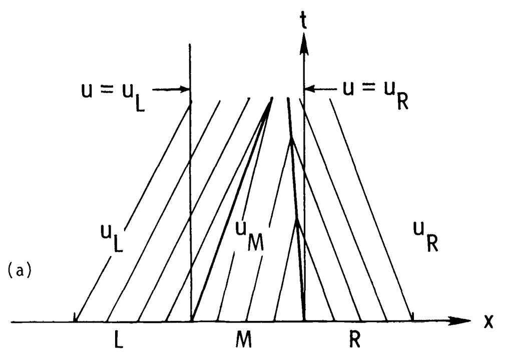
Colella [16] has pointed out that, in the approximate solution to the Riemann problem, two steps can be distinguished. The first step is to determine the speeds and amplitudes of the finite-amplitude waves; the second step is to compute the full solution as a function of x/t. It is only in the second step that the method of Roe errs. A rarefaction wave must always be given a finite spread; for Burgers' equation this boils down to replacing Roe's scheme by Godunov's.

The remedy that Roe [14] proposes cannot be formulated as a change in the approximate Riemann solution. It is based on regarding the initial values as nodal values of a smooth distribution, rather than zone averages of a piecewise uniform distribution. For neighboring zones L and R enclosing a sonic point we may write down the following upwind formulas:

$$\left.\begin{aligned}\left(\frac{\partial u}{\partial t}\right)_L &= -u_L(u_R-u_L)/\Delta x,\\ \left(\frac{\partial u}{\partial t}\right)_R &= -u_R(u_R-u_L)/\Delta x,\end{aligned}\right\} \quad u_L \leqq 0 \leqq u_R, \tag{29.2}$$

from which follows

$$(\partial u/\partial t)_L/(\partial u/\partial t)_R = u_L/u_R. \tag{29.3}$$



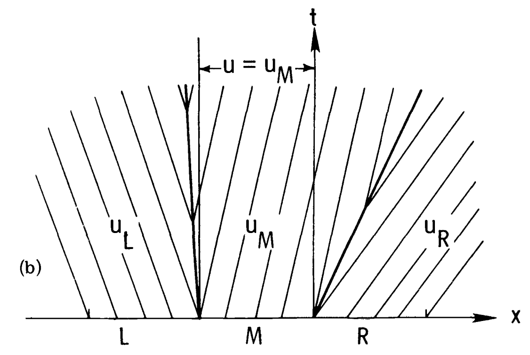


FIG. 8. *Two (x, t) diagrams showing the influence, through Godunov's scheme, of the adjacent zones on a zone inside a shock* (a) *and the influence, through Roe's scheme, of an almost sonic zone on the adjacent zones* (b). *In* (a), *the fluxes at the boundaries of zone M do not depend on u_M*; *in* (b) *they depend only on u_M.*

The latter relation can be forced onto the numerical flux $F_{tx}(u_L, u_R)$ for a transonic expansion:

$$(29.4)\qquad \{\tfrac{1}{2}u_R^2 - F_{tx}(u_L, u_R)\}/\{F_{tx}(u_L, u_R) - \tfrac{1}{2}u_L^2\} = u_L/u_R,$$

leading to

$$(29.5)\qquad F_{tx}(u_L, u_R) = \tfrac{1}{2}u_L u_R, \qquad u_L \leqq 0 \leqq u_R.$$

This unphysical flux (it is negative!) breaks down an expansion shock *faster* than the zero flux dictated by the exact Riemann solution. This is a direct consequence of the underlying assumption of smooth initial values.

While not fitting into the general framework of approximate Riemann solutions, the above approach has its appeal. The piecewise uniform values associated with a Riemann problem are not necessarily a good representation of the solution near a sonic point. In particular, the errors in the signal speed $df(u)/du$ $(= u)$ locally are of the order of the speed itself. The sonic point is always moved to a zone boundary and, in consequence, the flux gradient computed for each bordering zone becomes independent of the true gradient of the solution. In numerical solutions obtained with first-order schemes this shows up as a transonic expansion shock with amplitude $O(\Delta x)$. For a second-order scheme the effect disappears or reduces to $O[(\Delta x)^3]$.

In contrast, first-order accurate solutions obtained with the transonic flux (29.5) do not show these weak expansion shocks.

6. Inclusion of a source term. When approximating the equation

$$u_t + [f(u)]_x = s(x) \tag{30}$$

with any upwind scheme, it is not sufficient to add the source term to the right-hand side of (2); we must also change the initial-value distribution implied in our numerical model. Instead of assuming that it is piecewise uniform, as in (3.1), we rather take it to be *piecewise stationary*, that is,

$$[f(u^n(x))]_x = s(x), \qquad x_i - \Delta x/2 < x < x_i + \Delta x/2, \tag{31.1}$$

with

$$\frac{1}{\Delta x}\int_{x_i-\Delta x/2}^{x_i+\Delta x/2} u^n(x)\,dx = u_i^n. \tag{31.2}$$

This was first proposed by Liu [17] in constructing a random-choice method for the inhomogeneous conservation law (30).

The extension preserves a fundamental property of the homogeneous scheme, namely, that a zone-average can change only through the finite-amplitude waves entering from the zone boundaries. Moreover, if the numerical solution globally tends to a steady state, it will be the zone-averaged exact steady state almost everywhere (that is, provided that the scheme, like the three considered here, can render a shock transition in a finite number of zones). We shall come back to this further below.

As before, the initial values on either side of a zone boundary become the arguments of the numerical flux-function. Transient effects caused by a shock wave returning to the zone boundary under the influence of the source term, or by a curved transonic rarefaction fan, are ignored in order to keep the flux-function independent of Δt. The full scheme reads

$$(u_i^{n+1} - u_i^n)/\Delta t + \{F(u_{(i+1/2)-}^n, u_{(i+1/2)+}^n) - F(u_{(i-1/2)-}^n, u_{(i-1/2)+}^n)\}/\Delta x = s_i^n, \tag{32.1}$$

with

$$s_i \equiv \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} s(x)\,dx. \tag{32.2}$$

For the inhomogeneous Burgers equation

$$u_t + (\tfrac{1}{2}u^2)_x = s(x) \tag{33}$$

the initial-value representation must satisfy

$$\frac{1}{2}[u^n(x)]^2 = \frac{1}{2}[u^n(x_{(i-1/2)+})]^2 + \int_{x_{i-1/2}}^{x} s(x')\,dx', \qquad x_i - \Delta/2 < x < x_i + \Delta/2, \tag{34}$$

under the constraint (31.2). This constraint, however, is not strong enough to define a unique distribution in each zone, since the distribution may (and, in some zones, must) contain a discontinuity. Uniqueness can be achieved by selecting the distribution with, say, the weakest possible shock.

If we take Δx small enough to ensure that $s(x)$ does not change sign more than once in any zone, the following algorithm for the piecewise construction of $u^n(x)$ is adequate. First, determine for the zone under consideration the smallest continuous solution (see Fig. 9). Let ω_i^n be the average value on the positive branch of this solution; then the stationary distribution inside zone i will be continuous if

$$|u_i^n| \geqq \omega_i^n. \tag{35}$$

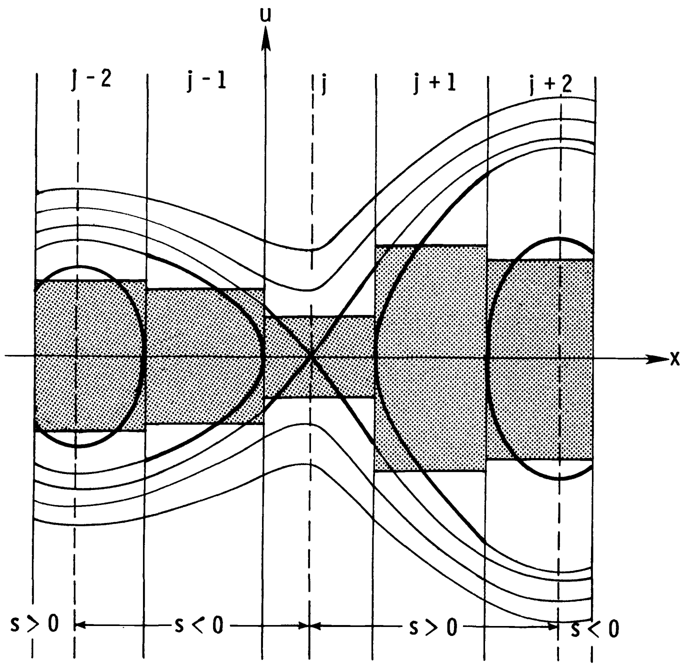


FIG. 9. *Stationary solutions of* (30) *for some source term* $s(x)$. *In each zone the heavy line traces the positive and the negative branch of the smallest continuous steady solution. The average value on these branches is indicated by the upper and lower boundary of the shaded areas. If the average value in a zone falls in the range of the shaded area, a continuous stationary solution cannot be realized.*

If this condition is satisfied, we search, iteratively, for the continuous distribution with the proper zone average. If it is violated, the stationary distribution will include parts of the upper and lower branch of the smallest continuous distribution, connected by a shock positioned so as to achieve the proper average value (see Fig. 10).

When using the upwind scheme to approach a globally stationary solution of (33), any zone containing a sonic point must be treated with special care, since the chance of numerically realizing the exact transonic structure without a shock is zero.

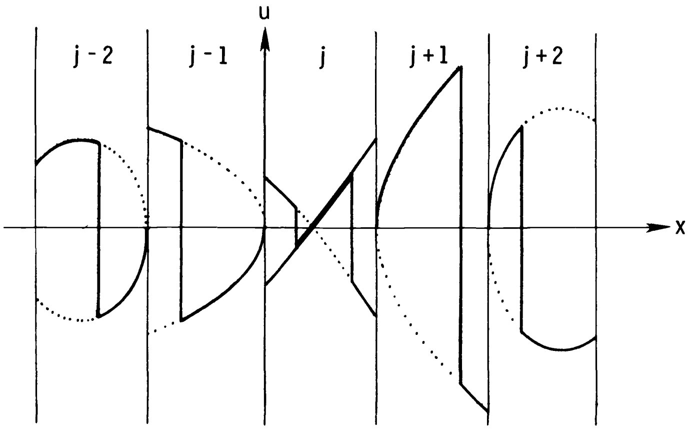


FIG. 10. *Examples of steady structure with a discontinuity, in the zones shown in Fig.* 9. *A* (*steady*) *shock connects the upper and lower branch of the smallest continuous steady solution for the zone considered. In zone j two different possibilities are shown.*

Specifically, in order to prevent the zone-boundary values from flipping sign, making global convergence impossible, smoothing may be introduced (see Fig. 11).

In a zone containing a stationary shock, no particular interior structure is needed to ensure global convergence, since the zone cannot influence its neighbors (see earlier Fig. 8a). After convergence one may insert the proper structure by enforcing continuity across the zone boundaries (see Fig. 12).

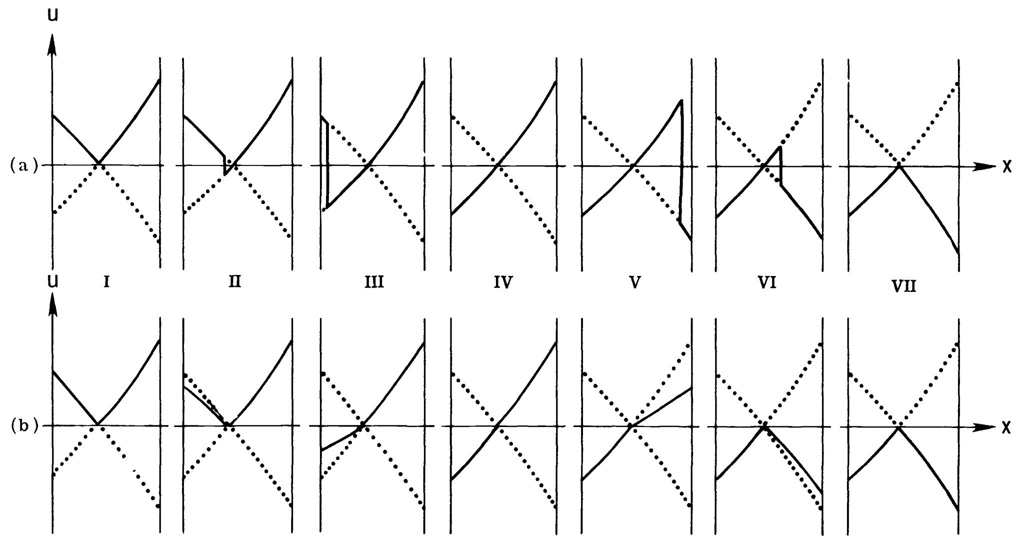


FIG. 11. (a) *A sequence of stationary structures close to the transonic expansion (graph* IV), *ordered by decreasing zone-average. Going from* III *to* IV, *and from* IV *to* V, *the slight change in zone-average causes one boundary value to flip its sign.* (b) *The solutions have been smoothed on the side where the shock occurs, so that the boundary values now vary continuously with the zone-average. The structures are not stationary but will allow convergence to the smooth transonic solution* IV.

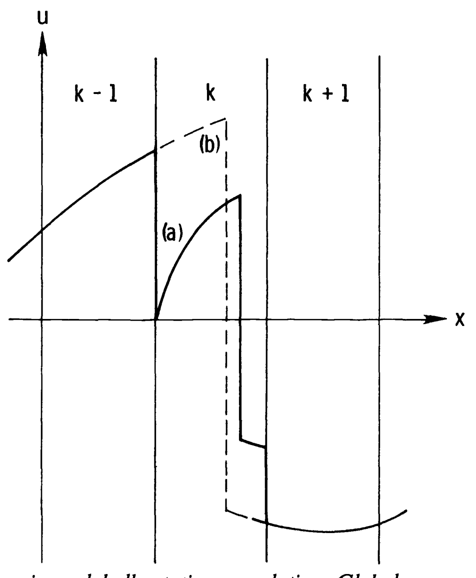


FIG. 12. *Shock structure in a globally stationary solution. Global convergence of the zone averages to a steady solution containing a shock has been obtained, say, with Godunov's scheme. For zone k containing the shock the scheme had adopted the structure* (a), *based on the smallest continuous steady solution for that zone. This may now be replaced by the stationary solution* (b) *that smoothly connects to the solution in zones* $k-1$ *and* $k+1$.

The scheme described above must be considered a research tool from which more practical schemes may be derived. In the numerical experiment of § 8 we simplified the procedure by allowing only s_i to enter the zone-boundary values. Away from sonic points or shocks the boundary values $u_{(i\pm 1/2)\mp}$ can be approximated by

(36.1) $$u^n_{(i\pm 1/2)\mp} = u^n_i \pm \tfrac{1}{2} s_i \Delta x / u^n_i + O\{(\Delta x)^2\}$$

or

(36.2) $$u^n_{(i\pm 1/2)\mp} = \sqrt{(u^n_i)^2 \pm s_i \Delta x} + O\{(\Delta x)^2\}.$$

We used (36.2), extending it, for use in sonic and shock zones, to

(36.3) $$u^n(x_{(i\pm 1/2)\mp}) = \operatorname{sgn}(u^n_i)\sqrt{(u^n_i)^2 \pm \min\{|s_i|\Delta x, (u^n_i)^2\} \operatorname{sgn}(s_i)}.$$

Note that (34), with $x = x_{i+1/2}$, is satisfied everywhere, but transonic structure in a zone is avoided. For zones that should include a sonic point the error in the boundary values is $O(\Delta x)$.

The technique of computing the boundary values in a zone from a locally stationary solution can easily be extended to systems of conservation laws of the form

(37) $$u_t + [f(u, x)]_x = s(x).$$

7. Second-order upwind schemes. Any numerical flux-function used in a first-order upwind scheme for (30) can be used in a second-order upwind scheme. For second-order accuracy in space we must introduce structure inside the zones by interpolation, while second-order accuracy in time may be achieved by advancing the cell-boundary values, to be used in the flux-function, to the intermediate time level $t^{n+1/2} = t^n + \frac{1}{2}\Delta t$. In predicting these time-centered values, *the interaction between cells can be fully ignored.* This observation, due to Hancock [9], has led to a drastic simplification of second-order upwind schemes since these first were formulated for systems of conservation laws by van Leer [10]. As in [10] we assume that the initial values form a *piecewise linear* distribution

(38) $$u^n(x) = u^n_i + (x - x_i)\frac{(\delta u)^n_i}{\Delta x}, \qquad x_{i-1/2} < x < x_{i+1/2},$$

with

(39) $$(\delta u)^n_i = \operatorname{ave}(u^n_{i+1} - u^n_i, u^n_i - u^n_{i-1});$$

ave (a, b) is an averaging procedure to be specified later. We particularly need the initial boundary values inside cell i

(40.1) $$u^n_{(i\pm 1/2)\mp} = u^n_i \pm \tfrac{1}{2}(\delta u)^n_i.$$

These boundary values are now advanced to $t^{n+1/2}$ by

(40.2) $$u^{n+1/2}_{(i\pm 1/2)\mp} = u^n_{(i\pm 1/2)\mp} - \frac{\Delta t}{2\Delta x}\{f(u^n_{(i+1/2)-}) - f(u^n_{(i-1/2)+})\} + \frac{\Delta t}{2} s_i.$$

The full scheme becomes

(41) $$(u^{n+1}_i - u^n_i)/\Delta t + \{F(u^{n+1/2}_{(i+1/2)-}, u^{n+1/2}_{(i+1/2)+}) - F(u^{n+1/2}_{(i-1/2)-}, u^{n+1/2}_{(i-1/2)+})\}/\Delta x = s_i.$$

The function ave (a, b) is chosen such that it tends to $\frac{1}{2}(a+b)$ if a and b are subsequent finite differences of a smooth solution, but tends to the smallest value where the solution is not smooth. Examples can be found in [11], [10]; we chose a refined formula due to Van Albada [12]

$$\text{ave}\,(a, b)=\frac{(b^2+c^2)a+(a^2+c^2)b}{a^2+b^2+2c^2}, \tag{42.1}$$

where c^2 is a small bias of the order $O((\Delta x)^3)$.

The weighted averaging prevents central differencing across a discontinuity in the solution or in its first derivative which would lead to numerical oscillations. It is an effective way of administering artificial dissipation wherever needed and nowhere else. By rewriting (42.1) as

$$\text{ave}\,(a, b)=\frac{a+b}{2}\left\{1-\frac{(a-b)^2}{a^2+b^2+2c^2}\right\}, \tag{42.2}$$

we see that, wherever the solution is smooth, the artificial-viscosity coefficient generally is of the order $O((\Delta x)^2)$. In a smooth extremum the coefficient grows to $O(\Delta x)$; the bias $2c^2$ in the denominator prevents a further increase of the viscosity that could lead to an undesirable clipping phenomenon (see [13, § 3(e)]).

With regard to Burgers' equation, an acceptable expression for the bias is

$$c^2=(u_{\max}-u_{\min})^2(\Delta x)^3/(x_{\max}-x_{\min})^3, \tag{43}$$

where $u_{\max}$ and $u_{\min}$ are certain upper and lower bounds of the numerical solution, fixed a priori or determined at each time level and $x_{\max}-x_{\min}$ is the length-scale of the problem.

8. Numerical comparison. The performance of the three schemes was tested on the basis of the periodic intial-value problem

$$u_t+(\tfrac{1}{2}u^2)_x=(\pi/2)\sin[2\pi(x-\xi)], \qquad 0\leqq x\leqq 1, \quad 0\leqq \xi \ll 1, \tag{44.1}$$

$$u(x, 0)=0, \tag{44.2}$$

$$u(0, t)=u(1, t). \tag{44.3}$$

The solution tends to the steady state

$$u(x, \infty)=\begin{cases}+\sin \pi(x-\xi), & 0\leqq x<\xi+\frac{1}{2},\\ -\sin \pi(x-\xi), & \xi+\frac{1}{2}<x\leqq 1,\end{cases} \tag{45}$$

including a sonic point at $x=\xi$ and a shock at $x=x_S=\xi+\frac{1}{2}$. We tested the ability of the schemes to approximate this steady state, and the accuracy of the approximation. For Δx we chose a value of $\frac{1}{16}$; the zones were centered on $x_i=(i-\frac{1}{2})\Delta x$, $i=1,\cdots,16$. For ξ we chose 0, $\frac{1}{4}\Delta x$ and $\frac{1}{2}\Delta x$, in order to achieve different subgrid shock positions. The Courant–Friedrichs–Lewy condition on the timestep, based on the maximum characteristic speed ($=1$) in the steady solution is

$$\frac{\Delta t}{\Delta x}\leqq 1; \tag{46}$$

accordingly we used $\Delta t=\frac{1}{2}\Delta x$ or $\Delta t=\Delta x$.

Table 1 shows the number of steps N_ε it took the schemes to converge according to the L_1-criterion

$$\sum_{i=1}^{16} |u_i^n - u_i^{n-1}| < \varepsilon, \tag{47}$$

with $\varepsilon = 1\times 10^{-3}$ or 1×10^{-6} and $\Delta t = \frac{1}{2}\Delta x$ or (for Godunov's scheme only) $\Delta t = \Delta x$. When the time-step is doubled, N_ε appears to be halved. The L_1-error E_ε was evaluated with respect to the zone-averaged exact solution, for the smallest value of ε. The subgrid shock positions in Table 1 were calculated with aid of (26.1) and (28.1), approximately valid because the source term is small in the shock region.

TABLE 1

Number of steps N_ε till convergence, L_1-error E_ε and steady-shock position x_S, for the initial-value problem (44), *solved with the first-order schemes of Godunov* (G), *Engquist–Osher* (EO) *and Roe* (R). *Mesh*: $\Delta x = \frac{1}{16}$, $\Delta t = \frac{1}{2}\Delta x$ *or* Δx (G *only, numbers between brackets*).

$\xi/\Delta x$	N_ε				E_ε		$(x_S - \frac{1}{2})/\Delta x$	
	$\varepsilon = 1\times 10^{-3}$		$\varepsilon = 1\times 10^{-6}$		$\varepsilon = 1\times 10^{-6}$		$\varepsilon = 1\times 10^{-6}$	
	G, R	EO	G, R	EO	G, R	EO	G, R	EO
0	62	61	112 [55]	111	8.8×10^{-3}	4.6×10^{-2}	0	0
$\frac{1}{4}$	68	66	138 [70]	136	9.6×10^{-3}	1.8×10^{-2}	.23	.26
$\frac{1}{2}$	52	52	88 [42]	88	4.6×10^{-3}	4.6×10^{-3}	$\frac{1}{2}$	$\frac{1}{2}$

The converged solutions for $\xi = \frac{1}{4}\Delta x$, with $\varepsilon = 1\times 10^{-6}$, are listed in Table 5; these are independent of Δt. The zone-averaged exact solution is given for comparison. The error in the zone with the sonic point is comparable to the value in the zone, as anticipated with the use of (36.3).

The results of Roe's scheme happen to be identical to those of Godunov's scheme; in particular, no expansion shock shows up. The reason is that, with the use of the boundary values (36.3), a sonic point in a zone is always moved to a boundary; in this case Roe's flux function (23) yields the same value ($=0$) as Godunov's (7.2). Furthermore, the initial average value in the zone ultimately containing the sonic point was close to the steady value to begin with.

The results of the Engquist–Osher scheme are identical to the Godunov–Roe results except for the expected spread in the shock profile for $\xi = 0$ and $\frac{1}{4}\Delta x$. The subgrid shock positions are slightly, but not significantly, more accurate than for the other schemes. Likewise, convergence with the Engquist–Osher scheme is faster, but not significantly so.

Better results were obtained with the second-order versions of the schemes. Convergence was achieved in 10–35% fewer steps than with the first-order schemes (Table 2) and the local error near the sonic point is now of the same order as elsewhere outside the shock (Table 5).

TABLE 2
The quantities N_ε, E_ε and x_S ($\varepsilon = 1\times 10^{-6}$) for the initial-value problem (44), *solved with the second-order two-step schemes based on the numerical flux-functions of Godunov* (G2, G2a), *Engquist–Osher* (EO2) *and Roe* (R2). *In* G2a *the algebraic average* (48) *is used instead of* (42.1). *Mesh*: $\Delta x = \frac{1}{16}$, $\Delta t = \frac{1}{2}\Delta x$.

$\xi/\Delta x$	N_ε			E_ε			$(x_S - \frac{1}{2})/\Delta x$		
	G2, R2	G2a	EO2	G2, R2	G2a	EO2	G2, R2	G2a	EO2
0	75	76	75	1.4×10^{-3}	3.1×10^{-2}	2.9×10^{-2}	0	0	0
$\frac{1}{4}$	89	92	89	1.3×10^{-3}	2.2×10^{-3}	4.7×10^{-3}	.24	.19	.26
$\frac{1}{2}$	79	79	79	1.3×10^{-3}	1.7×10^{-3}	1.3×10^{-3}	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

The Engquist–Osher scheme still yields a shock structure with two interior zones. For the second-order Godunov scheme we checked that the use of algebraic averaging of differences,

$$\text{ave}\,(a, b) = \tfrac{1}{2}(a + b), \tag{48}$$

causes the numerical shock to overshoot and undershoot (Table 5). The accuracy of the shock position also suffers (Table 2).

The results of Roe's scheme are practically, but not exactly, identical to the Godunov results, indicating a slightly different transient behavior. Again, the internal structure (36.3) in the zones, in combination with the particular choice of initial values, provides a safeguard against the occurrence of an expansion shock.

In order to make the problem more challenging, we changed the initial values (44.2), for $\xi = 0$, into

$$u_i^0 = \begin{cases} 1, & i = 1, \cdots, 8, \\ -1, & i = 9, \cdots, 16, \end{cases} \tag{49}$$

including an expansion shock at $x = 0$. The results are listed in Tables 3 and 6. Among the second-order schemes Roe's scheme now requires 30% more steps than the other schemes: apparently, the expansion shock is not so easily dissipated by Roe's scheme. Among the first-order schemes the discrepancy gets worse: the results of Roe's scheme quickly converge to the wrong solution, including the full initial expansion shock. This

TABLE 3
The quantities N_ε and E_ε ($\varepsilon = 1\times 10^{-6}$) for the initial-value problem (44.1), (49), (44.3), *solved with both first- and second-order upwind schemes. Scheme* R *converges to the wrong weak solution. The schemes* R* *and* R2* *incorporate the flux-function* (29.5) *for a transonic expansion. Mesh*: $\Delta x = \frac{1}{16}$, $\Delta t = \frac{1}{2}\Delta x$.

$\xi/\Delta x$	N_ε								E_ε	
	G	R	EO	G2	R2	EO2	R*	R2*	R	R2
0	170	30	169	77	99	76	103	71	5.7×10^{-1}	1.4×10^{-3}

is because the zones on either side of the expansion shock cannot be influenced by their neighbors (see Fig. 8b), and they cannot change by themselves because of (36c).

We then replaced the transonic value of the flux-function in Roe's scheme by Roe's new formula (29.5), with dramatic results. Not only was the proper solution obtained, but in considerably fewer steps than required by the other schemes (Table 3). Even for the second-order scheme convergence was significantly faster.

Finally, Table 4 shows the same quantities as Table 1 for experiments in which, in the first-order schemes, the source-term dependence of zone-boundary values was dropped. It took the schemes of Engquist–Osher and Godunov 15–25% more steps than previously to converge to a much less accurate solution (Table 5). Roe's scheme is now unstable: it yields an expansion shock where a smooth transonic transition should be; the amplitude of this shock grows linearly with time. The reason is that

TABLE 4

The quantities N_ε, E_ε and x_S ($\varepsilon = 1\times 10^{-6}$) for the initial-value problem (44), *solved with the first-order upwind schemes under the assumption of piecewise uniform initial values* (3.1). (*The label* "u" *in* Gu, Ru, EOu *stands for* "*uniform*".) *Mesh*: $\Delta x = \frac{1}{16}$, $\Delta t = \frac{1}{2}\Delta x$.

$\xi/\Delta x$	N_ε			E_ε			$(x_S - \frac{1}{2})/\Delta x$		
	Gu	Ru	EOu	Gu	Ru	EOu	Gu	Ru	EOu
0	135	-	135	6.0×10^{-2}	unstable	9.5×10^{-2}	0	-	0
$\frac{1}{4}$	174	-	172	6.1×10^{-2}	unstable	6.7×10^{-2}	.22	-	.24
$\frac{1}{2}$	103	103	103	4.7×10^{-2}	4.7×10^{-2}	4.7×10^{-2}	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

TABLE 5

Converged numerical solutions and zone-averaged asymptotic solutions for the experiments of Tables 1, 2 *and* 4, *with* $\xi = \frac{1}{4}\Delta x$, $\varepsilon = 1\times 10^{-6}$, $\Delta t = \frac{1}{2}\Delta x$.

i	$u_i^{N_\varepsilon}$, u_i^∞							
	$\xi/\Delta x = \frac{1}{4}$							
	G, R	EO	G2, R2	G2a	EO2	Gu	EOu	exact
1	.09778	.09778	.04916	.04917	.04916	.13828	.13828	.04899
2	.25516	.25516	.24249	.24244	.24248	.33330	.33330	.24259
3	.43185	.43184	.42635	.42610	.42639	.51176	.51175	.42687
4	.59602	.59602	.59379	.59344	.59364	.66976	.66976	.59474
5	.73869	.73869	.73871	.73726	.73933	.80171	.80171	.73976
6	.85367	.85367	.85462	.85711	.85231	.90266	.90266	.85635
7	.93631	.93631	.94080	.91445	.95106	.96879	.96879	.94003
8	.98329	.88033	.98172	1.15231	.95622	.99759	.89488	.98759
9	-.52996	-.42699	-.49587	-.60164	-.47889	-.54359	-.44088	-.49739
10	-.96441	-.96441	-.96449	-1.01358	-.96440	-.98796	-.98796	-.96847
11	-.89927	-.89927	-.90232	-.89318	-.90235	-.94026	-.94026	-.90254
12	-.79997	-.79997	-.80042	-.80067	-.80041	-.85632	-.85632	-.80192
13	-.67047	-.67047	-.66948	-.66861	-.66948	-.73932	-.73932	-.67048
14	-.51616	-.51616	-.51252	-.51223	-.51252	-.59368	-.59368	-.51328
15	-.34426	-.34426	-.33603	-.33589	-.33603	-.42473	-.42473	-.33635
16	-.16827	-.16827	-.14650	-.14649	-.14650	-.23797	-.23797	-.14649

TABLE 6

Converged numerical solutions and zone-averaged asymptotic solution for the experiments of Table 3 *with schemes* R *and* R2. *Note the expansion shock in the results of* R. *Parameters*: $\xi = 0, \varepsilon = 1 \times 10^{-6}, \Delta t = \frac{1}{2}\Delta x$.

i	$u_i^{N_\varepsilon}, u_i^\infty$		
	R	R2	exact
1	1.00000	.09815	.09786
2	1.03596	.28976	.28982
3	1.09933	.47012	.47064
4	1.17699	.63242	.63337
5	1.25564	.77076	.77177
6	1.32417	.87880	.88051
7	1.37431	.95648	.95540
8	1.40069	.98762	.99359
9	-1.40069	-.98762	-.99359
10	-1.37431	-.95648	-.95540
11	-1.32417	-.87880	-.88051
12	-1.25564	-.77076	-.77177
13	-1.17699	-.63242	-.63337
14	-1.09933	-.47012	-.47064
15	-1.03596	-.28976	-.28982
16	-1.00000	-.09815	-.09786

the zone containing the sonic point (say, zone j), not influenced by its neighbors, no longer has a steady internal structure, so that the scheme locally reduces to

$$u_j^n = u_j^0 + n\Delta t s_j. \tag{50}$$

9. Recommendations. For a scalar conservation law like Burgers' equation there seems to be little reason to abandon Godunov's first-order scheme in favor of the first-order Engquist–Osher scheme. The slight simplification in the flux calculation is accompanied by a degradation of steady-shock representation and no significant acceleration of the convergence to a steady state. The simplification achieved in Roe's first-order scheme is too drastic: the scheme cannot be used "as is" near a sonic point. With the modification (29.5), however, the scheme surpasses Godunov's scheme.

Both Godunov and Engquist–Osher schemes become appreciably more elaborate when applied to a nonlinear hyperbolic system like the one-dimensional Euler equations. The interpretation of the Engquist–Osher scheme as a Godunov-type scheme in which any shock in the solution of a local Riemann problem is replaced by an overturned centered compression wave remains valid. This modification makes it possible to compute $F_{EO}(u_L, u_R)$ explicitly, while $F_G(u_L, u_R)$ must be determined iteratively.

The Engquist–Osher scheme, on the other hand, requires two interior points in a discrete stationary shock [5], while Godunov's scheme probably requires only one (this has not yet been proven in general).

For the Euler equations, however, Roe's more drastic simplification of Godunov's method will pay off, even though the scheme must be somewhat modified in order to reject (almost) stationary expansion shocks. This can be achieved by spreading of rarefaction waves, as in [16], or by generalizing (29.3). Numerical experience with the latter technique is still lacking.

The first-order schemes, when formulated as in § 6, have the potential of achieving any desired order of spatial accuracy in a steady numerical solution. To what degree this potential can be realized for the one-dimensional Euler equations is at present not clear. Meanwhile, the second-order two-step schemes, intended primarily for solving transient problems, seem to outperform the first-order schemes based on (36.3) in obtaining a steady solution for Burgers' equation. Experiments for the Euler equations with both kinds of schemes, conducted for [13], but not fully reported therein, indicate a similar performance.

All schemes discussed in this paper are explicit. When using their numerical flux-functions in an implicit configuration, which may be desirable in approaching a steady state, the above recommendations do not automatically carry over. As noted by Engquist and Osher [4], the nonsmooth dependence of $F_G(u_L, u_R)$ on its arguments in the case (iii) of a transonic shock, that is: $F_G(u_L, u_R) \in C^{(0)}$, invalidates the linearization needed to invert the implicit difference equations. In contrast, $F_{EO}(u_L, u_R) \in C^{(1)}$ in all cases (i), (ii) and (iii). Whether this makes a difference in practice remains to be shown.

Acknowledgments. The ideas of Steve Hancock (Physics International, San Leandro, California) and Dick van Albada (University of Groningen, Netherlands), embodied in (40) and (42), respectively, are greatly appreciated, while discussions with Stanley Osher (University of California at Los Angeles) strengthened my belief in the treatment of source terms discussed in § 6. Phil Colella (Lawrence Berkeley Laboratory) pointed out the weak transonic expansion shocks arising in the use of the exact Riemann solution, while Phil Roe (Royal Aircraft Establishment, United Kingdom) kindly gave me permission to discuss his hitherto unpublished sonic flux-function.

REFERENCES

[1] S. K. Godunov, *Finite-difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics*, Mat. Sbornik, 47 (1959), pp. 271–306. (In Russian.)

[2] S. K. Godunov, A. W. Zabrodyn and G. P. Prokopov, *A difference scheme for two-dimensional unsteady problems of gas dynamics and computation of flow with a detached shock wave*, Z. Vyčisl. Matem. Mat. Fiz., 1 (1961), pp. 1020–1050. (In Russian.)

[3] A. Harten, P. D. Lax and B. van Leer, *On upstream differencing and Godunov-type schemes for hyperbolic conservation laws*, ICASE Report No. 82-5, 1982; SIAM Rev., 25 (1983), pp. 35–62.

[4] B. Engquist and S. Osher, *Stable and entropy satisfying approximations for transonic flow calculations*, Math. Comp., 34 (1980), pp. 45–75.

[5] S. Osher and F. Solomon, *Upwind difference schemes for hyperbolic systems of conservation laws*, Math. Comp. (1983), to appear.

[6] P. L. Roe, *The use of the Riemann problem in finite-difference schemes*, Lecture Notes in Physics, 141, Springer-Verlag, New York, 1981, pp. 354–559.

[7] ———, *Approximate Riemann solvers, parameter vectors and difference schemes*, J. Comp. Phys., 43 (1981), pp. 357–372.

[8] E. M. Murman, *Analysis of embedded shock waves calculated by relaxation methods*, AIAA J., 12 (1974), pp. 626–633.

[9] S. L. Hancock, private communication (1980).

[10] B. van Leer, *Towards the ultimate conservative difference scheme. V. A second order sequel to Godunov's method*, J. Comp. Phys., 32 (1979), pp. 101–136.

[11] ———, *Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection*, J. Comp. Phys. 23 (1977), pp. 276–299.

[12] G. D. van Albada, private communication (1981).

[13] G. D. van Albada, B. van Leer and W. W. Roberts, Jr., *A comparative study of computational methods in cosmic gas dynamics*, Astron. Astrophys., 108 (1982), pp. 76–84.

[14] P. L. ROE, private communication (1981).
[15] A. HARTEN AND P. D. LAX, *A random-choice finite-difference scheme for hyperbolic conservation laws*, SIAM J. Numer. Anal., 18 (1981), pp. 289–315.
[16] P. COLELLA, *Glimm's method for gas dynamics*, this Journal, 3 (1982), pp. 76–110.
[17] T.-P. LIU, *Quasilinear hyperbolic systems*, Comm. Math. Phys., 68 (1979), pp. 141–172.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

MULTIPLE STEADY STATES FOR 1-D TRANSONIC FLOW

PEDRO EMBID,† JONATHAN GOODMAN‡ AND ANDREW MAJDA§

Abstract. The existence of multiple steady states with the same farfield behavior is discussed for simple 1-D transonic model problems. These multiple solutions all have only entropy satisfying compressive steady shock waves. Only some of these solutions are stable in the time-dependent system and are accessible through physical time-dependent perturbations. This is demonstrated by some elementary explicit solutions in a scalar model problem. However, for a large class of initial data and large C.F.L. numbers, numerical experiments show that implicit schemes can converge to the physically unstable steady states and this phenomenon is analysed. The scalar model is also discussed as a very simple numerical test problem for implicit schemes with rich structure in both the steady state and time-dependent regimes.

Key words. transonic flow, multiple steady state, false stability

Introduction. One of the main objectives in 2-D transonic flow calculations is to compute the steady flow past a given airfoil at a fixed angle of attack, α, and with a prescribed subsonic free stream Mach number, M_∞. At the present time, most calculations of this type are performed within the framework of the potential flow or Euler equations with highly nonuniform stretched grids to concentrate many grid points near the airfoil. Most methods developed thus far for these steady state calculations are implicit so that no C.F.L. restrictions on the time steps are imposed for stability (see [1]). Implicit methods are used for two reasons: (1) large time steps can be used when the solution is near steady state and therefore slowly varying in time; (2) with the radically stretched grids, explicit schemes would require extremely small time steps to maintain stability.

We digress for the moment and consider the simplest nonlinear time-dependent mathematical model, the scalar autonomous O.D.E.

$$\frac{dy}{dt}=f(y), \qquad y(0)=y_0, \tag{1.1}$$

where $f(y)$ is a given smooth function and y_0 is some prescribed constant initial value. The steady state solutions of (1.1) are the zeros of $f(y)$, i.e., the points y_j, $j=1,\cdots,N$, where

$$f(y_j)=0. \tag{1.2}$$

Regarding (1.1) as a simple model for a physical process, from a practical viewpoint, one is interested not only in finding the steady state solutions but also the "physically realizable" stable steady states. For simplicity, we assume that all the zeros of $f(y)$ are simple, i.e., $f'(y_j)\neq 0$. By linearizing (1.1) about y_j, we see that these physically accessible stable steady states are the zeros, y_j, where

$$f'(y_j)<0 \tag{1.3}$$

* Received by the editors March 10, 1982, and in revised form September 29, 1982.

† Center for Pure and Applied Mathematics, University of California, Berkeley, California 94720.

‡ Stanford University, Stanford, California 94305.

§ Center for Pure and Applied Mathematics, University of California, Berkeley, California 94720.
The research of this author was supported by NASA Consortium Agreement NCA2-0R050-101 with the Computational Fluid Dynamics Branch of the Ames Research Center and by National Science Foundation grant MCS-8102360.

and for y_0 close to y_j, the solution of (1.1) adjusts rapidly to y_j and satisfies the estimate,

$$|y(t)-y_j|<c\,e^{f'(y_j)t}|y_0-y_j| \tag{1.4}$$

for some constant $c>0$. For those zeros y_j with $f'(y_j)>0$, the solution to (1.1) with y_0 close to y_j diverges from y_j at an exponential rate and these steady states for (1.1) are physically inaccessible and unstable.

Next, analogous to the computational transonic program sketched in the first paragraph, we consider the numerical problem of finding a steady state solution of (1.1). Typically, one would use Newton's method,

$$y_{N+1}=y_N-(f'(y_N))^{-1}f(y_N), \qquad y_0=y_0, \tag{1.5}$$

and we are guaranteed that if y_0 is close to y_j then

(1.6) The algorithm in (1.5) always converges rapidly to y_j whether y_j is a physically stable or unstable steady state.

The use of Newton's method for computing the steady states of the model in (1.1) is analogous to the use of large time step implicit methods for computing transonic steady states—physical time accuracy is completely ignored. In fact, implicit methods often approach Newton's method as $\Delta t\to\infty$ (see [16]). With the behavior in (1.6) in a trivial model problem, the following two questions arise:

(1.7)
(1) Given a fixed airfoil, angle of attack, α, and free stream Mach number, M_∞, can multiple steady states occur for the potential flow or Euler equations?
(2) If there are multiple steady states, can numerical methods converge to the physically unstable steady solutions?

Given the fact that both (1) and (2) occur in a context, it is important either to check the stability of the computed steady state by a posteriori methods or to develop a convergence criterion guaranteeing convergence to a physically stable steady state.

Recently, Jameson and Steinhoff [9], [10] have given very convincing numerical evidence that the phenomenon of (1.7)(1) occurs for the potential flow equations by producing three distinct solutions for symmetric Joukowski profiles with zero angle of attack for a range of free stream Mach numbers. To our knowledge, no calculations addressing the issue of (1.7)(2) have been performed, but we strongly suspect, by analogy with much simpler bifurcation problems, that all three solutions are not simultaneously stable—in fact, the symmetric solution is likely to lose its stability in a range of M_∞. However, all three have been numerically computed through a variety of implicit and multigrid methods ([9]) so there is a real possibility that the phenomenon in (2) of (1.7) also occurs in these calculations.

Our main objective here is a detailed investigation of the two questions raised in (1.7) in simpler 1-D transonic model problems. Computing solutions of the averaged 1-D duct equations (see (2.1)–(2.3)) has become a popular 1-D test problem [8], [11] for multidimensional numerical shock algorithms because one can construct explicit steady shock wave solutions with considerable smooth structure (and often resembling the pressure profiles near airfoils). In § 2, we study steady solutions of the 1-D duct equations where the area variation is confined to $0\leqq x\leqq 1$. We fix a constant supersonic state, (ρ_0, m_0, E_0), for $x \leqq 0$ and a subsonic state, (ρ_1, m_1, E_1), for $x > 0$. (This is the analogue in the duct case of prescribing M_∞ and α in the 2-D transonic case.) We ask the general question which is the analogue of (1.7)(1).

(1.8) Are there multiple steady states with *entropy satisfying* steady shock waves with the prescribed values (ρ_0, m_0, E_0) and (ρ_1, m_1, E_1) for $x \leqq 0$ and $x \geqq 1$ respectively?

In § 2, we give a complete answer to this question and give a simple recipe for constructing all the multiple steady states with a single standing shock wave in a given duct geometry with the above prescribed exit and entry values. In particular, for the example of a converging-diverging nozzle, there are often *two distinct* steady states with fixed transonic exit and entry conditions. One of these solutions has an "entropy satisfying" compressive steady shock in the expanding portion of the duct while the second solution has an "entropy satisfying" compressive steady shock in the contracting portion of the duct (see Fig. 2.1) for the isentropic case). The first of these solutions is well known and is expected to be stable. Since steady compressive shocks are never observed in the contracting portion of a duct, this second solution is expected to be unstable as regards perturbations with time-dependent solutions of the duct equations even though it is a bonafide steady state weak solution with a compressive, "entropy satisfying" shock. In fact, it is possible to construct general duct geometries with arbitrarily many multiple steady states of this type. Our analysis in § 2 is motivated by earlier work of Liu ([6]) on a related problem although our approach has the advantages of being simpler, explicit and "in the large" for the problems we study here. In [6], Liu considers "noninteracting" *time-dependent* solutions of the duct equations and finds, for example, three time-dependent "noninteracting" wave patterns in a contracting duct with the same asymptotic values at $t = 0$.

In § 3a, we describe an explicit scalar equation model and demonstrate very similar multiple steady state structure as we have described above for the 1-D duct equations—see Fig. 3a). (In fact, the reader interested in the ideas of our construction in § 2 without the algebraic details should first look at the analogous and much simpler construction in § 3.) This model problem has the form

$$u_t + (\tfrac{1}{2}u^2)_x = a(x)u, \qquad u(x, 0) = u_0(x), \tag{1.9}$$

where $a(x)$ is nonzero only for $0 \leqq x \leqq 1$. In § 3b, we determine some interesting explicit time-dependent solutions of (1.9) with structure and moving shock waves and use these exact solutions to develop a rigorous but elementary nonlinear stability analysis of the multiple steady state solutions of (1.9)—in fact, only the stability analysis for the nonlinear O.D.E. already discussed in (1.1)–(1.4) needs to be applied in our construction. Recently, Liu ([7]) has succeeded in rigorously proving the instability of a steady shock in a contracting duct for the much more difficult full system of 1-D duct equations by utilizing his modification of the Glimm scheme and wave interaction estimates. In § 4, we discuss a variety of numerical experiments with the model equation in (1.9). We have used two implicit upwind schemes, implicit modifications with $a(\rho) \neq 0$ of the two explicit upwind schemes of Engquist and Osher [3], [4] and the explicit MacCormack method in these calculations. Through numerical experiments, we demonstrate that there is a large range of C.F.L. numbers and a large set of initial data where the implicit schemes converge to an unstable multiple steady state. We call these "falsely stable" steady states since they are stable for the numerical method but not for the time-dependent physical model problem. Thus, the difficulty addressed in question (1.7)(2) already occurs for this very simple transonic model problem in (1.9).

We also advocate (1.9) as a simple model problem with rich spatial structure in shock solutions besides possessing multiple steady states. For "transonic" exit and entry conditions, the advantage of the model in (1.9) is that all variables have prescribed physical boundary conditions and the effects of additional numerical boundary conditions on the stability and the quality of computed solutions with the tested numerical methods at large C.F.L. numbers is relatively unimportant. We also recommend the use of the explicit solutions constructed in § 3b for time-dependent test calculations (see § 4). Yee, Beam and Warming ([11]) have demonstrated the striking effect of numerical boundary conditions on the stability and quality of calculations for implicit methods at large C.F.L. numbers for the 1-D duct equations where additional purely numerical boundary conditions are needed at the subsonic wall. Although we have not pursued this here, it also seems interesting to test the variety of 2-D multigrid methods for transonic flow and also the steady shock tracking method of [8] on solutions of the simple model problem in (1.9). Also, we have intentionally avoided any intrinsically 1-D or scalar methods for computing (1.9) such as via singular perturbation. Many additional details and results of computations are presented in our report [15].

2. Multiple steady transonic patterns for the duct flow. The equations that model the isentropic flow of a gas through a duct of variable area are

$$\rho_t + m_x = -\frac{A'(x)}{A(x)} m, \tag{2.1}$$

$$m_t + \left(\frac{m^2}{\rho} + p\right)_x = -\frac{A'(x)}{A(x)} \frac{m^2}{\rho}. \tag{2.2}$$

(The full gas dynamic equations are discussed in [15].)

Where ρ is the density, m is the mass flux density, p is the pressure and $A(x)$ is the area of the cross section of the duct at the point x. Also $p_\rho > 0$, $p_{\rho\rho} > 0$ and p and p/ρ tend to 0 with ρ. As a consequence the speed of sound $c = \sqrt{p_\rho}$ is a well-defined quantity. Finally we assume $A(x)$ is constant for $x \leqq 0$ and $1 \geqq x$ and we consider only solutions of (2.1)–(2.2) for which $m > 0$.

We are interested in finding time-independent solutions of (2.1)–(2.2) that take a certain fixed value upstream ($x \leqq 0$) and another fixed value downstream ($x \geqq 1$), the value upstream being supersonic ($u > c$) and the one downstream subsonic ($u < c$). These "steady transonic patterns" will exhibit in general a finite number of standing shocks, and there the Rankine–Hugoniot jump condition

$$m_L = m_R, \tag{2.3}$$

$$\frac{m_L^2}{\rho_L} + p_L = \frac{m_R^2}{\rho_R} + p_R \tag{2.4}$$

and the entropy condition

$$\frac{m_L}{\rho_L} - c_L > 0 > \frac{m_R}{\rho_R} - c_R \tag{2.5}$$

must be satisfied. Here the subscripts L and R indicate that the limiting values of the parameter from the left and the right along the shock are considered.

We are especially interested in steady transonic patterns exhibiting only one standing shock. In general these patterns are not uniquely determined by the values upstream and downstream and the idea behind the construction of these patterns is the following. Let $w_0=(\rho_0, m_0)$ and $w_1=(\rho_1, m_1)$ be the values fixed upstream and downstream respectively. Suppressing the time dependence in (2.1)–(2.3) we get a system of ordinary differential equations. Solve that system for $x \geqq 0$ with initial condition w_0 at $x=0$ and call the solution $w_L(x)$ "the branch emerging from the left". Similarly, solve the system for $x \leqq 1$ with initial condition w_1 at $x=1$ and call the solution $w_R(x)$ "the branch emerging from the right". In § 2a we will give sufficient conditions under which $w_L(x)$ can be defined for all $x \geqq 0$ and $w_R(x)$ for all $x \leqq 1$. Finally, let x_1 be any point in (0, 1) for which $w_L(x_1)$, $w_R(x_1)$ satisfy (2.3)–(2.5); then

$$w(x)=\begin{cases} w_L(x), & x<x_1, \\ w_R(x), & x>x_1, \end{cases}$$

defines a steady transonic pattern with values w_0 for $x \leqq 0$ and w_1 for $x \geqq 1$ and with only one standing shock. Moreover, it is clear that all such patterns can be obtained in this way. Thus, everything reduces to the problem build the left and right branches and to find all the possible points for which a jump from the left branch to the right one can be accomplished.

2a. Construction of steady smooth solutions. If we look for time-independent solutions of the system (2.1)–(2.2), it reduces to the system

$$m_x = -\frac{A'(x)}{A(x)}m, \tag{2.1'}$$

$$\left(\frac{m^2}{\rho}+p\right)_x = -\frac{A'(x)}{A(x)}\frac{m^2}{\rho}. \tag{2.2'}$$

After some straightforward algebraic manipulations we get the system

$$mA = \text{constant} = m_0A_0, \tag{2.1''}$$

$$\frac{m^2}{2\rho^2}+h(\rho)=\text{constant}, \tag{2.2''}$$

where $(\rho_0, m_0)=w_0$ is given at $x=0$ and $h(p)$ is an antiderivative of $p_\rho(\rho)/\rho$.

Equation (2.1″) says that the flux of mass is the same across any cross section of the duct. Equation (2.2″) is Bernoulli's law, it says that the total enthalpy is constant along the duct.

The quantity $m(x)$ can be computed from (2.1″). However, in order to compute $\rho(x)$ we must study in the phase plane (ρ, m) the curve

$$\frac{m^2}{2\rho^2}+h(\rho)=B=\text{constant}. \tag{2.6}$$

We call this curve the "Bernoulli curve" and the constant B the "Bernoulli constant".

From (2.6) we get $m=\sqrt{2\rho^2(B-h(\rho))}$. It is easy to see that m is a concave function of ρ and attains its maximum value m^* at a unique point ρ^*. Moreover (ρ^*, m_*) is the unique point of intersection (besides (0, 0)) of the Bernoulli curve and the "sonic line" $m=\rho c(\rho)$. The Bernoulli curve is divided into two branches, one contained in the supersonic region $m>\rho c(\rho)$ and the other contained in the subsonic region $m<\rho c(\rho)$.

Being a concave function of ρ, the Bernoulli curve intersects the line $m = m(x)$ at the most at two points, one of them in the supersonic region and the other in the subsonic region. This provides two possible choices for $\rho(x)$ and the choice will depend upon the region where the initial data is; for example, for the left branch we will choose the supersonic point and for the right branch the subsonic one.

So far we have seen how to construct a smooth solution.

As a final remark, this construction of smooth solutions is possible as long as the Bernoulli curve intersects the line $m = m(x)$, that is, if $m(x) \leqq m^*$, or equivalently, $m_0 A_0 \leqq m_* A(x)$ for all x. *Moreover, if* $m(x) < m^*$ *then the solution always remains either in the supersonic region or in the subsonic one.*

In the construction of the transonic patterns the next step is the determination of all the possible locations for the standing shock. We will do this in § 2b for the isentropic approximation of two reasons: first, because the two-dimensional graphical interpretation for the isentropic case is revealing; and second, because as long as the solution is smooth the entropy remains constant and when there is a jump the increase of entropy is only of the third order in the shock strength. All our constructions will generalize for the nonisentropic case but the graphical interpretation is lost (see [15]).

2b. Steady transonic patterns for the isentropic model. We recall that the Rankine–Hugoniot jump conditions for (2.1), (2.2) are given in (2.3), (2.4) and the entropy condition is given in (2.5).

The solutions we are looking for must take the value (ρ_0, m_0) for $x \leqq 0$ and (ρ_1, m_1) for $x \geqq 1$, (ρ_0, m_0) lies in the supersonic region and (ρ_1, m_1) lies in the subsonic one. Consider the Bernoulli curves going through (ρ_0, m_0) and (ρ_1, m_1) with corresponding Bernoulli constants $B_0 = m_0^2/2\rho_0^2 + h(\rho_0)$ and $B_1 = m_1^2/2\rho_1^2 + h(\rho_1)$, and let $(m_0)_*$ and $(m_1)^*$ be their respective maximum values when they are regarded as functions of ρ. If the left branch and the right branch are such that $m_L(x) < (m_0)_*$ and $m_R(x) < (m_1)_*$ for all x, then as we have already seen in § 2a the left branch will take its values in the supersonic region and the right branch will take them in the subsonic one, i.e. $m_L(x)/\rho_L(x) - c_L(x) > m_R(x)/\rho_R(x) - c_R(x)$ for all x and the entropy condition (2.5) will be automatically satisfied.

Next we are going to study the jump conditions (2.3), (2.4). We want to show that for any given supersonic state (ρ_L, m_L) there is a unique subsonic state (ρ_R, m_R) such that (2.3), (2.4) hold.

Consider the curve $m^2/\rho + p(\rho) = P$, where $P = m_L^2/\rho_L + p(\rho_L)$. This defines m as a function of ρ and it can be easily shown that this function is concave and that its unique maximum is in the sonic line $m = \rho c(\rho)$. Since (m_L, ρ_L) belongs to the curve and it is not the maximum (because it is in the supersonic region) then by concavity of the curve $m^2/\rho + p(\rho) = P$ it will intersect the line $m = m_L$ at two points: one is (ρ_L, m_L) and the other, (ρ_R, m_R) is in the subsonic region (Fig. 2.1). Clearly these two points satisfy (2.3), (2.4) and (2.5).

As we saw in § 2a, $(\rho_L(x), m_L(x))$ belongs to the branch of Bernoulli curve $m^2/2\rho^2 + h(\rho) = B_0$ that lies in the supersonic region and since this curve is concave this branch can be parametrized by m. If we associate to each point of this branch its unique shock state we get a "curve of shocks" (ρ_σ, m_σ) that is also parametrized by m. This curve contains all the shock states associated with the left branch $(\rho_L(x), m_L(x))$, i.e., all the points where the jump can occur. The main property of this curve of shocks is given in the following proposition which provides the key to the multiple steady state construction.

PROPOSITION 2.1. *The Bernoulli constant $B(\rho, m) = m^2/2\rho^2 + h(\rho)$ increases with m along the curve of shocks.*

The proof, via straightforward calculation, is given in [15].

Now we are able to carry out the construction of the steady transonic patterns. For $m(x)$ the construction is straightforward: for smooth flows we know that mA is constant and at any standing shock $m_L = m_R$, but since $A(x)$ is smooth it follows that mA is constant also across a standing shock. Therefore $m(x) = m_L(x) = m_R(x) = m_0A(0)/A(x)$ and $m(x)$ does not exhibit jumps along the duct. Finally let us remark that m_0 and m_1 cannot be arbitrarily prescribed since $m_0A(0) = m_1A(1)$.

The construction of $\rho(x)$ is as follows: assume for the moment that $B_0 > B_1$. Since the Bernoulli constant increases from 0 to B_0 with m along the curve of shocks there is a unique point (ρ_1^J, m^J) of intersection of the curve of shocks with the Bernoulli curve B_1. The associated supersonic state (ρ_0^J, m^J) lies on the Bernoulli curve B_0 by construction. Therefore the only way to jump from the Bernoulli curve B_0 to B_1 in the (ρ, m) phase space is through these points. On the other hand we know that values $(\rho_L(x), m_L(x))$ belong to the branch of the Bernoulli curve B_0 contained in the supersonic region and $(\rho_R(x), m_R(x))$ belong to the branch of the Bernoulli curve B_1 contained in the subsonic region. Therefore if we want to jump in the x-physical space from $\rho_L(x)$ to $\rho_R(x)$ we must find all the points x_j for which $m(x_j) = m^J$ and define

$$\rho(x) = \begin{cases} \rho_L(x), & x < x_j, \\ \rho_R(z), & x > x_j. \end{cases}$$

Finally let us remark that for $B_0 < B_1$ no construction is possible because the maximum value attained by the Bernoulli constant on the curve of shocks associated with B_0 is B_0, but $B_0 < B_1$ and the curve of shocks never intersects B_1.

Summarizing:

(a) If $B_0 < B_1$, no steady transonic pattern with fixed values at $x \leqq 0$ and $x \geqq 1$ exists.

(b) If $B_0 > B_1$, all the possible patterns are obtained by jumping from the left branch to the right at any point x_j for which $m(x_j) = m^J$. Therefore the geometry of the duct is incorporated in the last step of solving the equation $m(x_j) = m^J$.

We finish this section by showing the possible patterns appearing in the contracting–expanding nozzle (de Laval nozzle) (Fig. 2.1), where two different steady solutions of the duct equations with the same entry and exit conditions and each one exhibiting a single standing shock are displayed (in this case the equation $m(x_j) = m^J$ has only two roots).

3. A scalar model with multiple steady states.

Here we consider the simple scalar model problem,

$$u_t + (\tfrac{1}{2}u^2)_x = a(x)u, \qquad u(x, 0) = u_0(x), \tag{3.1}$$

where $a(x)$ varies for $0 < x < 1$ and vanishes for $x < 0$ or $x > 1$. Given a shock curve $(x(t), t)$, the Rankine–Hugoniot jump conditions for (3.1) are given by

$$\frac{dx}{dt} = \frac{u^L(x(t), t) + u^R(x(t), t)}{2} \tag{3.2}$$

while the entropy condition (see [5]) requires

$$u^L(x(t), t) > u^R(x(t), t), \tag{3.3}$$

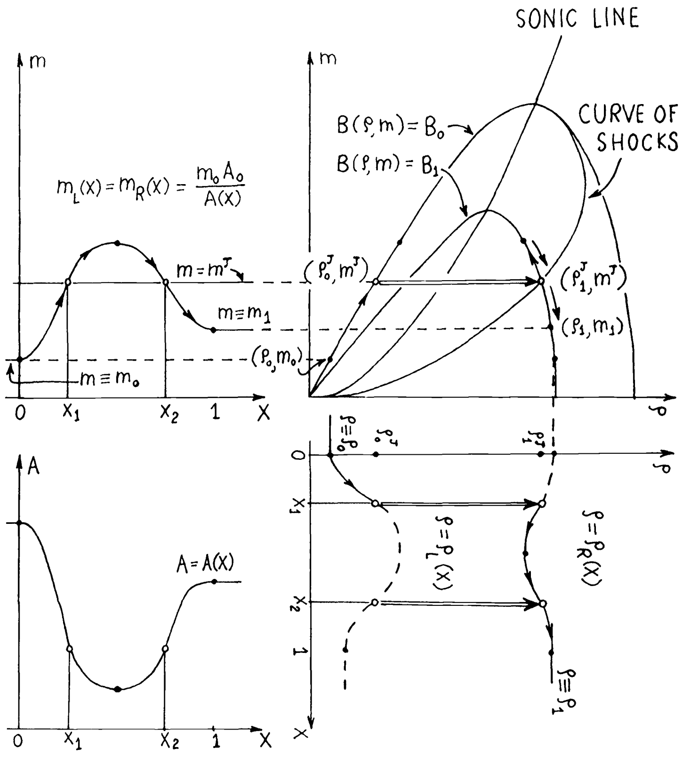


FIG. 2.1. *Converging–diverging nozzle showing the two possible solutions: one with a single standing shock at $x = x_1$ and the other with a single standing shock at $x = x_2$.*

where u^L, u^R are the limiting values from the left and right along the shock. We have two objectives in this section. First, we show that the simple equation in (3.1) already exhibits analogous multiple steady state transonic profiles with fixed exit and entry conditions. Then, we give a simple, explicit and rigorous nonlinear stability analysis for the multiple steady states in the model problem with suitably perturbed time-dependent solutions of (3.1).

3a. Multiple steady states for the model. We are interested in finding the steady state solutions of (3.1) with prescribed constant valus u_0, for $x \leqq 0$, and u_1, for $x \geqq 1$. In the model, it is a simple matter to compute the left and right branches of the smooth steady solutions of (3.1) emanating from u_0 and u_1 since they satisfy the O.D.E., $u_x = a(x)$. In fact, we have

$$u^L(x) = u_0 + A(x),\qquad u^R(x) = u_1 + A(x) - A_1, \tag{3.4}$$

where $A(x) = \int_0^x a(s)\,ds$ and $A_1 = \int_0^1 a(s)\,ds$. We remark that the condition

$$u^L(x) > u^R(x) \tag{3.5}$$

is valid for all x provided that the three constants, u_0, u_1, A_1 satisfy

$$u_0 > u_1 - A_1. \tag{3.6}$$

For the remainder of this section, we assume that the condition in (3.6) is satisfied for these constants. To find the steady state solutions in the model, we introduce the function

$$f(x)=\frac{u^L(x)+u^R(x)}{2}\equiv\frac{u_0+u_1-A_1}{2}+A(x). \tag{3.7}$$

If x_j is a root of $f(x_j)=0$, i.e., $A(x_j)=-(u_0+u_1)/2+A_1/2$, it follows from the jump conditions in (3.2) with $dx/dt\equiv 0$ that

$$u(x)=\begin{cases} u^L(x), & x<x_j,\\ u^R(x), & x>x_j,\end{cases} \tag{3.8}$$

is a steady solution of (3.1) and furthermore, by using (3.5) and (3.6), we see that $u(x)$ automatically stisfies the entropy condition in (3.3). Thus, we have the following:

Criterion for multiple steady states. Given u_0, u_1, A_1, satisfying (3.6) the transonic steady state solutions of (3.1) with fixed exit and entry values, u_0, u_1, and a single standing shock wave are in one to one correspondence with the distinct roots of the function, $f(x_j)=0$, from (3.7) via the formula in (3.8). Thus, several distinct roots of (3.7) imply that there are multiple steady states. In fact, we can choose $a(x)$ to guarantee an arbitrarily large number of multiple steady states satisfying the entropy conditions for fixed exit and entry conditions, u_0, u_1.

We illustrate this criterion with an example which is the analogue for the model in (3.1) of the converging–diverging nozzle discussed in § 2. In the model, $A(x)=\int_0^x a(s)\,ds$ has the same role as the cross sectional area in the duct case. Thus, for the analogue of the converging–diverging nozzle, $A(x)$ has a graph as depicted in Fig. 3.1. Of course, in this case, $A_1<A_{\max}$ and the equation, $f(x)=0$, has *two distinct* roots in [0, 1] provided that the constants, u_0, u_1, A_1 satisfy (see Fig. 3.1)

$$\frac{A_1}{2}<\frac{-(u_0+u_1)}{2}<A_{\max}-\frac{A_1}{2}. \tag{3.9}$$

To guarantee the entropy condition in (3.3), we also need to require (from (3.6))

$$u_0>u_1-A_1. \tag{3.10}$$

We claim that given the constant, A_1, determined from the geometry, there are many choices of u_0, u_1 satisfying (3.9) and (3.10). (For example, we set

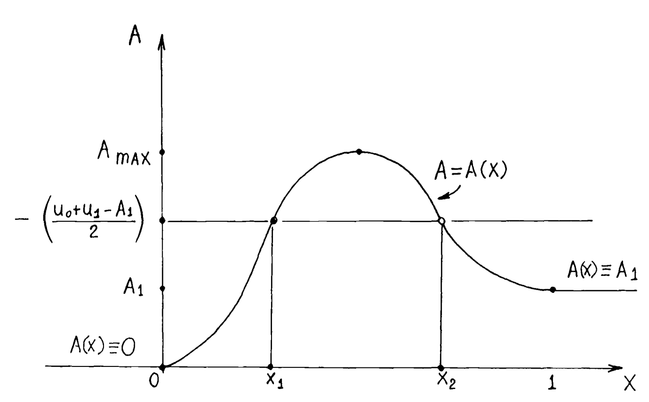


FIG. 3.1. *Plot of typical* $A(x)=\int_0^x a(s)\,ds$ *with analogous behavior as in contracting expanding nozzle.* x_i, $i=1,2$ *are the roots of* $0=f(x)=A(x)+(u_0+u_1-A_1)/2$.

$$u_0 = u_1 - A_1 + d \quad \text{with } d > 0 \tag{3.11}$$

so that (3.10) is automatically satisfied. For details, see [15].) See the graphs drawn in Fig. 3.2. This is the analogous behavior to that occurring in the converging–diverging nozzle discussed in § 2.

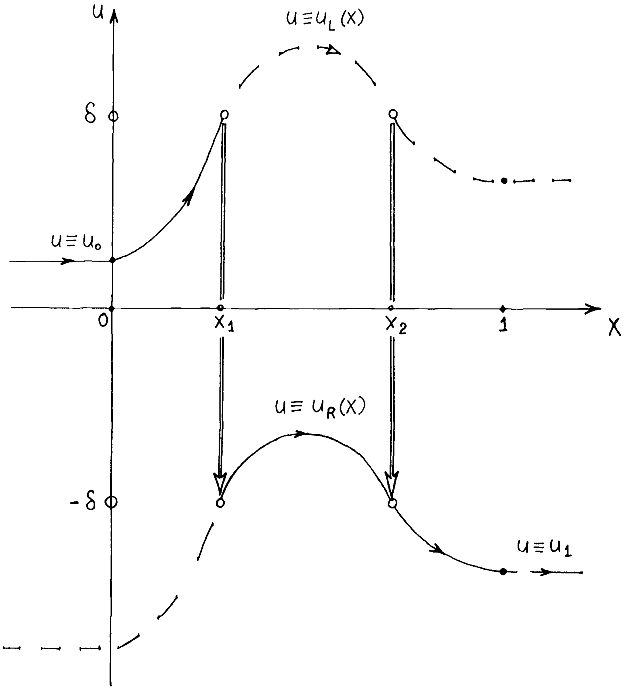


FIG. 3.2. *A typical case with two distinct steady states with the same exit conditions u_0, u_1 with a single entropy satisfying standing shock wave at either x_1 or x_2, respectively, for the model with $A(x)$ as in Fig.* 3.1.

3b. Explicit nonlinear stability analysis. For the case of a converging–diverging nozzle, it is well known that shocks in the diverging part of the duct are the only ones experimentally observed. This implies that the duct geometry is a determining factor in the stability of such transonic patterns. In the model problem, the analogous stability phenomena are expressed by the:

Conjecture. Standing shock waves at locations x_j with $a(x_j) < 0$ ($a(x_j) > 0$) are stable (unstable) against small time-dependent perturbations of (3.1).

Our purpose here is to give an elementary rigorous nonlinear stability analysis strongly supporting this conjecture and to build some interesting explicit time-dependent solutions of (3.1) with structure. For a fixed u_0, u_1, we assume that we have a finite number of multiple steady state solutions of (3.1) determined by the formula in (3.8), where x_j are the roots, $f(x_j) = 0$, from (3.7). We observe that $f'(x) = a(x)$, thus

$$f'(x_j) < 0 (>0) \quad \text{if and only if} \quad a(x_j) < 0 (>0). \tag{3.12}$$

We will not study general perturbations for (3.1) but instead explicitly analyse the time-dependent behavior of solutions of (3.1) defined by initial data with the form,

$$u^0(x, \alpha) = \begin{cases} u^L(x), & x < \alpha, \\ u^R(x), & x > \alpha, \end{cases} \tag{3.13}$$

where α is a parameter. These are initial data with a moving shock wave located initially at $x=\alpha$—of course, for $\alpha = x_j$, we have the steady state solutions from (3.8). With the above initial data, we can construct the exact solution of the time-dependent equation in (3.1) by using only a single quadrature. The idea is that $u^L(x)$, $u^R(x)$ are both exact smooth solutions of the model equation in (3.1) so that we only need to determine the location of the shock at later times to build the explicit solution to (3.1). Thus, we claim that a time-dependent solution of (3.1) with the initial data from (3.13) exists with the form,

$$u(x,t,\alpha)=\begin{cases} u^L(x), & x<x(t,\alpha), \\ u^R(x), & x>x(t,\alpha). \end{cases} \tag{3.14}$$

The shock location is determined by requiring that the Rankine–Hugoniot jump conditions from (3.3) are satisfied across $x(t,\alpha)$. Thus, $x(t,\alpha)$ should satisfy the autonomous nonlinear O.D.E.,

$$\begin{aligned} \frac{dx(t,\alpha)}{dt} &= \frac{u^L(x(t,\alpha))+u^R(x(t,\alpha))}{2} \equiv f(x(t,\alpha)), \\ x(t,\alpha)|_{t=0} &= \alpha, \end{aligned} \tag{3.15}$$

where $f(x)$ was defined in (3.7). We recall that the solution of (3.15) for $\alpha \neq x_j$ is determined by the quadrature formula,

$$t(x,\alpha)=\int_\alpha^x \frac{dx}{f(x)}. \tag{3.16}$$

By reversing the above steps, we see that provided (3.15) is satisfied, the function in (3.14) defines an explicit solution of (3.1) with the initial data in (3.13). Furthermore, since (3.6) guarantees (3.5), the entropy condition in (3.3) is automatically satisfied for this moving shock wave. The stationary points of the autonomous O.D.E. in (3.15) are the points, x_j, with $f(x_j)=0$ and these are precisely the locations of the steady shocks in (3.8). Thus, we have reduced the nonlinear stability analysis for the special data in (3.13) to that for the scalar autonomous O.D.E. discussed in the introduction.

Let's illustrate this explicit nonlinear stability analysis for the two distinct steady state solutions discussed at the end of § 3a and depicted in Fig. 3.2 (the analogue of the converging–diverging nozzle). Recall that at x_1, $f(x_1)=0$ but $f'(x_1)=a(x_1)>0$ and intuitively, the corresponding steady solution from (3.8) is unstable. We have $f(x)<0$ for $x<x_1$ so from (3.15) and our remarks in the introduction, if we choose the initial shock location, $\alpha<x_1$, the explicit shock wave solution in (3.14) moves to the left becoming after a *finite time*, *T*, the time-dependent steady shock with constant velocity given by

$$u(x,t)=\begin{cases} u_0, & x<\left(\dfrac{u_0+u_1-A_1}{2}\right)(t-T), \\ u_1-A_1, & 0\geqq x>\left(\dfrac{u_0+u_1-A_1}{2}\right)(t-T) \end{cases}$$

for $t>T$. On the other hand, if we choose α with $x_1<\alpha<x_2$, $f(x)>0$ for $x_1<x<x_2$ and from (3.15) the explicit solution from (3.14) approaches the steady state of (3.8) with $f(x_2)=0$ (where $f'(x_2)0=a(x_2)<0$) asymptotically as $t\to\infty$. The rate of adjustment of the shock location, $x(\alpha, t)$, to x_2 can be estimated for large t by

$$|x(\alpha,t)-x_2|\leqq c_0\, e^{a(x_2)t}.$$

Thus, we have rapid adjustment of these perturbed shock profiles to the steady state with shock location at x_2. The same remark applies if $\alpha > x_2$ since $f(x) < 0$ for $\alpha > x_2$ and we have rapid adjustment to this same steady profile with shock located at x_2. Therefore, for these special initial data, we have explicitly demonstrated the conclusion of the conjecture.

In fact, the reader can see that by appealing to the elementary stability properties of scalar O.D.E.'s ((1.2)–(1.4)), the above arguments supporting the conjecture are valid for general duct geometries with many multiple steady states for the model problem. By using the quadrature formula in (3.16), one can even discuss nonlinear stability for multiple roots x_j with $f(x_j) = 0, f'(x_j) = 0$. We omit this additional straightforward analysis here.

Finally, we remark that our analysis does not rigorously prove the stability of the shock wave located at x_1 to *all* small perturbations. However, this analysis does support the conjecture and we have rigorously demonstrated the instability of the steady shock at x_1.

4. Numerical results on the scalar model problem. In this section we describe our numerical experiments on the scalar model problem:

$$\begin{aligned} &u_t + \tfrac{1}{2}(u^2)_x = a(x)u, \qquad 0 < x < 1, \\ &u_0 = u(0, t) > 0, \qquad u_1 = u(1, t) < 0. \end{aligned} \tag{4.1}$$

We exhibit the falsely stable, but entropy satisfying, steady solutions to the difference schemes and show that these nonphysical solutions have a sizeable domain of attraction under the numerical relaxation process. We illustrate the use of (4.1) as a model problem for numerical computations by solving (4.1) in cases both steady and unsteady, where explicit solutions are known (see § 3).

The problem (4.1) differs from (3.1) only by the imposition of boundary conditions at $x = 0$ and $x = 1$. However, all solutions in § 2 were constant for $x > 1$ or $x < 0$ so they are also solutions of (4.1). Our numerical test problem (4.10) does not have $a'(x)$ going smoothly to zero at $x = 0$ or $x = 1$. This could be a source of numerical problems if the numerical boundary treatment involved extrapolating u or a outside the computational domain $0 \leqq x \leqq 1$, where u and a are smooth. We have not used such boundary schemes.

A difference approximation to the time-dependent problem (4.1) can be used either to compute the time evolution of the system or as a relaxation method for solving the steady state difference equations. In the second case we take for efficiency the largest time steps for which the method is stable. However, if the time steps are too large to resolve the time evolution of (4.1) (see Fig. 4.5), then the stability properties of a steady state solution may change. A steady solution to (4.1) which satisfies the entropy condition but is unstable to small perturbations in physical time may be a stable steady solution to the numerical time step iteration. We call this phenonemon *false stability* and demonstrate that it can occur for large time step implicit methods with Courant number as small as 23. As mentioned in the introduction, we believe that analogous numerical phenomena are occurring in the multiple steady state solutions to the steady two-dimensional transonic potential equations in various regimes of free stream Mach number, air foil shape and angle of attack as reported in [9] and [10]. By contrast [13] reports spurious multiple solutions to the nonlinear difference equations which are not related to multiple solutions of a differential equation.

We believe that (4.1) is a useful model problem because it is simple and yet has solutions with enough interesting structure to be a genuine test of a numerical scheme (see in particular our remarks on the first order upwind scheme). Like the inviscid Burgers equations, (4.1) has many explicit analytical solutions both steady and unsteady. With boundary conditions satisfying the inequalities in (4.1) there are only inflow boundaries so no special boundary conditions are needed. On the other hand, steady solutions to (4.1) are not piecewise constant and may even have smooth sonic point transitions, depending on the choice of $a(x)$, u_0 and u_1.

In the rest of this section we first describe three numerical methods we have used, a first and second order accurate upwind scheme and the explicit MacCormack scheme. We present steady solutions of the difference schemes and compare them with the exact solution computed from (3.16) using higher order Gauss quadrature; then we present the falsely stable steady states and show how the domain of attraction of a falsely stable solution depends on the Courant number. Next we examine the time accuracy of the schemes for the full time-dependent problem (4.1) in cases where exact solutions are given in § 3. Finally we analyze the large boundary errors of the upwind schemes and show that they are not due to improper boundary treatment but to the exceptional low accuracy of the first order scheme near a sonic point. This leads to a simple modification of the first order scheme, analogous to the box scheme, which eliminates the sonic point problem and gives second order accurate steady profiles.

I. *The methods.*

a) *The first order upwind scheme.* We use the form of upwind differencing proposed by Engquist and Osher ([3]) together with backward implicit Euler time differencing to get

$$u(x,t+k)=u(x,t)-\frac{\lambda}{2}D^u[u^2(x,t+k)]+ka(x)u(x,t+k), \tag{4.2}$$

where D^u is their upwind scheme with switching through zero. Since this is a three point scheme it could be used at every interior grid point. With all inflow boundaries we simply gave the specified boundary values at the boundary grid points.

The implicit equations (4.2) are a coupled system of nonlinear equations with a tridiagonal Jacobian matrix $J(u(\cdot,t+k),\lambda,k)$. Rather than solving the system exactly at each time step we took only one Newton iteration per time step using $u(\cdot,t)$ as the initial guess. This linearized implicit scheme takes the form

$$J(u(\cdot,t),\lambda,k)\cdot(u(\cdot,t+k)-u(\cdot,t))=-\frac{\lambda}{2}D^u u^2(\cdot,t)+kau(\cdot,t). \tag{4.3}$$

In our experience the linear system $Ju=0$ can always be solved without pivoting using a bidiagonal LU factorization of J. This makes (4.3) an efficient method. As was pointed out by Engquist and Osher, one advantage of (4.2) is that the resulting Jacobian J is a continuous function of u so that Newton's method will converge quadratically. Godunov's version of the upwind scheme does not have this property.

b) *The second order upwind scheme.* The second order three point one-sided difference formula is

$$\begin{aligned}\frac{df}{dx}&=\frac{1}{h}\left(-\frac{1}{2}f(x+2h)+2f(x+h)-\frac{3}{2}f(x)\right)+O(h^2)\\&=\frac{1}{h}D^{++}f(x)+O(h^2).\end{aligned} \tag{4.4}$$

As in the first order method we retain conservation form by using (D^{--} is the backward analogue of D^{++})

$$(u^2)_x \cong \frac{1}{h}[D^{++}(\psi(u)u^2)+D^{--}(\phi(u)u^2)] = \frac{1}{h}D^{uu}(u^2), \tag{4.5}$$

(with $\psi(u)=\frac{1}{2}(\operatorname{sgn} u+1)$, $\phi(u)=-1\psi(u)$. Since this is a five point scheme it could not be used at the interior grid points adjacent to boundary points. At these points we used the first order upwind difference. Other second order one-sided schemes with more complex switches are developed in [4].

To achieve large Courant numbers we want a linearized implicit method like (4.3) but we want to avoid factoring the pentadiagonal J' resulting from (4.5). Following a suggestion of R. Warming we tried using the matrix J from (4.3),

$$J(u(\cdot,t),\lambda,k)\cdot(u(\cdot,t+k)-u(\cdot,t)) = -\frac{\lambda}{2}D^{uu}u^2(\cdot,t)+kau(\cdot,t).$$

This gave an improved but still limited stability bound on λ. We finally achieved a very large λ bound by using an ad·hoc tridiagonal matrix $M(\alpha)$. If J'' is the tridiagonal restriction of J' gotten from J' by simply setting the off-diagonal elements to zero, then

$$M(\alpha)=\alpha J''(u)+(1-\alpha)J(u), \tag{4.6}$$

$$M(\alpha)(u(\cdot,t+k)-u(\cdot,t)) = -\frac{\lambda}{2}D^{uu}u^2(\cdot,t)+kau(\cdot,t) \tag{4.7}$$

was stable with $\lambda=600$, $\alpha=0.45$. As with the first order scheme, pivoting was never needed.

Both (4.3) and (4.7) are in "Δ form" (see Beam and Warming [12]) so that the steady solution, if it exists, is independent of λ.

c) *The explicit MacCormack scheme.* With $v(x,t+k)$ defined by

$$v(x,t+k)=u(x,t)-\frac{\lambda}{2}D^{+}u^2(x,t)+ka(x)u(x,t) \tag{4.8}$$

we maintain second order accuracy and get a completely explicit method

$$\begin{aligned} u(x,t+k) &= u(x,t)-\frac{\lambda}{4}[D^{+}v^2(x,t+k)+D^{-}u^2(x,t)] \\ &\quad +k\frac{a(x)}{2}[u(x,t)+v(x,t+k)]. \end{aligned} \tag{4.9}$$

This is MacCormack's explicit method modified slightly to take into account the additional term $a(x)u$ in (4.1). For linear constant coefficient problems (4.8), (4.9) is stable when k is small enough and we had no problems computing with $\lambda=0.5$. At the right boundary we used (4.8) to predict $v(1,t+k)$ rather than using the known boundary value u_1. The local third order truncation error is preserved in this way.

In the above we gave λ values which were the ratio of the time step size k to the space step h. The Courant number is this ratio nondimensionalized by multiplying by the fastest wave speed in the problem. In our test problem (4.10) this is $\frac{1}{2}$ since the largest u value is $u=1$ at $x=0$. Thus, we ran the explicit MacCormack scheme at Courant number $\frac{1}{4}$ and scheme (4.7) at Courant number 300.

II. *Numerical results and discussion.* For a specific test problem we used (4.1) with

$$a(x)=6x-3, \quad u_0=1, \quad u_1=-0.1. \tag{4.10}$$

Using the analysis from § 3 we see that there are two entropy satisfying steady solutions. One is stable in time with a standing shock at $x_1=0.18$ and one has an unstable standing shock at $x_2=0.82$. Here, unlike the example discussed in § 3, x_1 is stable and x_2 is unstable since the sign of a is reversed. In all our runs we took initial data to follow the solution branches $u(x)=u_l(x)$ for $x \leqq x_0$ $u(x)=u_r(x)$ for $x>x_0$ (see (3.4)). This guarantees that the exact solution to the time-dependent problem always follows these solution branches with a time-dependent jump location $x(t)$. The right boundary value $u_1=-0.1$ is close to the sonic point $u=0$ and this causes problems for the upwind schemes.

a) *Accuracy of numerical approximations to the time stable steady solution.* We computed the steady profiles by taking the initial jump $x_0=x_1$ and marching in time until the convergence criterion,

$$\sup_x |u(x,t+k)-u(x,t)| \leqq C_{\text{conv}},$$

was satisfied. We found little difference in the solution when C_{conv} ranged between 10^{-5} and 10^{-10} or when the Courant number ranged from 0.1 to 0.75 to 500 for the explicit and implicit schemes, respectively.

For the first order upwind scheme (4.3) the accuracy is low (see Fig. 4.1). This should be contrasted with the high accuracy of the scheme for steady solutions of Burgers' equation (Engquist and Osher [9]) which we regard as somewhat accidental. However, the scheme being monotone, there are no numerical oscillations around the shock.

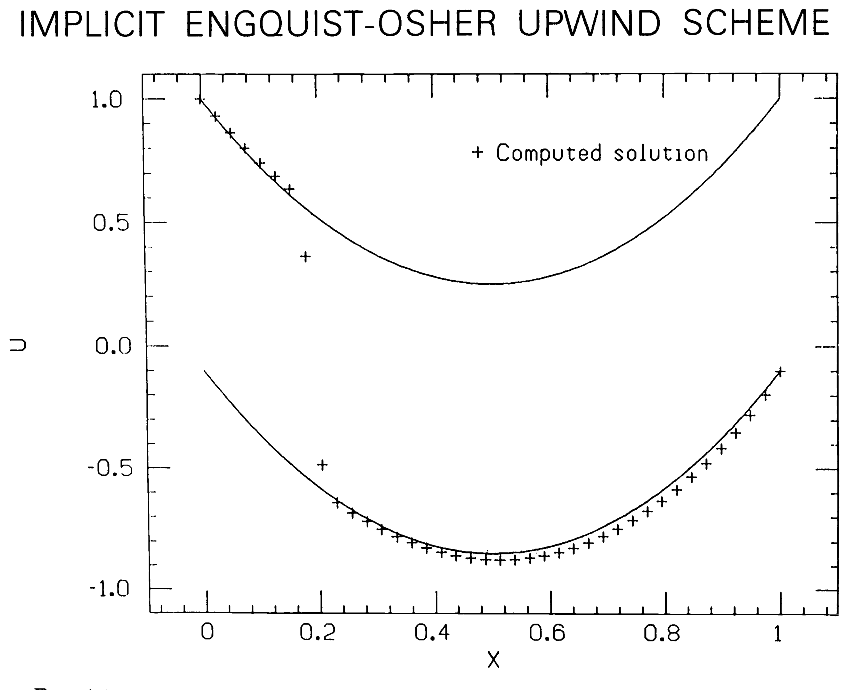


FIG. 4.1. *Computer stable steady profile, implicit Engquist–Osher scheme,* 40 *points.*

The second order upwind scheme (4.7) is not monotone so we should not be surprised that the shock profile is not monotone. One point of overshoot on either side of the shock is characteristic of the method (see Fig. 4.2). The accuracy is much better than the first order scheme except for the large error at the right boundary caused by the proximity of the boundary value $u = -0.1$ to the sonic point $u = 0$. We give a discussion and remedy at the end of the section.

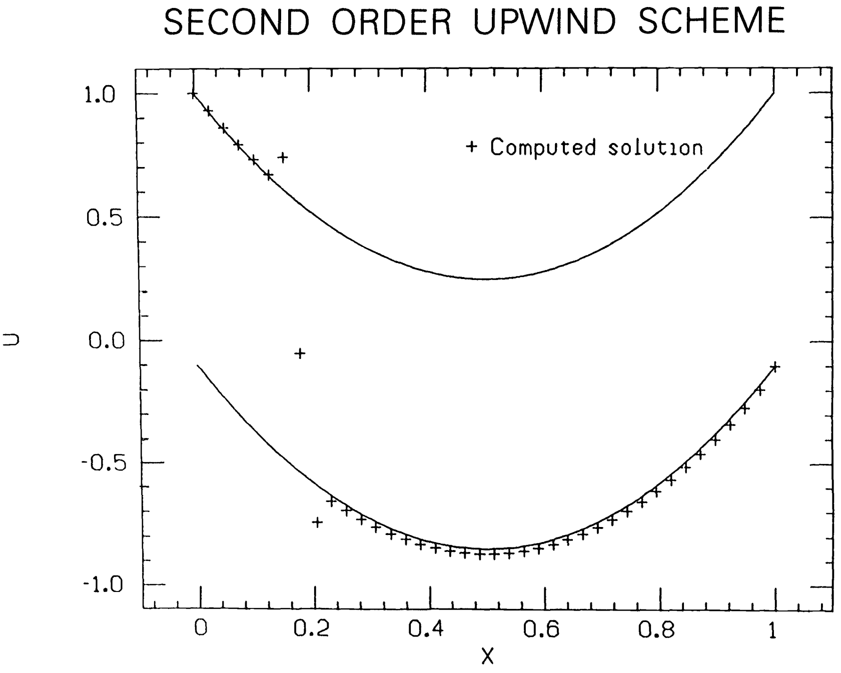


FIG. 4.2. *Computed stable steady profile, second order upwind scheme* 40 *points.*

The explicit MacCormack scheme is also accurate with more than 10 points (see Fig. 4.3), but there may be oscillations around the shock. We note that this is a particularly severe test of the scheme since we used no extra artificial damping.

b) *Convergence to falsely stable steady solutions.* We took initial data with a jump at

$$x_0 = dx_1 + (1-d)x_2.$$

We report Courant numbers, c, with $c = \lambda/2$, since the fastest wave speed is $\frac{1}{2}$ which occurs at the right boundary where $u = 1$. We found, for example, that using the second order upwind scheme (4.7) with $\alpha = 0.45$, $c = 300$ and $0 \leqq d \leqq 0.3$ the numerical solution would converge to the unstable steady solution (see Fig. 4.4) with a shock at x_2. These falsely stable solutions are accurate approximations to exact entropy satisfying but physically unstable solutions of (4.1).

For the first order scheme (4.3) we did systematic experiments with false stability. When the initial data was not far from the unstable solution we found three ranges of Courant number. For $c < c_i$ the numerical solution would converge to the physically stable solution. For $c_i < c < c_f$ the time integration became unstable. See Fig. 4.5 for signs of the onset of this instability. For $c > c_f$ we got convergence to the physically unstable steady state. Table 1 gives experimental values for c_i and c_f as functions of d.

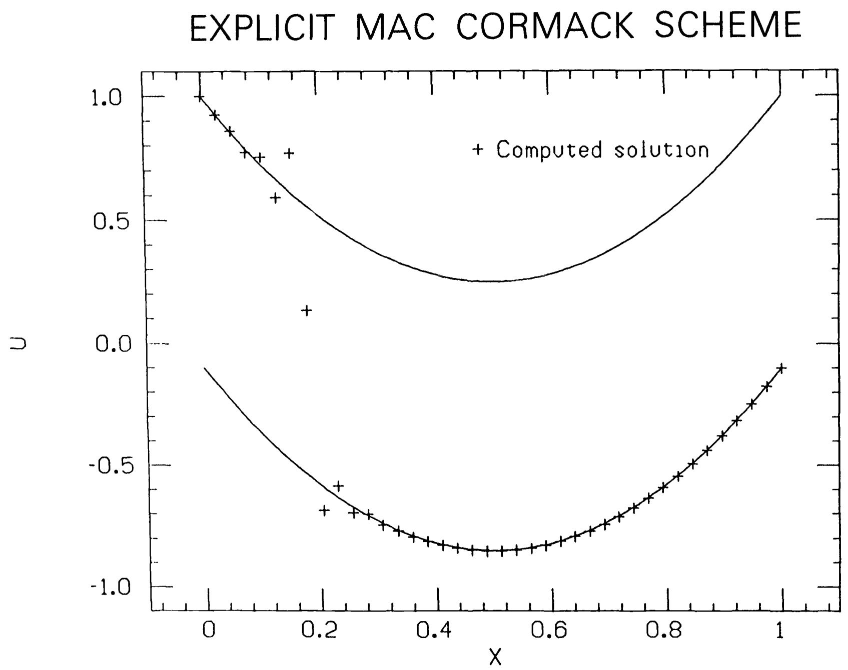


FIG. 4.3. *Computed stable steady profile, MacCormack scheme,* 40 *points.*

TABLE 1

d	0	0.1	0.2	0.3	0.35	0.4
c_i	3.1	3.1	3.7	4.4	6.1	12
c_f	22.5	23	33	60	143	*

* Could not get convergence to the unstable steady state.

c) *Accuracy of time-dependent solutions.* With an initial shock at $x_0 = 0.7$ (close to the unstable standing shock location) we computed solutions of (4.1) with time steps small enough to resolve the time evolution. First we report some results of time-dependent calculations given in detail in [15]. With 20 points and Courant number $c = \frac{1}{4}$ we observed that the upwind scheme (4.3) had roughly the correct time-dependent shock position but, as for the steady profiles, low overall accuracy. Engquist and Osher report [4] that for Burgers' equation moving shocks are more smeared than standing shocks. Here we did not observe any extra smearing in the moving shocks. The smearing is less than that of a typical monotone first order scheme. In the same situation (20 points, $c = \frac{1}{4}$) the explicit MacCormack scheme did rather poorly with large oscillations on one side of the shock. These oscillations disappeared as the solution approached the steady state. We repeat that this is a particularly severe test of MacCormack's scheme since we added no extra artificial dissipation.

In Figs 4.5, 4.6, 4.7 we show the upwind scheme with $c = 3$ and 40 points. This value of c is near to the onset of the instability reported in Table 1. The instability is localized around the shock front since the scheme is stable for smooth solutions.

d) *Upwind differencing at a sonic point.* As a demonstration that the error at the right endpoint is due to proximity to a sonic point, and not due to the numerical boundary for upwind schemes, we computed the profiles for $0 \leqq x \leqq 0.9$ so that the

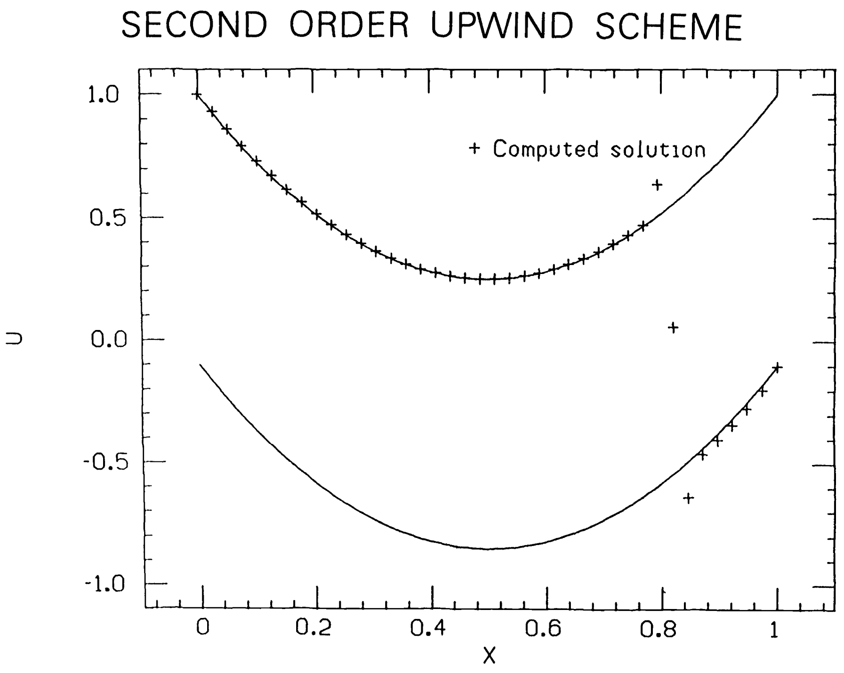


FIG. 4.4. *Computed falsely stable steady profile, second order upwind scheme,* 40 *points.*

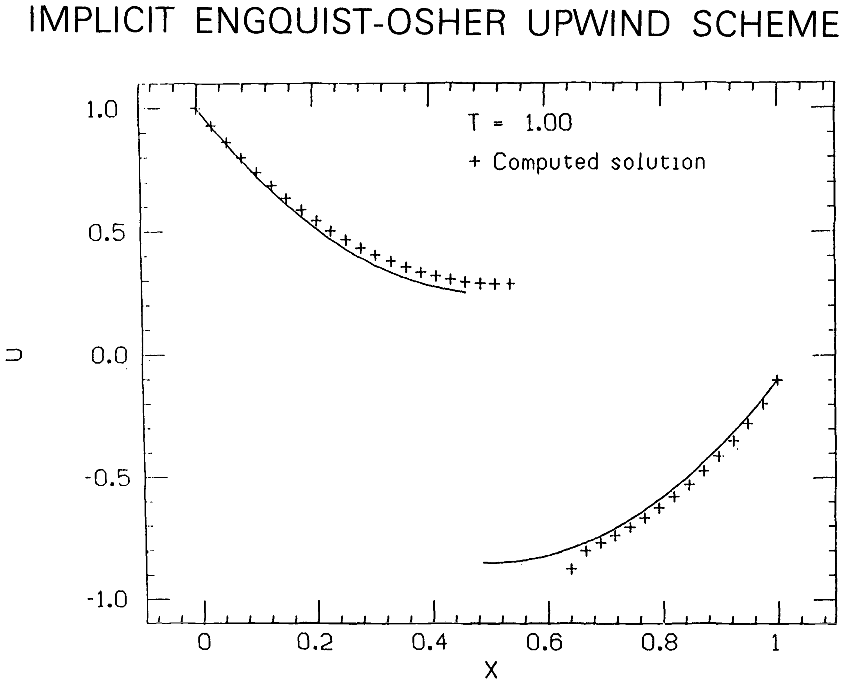


FIG. 4.5. *Time-dependent computations, implicit Engquist–Osher scheme, Courant number* 3.0, 40 *points.*

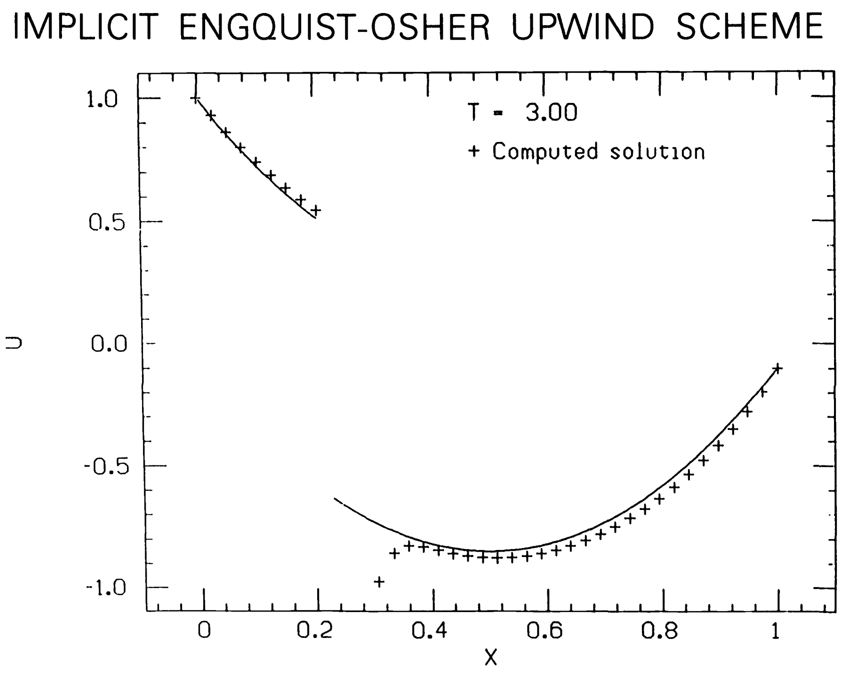


FIG. 4.6. *Time-dependent computations, implicit Engquist–Osher scheme, Courant number* 3.0, 40 *points.*

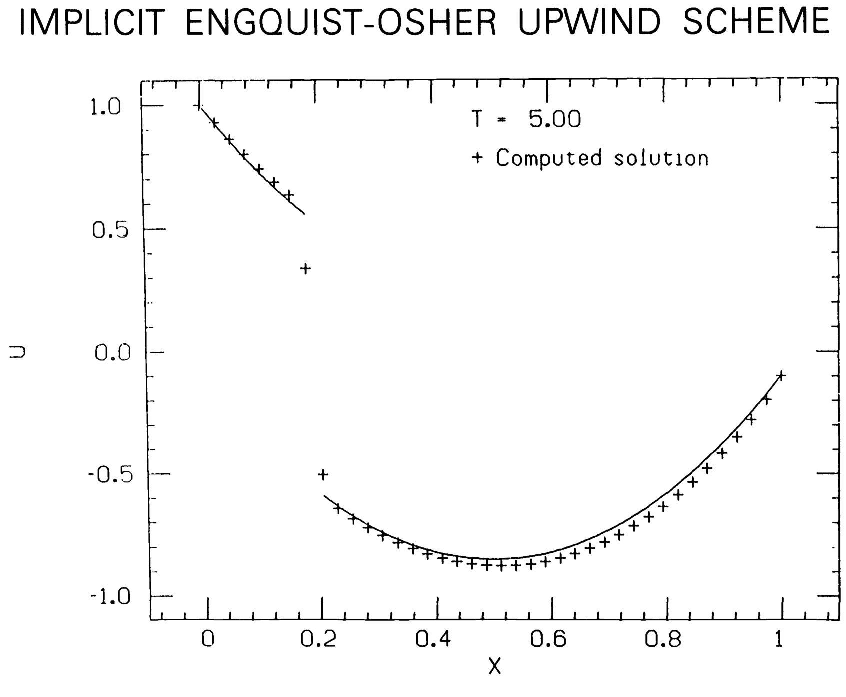


FIG. 4.7. *Time-dependent computations, implicit Engquist–Osher scheme, Courant number* 3.0, 40 *points.*

right boundary value would be far from the sonic point. Figs. 4.8, 4.9 show that the large boundary errors disappeared in this calculation. In [15], we give a more elaborate discussion of this error, another remedy and also many other computational results.

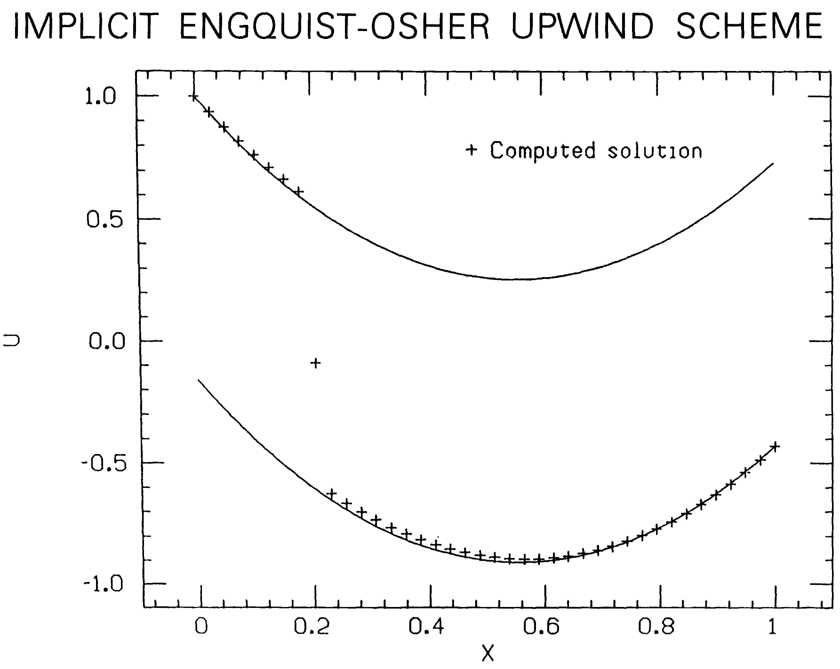


FIG. 4.8. *Computed steady profile with nonsonic boundary conditions, implicit Engquist–Osher scheme, 40 points.*

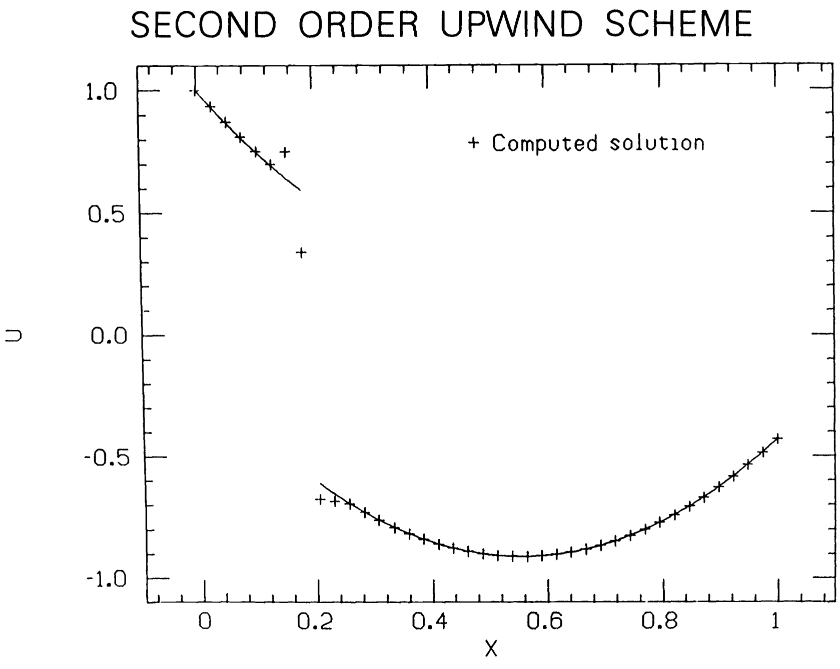


FIG. 4.9. *Computed steady profile with nonsonic boundary condition, second order upwind scheme, 40 points.*

Acknowledgments. The authors recognize the influence of the work of T. P. Liu on some of the results presented here and also thank him for a helpful conversation regarding his work in [7] which lead to our explicit results in § 3b. We also thank R. Warming and P. Colella for several interesting comments and their constructive criticism.

BIBLIOGRAPHY

[1] AIAA, *Computational Fluid Dynamics Conference Proceedings*, CP814, Palo Alto, CA, June 1981.

[2] R. COURANT AND K. O. FRIEDRICHS, *Supersonic Flow and Shock Waves*, Springer-Verlag, New York, 1948.

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

[4] ———, *One-sided difference approximations for nonlinear conservation laws*, Math. Comp., 36 (1981), pp. 321–351.

[5] P. D. LAX, *Hyperbolic Systems of Conservation laws and the Mathematical Theory of Shock Waves*, CBMS Regional Conference Series in Applied Mathematics, 11, Society for Industrial and Applied Mathematics, Philadelphia, 1973.

[6] T. P. LIU, *Transonic gas flow in a variable area duct*, Arch. Rat. Mech. Anal., to appear.

[7] ———, *Stability of transonic flows along a nozzle*, to appear.

[8] G. R. SHUBIN, A. B. STEPHENS AND H. M. GLAZ, *Steady shock tracking and Newton's method*, Appl. Math. Branch, Rep. 33, Naval Surface Weapons Center, Silver Spring, MD, 1980.

[9] J. STEINHOFF AND A. JAMESON, *Multiple solutions of the transonic potential flow equation*, AIAA #81-1019, pp. 347–353, 1980.

[10] ———, *Nonuniqueness in transonic potential flows*, Proc. GAMM Specialist Workshop for Numerical Methods in Fluid Mechanics, Stockholm, September, 1979.

[11] H. C. YEE, R. M. BEAM AND R. F. WARMING, *Boundary approximations for implicit schemes for the one-dimensional inviscid equations of gas dynamics*, AIAA J., 20 (1982), pp. 1203–1211.

[12] R. M. BEAM AND R. F. WARMING, *An implicit finite difference algorithm for hyperbolic systems in conservation form*, J. Comp. Phys., 22 (1976), pp. 87–110.

[13] G. R. SHUBIN, A. B. STEPHENS, H. M. GLAZ, A. B. WARDLAW AND L. B. HACKERMAN, *Steady shock tracking, Newton's method, and the supersonic blunt body problem*, this Journal, 3 (1982), pp. 127–144.

[14] W. D. GROPP, *A test of moving mesh refinement for* 2-D *scalar hyperbolic problems*, this Journal, 1 (1980), pp. 191–197.

[15] P. EMBID, J. GOODMAN AND A. MAJDA, *Multiple steady states for* 1-D *transonic flow*, CPAM Rep. 69, Univ. California, Berkeley, February 1982.

[16] W. R. BRILEY AND H. MCDONALD, *On the structure and use of linearizèd block implicit schemes*, J. Comp Phys., 34 (1980), pp. 54–73.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

ADDITION OF POINTS TO GAUSS-LAGUERRE QUADRATURE FORMULAS*

D. K. KAHANER,† J. WALDVOGEL‡ AND L. W. FULLERTON§

Abstract. The Gauss-Laguerre quadrature formula is defined by

$$If=\int_0^\infty e^{-x}f(x)\,dx \simeq \sum_{i=1}^{n} \alpha_i^{(n)} f(\xi_i^{(n)}),$$

where the numbers $\alpha_i^{(n)}$ and $\xi_i^{(n)}$ are weights and nodes. A common method of estimating the error of this rule is to evaluate the quadrature rule for two different values of n and to then compare the difference in the answers. Unfortunately, none of the nodes are in common for the two different quadrature rules, and so the function must be evaluated at each separate node.

We investigate in this paper the addition of points to the Gauss-Laguerre rule such that the new points are real, lie in the interval of integration, and the associated weights are positive. Such rules enable one to estimate economically the error of quadrature, because the function values at the Gauss-Laguerre abscissae are reused. The weights and nodes for some suitable low-order formulae are given in Table 2.

Key words. automatic quadrature, adaptive quadrature, extrapolation

1. Introduction.

Many practical applications require the evaluation of integrals of the form

$$If=\int_0^\infty e^{-x}f(x)\,dx. \tag{1.1}$$

When $f(x)$ is a well-behaved function, the best known method for estimating this integral is by the use of Laguerre-Gauss quadrature [1]: we approximate If by the sum

$$Q_nf=\sum_{i=1}^{n} \alpha_i^{(n)} f(\xi_i^{(n)}) \simeq If. \tag{1.2}$$

The numbers $\alpha_i^{(n)}$, $\xi_i^{(n)}$, called weights and nodes, are determined by the requirements that $Q_nf=If$ whenever f is a polynomial of degree $2n-1$. These nodes and weights have been extensively tabulated [2], or may be computed directly [7]. For many functions, Q_nf provides a good estimate of If, but for practical purposes this number by itself is not sufficient because it is not accompanied by an error estimate. A common approach is to compute Q_nf for two different values of n and utilize the difference or something derived therefrom to estimate the error. Unfortunately, the set of nodes $\{\xi_i^{(n)}\}$ is different for each n; there are no points in common. In this paper we provide new formulas of the type

$$K_{n,k}f\equiv \sum_{i=1}^{n} A_i^{(n,k)} f(\xi_i^{(n)}) + \sum_{j=1}^{k} B_j^{(n,k)} f(x_j^{(n,k)}) \simeq If. \tag{1.3}$$

We call $K_{n,k}f$ an extended Gauss-Kronrod quadrature rule. These formulas make maximum use of old function values $f(\xi_i^{(n)})$ but augment them with k new evaluations $f(x_j^{(n,k)})$. When no confusion can result we drop the superscripts. In analogy with (1.2) the new weights and nodes A_i, B_j, x_j in (1.3) are determined by the requirement that $K_{n,k}f=If$ whenever f is a polynomial of degree $2k+n-1$.

* Received by the editors December 23, 1981, and in revised form November 1, 1982.

† Scientific Computing Division, National Bureau of Standards, Washington, DC 20234.
‡ ETH-Zurich, Switzerland.
§ IMSL, Houston, Texas 77036.

In §§ 2 and 3 we describe the technique which has been used to compute the nodes and weights. (As will be seen, this technique is perfectly general although we have only applied it in detail to the Laguerre case.) Henceforth when explicit integrals are required we use the inner product notation $\langle f, g\rangle$ to denote

$$If g = \langle f, g\rangle = \int w(x)f(x)g(x)\,dx \tag{1.4}$$

for nonnegative, integrable $w(x)$. Section 4 illustrates the new Laguerre weights and nodes for various n and k, and provides some examples. A more complete tabulation is given in [8]. Section 5 gives, informally, more details about the numerical accuracy and stability of the techniques we utilized to obtain the new formulas.

Formula (1.3) was first considered in [3] for integration on a finite interval and with $k = n+1$. If $k \leqq n$ it is shown in [6, Lemma 1] for $[-1, 1]$, $w(x) = 1$ that the formulas do not exist. This result generalizes easily for any $w(x) > 0$. For later work and more theoretical background see [4] and [5]. In the case of Gaussian formulas it is very well known that the nodes ξ_i are on the integration interval and the weights α_i are always positive. To be practical the new formulas (1.3) must have similar properties, e.g., $A_i > 0$, $B_j > 0$, $x_j \geqq 0$ (in the Laguerre case). Unfortunately for certain values of n and k one or more of these inequalities fails (see Table 1), and there is yet no successful theory to predict this failure in advance with any generality. Thus, there are several combinations of n and k (but not all combinations) from which it is possible to select a formula. For estimating errors in the Gauss formulas, the $k = n+1$ formula is most natural. In those cases where the latter does not exist the smallest $k > n+1$ takes its place. We include for consideration some larger values of k because such formulas may be of independent interest.

2. Computing quadrature nodes. Let $L_j(x)$ denote the orthogonal polynomial of degree j with respect to the inner product (1.4)

$$L_j(x) = \prod_{i=1}^{j} (x - \xi_i^{(j)}). \tag{2.1}$$

We are considering a formula of type (1.3) with k additional nodes, and so we define

$$E_k(x) = \prod_{j=1}^{k} (x - x_j), \qquad k > n \tag{2.2}$$

and

$$P_{n,k}(x) = L_n(x)E_k(x). \tag{2.3}$$

A characterization of the numbers x_j, which is necessary for (1.3) to be exact for polynomials of degree $2k + n - 1$ is [6]

$$\langle P_{n,k}, x^j\rangle = 0, \qquad 0 \leqq j < k.$$

$E_k(x)$ is thus orthogonal, with respect to the modified weight function $w(x)L_n(x)$, to lower degree polynomials. Our approach to finding the points x_j is to construct $P_{n,k}$ and find its zeros explicitly. Since the zeros of $L_n(x)$ are well known, it is easy to check the accuracy of the zero finding by seeing how well it reproduces the known zeros ξ_i. As we will see, in those cases when (2.3) has nicely spaced zeros, the numbers x_j tend to interlace with ξ_i. Since the zero finding procedure does not play favorites, both sets are computed to comparable accuracy.

Expanding $P_{n,k}(x)$ in terms of $\{L_j\}$ we have

$$P_{n,k}(x) = \sum_{j=0}^{n+k} \alpha_j L_j(x), \qquad \alpha_{n+k} \equiv 1. \tag{2.4}$$

By the orthogonality of $E_k(x)$

$$\langle E_k \cdot L_j, L_n \rangle = 0 = \alpha_j, \qquad j < k, \tag{2.5}$$

so

$$P_{n,k}(x) = \sum_{j=k}^{n+k} \alpha_j L_j(x), \qquad \alpha_{n+k} \equiv 1. \tag{2.6}$$

By the definition (2.3) of $P_{n,k}$, the right-hand side of (2.6) has $L_n(x)$ as a factor, or equivalently

$$\sum_{j=k}^{k+n} \alpha_j L_j = 0 \mod L_n. \tag{2.7}$$

This result suggests the following definition

$$L_j^* := L_j \mod L_n. \tag{2.8}$$

Obviously

$$L_0^* = L_0, \quad L_1^* = L_1, \quad \cdots, \quad L_{n-1}^* = L_{n-1}, \quad L_n^* = 0. \tag{2.9}$$

As the L_j's satisfy a 3-term recursion

$$L_{j+1} = (x - a_j)L_j + b_j L_{j-1}, \tag{2.10}$$

so do the L_j^*

$$L_{j+1}^* = (x - a_j)L_j^* + b_j L_{j-1}^* \mod L_n. \tag{2.11}$$

For example,

$$L_{n+1} = (x - a_n)L_n + b_n L_{n-1}, \qquad L_{n+1}^* = b_n L_{n-1}. \tag{2.12}$$

Similarly

$$L_{n+2}^* = L_{n-1} b_n (a_{n-1} - a_{n+1}) - b_n b_{n-1} L_{n-2}.$$

In general we write

$$L_j^* = \sum_{t=0}^{n-1} \alpha_{j,t} L_t. \tag{2.13}$$

The numbers $\alpha_{j,t}$ can be computed by a recursion:

$$\begin{aligned} &\alpha_{j,t} = \delta_{j,t}, \qquad j, t < n, \\ &\alpha_{j+1,t} = \alpha_{j,t}(a_t - a_j) + \alpha_{j-1,t} b_j - \alpha_{j,t+1} b_{t+1}, \qquad t = 0, 1, \cdots, n-1, \quad j \geqq 0, \\ &\alpha_{j,-1} = \alpha_{j,n} = 0. \end{aligned} \tag{2.14}$$

(In fact in the Laguerre case, as the a_n's and b_n's are rational, these numbers can be computed in exact rational form.) We also note that each of $L_{n+1}^*, L_{n+2}^*, \cdots, L_{2n}^*$ contains one additional term. Because (2.7) is equivalent to

$$\sum_{j=k}^{k+n} \alpha_j L_j^* = 0, \qquad \alpha_{n+k} = 1, \tag{2.15}$$

substituting (2.13) into (2.15) leads to the system of equations

$$\begin{bmatrix} \alpha_{k,0} & \alpha_{k+1,0} & \cdots & \alpha_{k+n,0} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{k,n-2} & \alpha_{k+1,n-2} & \cdots & \alpha_{k+n,n-2} \\ \alpha_{k,n-1} & \alpha_{k+1,n-1} & \cdots & \alpha_{k+n,n-1} \end{bmatrix} \begin{pmatrix} \alpha_k \\ \vdots \\ \alpha_{k+n-1} \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}. \tag{2.16}$$

By using the reduced matrix composed of the first n columns, we see that (2.16) becomes a system of n equations for $\alpha_k, \cdots, \alpha_{k+n-1}$. From the remark above (2.15) this matrix is lower cross triangular ($\alpha_{t,s} = 0$, $s < n - t - 1$, $0 \leqq s$, $t < n$) if $k = n + 1$ (the "traditional" Kronrod case) and each successive k fills in one further upper cross diagonal. This result has also been noted by Monegato [6] in the case $k = n + 1$, and he used this result to prove that the matrix is nonsingular. We have not utilized this triangular structure in our algorithm, although for some special situations, e.g., Chebyshev polynomials of the first or second kind, it is easy to show that some of the higher degree T_j^*· only involve two of the first nT_i. We also have not been able to prove that the matrix is in general nonsingular, although for all the cases we have studied it has been.

3. Computing quadrature weights. The quadrature weights A_i, B_j are well defined by the requirement that $K_{n,k}f$ be exact for polynomials of highest possible degree. Once the ξ_i and x_j are given, though, the weights can in principle be determined by several methods. We use the following.

For convenience we merge the ξ's and x's, calling both t's and then rewrite (1.3) as

$$K_{n,k}f = \sum_{i=1}^{n+k} w_i f(t_i). \tag{3.1}$$

Define

$$Q(x) := \sum_{i=1}^{n+k} w_i L_k(t_i)/(x - t_i) \tag{3.2}$$

$$= \frac{r(x)}{P_{n,k}(x)}, \tag{3.3}$$

where $r(x)$ is a polynomial of degree $n + k - 1$. In fact,

$$r(x) = \sum_{i=1}^{n+k} P_{n,k}(x) w_i L_k(t_i)/(x - t_i). \tag{3.4}$$

By evaluating r at $x = t_i$, we get

$$r(t_i) = P'_{n,k}(t_i) w_i L_k(t_i) \tag{3.5}$$

$$= E_k(t_i) L'_n(t_i) w_i L_k(t_i), \tag{3.6}$$

$$r(t_i) = \left\{ \sum_{j=k}^{k+n} \alpha_j L'_j(t_i) \right\} w_i L_k(t_i), \qquad \alpha_{k+n} = 1, \tag{3.7}$$

where (3.6) follows from (2.3) and (3.7) follows from (2.6).

The calculation of L'_j in (3.7) as well as $L_k(t_i)$ can be done quite stably via the recursion for the L_j's. Thus w_i can be calculated if the left-hand side, $r(t_i)$, is known. Interestingly enough, although $r(x)$ is nominally a polynomial of degree $n + k - 1$, in fact it is a much lower degree polynomial, $n - 1$. This fact simplifies the calculation considerably. For example, to compute weights for the case of one Gauss point and k new points, $r(x)$ is a constant.

By expanding the denominator in (3.2) we have

$$Q(x)=\sum_{\nu=0}^{\infty}\frac{1}{x^{\nu+1}}\sum_{i=1}^{n+k}w_iL_k(t_i)t_i^{\nu} \tag{3.8}$$

$$=\left[\sum_{\nu=0}^{n+k-1}+\sum_{\nu=n+k}^{\infty}\right]\frac{1}{x^{\nu+1}}\sum_{i=1}^{n+k}w_iL_k(t_i)t_i^{\nu}$$

$$=\sum_{\nu=0}^{n+k-1}\frac{1}{x^{\nu+1}}\langle L_k,x^{\nu}\rangle+O\left(\frac{1}{x^{n+k+1}}\right) \tag{3.9}$$

$$=\sum_{\nu=k}^{n+k-1}\frac{1}{x^{\nu+1}}\langle L_k,x^{\nu}\rangle+O\left(\frac{1}{x^{n+k+1}}\right). \tag{3.10}$$

Here (3.9) follows from the exactness of (3.1) for polynomials of degree $n+2k-1$ and (3.10) from the orthogonality of L_k. Thus we have

$$Q(x)=c_1x^{-(k+1)}+\cdots+c_nx^{-(n+k)}+O(x^{-(n+k+1)}), \tag{3.11}$$

where

$$c_i=\langle L_k,x^{k+i-1}\rangle,\qquad i=1,2,\cdots,n. \tag{3.12}$$

So from (3.3)

$$r(x)=Q(x)P_{n,k}(x)=\sum_{i=1}^{n}c_ix^{-k-i}P_{n,k}(x),$$

and we see that $r(x)$ is a polynomial of degree $n-1$. The contributions of Q to $r(x)$ are only from the first n terms in (3.11), that is,

$$c_1x^{-k-1}+\cdots+c_nx^{-k-n}. \tag{3.13}$$

The c_i's in (3.12) can be computed via the recursion (2.10), so

$$r(x)=\text{poly}\,[P_{n,k}(x)Q(x)],\quad \deg r=n-1. \tag{3.14}$$

4. Formulas and examples.

4. Formulas and examples. This section contains two sets of tables for use with the extended Gauss–Laguerre quadrature formulas (1.3).

Table 1 lists those values of n and k for which formulas have been computed.

TABLE 1
Summary information of extended Gauss–Laguerre–Kronrod quadrature

	k																	
n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1		Z	6	8	10	12	14	Z	Z	Z	Z	24						
2			C	N	11	13	15	ZN	19	21	23							
3				C	C	14	16	18	Z	22	C							
4					C	C	C	19	21	ZN	25	C						
5						C	C	C	22	24	Z	28	C					
6							C	C	C	C	27	29	N	N				
7								C	C	C	C	30	32	C	36			
8									C	C	C	C	33	ZN	N	39		
9										C	C	C	C	C	C	C	42	
10											C	C	C	C	C	C	C	45

TABLE 2
Nodes and weights for extended Gauss–Laguerre quadrature

4-point Laguerre-Kronrod Quadrature Formula
with 1 Gauss point and 3 additional points

i	nodes	weights	Gauss weights
1	1.0000000000000000	$5.90163934426230\cdot10^{-1}$	$1.00000000000000\cdot10^{0}$
2	0.0568976250010772	$2.96432485383635\cdot10^{-1}$	
3	3.3698392313109231	$1.10974454218907\cdot10^{-1}$	
4	7.8232631436879997	$2.42912597122808\cdot10^{-3}$	

5-point Laguerre-Kronrod Quadrature Formula
with 1 Gauss point and 4 additional points

i	nodes	weights	Gauss weights
1	1.0000000000000000	$4.79600333055787\cdot10^{-1}$	$1.00000000000000\cdot10^{0}$
2	0.1576954801215496	$3.69326094452073\cdot10^{-1}$	
3	2.8500082985710812	$1.41243684774939\cdot10^{-1}$	
4	6.0165843328122931	$9.73983652570679\cdot10^{-3}$	
5	11.2423785551617429	$9.00511914942123\cdot10^{-5}$	

6-point Laguerre-Kronrod Quadrature Formula
with 1 Gauss point and 5 additional points

i	nodes	weights	Gauss weights
1	1.0000000000000000	$4.34245046892434\cdot10^{-1}$	$1.00000000000000\cdot10^{0}$
2	0.1829603989824868	$3.94762075757311\cdot10^{-1}$	
3	2.5922508199253002	$1.52157044365239\cdot10^{-1}$	
4	5.1384848875831238	$1.82338739548839\cdot10^{-2}$	
5	8.9529512477548957	$5.99218164468885\cdot10^{-4}$	
6	14.8297812171827648	$2.74086566288354\cdot10^{-6}$	

7-point Laguerre-Kronrod Quadrature Formula
with 1 Gauss point and 6 additional points

i	nodes	weights	Gauss weights
1	1.0000000000000000	$4.22142618082264\cdot10^{-1}$	$1.00000000000000\cdot10^{0}$
2	0:1879695887504464	$4.00617850433272\cdot10^{-1}$	
3	2.5026000900489822	$1.53198633880393\cdot10^{-1}$	
4	4.7795963847097300	$2.27446130105297\cdot10^{-2}$	
5	7.9886543552868417	$1.27553308646018\cdot10^{-3}$	
6	12.4485130629567814	$2.07047420621403\cdot10^{-5}$	
7	18.9899638155445156	$4.67650188680929\cdot10^{-8}$	

28-point Laguerre-Kronrod Quadrature Formula
with 10 Gauss points and 18 additional points

i	nodes	weights	Gauss weights
1	0.1377934705404924	$8.09373580983253\cdot10^{-2}$	$3.08441115765020\cdot10^{-1}$
2	0.7294545495031705	$2.31849330323753\cdot10^{-1}$	$4.01119929155274\cdot10^{-1}$
3	1.8083429017403160	$9.36730669347739\cdot10^{-2}$	$2.18068287611809\cdot10^{-1}$
4	3.4014336978548995	$3.34446813206487\cdot10^{-2}$	$6.20874560986777\cdot10^{-2}$
5	5.5524961400638036	$4.46422296677345\cdot10^{-3}$	$9.50151697518110\cdot10^{-3}$
6	8.3301527467644967	$3.90850918672935\cdot10^{-4}$	$7.53008388587539\cdot10^{-4}$
7	11.8437858379000656	$1.36832206767955\cdot10^{-5}$	$2.82592334959957\cdot10^{-5}$
8	16.2792578313781021	$2.19106173393734\cdot10^{-7}$	$4.24931398496269\cdot10^{-7}$
9	21.9965858119807620	$8.09142634024867\cdot10^{-10}$	$1.83956482397963\cdot10^{-9}$
10	29.9206970122738916	$4.15812728328488\cdot10^{-13}$	$9.91182721960901\cdot10^{-13}$
11	0.0429781731482718	$9.53075626780387\cdot10^{-2}$	
12	0.3289176194809725	$2.21683636061387\cdot10^{-1}$	
13	1.2579673664649055	$1.57022936268989\cdot10^{-1}$	
14	2.4863322728487988	$6.70417135087898\cdot10^{-2}$	
15	4.4506073160317271	$1.25640886594998\cdot10^{-2}$	
16	6.8155915356307183	$1.52679882245378\cdot10^{-3}$	
17	10.0238077719889290	$7.77820143339117\cdot10^{-5}$	
18	13.8883101949033859	$2.05235014220785\cdot10^{-6}$	
19	19.0240878575067498	$1.58752146509375\cdot10^{-8}$	
20	23.9387730886657401	$4.67591923429223\cdot10^{-11}$	
21	26.1322043237291165	$1.50305555137104\cdot10^{-11}$	
22	34.2870595757048108	$5.95338419581797\cdot10^{-15}$	
23	39.1942003225497253	$4.94419421427085\cdot10^{-17}$	
24	44.7203619733152653	$2.22464800465781\cdot10^{-19}$	
25	51.0038952740557615	$4.75580633904356\cdot10^{-22}$	
26	58.2796092772633328	$3.86103275520259\cdot10^{-25}$	
27	66.9996954695658746	$7.75633106693722\cdot10^{-29}$	
28	78.3270796488236109	$1.30646368161741\cdot10^{-33}$	

Associated with a pair (n, k) the reader will find in the table a *number*, one or more of the symbols C, Z, N, or *blank*. The meanings are:

number	This formula is suitable for computation. The nodes x_j are all in $[0, \infty)$ and all the weights A_i, B_j are positive. The *number* in this position is the highest degree polynomial $f(x)$ that formula (1.3) will integrate exactly. For this formula the weights A_i, B_j, the Gauss nodes ξ_i and the new nodes x_j are given in Table 2 [8, Table II]. To illustrate the formulas we reproduce a few of these entries later. Entries in this table are accurate to all the digits printed. As a convenience the Gaussian weights $\alpha_i^{(n)}$ (see (1.2)) are also given.
C	One pair or more of the nodes x_j is complex.
Z	All the nodes x_j are real, but at least one is negative.
N	This formula may be suitable for computation. The nodes x_j are all real and nonnegative, but at least two are close together. Accompanying this, one or more weights A_i, B_j is negative. These rules will also be found in Table 2 [8].
blank	No computations were done to derive this formula. Note that for $k \leqq n$ the formula does not exist.

In Figs. 1, 2, 3 we illustrate the application of the formulas from Table 2 to three sample integrals

$$\int_0^\infty e^{-x} x \ln x \, dx, \quad \int_0^\infty e^{-x} \sqrt{x} \, dx, \quad \int_0^\infty \frac{e^{-x}}{(1+x)} dx.$$

In each we plot the error in estimating the integral using Gauss–Laguerre and extended Gauss–Kronrod quadrature rules for various total numbers of points. We note the very smooth decrease of the error with Gauss–Laguerre and the rather erratic behavior of the extended formulas. On the other hand, most of these errors are substantially less than the Gauss errors with the same number of points.

Figure 4 represents graphically the nodes of these new formulas and illustrates the interlacing of the new extended Kronrod nodes with the Gauss–Laguerre nodes.

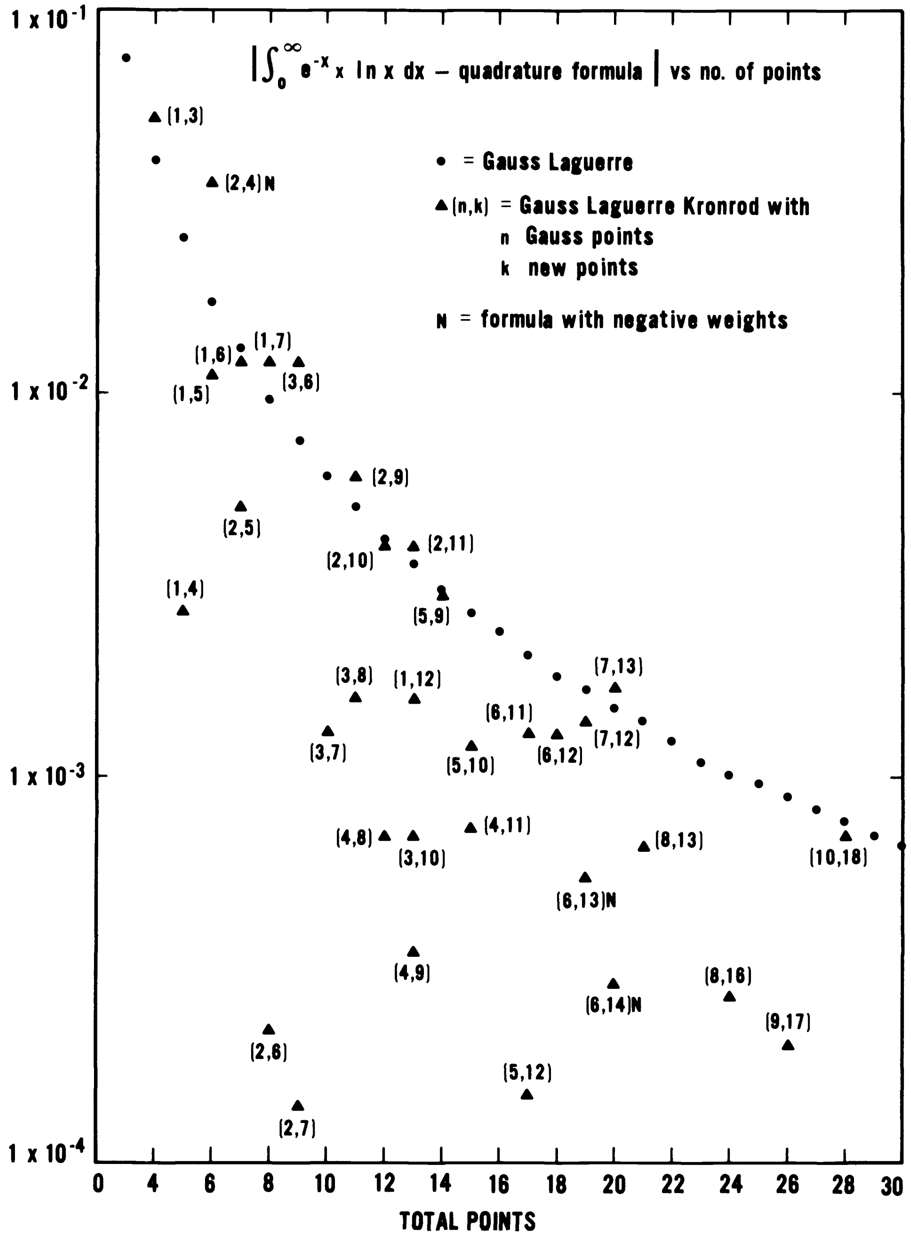
5. Some numerical considerations. Here we provide a few brief details about some of the numerical issues faced in obtaining Tables 1 and 2.

The nodes were computed as the zeros of (2.4). The coefficients a_j were obtained quite accurately by first using rational arithmetic to compute the matrix elements (2.14) and then solving (2.16) in multiple precision. The zeros of (2.3) were obtained via a Newton–Raphson iteration in CDC double precision. During this phase of the calculation, we experienced a considerable loss of accuracy. For small x the function $P_{n,k}(x)$ has very small values (e.g. $O(10^{-6})$ in our largest cases) which are produced by subtraction. This evidences itself as

$$P_{n,k}(0) = \sum_{j=k}^{n+k} \alpha_j \ll \max_j |\alpha_j|.$$

Hence the small zeros can be inaccurate even with exact a_j's.

The confirmation of the 15-digit accuracy in all the zeros we publish comes from verifying that the Laguerre zeros, a subset of those of $P_{n,k}$, where correct to at least 20 digits, via CDC double precision. An independent check on the accuracy of the zeros was obtained by carefully forming E_k in powers of x and finding its zeros by an eigenvalue calculation with 80 digit arithmetic. This latter was done on a VAX 11/750 (Comet) using Brent's multiprecision subroutine package. This also served to check



$\left|\int_0^\infty e^{-x} x \ln x\, dx$ – quadrature formula$\right|$ vs no. of points
● = Gauss Laguerre
▲(n,k) = Gauss Laguerre Kronrod with n Gauss points k new points
N = formula with negative weights
1 x 10^-1
1 x 10^-2
1 x 10^-3
1 x 10^-4
0 4 6 8 10 12 14 16 18 20 22 24 26 28 30
TOTAL POINTS
(1,3)
(2,4)N
(1,6)
(1,7)
(1,5)
(3,6)
(2,9)
(2,5)
(2,10)
(2,11)
(1,4)
(5,9)
(3,8)
(1,12)
(7,13)
(6,11)
(3,7)
(5,10)
(6,12)
(7,12)
(4,8)
(3,10)
(4,11)
(8,13)
(10,18)
(6,13)N
(4,9)
(6,14)N
(8,16)
(9,17)
(2,6)
(5,12)
(2,7)

FIG. 1

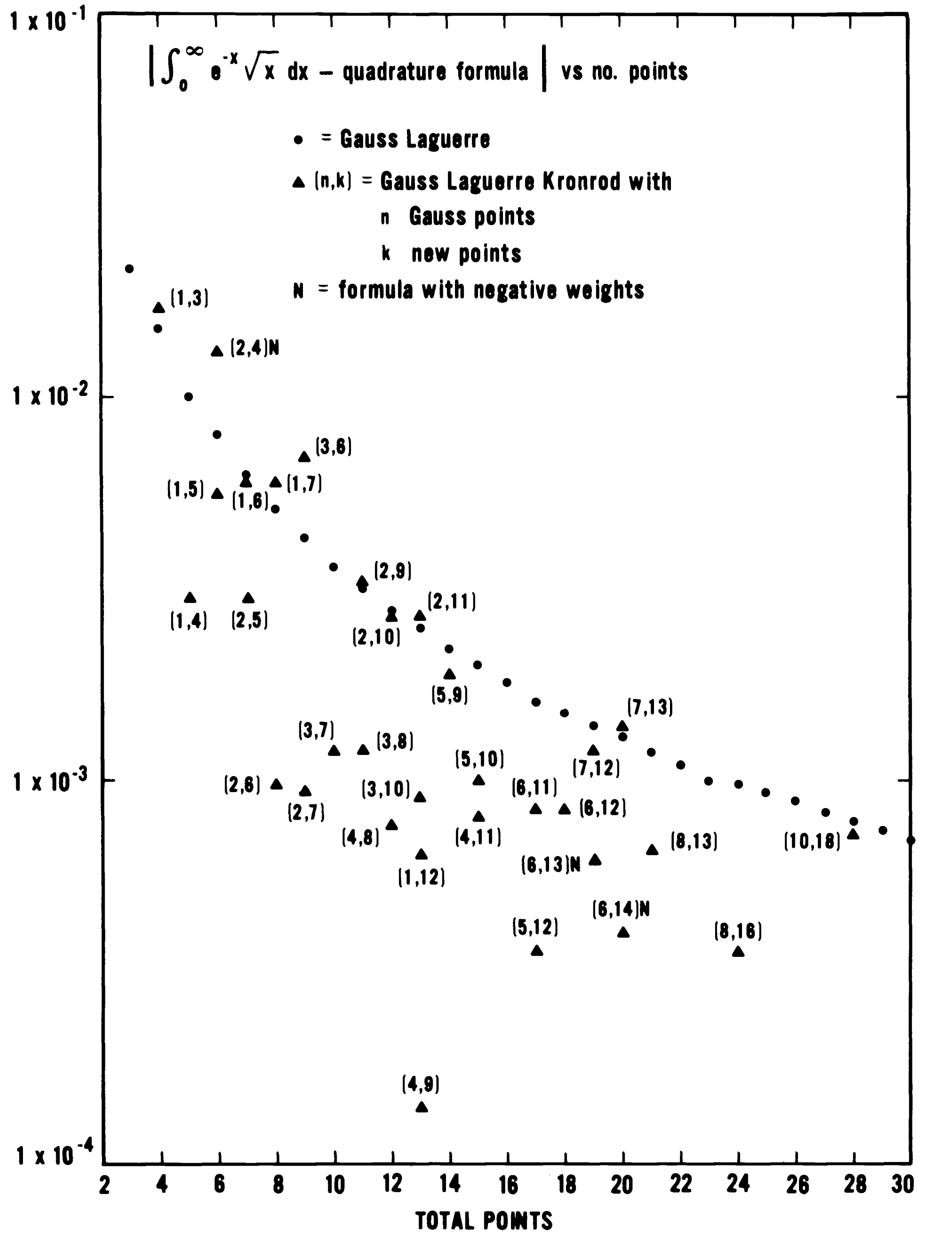

$\left| \int_0^\infty e^{-x}\sqrt{x}\,dx - \text{quadrature formula} \right|$ vs no. points
● = Gauss Laguerre
▲ (n,k) = Gauss Laguerre Kronrod with
n Gauss points
k new points
N = formula with negative weights
1 x 10^-1
1 x 10^-2
1 x 10^-3
1 x 10^-4
(1,3)
(2,4)N
(3,6)
(1,5)
(1,6)
(1,7)
(2,9)
(1,4)
(2,5)
(2,11)
(2,10)
(5,9)
(7,13)
(3,7)
(3,8)
(5,10)
(7,12)
(2,6)
(3,10)
(6,11)
(6,12)
(2,7)
(4,8)
(4,11)
(8,13)
(10,18)
(1,12)
(6,13)N
(6,14)N
(5,12)
(8,16)
(4,9)
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
TOTAL POINTS

FIG. 2

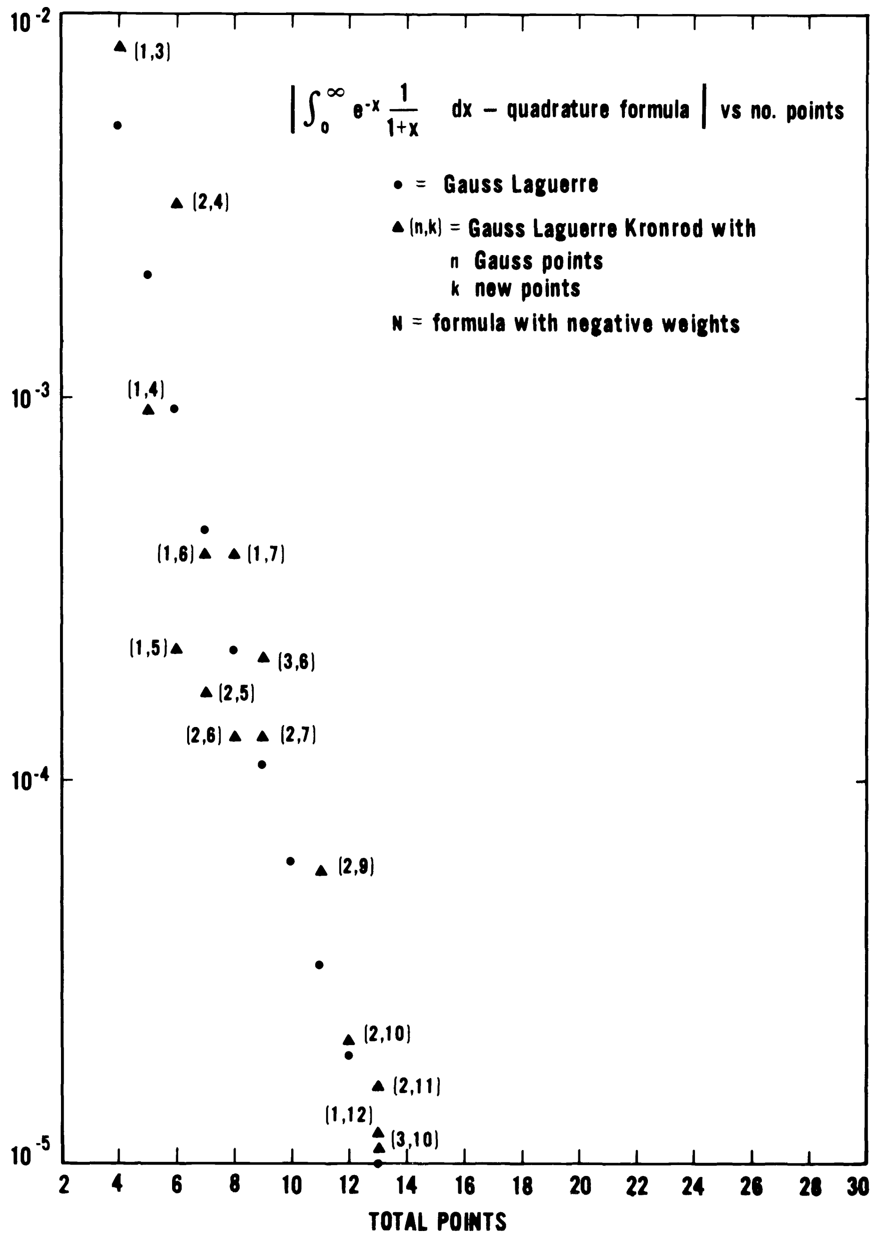
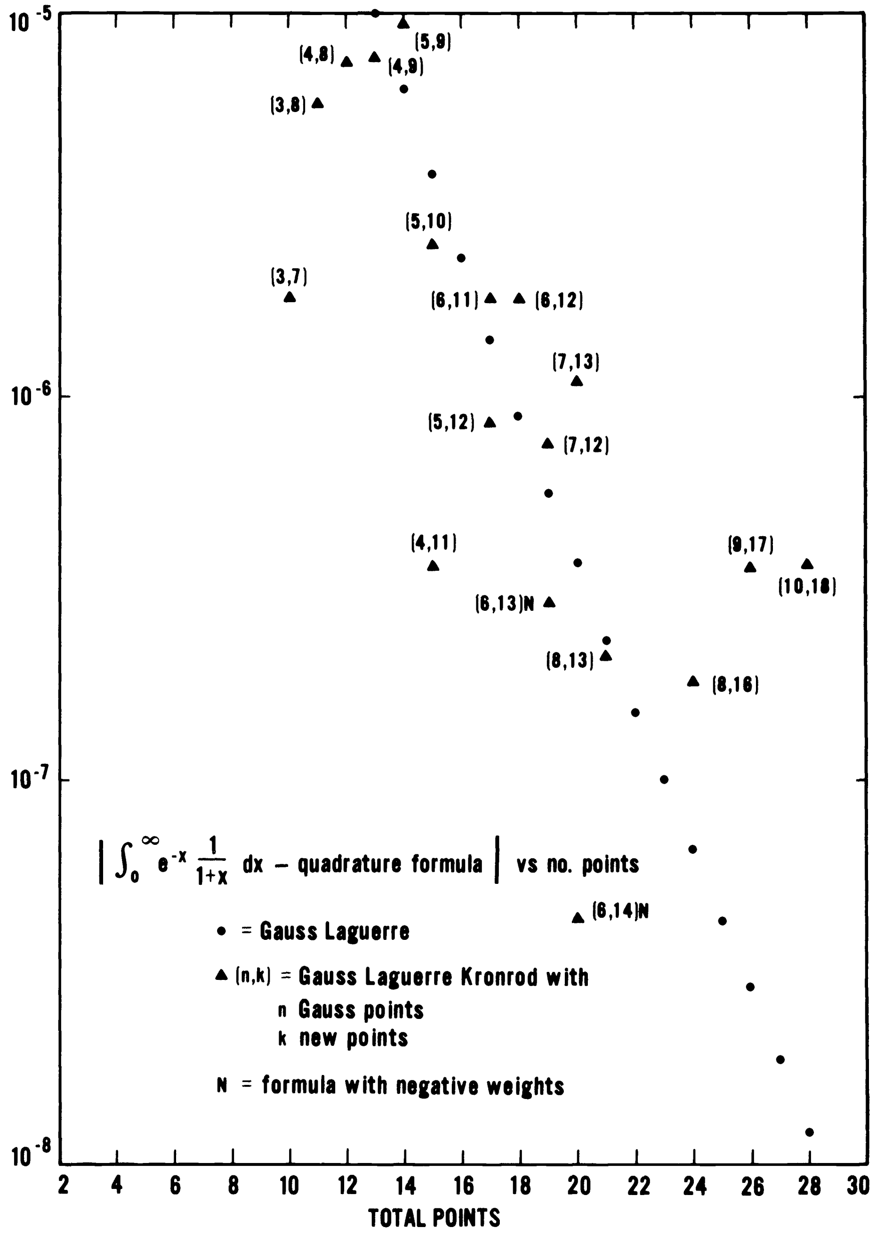

$\left|\int_0^\infty e^{-x}\frac{1}{1+x}\,dx - \text{quadrature formula}\right|$ vs no. points
● = Gauss Laguerre
▲(n,k) = Gauss Laguerre Kronrod with
n Gauss points
k new points
N = formula with negative weights
(1,3)
(2,4)
(1,4)
(1,6)
(1,7)
(1,5)
(3,6)
(2,5)
(2,6)
(2,7)
(2,9)
(2,10)
(2,11)
(1,12)
(3,10)
10^{-2}
10^{-3}
10^{-4}
10^{-5}
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
TOTAL POINTS

FIG. 3



10^-5
10^-6
10^-7
10^-8
(4,8)
(5,9)
(4,9)
(3,8)
(5,10)
(3,7)
(6,11)
(6,12)
(7,13)
(5,12)
(7,12)
(4,11)
(9,17)
(10,18)
(6,13)N
(8,13)
(8,16)
(6,14)N
$\left| \int_0^\infty e^{-x} \frac{1}{1+x}\, dx$ – quadrature formula $\right|$ vs no. points
● = Gauss Laguerre
▲ (n,k) = Gauss Laguerre Kronrod with n Gauss points k new points
N = formula with negative weights
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
TOTAL POINTS

FIG. 3 (*cont.*)

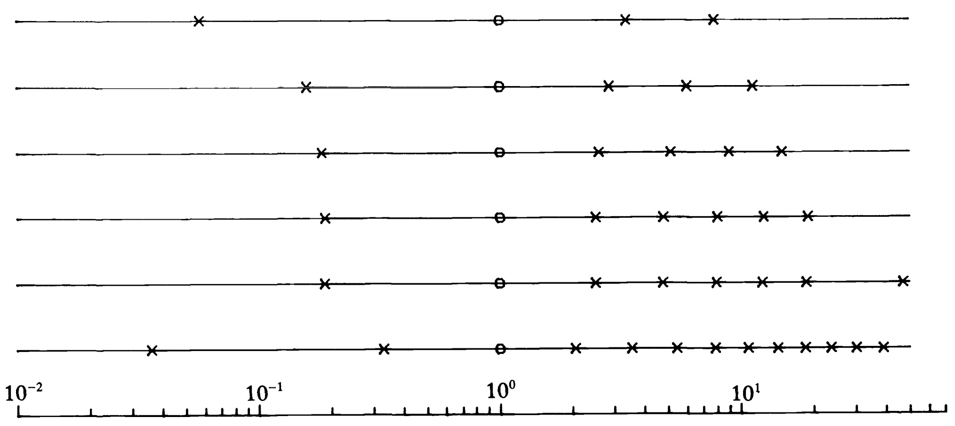


FIG. 4a). 1-*Gauss–Laguerre node,* O, *extended Kronrod nodes,* ×.

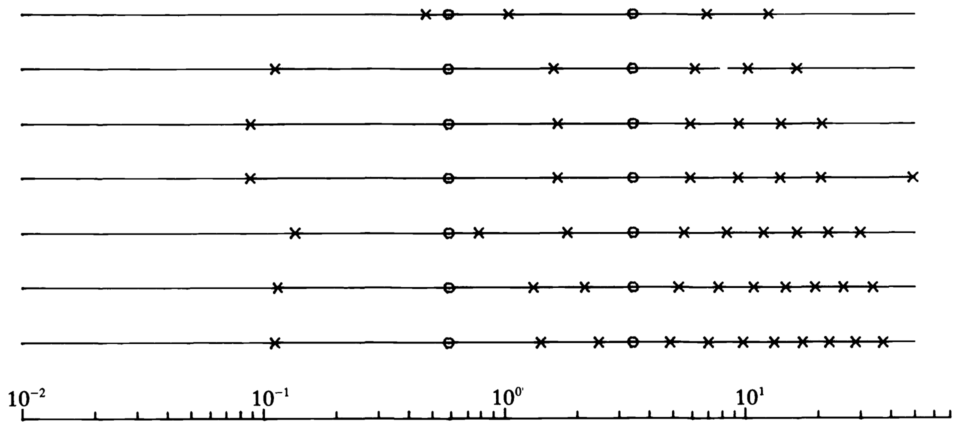


FIG. 4b). 2-*Gauss–Laguerre nodes,* O, *extended Kronrod nodes,* ×.

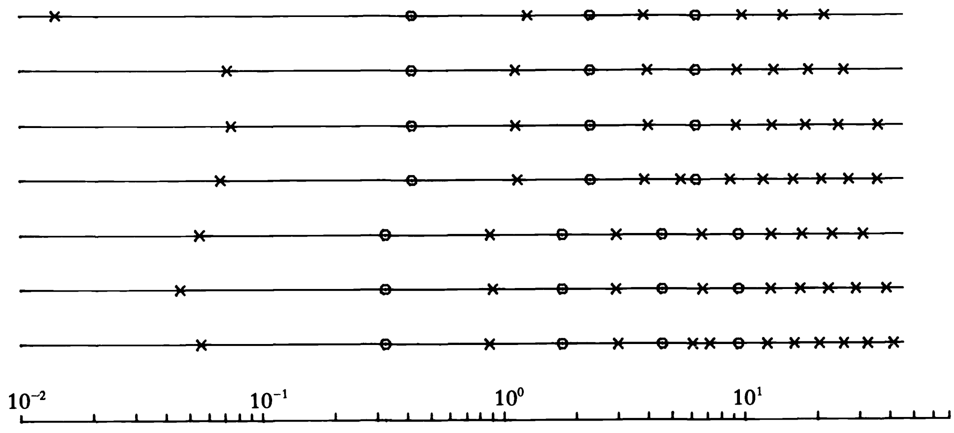


FIG. 4c). 3- *and* 4-*Gauss–Laguerre nodes,* O, *extended Kronrod nodes,* ×.

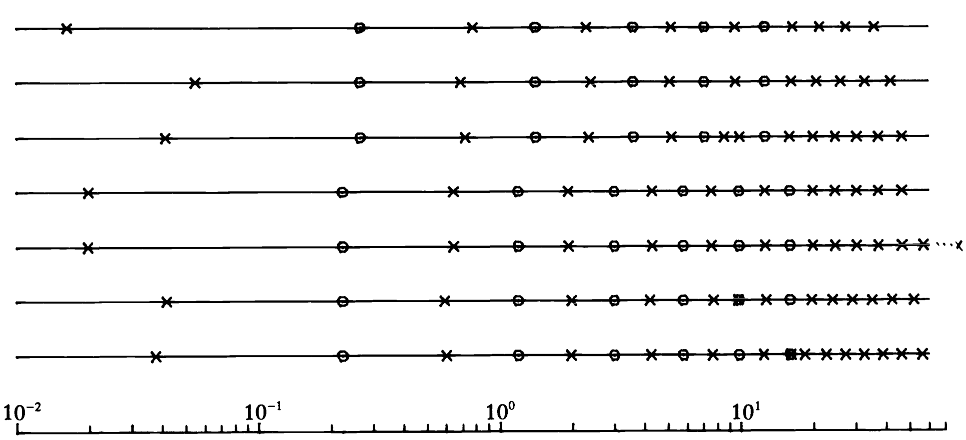


FIG. 4d). 5- *and* 6-*Gauss–Laguerre nodes,* O, *extended Kronrod nodes,* ×.

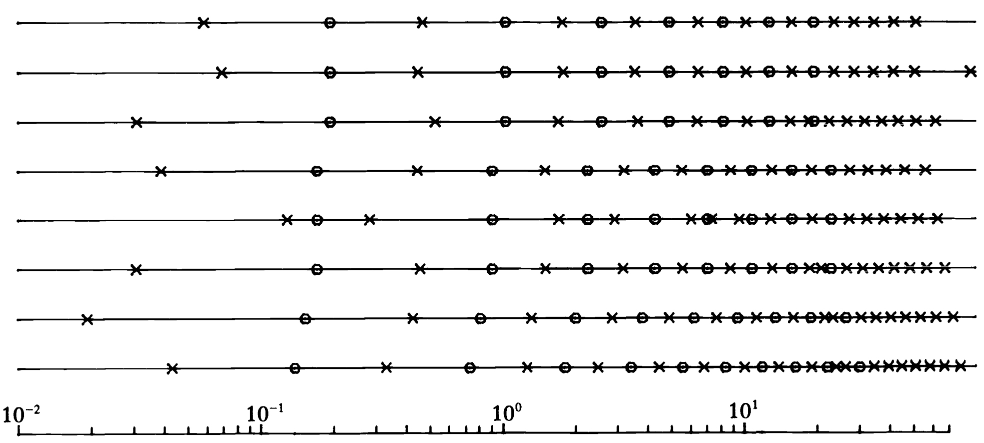


FIG. 4e). 7-, 8-, 9- *and* 10-*Gauss–Laguerre nodes,* O, *extended Kronrod nodes,* ×.

on those cases for which there are complex zeros. As mentioned above, the (n, k) values leading to real zeros cannot be predicted in advance.

The weight calculations were done as decribed in § 3 in CDC single and double precision, and then by using a more direct technique in 80 digit multiple precision. The last two agreed to at least every figure in Table 2.

In addition to comparing the results at the three different precisions, we also tested the quadrature sum (1.3) on the monomials $f(x)=x^i$, $0\leqq i\leqq 2k+n-1$. These functions ought to be integrated exactly. Using the double precision versions of the nodes and weights we found no relative error more than 10^{-15}. For all $n+k\leqq 20$ the maximum relative error was less than 10^{-20}.

REFERENCES

[1] P. J. DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration*, Academic Press, New York, 1975.
[2] H. ENGELS, *Numerical Quadrature and Cubature*, Academic Press, New York, 1980.
[3] A. S. KRONROD, *Nodes and Weights of Quadrature Formulas*, Consultants Bureau, New York, 1965.

[4] P. RABINOWITZ, *The exact degree of precision of generalized Gauss–Kronrod integration rules*, Math. Comp., 35 (1980), pp. 1275–1283.

[5] G. MONEGATO, *A note on extended Gaussian quadrature rules*, Math. Comp., 30 (1976), pp. 812–817.

[6] ———, *An overview of results and question related to Kronrod schemes*, in Number Integration, G. Hammerlin, ed., ISNM45, Birkhauser Verlag, Basel 1979, pp. 231–240.

[7] G. GOLUB AND S. WELSCH, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.

[8] D. KAHANER, J. WALDVOGEL AND L. W. FULLERTON, *Addition of points to Gauss–Laguerre quadrature formulas*, IMSL Technical Report Series, 8205, IMSL, Houston 1982.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

FINITE DIFFERENCE METHODS FOR THE STOKES AND NAVIER-STOKES EQUATIONS*

JOHN C. STRIKWERDA†

Abstract. This paper presents a new finite difference scheme for the Stokes equations and incompressible Navier–Stokes equations for low Reynolds number. The scheme uses the primitive variable formulation of the equations and is applicable with nonuniform grids and nonrectangular geometries. Several other methods of solving the Navier–Stokes equations are also examined in this paper and some of their strengths and weaknesses are described. Computational results using the new scheme are presented for the Stokes equations for a region with curved boundaries and for a disk with polar coordinates. The results show the method to be second-order accurate.

Key words. finite differences, Navier–Stokes equations, Stokes equations

AMS(MOS) subject classifications. 65N05, 76D05

1. Introduction. In this paper we examine several common methods for solving the incompressible Navier–Stokes equations by finite differences and we present a new second-order accurate finite difference scheme for these equations. This new scheme is designed to be applied with nonuniform grids and nonorthogonal coordinate systems. Numerical experiments with the Stokes equations illustrate the versatility and accuracy of the scheme.

The steady-state Stokes equations on a domain Ω in $\mathbb{R}^n$ are given by

$$-\nabla^2\mathbf{u}+\nabla p=\mathbf{f}(x), \qquad \nabla\cdot\mathbf{u}=g(x) \tag{1.1}$$

and the steady-state Navier–Stokes equations are

$$-R^{-1}\nabla^2\mathbf{u}+(\mathbf{u}\cdot\nabla)\mathbf{u}+\nabla p=\mathbf{f}(x), \qquad \nabla\cdot\mathbf{u}=g(x) \tag{1.2}$$

where R is the Reynolds number. We will consider the systems (1.1) and (1.2) with Dirichlet boundary conditions

$$\mathbf{u}(x)=\mathbf{b}(x) \quad \text{on } \partial\Omega. \tag{1.3}$$

A necessary condition for (1.1) or (1.2) to have a solution is that the data $g(x)$ and $\mathbf{b}(x)$ satisfy the integrability condition

$$\int_\Omega g=\int_{\partial\Omega}\mathbf{b}\cdot\mathbf{n}, \tag{1.4}$$

where $\mathbf{n}$ is the outer unit normal to $\partial\Omega$. For the mathematical theory of the systems (1.1) and (1.2) we refer to Ladyzhenskaya (1963) and Temam (1979).

We will be concerned only with methods that solve the systems (1.1) and (1.2) in the primitive variables $\mathbf{u}$ and p and not with methods such as the vorticity and stream-function reformulation. Also our methods are applicable in two or three dimensions although our examples will be only in two dimensions.

We emphasize that the scheme presented here is designed to be easily applicable with nonrectangular geometries and nonuniform grids. The vast majority of papers on the numerical solution of the incompressible Navier–Stokes equations limit

* Received by the editors May 17, 1982, and in revised form October 20, 1982. This research was sponsored by the United States Army under contract DAAG29-80-C-0041. Portions of this research were performed under NASA contracts NAS-15810 and NASI-16394 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

† Mathematics Research Center, University of Wisconsin-Madison, Madison, Wisconsin 53706.

themselves to examples using rectangular geometry and uniform grids. By way of contrast, computations with the compressible Navier–Stokes equations routinely use nonrectangular geometries and nonuniform grids.

The scheme proposed in this paper uses central finite differences with a regularizing term. Chorin (1967), (1968), (1969) proposed schemes with standard central finite differences for the time-dependent Navier–Stokes equations. There are several difficulties with central differencing schemes and the scheme presented here avoids them. The first difficulty is that with central differences the grid decomposes into two staggered grids which are coupled only at the boundaries and this can cause the pressure field to be oscillatory (see for example Patterson and Imberger (1980) and Ghia et al. (1977) reporting on the method of Chorin (1968) and the paper by Zoby in Rubin and Harris (1975) reporting on the method of Chorin (1967)). Since the pressure is of significant importance in engineering applications this oscillatory behavior of the pressure is undesirable. Another difficulty is with the possible nonexistence of a solution to the finite difference equations. We discuss this point in § 4. The regularized central difference scheme proposed here removes the grid decoupling and thus the pressure oscillations and, furthermore, the incompressibility constraint is modified to guarantee the existence of a solution.

The most common finite difference methods for the Navier–Stokes equations are the staggered mesh schemes. They avoid the grid decoupling of the standard central differencing schemes but require more care when used with nonrectangular geometry. These schemes are discussed in more detail in § 3.

In § 5 results are given of the regularized central scheme being used to solve the Stokes equations on a nonrectangular region and the results are shown to be second-order accurate in both the velocities and the pressure. To our knowledge no other finite difference scheme for the Stokes or Navier–Stokes equations in the primitive variables has been shown to be second-order accurate for nonrectangular geometry. For rectangular geometry with uniform grid one can give proofs of the convergence of schemes, e.g. Chorin (1969), Temam (1979), Kzivickii and Ladyzhenskaya (1966), but these proofs fail for nonrectangular geometry (see § 4). Chorin (1967), (1968) gives results of using central differences for rectangular regions but the results are for only one grid spacing.

Peskin (1977) has used Chorin's method with moving boundaries which are superimposed on a rectangular grid and Viecelli (1971) has a similar method for staggered grid schemes. Liu and Krause (1979) have developed a staggered grid scheme for general geometry. However, the order of accuracy of these schemes has not been demonstrated. Thompson (1980) used central differencing with nonrectangular geometry but also used the elliptic equation for the pressure (see § 2), which makes the accuracy uncertain.

The outline of the remaining sections of the paper is as follows. In § 2 we discuss the strengths and weaknesses of some common approaches to solving the systems (1.1) and (1.2) and in § 3 we discuss finite difference schemes for these systems. The finite difference integrability condition is discussed in § 4, and computational results are given in § 5. The numerical examples of § 5 demonstrate that the new scheme given here can be used to give second-order accurate solutions to the Stokes equations for nonrectangular geometries. To our knowledge no other finite difference schemes for the Stokes or incompressible Navier–Stokes equations in the primitive variables have been shown to be second-order accurate for nonrectangular geometries. Computations using the new scheme for the incompressible Navier–Stokes equations are currently being made and will be reported when complete.

2. Solution techniques. In this section we review some approaches to solving the Navier–Stokes and Stokes equations numerically. Few researchers have treated the system (1.2) in the given form, most have altered it in some way. Before examining the altered forms of (1.2) we look at the system in the given form.

The Stokes equations (1.1) and the Navier–Stokes equations (1.2) are elliptic systems of $n+1$ equations in $n+1$ dependent variables. The definition of an elliptic system, as given by Douglis and Nirenberg (1957), requires that the determinant of the principal symbol of the system not vanish for nonzero values of dual variables. For the Navier–Stokes equations the determinant of the principal symbol is

$$\det\begin{pmatrix} (1/R)|\xi|^2 I_n & i\boldsymbol{\xi} \\ i\boldsymbol{\xi}^T & 0 \end{pmatrix} = R^{-(n-1)}|\xi|^{2n}, \tag{2.1}$$

which is nonzero for $|\xi| \neq 0$. Moreover, since the determinant is a polynomial of degree $2n$ in the variables $\xi = (\xi_1, \cdots, \xi_n)$ the system requires n boundary conditions at each point of the boundary (Agmon, Douglis and Nirenberg (1964)). These boundary conditions will usually be Dirichlet or Neumann conditions on the velocity $\mathbf{u}$.

One of the most common ways of modifying the Navier–Stokes equations (1.2) is to replace it by the system

$$\begin{gathered} -R^{-1}\nabla^2\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \mathbf{f}(x), \\ \nabla^2 p = \nabla\cdot\mathbf{f} + R^{-1}\nabla^2 g - \sum_{i,j} u^i_{x_j} u^j_{x_i} - \mathbf{u}\cdot\nabla g. \end{gathered} \tag{2.2}$$

The last equation of (2.2) is obtained by taking the divergence of the first equation of (1.2) and then using the last equation of (1.2) to eliminate the divergence of velocity. The system (2.2) has the advantage over (1.2) in that, when discretized, it can be solved using standard methods for inverting the discrete Laplacian. However, the system (2.2) has a grave disadvantage in that it requires $n+1$ boundary conditions, one for each elliptic equation, as opposed to (1.2) which requires only n boundary conditions. Thus any attempt to solve (1.2) via (2.2) would require some means of determining the correct additional boundary condition. Without the correct boundary condition solutions of (2.2) will not be solutions of (1.2).

Roache (1972, p. 194) suggests that the additional boundary condition be given by the normal derivative of pressure as determined by the first equation of (1.2) or (2.2) evaluated on the boundary. This, however, is not satisfactory as a boundary condition since it is not independent of the system of differential equations. Roache's suggestion leaves the system (2.2) underdetermined.

Another boundary condition which is commonly used along boundaries corresponding to physical surfaces is to set the normal derivative of the pressure to zero, which is valid in the limit for high Reynolds number flow. With this boundary condition and (1.3) the system (2.2) has the proper number of boundary conditions, however, its solutions are not solutions of (1.2).

As one would expect, the methods using (2.2) or similar systems have difficulty with the accuracy of the pressure field and with satisfying the incompressiblity condition on the velocities (see for example the work by Boney, Hefner, Hirsh and Zoby reported in Rubin and Harris (1975)).

The above mentioned difficulties are seen in computations with the time-dependent Navier–Stokes equations as well. Roache (1972) has a discussion of the difficulties of obtaining a zero divergence for the velocity field when using the above approach for time-dependent flows (see also Harlow and Welch (1964)).

Because of these difficulties, it seems best not to use the derived system (2.2) but to use the original system (1.2). This is also the approach used by Chorin (1968).

Another approach to solving the Navier–Stokes equations (1.2) is the artificial compressiblity method. The basic idea of this method is to solve a time-dependent system of equations, whose steady-state solutions solve (1.2), until a steady state is reached. Methods have been proposed by Chorin (1967) and Yanenko (1967). The convergence rate of these methods is dependent on the choice of finite difference method used to solve the system. Moreover, as will be discussed in § 4, it may happen that the finite difference equations do not have a steady-state solution, so the method cannot converge. Taylor and Ndefo (1970) reported difficulty in getting Yanenko's method to converge, most likely because there was no solution.

Another common method is to use the "parabolized" Navier–Stokes equations which the second-derivatives in the stream-wise direction are removed. Because of its limited applicability and uncertain justification we will not discuss this method here except to note that often an analogue of (2.2) is derived and thus some of our observations on (2.2) also apply to the parabolized equations. Raithby and Schneider (1979) discuss these difficulties for three-dimensional flow problems.

3. Finite difference schemes. In this section we discuss the staggered mesh and central finite difference schemes for (1.1) and (1.2) and introduce a new scheme. The second-order accurate staggered mesh scheme for a uniform Cartesian grid assigns the values of each of the velocity components and the pressure to different interlaced grids. In two dimensions with velocity components u and v, one may assign values of u to grid locations $((i+\frac{1}{2})h, jh)$, values of v to $[ih, (j+\frac{1}{2})h)$, and values of p to (ih, jh) (see e.g. Harlow and Welch (1965), Patankar and Spalding (1972), Raithby and Schneider (1979), Brandt and Dinar (1979)). This method works very well as long as the geometry is rectangular and the grid is uniform. Nonuniform grids and grid mapping techniques cannot be conveniently handled, although Liu and Krause (1979) have developed a staggered mesh scheme for use with general geometries.

The staggered mesh schemes also have some difficulty at boundaries. For example, when both velocity components are specified at a boundary then that velocity component whose mesh lines do not lie on the boundary requires some special treatment.

The central difference scheme on a uniform rectangular mesh assigns values of all the variables to each grid point. The divergence and gradient operators are approximated using central differences and the Laplacian is approximated by the standard five-point discrete Laplacian. Central difference schemes have been used by Chorin (1967), (1968) in time-dependent calculations.

An important concept for finite difference schemes for elliptic systems such as (1.1) and (1.2) is that of regularity (see Bube and Strikwerda (1983), and also Frank (1968), Brandt and Dinar (1979)). Regular schemes give rise to regularity estimates analogous to those in the theory of elliptic systems of differential equations. Solutions to regular difference schemes will in general be smoother than solutions to nonregular schemes and also will be more accurate approximations to the solutions of the differential equations.

The central difference scheme is nonregular (Bube and Strikwerda (1983)), which results in nonsmooth solutions. The lack of smoothness is most noticeable in the pressure. The staggered mesh scheme is regular. The advantage of the central difference scheme is that it is easily implemented with nonuniform grids as introduced by coordinate changes.

It should be emphasized that none of the difficulties mentioned above are insurmountable. Both the staggered mesh and central differencing schemes have been used and often quite successfully. However we will consider a new scheme which incorporates both regularity and ease of implementation with coordinate grid mapping techniques.

Before introducing the new scheme we will discuss the concept of regularity for difference schemes as given in Bube and Strikwerda (1983). A difference operator A may be written as

$$Af(x)=\sum_{\mu} a_{\mu}(h,x)T^{\mu}f(x),$$

where T^{μ} is the translation operator given by

$$T^{\mu}f(x_{\nu})=f(x_{\nu+\mu})$$

for multi-indices ν and μ.

The symbol of A is given by

$$a(h,x,\zeta)=\sum_{\mu} a_{\mu}(h,x)\,e^{i\mu\cdot\zeta}.$$

For example, the first-order central difference operator in the kth coordinate direction has symbol

$$\frac{e^{i\zeta_k}-e^{-i\zeta_k}}{2h_k}=ih_k^{-1}\sin\zeta_k,$$

and the standard second-order accurate Laplacian in n variables has the symbol

$$-\sum_{k=1}^{n} 4h_k^{-2}\sin^2\tfrac{1}{2}\zeta_k.$$

A finite difference scheme for the Stokes equations is regular elliptic if the determinant of the matrix of symbols of the scheme vanishes only for ζ equal to zero modulo 2π. For the Stokes equations with central differencing, and $\Delta x=\Delta y=h$, this determinant is

$$\det\begin{pmatrix} 4h^{-2}(\sin^2\frac{1}{2}\zeta_1+\sin^2\frac{1}{2}\zeta_2) & 0 & ih^{-1}\sin\zeta_1 \\ 0 & 4h^{-2}(\sin^2\frac{1}{2}\zeta_1+\sin^2\frac{1}{2}\zeta_2) & ih^{-1}\sin\zeta_2 \\ ih^{-1}\sin\zeta_1 & ih^{-1}\sin\zeta_2 & 0 \end{pmatrix} \tag{3.1}$$
$$=4h^{-4}(\sin^2\tfrac{1}{2}\zeta_1+\sin^2\tfrac{1}{2}\zeta_2)(\sin^2\zeta_1+\sin^2\zeta_2).$$

This determinant vanishes for the dual variables ζ_1 and ζ_2 equal to π, and thus the scheme is not regular. One sees that the nonregularity comes from the form of the differencing used for the gradient and divergence terms. Our new scheme is a modification of the central differencing scheme so as to make the scheme regular.

The new scheme we consider will be called the regularized central difference scheme. In this scheme the derivatives of pressure are approximated as

$$\frac{\partial p}{\partial x_k}\simeq \delta_{k0}p-\alpha h_k^2\delta_{k-}\delta_{k+}^2 p \tag{3.2}$$

and the first derivatives of the velocity in the divergence equation are approximated as

$$\frac{\partial u^k}{\partial x_k}\simeq \delta_{k0}u^k-\alpha h_k^2\delta_{k+}\delta_{k-}^2 u^k, \tag{3.3}$$

where α is a nonzero constant and δ_{k0}, δ_{k+} and δ_{k-} are the centered, forward, and backward divided differences, respectively. The Laplacian is approximated with the usual five-point scheme. For a Cartesian grid in two dimensions the determinant of the symbol is

$$\det\begin{pmatrix} 4h^{-2}(\sin^2\frac{1}{2}\zeta_1+\sin^2\frac{1}{2}\zeta_2) & 0 & d(\zeta_1) \\ 0 & 4h^{-2}(\sin^2\frac{1}{2}\zeta_1+\sin^2\frac{1}{2}\zeta_2) & d(\zeta_2) \\ -\overline{d(\zeta_1)} & -\overline{d(\zeta_2)} & 0 \end{pmatrix}$$

$$=4h^{-2}(\sin^2\tfrac{1}{2}\zeta_1+\sin^2\tfrac{1}{2}\zeta_2)(|d(\zeta_1)|^2+|d(\zeta_2)|^2),$$

where

$$\begin{aligned} d(\zeta) &= ih^{-1}\sin\zeta-\alpha h^{-1}e^{1/2i\zeta}(2i\sin\tfrac{1}{2}\zeta)^3 \\ &= 2ih^{-1}\sin\tfrac{1}{2}\zeta(\cos\tfrac{1}{2}\zeta+4\alpha\, e^{1/2i\zeta}\sin^2\tfrac{1}{2}\zeta). \end{aligned}$$

since $d(\zeta)$ is not zero for any nonzero value of ζ, when α is nonzero, the scheme is regular. Note that for α equal to one-sixth the approximations (3.2) and (3.3) are third-order accurate.

Since the regularized central difference scheme is a variant of the central difference scheme it is easy to implement with coordinate maps. At those boundary points where the correction term would require points beyond the boundary we use the correction term which interchanges the forward and backward operators. This scheme also requires the use of extrapolation to compute the pressure values on the boundary. It has been found that third order extrapolation gave quite good results, e.g.

$$p_{0j}=3p_{1j}-3p_{2j}+p_{3j} \tag{3.4}$$

at the boundary $x=0$ in two dimensions.

The use of the extrapolation boundary condition (3.4) should not be confused with the discussion in § 2 about the extra boundary condition for the system of differential equations (2.2). The boundary condition (3.4) is required only by the finite difference scheme and is not required by the system of differential equations (1.1) or (1.2). Therefore the solution of the system (1.1) or (1.2) will not satisfy any extra nontrivial condition analogous to (3.4), even though the difference approximations will satisfy (3.4). Other extrapolations may be used in place of (3.4); however, if the order of extrapolation is too low the accuracy may be degraded at the boundary.

A number of first-order accurate schemes for the Stokes and Navier–Stokes equations have been presented e.g. Kzivickii and Ladyzhenskaya (1966) and Temam (1979, p. 48). In this paper we are concerned only with second-order accurate schemes.

4. The integrability condition. Each of the schemes for the Stokes equations which have been discussed in the previous section can be written as

$$(4.1)\quad \begin{array}{ll} \text{(a)} & L_h\mathbf{u}_h+\mathbf{G}_hp_h=\mathbf{f}_h, \\ \text{(b)} & \mathbf{D}_h\cdot\mathbf{u}_h=g_h, \end{array} \quad \text{on } \Omega_h,$$

with Dirichlet boundary conditions

$$\mathbf{u}_h=\mathbf{b}_h\quad \text{on } \partial\Omega_h.$$

The difference operators L_h, $\mathbf{G}_h$, and $\mathbf{D}_h$ are approximations to the differential operators in (1.1). The discrete functions $\mathbf{f}_h$, g_h, and $\mathbf{b}_h$ are approximations to $\mathbf{f}$, g and $\mathbf{b}$ on the mesh Ω_h, where h is some measure of the fineness of the mesh Ω_h.

Now let us compare the system (4.1) with the system (1.1). First note that if $\mathbf{G}_h$ is a consistent approximation to the gradient then the discrete pressure p_h is determined only up to a constant. This means that the system of linear equations (4.1) does not have full column rank. If there are as many equations in (4.1) as there are unknowns, and this is the case for each scheme we have considered, then the system (4.1) does not have full row rank either. This implies that there is a constraint which the data must satisfy to guarantee a solution; in particular, the discrete integrability condition analogous to (1.4) must be satisfied.

There are at least two ways to satisfy the discrete integrability condition. The first method would be to analyze the matrix corresponding to (4.1) and determine the null space of the adjoint matrix. If the data is constrained to be orthogonal to this null space then a solution will exist. This approach is impractical for many situations, especially if coordinate changes have been employed, since then the matrices are not easy to analyze.

A second approach, which will be adopted here, is to replace (4.1b) by

$$\mathbf{D}_h \cdot \mathbf{u}_h = g_h + \delta_h \tag{4.1b$'$}$$

where δ_h is a constant chosen to guarantee a solution. The value of δ_h must be determined as part of the solution. As shown in the examples in § 5 δ_h is at least $O(h^2)$ for the regularized central scheme. We will refer to the equations (4.1a, b′, c) as (4.2′).

Another way of looking at the condition (4.1b′) is as follows. As shown by Temam (1979) and others, any discrete divergence operator D_h, defined only on the interior of the grid, has a corresponding gradient operator $\mathbf{G}'_h$ defined by

$$(D_h\mathbf{u}, \phi) + (\mathbf{u}, \mathbf{G}'_h\phi) = 0 \tag{4.2}$$

for all grid vector functions $\mathbf{u}$ and scalar functions ϕ which vanish on the boundary. If one wishes to satisfy (4.1b) at each point of the interior then (4.2) with ϕ taken to be one at each interior point gives the requirement that

$$(g, 1) + (\mathbf{u}, \mathbf{G}'_h 1) = 0 \tag{4.3}$$

must be satisfied. This formula is the analogue of the integrability condition (1.4), and the second term in (4.3) will usually involve only the values of $\mathbf{u}$ on and near the boundary. If the constraint (4.3) is not satisfied then the data must be modified so that (4.3) is satisfied. The use of (4.1b′) in place of (4.1b) is one way by which (4.3) can be satisfied. An advantage of using (4.1b′) over approaches which would modify the boundary data of $\mathbf{u}$ is that (4.1b′) requires no explicit knowledge of $\mathbf{G}'_h$. Note that it is not necessary for $\mathbf{G}'_h$ to be the same as $\mathbf{G}_h$.

It is interesting to note that for the staggered mesh scheme on a uniform grid one can easily satisfy the discrete integrability condition since the calculus of finite differences mimics the differential calculus very closely, see e.g. Kzivickii and Ladyzhenskaya (1966). Similarly, Chorin (1969) proves the convergence of a central difference scheme for the time-dependent Navier–Stokes equations on a periodic rectangular mesh. An essential element of these proofs is that one has a convenient form of the finite difference analogue of the divergence theorem of the differential calculus. Liu and Krause (1979) develop a staggered grid scheme for nonrectangular grids and the success of their scheme is due to their careful treatment of the integrability constraint. Also, Ghia, Hankey and Hodge (1977) mention being unable to obtain a solution to the discrete Navier–Stokes equations for certain situations. We conjecture that this difficulty was caused by the discrete integrability condition not being satisfied.

There is the possibility that the null space of the discrete operator (4.1) has dimension greater than one. The regularized central scheme with the third-order extrapolation (3.4) appears to have only a one-dimensional null space. However, for α equal to zero numerical experiments indicate that there are solutions which are effectively null vectors in that they solve (4.1) with $\mathbf{f}_h$ and g_h smaller than the norm of the solution by a factor proportional to h or h^2. The dimension of the space of nearly null vectors and null vectors appears to be four for the central differencing scheme. These vectors correspond to the four zeros of the determinant of the symbol of the difference operator.

These nearly null vectors and null vectors, other than the usual constant pressure null solution, make solving the discrete system very difficult. On the other hand the regular discrete systems can be solved easily by the iterative procedure given in Strikwerda (1983).

5. Computational results. In this section we present the results of testing the new scheme described in § 3. In the examples discussed here the discrete Stokes equations were solved using test problems which illustrate various features of the schemes. For each example an exact analytical solution is known and the approximate solutions were compared to the exact solutions to study the accuracy of the method. The value of α, the regularity parameter, was one-sixth in all cases. We restrict ourselves here to the Stokes equations for reasons of simplicity. For low Reynolds numbers the nonlinearity of the Navier–Stokes equations usually does not present difficulties as great as those addressed in this paper. For higher Reynolds numbers the nonlinear effects cause additional computational problems which we do not wish to address here. The schemes presented here are being used in computations for the incompressible Navier–Stokes equations and the results will be reported when complete.

The iterative procedure which was used to solve the system of finite difference equations is described at length in Strikwerda (1983). The method consists of alternatively updating the velocity components by successive over-relaxation and updating the pressure by subtracting from the pressure at each grid point a multiple of the discrete divergence of the velocity field. This update of the pressure is of the same form as that used by Chorin (1968). The iterative method was stopped when the changes to the velocity field were sufficiently small and when the changes of the pressure were sufficiently close to being constant. The quantity δ_h was computed as the average value of the discrete divergence of the velocity minus the average value of g. The magnitude of δ_h is one measure of the truncation error of the scheme.

For the first test problem the Stokes equations were solved on the unit square with a uniform grid. The exact solution is

$$
\begin{aligned}
u &= (2\pi)^{-1} \sin \pi x \cos \pi y,\\
v &= (2\pi)^{-1} \cos \pi x \sin \pi y, \qquad (5.1)\\
p &= \cos \pi x \cos \pi y
\end{aligned}
$$

with $\mathbf{f} = 0$ and $g = \cos \pi x \cos \pi y$. For this example both the accuracy and symmetry of the solution were checked. The symmetry was checked to study the effect of the nonsymmetric regularizing term on the symmetry of the solution. The symmetry was measured by computing the quantities sym (u) and sym (p) given by

$$\text{(5.2)}\qquad \begin{aligned} \text{sym}\,(u) &= \left(\sum_{i,j=0}^{N} (u_{ij}+u_{N-i,N-j})^2\right)^{1/2} \Big/ \|u\|_2, \\ \text{sym}\,(p) &= \left(\sum_{i,j=0}^{N} (p_{ij}-p_{N-i,N-j})^2\right)^{1/2} \Big/ \|p-\bar{p}\|_2 \end{aligned}$$

for an $(N+1)\times(N+1)$ grid. The quantity $\bar{p}$ is the average value of the p_{ij} and the norm is the l^2-norm, e.g.

$$\|u\|_2 = \left(\sum_{i,j} u_{ij}^2\right)^{1/2}.$$

The second test problem demonstrates the ability of the scheme to produce second-order accurate solutions on a nonrectangular region. The exact solution is

$$\text{(5.3)}\qquad u=\xi^2+\eta^2, \quad v=-2\xi\eta+\xi^2, \quad p=4\xi+2\eta$$

on the region Ω which is the image of the unit square under the mapping

$$\xi = x\cosh(y), \qquad \eta = y-x^2$$

for (x, y) in the unit square, i.e. $0<x$, $y<1$. Thus the equations being solved on the unit square were

$$\begin{aligned} &x_\xi(x_\xi u_x)_x + x_\xi(y_\xi u_y)_x + y_\xi(x_\xi u_x)_y + y_\xi(y_\xi u_y)_y \\ &\quad + x_\eta(x_\eta u_x)_x + x_\eta(y_\eta u_y)_x + y_\eta(x_\eta u_x)_y + y_\eta(y_\eta u_y)_y - x_\xi p_x - y_\xi p_y = 0 \end{aligned}$$

for the first equation, with the second being similar, and

$$x_\xi u_x + y_\xi u_y + x_\eta v_x + y_\eta v_y = 0$$

for the third equation. The regularizing terms were added only to the terms corresponding to p_x in the first equation, p_y in the second, and u_x and v_y in the third. The regularizing terms were added to only these terms since that was sufficient to guarantee the regularity of the scheme.

In the third test problem the Stokes equations were solved on a disk using polar coordinates with uneven grid spacing in both the radial and angular direction. The exact solution is

$$\text{(5.4)}\qquad u=r^3\sin 2\theta, \quad v=2r^3\cos 2\theta, \quad p=6r^2\sin 2\theta$$

with **f** and g being zero. The uneven grid was given by

$$r_i = .75\rho_i + .25\rho_i^2, \qquad \theta_j = \varphi_j - .25\sin\varphi_j,$$

where ρ_i and φ_j were evenly spaced in the interval $[0, 1]$ and $[0, 2\pi]$ respectively. This uneven spacing was chosen merely to show the versatility of the scheme and is not intended to give a better resolution of the solution.

For completeness we give the Stokes equations in polar coordinates,

$$\text{(5.5)}\qquad \begin{aligned} &r^{-1}(ru_r)_r + r^{-2}u_{\theta\theta} - r^{-2}u - 2r^{-2}v_\theta - p_r = 0, \\ &r^{-1}(rv_r)_r + r^{-2}v_{\theta\theta} - r^{-2}v + 2r^{-2}u_\theta - r^{-1}p_\theta = 0, \\ &r^{-1}(ru)_r + r^{-1}v_\theta = 0. \end{aligned}$$

The difference formulas used in the numerical experiments were all second-order accurate. As an example of the formulas, the term $r^{-1}(ru_r)_r$ was differenced as

$$\left(\left(\frac{r_{i+1}+r_i}{r_{i+1}-r_i}\right)(u_{i+1,j}-u_{i,j})-\left(\frac{r_i+r_{i-1}}{r_i-r_{i-1}}\right)(u_{i,j}-u_{i-1,j})\right)\Big/\tfrac{1}{2}(r_{i+1}^2-r_{i-1}^2).$$

The results of the numerical experiments are shown in the following tables. Each table lists the errors incurred for grids with $N+1$ points on a side for values of N of 20, 30, 40 and 60. Tables 1, 2 and 3 list the relative errors for test problems 1, 2 and

TABLE 1
Errors for test problem 1 for grids with $N+1$ points on a side for four values of N. The numbers in parentheses are the decimal exponents, i.e., $.35(-3)=.35\times 10^{-3}$.

N	err (u)	err (p)	δ_h	sym (u)	sym (p)
20	.35 (−3)	.17 (−2)	−.44 (−5)	.68 (−3)	.13 (−2)
30	.11 (−3)	.86 (−3)	−.89 (−6)	.22 (−3)	.37 (−3)
40	.41 (−4)	.51 (−3)	−.53 (−6)	.82 (−4)	.15 (−3)
60	.19 (−4)	.23 (−3)	−.50 (−7)	.37 (−4)	.52 (−4)

TABLE 2
Errors for test problem 2

N	err (u)	err (p)	δ_h
20	.10 (−3)	.21 (−2)	−.24 (−3)
30	.45 (−4)	.92 (−3)	−.12 (−3)
40	.25 (−4)	.48 (−3)	−.74 (−4)
60	.11 (−4)	.22 (−3)	−.35 (+4)

TABLE 3
Errors for test problem 3

N	err (u)	err (p)	δ_h
20	.75 (−1)	.93 (−1)	−.33 (−2)
30	.33 (−1)	.34 (−1)	−.53 (−3)
40	.19 (−1)	.18 (−1)	−.15 (−3)
60	.83 (−2)	.75 (−2)	−.27 (−4)

3, respectively, and Table 1 also shows the symmetry errors for problem 1. The relative errors are measured in the l^2-norm, i.e.,

$$\text{err}\,(u)=\left(\sum (u_{ij}-u(x_i,y_i))^2\right)^{1/2}/\|u\|_2.$$

The error in pressure is computed similarly except that the norms are taken modulo additive constants, i.e.

$$\text{err}\,(p)=\|p_h-p_e-\overline{(p_h-p_e)}\|_2/\|p_e-\bar{p}_e\|_2$$

where p_h and p_e are the approximate and exact solutions, respectively. Also shown is the value of δ_h which is described in § 4. Table 4 displays the behavior of the error

TABLE 4
Computed order of accuracy for u, p, and δ_h for the test problems

N_1/N_2		1	2	3
	u	2.8	2.0	2.0
30/20	p	1.7	2.0	2.5
	δ_h	4.0	1.7	4.5
	u	3.4	2.0	1.9
40/30	p	1.8	2.3	2.2
	δ_h	1.9	1.7	4.4
	u	3.1	2.0	2.0
40/20	p	1.7	2.1	2.4
	δ_h	3.1	1.7	4.5
	u	2.5	2.0	2.0
60/30	p	1.9	2.1	2.2
	δ_h	4.2	1.8	4.3
	u	1.9	2.0	2.0
60/40	p	2.0	1.9	2.2
	δ_h	5.8	1.8	4.2

as the grid resolution is increased. The numbers shown are values of

$$-\frac{\log(\mathrm{err}_1/\mathrm{err}_2)}{\log(N_1/N_2)}$$

where err_1 and err_2 are the errors for grids of N_1+1 and N_2+1 points on a side, respectively. This value should be approximately 2.0 for a second-order scheme. The error reductions are shown for u, p and δ_h. The other velocity component had a similar error behavior in all the examples. All of the solutions were computed by the iterative method given in Strikwerda (1983).

That some of the errors were better than second-order accurate for test problems 1 and 3 can be attributed to the third-order accurate difference formulas used for the gradient and divergence terms. One might expect that some of the errors would behave as third-order errors for some value of N_1 and N_2. However, since the discrete Laplacian is second-order accurate, for N large enough the total scheme should be second-order accurate. It is not clear why δ_h should behave as a fourth-order error as seen in test problem 3 and for some values of N_1 and N_2 in test problem 1. Test problem 2 was no better than second-order accurate since the gradient and divergence were only second-order accurate. The third-order differences were only used on those terms which were necessary to achieve regularity of the scheme. The results show conclusively that the scheme has overall second-order accuracy.

Test problem 1 for $N=40$ is similar to the test problem of Chorin (1968) for the time-dependent Navier–Stokes equations with Reynolds number of 1.0. While the accuracy of the velocity components is of the same order of magnitude for both problems, the results for pressure are more accurate for the regularized central scheme than are Chorin's results by at least an order of magnitude. We attribute this increase in accuracy to both the regularized differencing and the use of the integrability variable δ_h.

In engineering computations it is very important to know the accuracy of one's results. For the incompressible Navier–Stokes calculations the pressure is important since it is used to compute drag and lift forces, however, the pressure is often computed with only indifferent accuracy, e.g. Rubin and Harris (1975). The second-order accuracy obtained for pressure by the regularized central scheme demonstrates the advantage of this scheme over other existing schemes.

6. Conclusion. In this paper we have examined several finite difference methods for the steady Stokes and incompressible Navier–Stokes equations in primitive variables. We have shown that the regularized centered difference scheme is second-order accurate and useful with nonrectangular regions. Although the numerical experiments were done using the Stokes equations, for which exact solutions were available, we believe the regularized central scheme is equally useful with the incompressible Navier–Stokes equations at moderate Reynolds number.

Acknowledgment. The author wishes to thank the referees for their very helpful suggestions and comments.

REFERENCES

S. Agmon, A. Douglis and L. Nirenberg (1964), *Estimates near the boundary for solutions of elliptic partial differential equations satisfying general boundary conditions*, II, Comm. Pure Appl. Math., 17, pp. 35–92.

A. Brandt and N. Dinar (1979), *Multi-grid solutions to elliptic flow problems*, Proc. Conference on Numerical Solutions of Partial Differential Equations, Mathematical Research Center, Madison, WI, October 1978.

K. Bube and J. Strikwerda (1983), *Interior regularity estimates for elliptic systems of difference equations*, SIAM J. Numer. Anal., 20, pp. 639–656.

A. J. Chorin (1967), *A numerical method for solving incompressible viscous flow problems*, J. Comp. Phys., 2, pp. 12-26.

——— (1968), *Numerical solution of the Navier–Stokes equations*, Math. Comp., 22, pp. 745–762.

——— (1969), *On the convergence of discrete approximations to the Navier–Stokes equations*, Math. Comp., 23, pp. 341–353.

A. Douglis and L. Nirenberg (1955), *Interior estimates for elliptic systems of partial differential equations*, Comm. Pure Appl. Math., 8, pp. 503–538.

L. Frank (1968), *Difference operators in convolutions*, Soviet Math. Dokl., 9, pp. 831–834.

K. N. Ghia, W. L. Hankey and J. K. Hodge (1977), *Study of incompressible Navier-Stokes equations in primitive variables using implicit numerical technique*, AIAA paper 77–648.

F. H. Harlow and J. E. Welch (1964), *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, Phys. Fluids, 8, pp. 2181–2189.

A. Kzivickii and O. A. Ladyzhenskaya (1966), *The method of nets for the non-stationary Navier–Stokes equations*, Proc. Steklov Inst., 92, pp. 105–112. (In Russian.)

O. A. Ladyzhenskaya (1963), *The Mathematical Theory of Viscous Incompressible Flows*, translated by R. A. Silverman, Gordon and Breach, New York.

N. S. Liu and E. Krause (1979), *Calculation of incompressible viscous flows in vessels with moving boundaries*, Acta Mech., 33, pp. 21–32.

S. V. Patankar and D. B. Spalding (1972), *A calculation procedure for heat, mass, and momentum in three dimensional parabolic flows*, Internat. J. Heat Mass Trans., 15, pp. 1787–1806.

J. Patterson and J. Imberger (1980), *Unsteady natural convection in a rectangular cavity*, J. Fluid Mech., 100, pp. 65–86.

C. S. Peskin (1977), *Numerical analysis of blood flow in the heart*, J. Comp. Phys., 25, pp. 220–252.

G. E. Raithby and G. D. Schneider (1979), *Numerical solution of problems in incompressible fluid flow: Treatment of the velocity-pressure coupling*, Num. Heat Trans., 2, pp. 417–440.

P. Roache (1972), *Computational Fluid Dynamics*, Hermosa, Albuquerque, NM.

S. Rubin and J. Harris, eds. (1975), *Numerical studies of incompressible viscous flow in a driven cavity*, NASA SP-378.

J. S. STRIKWERDA (1983), *An iterative method for solving the Stokes equations*, Tech. Summ. Rep. Mathematics Research Center, Univ. Wisconsin, Madison, 2490.

T. D. TAYLOR AND E. NDEFO (1970), *Computation of viscous flow in a channel by the method of splitting*, Proc. Second International Conference Numerical Methods in Fluid Dynamics, pp. 356–364.

R. TEMAM (1979), *Navier–Stokes Equations*, North-Holland, Amsterdam.

J. F. THOMPSON (1980), *Numerical solution of flow problems using body-fitted coordinate systems*, Computational Fluid Dynamics, W. Kollman, ed., Hemisphere Publ. Corp., Washington, DC.

J. A. VIECELLI (1971), *A computing method for incompressible flows bounded by moving walls*, J. Comp. Phys., 8, pp. 119–143.

N. N. YANENKO (1971), *The Method of Fractional Steps*; *The Solution of Problems of Mathematical Physics in Several Variables*, translated by M. Holt, Springer-Verlag, Berlin.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

HIGH ORDER DIFFERENCE SCHEMES FOR LINEAR PARTIAL DIFFERENTIAL EQUATIONS*

R. MANOHAR†‡ AND J. W. STEPHENSON†

Abstract. A procedure for deriving high order difference formulas using the local solutions of the differential equation is described. The same procedure can be used to incorporate the boundary conditions in the derivation of the difference formulas for the boundary mesh points. A simple example of a Poisson equation over a rectangle is chosen to demonstrate the method, although the same procedure can be applied to equations with variable coefficients over arbitrary regions.

Key words. finite difference schemes, partial differential equations, Poisson equation

Introduction. A procedure for deriving high order difference formulas for solving linear partial differential equations is described. This procedure is based upon expressing the solution locally about a given mesh point as a linear combination of the analytic solutions of the differential equation. The finite difference formulas are obtained by collocation over a set of mesh points surrounding the given mesh point for which the difference formula is derived. For a mesh point on the boundary additional constraints are provided by the boundary conditions, otherwise the procedure remains the same. In any case, the difference formula takes into account the local solution of the differential equation. Initially we considered the solutions obtained by the method of separation [1], however, it was found that the same difference formula can be obtained more efficiently by using a truncated power series solution of the differential equation. Once this choice is made, then our procedure has strong connections with the "Mehrstellenverfahren" of Collatz [2], and the works of Young and Dauwalder [3], and that of Lynch and Rice [4].

The results of Young and Dauwalder [3] for elliptic equations are more general in certain details than those derived here. However, we have intentionally chosen a very special and a simple example of the Poisson equation in two dimensions to demonstrate the procedural details. We feel that our procedure is simpler and more direct and the resulting algorithm can be used with ease on a computer to generate the difference formulas automatically in the case of more complex differential equations over arbitrary regions. The treatment of the boundary conditions is easily incorporated in the procedure.

Lynch and Rice [4] call their method by the acronym HODIE (High Order Differences with Identity Expansion). Their procedure is a generalization of "Mehrstellenverfahren" and the work of Young and Dauwalder. They have studied in great detail the computational effort required in generating the difference formulas and also solving the resulting system of linear equations. They have clearly demonstrated the usefulness of high order methods and have shown that such methods are computationally efficient. In the algorithm for the derivation of difference formulas, Lynch and Rice use the nodal values of the forcing function f, which is computationally more convenient but requires the "identity expansion" in their procedure. This often creates problems in the automatic generation of formulas due to the fact that one of the matrices appearing in the derivation becomes singular or nearly singular particularly

* Received by the editors October 23, 1981, and in revised form October 26, 1982.

† Department of Mathematics, University of Saskatchewan, Saskatoon, Saskatchewan, Canada S7N 0W0.

‡ On leave at the Department of Chemical Engineering, University of Alberta, Edmonton, Alberta, Canada T6G 2G1.

when a uniform mesh is used. Our procedure avoids this difficulty by separating the determination of the coefficients of the nodal values of the solution u and the coefficients of the nodal values of f. We first of all determine the coefficients of the nodal values of u using the Taylor coefficients of f. This gives us the "Mehrstellenverfahren" type formulas which can be used provided the Taylor coefficients of f are available. However, if it is necessary to replace the Taylor coefficients of f with nodal values of f, then the Taylor coefficients which appear in our formulas indicate an appropriate choice of the nodal points to be used for such an interpolation of f. This produces HODIE type formulas and avoids the difficulties inherent in the a priori selection of nodes in the HODIE method. In the final analysis the two procedures are equivalent in the sense that both procedures use local polynomial expansions. In the case of the Poisson equation, the two methods produce identical formulas.

It is implicitly assumed in all these procedures aimed at deriving high order difference formulas that the coefficients in the differential equation and also the solutions are sufficiently smooth. If the smoothness condition is satisfied then these procedures give difference formulas with the highest order of truncation error for a given set of mesh points. In this sense the difference formulas obtained are "optimal."

Although some of the difference formulas derived here to demonstrate our procedure are well known, the same procedure can be applied to more general linear elliptic equations with variable coefficients as in [3], [4] and [5] and also to other equations such as the parabolic equation given in [1]. A further generalization of our procedure has been applied to the biharmonic equation in which not only the nodal values of the solutions but also the values of the derivatives at the nodal points are used for collocation [7]. This generalization allows the handling of the boundary conditions with much greater ease.

Derivation of the difference formulas. All the important features of the method can be described by using a simple example of the Poisson equation

$$(1)\qquad \Delta u = u_{\xi\xi} + u_{\eta\eta} = f(\xi, \eta), \qquad 0 \leqq \xi, \quad \eta \leqq 1$$

over a square. It is assumed that the values of u are prescribed on three sides of the square and the normal derivative of u is given by a function $g(\eta)$ on the right boundary $\xi = 1$. Assume that the region is subdivided into square subregions and that the mesh size h is uniform. Let (ξ_0, η_0) be a typical mesh point for which a difference formula is to be derived. It is convenient to shift the origin to this point and use local coordinates (x, y). Two different types of difference formulas are needed to solve the problem, one for the interior mesh points and the other for the boundary mesh points.

Let us assume the following expansions for the solution $u(x, y)$, $f(x, y)$ and the boundary function $g(y)$,

$$(2)\quad u(x, y) = \sum a_{i,j} x^i y^j, \quad f(x, y) = \sum c_{i,j} x^i y^j, \quad g(y) = \sum b_i y^i, \qquad i, j = 0, 1, 2, \cdots.$$

The coefficients $a_{i,j}$ are unknown, while $c_{i,j}$ and b_j are either known or they can be calculated. The differential equation (1) imposes the following constraints

$$(3)\qquad (i+1)(i+2)a_{i+2,j} + (j+1)(j+2)a_{i,j+2} = c_{i,j}, \qquad i, j = 0, 1, 2, \cdots.$$

The additional constraints

$$(4)\qquad b_j = a_{1,j}, \qquad j = 0, 1, 2, \cdots$$

are imposed if the normal derivative boundary condition is applied at $x = 0$.

In order to derive a difference formula let us first approximate u by neglecting terms of degree greater than N say, by setting the corresponding coefficients $a_{i,j}$ to zero. This immediately determines the number of unknowns $a_{i,j}$ in the expansion of u say n, and the number of constraints (equations) that are retained in (3) or (4) say m_1 and m_2 respectively. In order to derive a difference formula for an interior mesh point choose $m = n - m_1$ mesh points around the origin. Let the coordinates of these m mesh points be (x_k, y_k), $k = 1, 2, \cdots, m$. Collocation of u at these mesh points gives the following m linear equations

$$u_k = u(x_k, y_k) = \sum a_{i,j} x_k^i y_k^j, \qquad k = 1, 2, \cdots, m.$$

These m equations along with the m_1 constraints in (3) determine the n unknowns $a_{i,j}$ in terms of the m nodal values $u_1, u_2, \cdots, u_m$ and the m_1 coefficients $c_{i,j}$, provided that the coefficient matrix is nonsingular. Once this is done, the difference formula is obtained from $u_0 = u(0, 0) = a_{0,0}$, where $a_{0,0}$ is expressed in terms of $u_1, u_2, \cdots, u_m$. For a difference formula for a boundary mesh point it is necessary to consider m_2 constraints given by (4) in addition to the m_1 constraints given by (3). Therefore, only $n - (m_1 + m_2)$ mesh points lying in the region and on the boundary are chosen for collocation. The expressions for the other coefficients such as $a_{1,0}$, $a_{0,1}$ etc. provide difference formulas for the derivatives of u.

There are three important points that may be mentioned in connection with the method of derivation proposed here. Firstly, it is not necessary to include all the terms of the highest degree N in the expansion of u, though, if any one term is chosen all those terms which are connected to it through any one of the constraints in (3) must be included. Secondly, if all the terms of degree N and lower are retained in the expansion of u, then the minimum order of the truncation can immediately be predicted to be h^{N-1} because of the fact that the difference formula is accurate for all polynomials of degree N or less. If the collocation points are chosen to take advantage of the symmetry of the operator, the difference formula can achieve a higher order of truncation error. This can be tested by substituting higher degree terms in the difference formula. Another way of asserting the higher order is to choose additional terms in the expansion and additional mesh points in such a way that the difference formula still remains the same i.e. the additional nodal values have the weight zero in the formula. Finally, for a certain distribution of the mesh points, the coefficient matrix may be singular. This indicates that additional mesh points and terms in the expansion are needed to get a high order formula. Although it is not necessary, we generally prefer to use mesh points which form a single cell around the origin. For equations with constant coefficients on a domain which can be partitioned into squares, this eliminates the necessity of special formulas for mesh points near the boundary.

These features can be demonstrated in the development of the classical 5-point formula for the Poisson equation. Higher order formulas can be derived in an analogous manner. Let a typical mesh point 0 be an interior mesh point as shown in Fig. 1. Let u in (2) be approximated by

$$u(x, y) = a_{0,0} + a_{1,0}x + a_{0,1}y + a_{2,0}x^2 + a_{0,2}y^2. \tag{5}$$

From (3) we get only one constraint given by

$$2(a_{2,0} + a_{0,2}) = c_{0,0}. \tag{6}$$

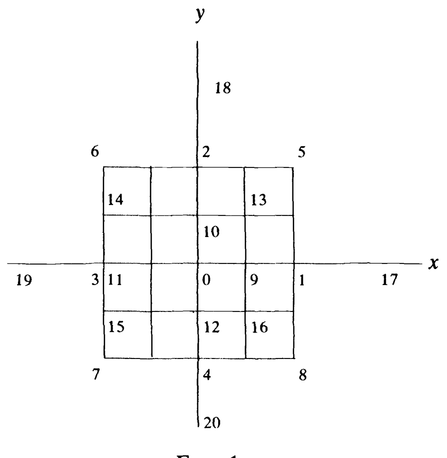


FIG. 1

The four points needed for collocation are chosen to be 1, 2, 3, and 4 in Fig. 1. The system of equations that follows is given by

$$(7)\qquad \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & -1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix} \begin{bmatrix} a_{0,0} \\ a_{1,0}h \\ a_{0,1}h \\ a_{2,0}h^2 \\ a_{0,2}h^2 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ c_{0,0}h^2 \end{bmatrix}$$

The solution of (7) gives

$$(8)\qquad a_{0,0} = u_0 = \tfrac{1}{4}(u_1 + u_2 + u_3 + u_4) - \tfrac{1}{4}c_{0,0}h^2.$$

If $c_{0,0}$ is replaced by $f_0 = f(0, 0)$, then (8) becomes the classical 5-point formula. If all the terms of degree 3 are included in the expansion of u, then there are 10 unknown coefficients and 3 constraints. If we choose any seven points in Fig. 1 including the points 1, 2, 3 and 4, we again get formula (8) by this procedure. This demonstrates that the formula (8) is of order h^2. The other coefficients $a_{1,0}$, $a_{0,1}$, $\cdots$ determined from (7) are useful, for example

$$[u_x]_{0,0} = a_{1,0}, \quad [u_y]_{0,0} = a_{0,1}, \quad [u_x]_{h,0} = a_{1,0} + 2a_{2,0}h + 3a_{3,0}h^2 + \cdots.$$

These formulas approximate the derivatives to an order one less than the corresponding formula for u. Numerical experiments carried out on several problems in particular see [1], show that these formulas give excellent results.

For the boundary mesh point shown in Fig. 2, in order to obtain a formula compatible with (8), we include all the terms up to degree 3 in the expansion of u. This means there are three constraints of the type (3) and three constraints of the type (4). The four points chosen for collocation are 1, 2, 3 and 4 shown in Fig. 2. The difference formula obtained by this procedure is

$$(9)\qquad u_0 = \tfrac{1}{4}(u_1 + u_2) + \tfrac{1}{2}u_4 + \tfrac{1}{2}b_0h - \tfrac{1}{6}b_2h^3 - \tfrac{1}{4}c_{0,0}h^2 + \tfrac{1}{12}c_{1,0}h^3.$$

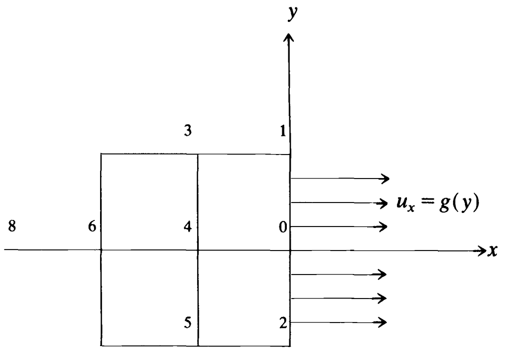


FIG. 2

If we replace $c_{0,0}$ by f_0 and $c_{1,0}$ by f_0-f_4 we get

$$u_0=\tfrac{1}{4}(u_1+u_2)+\tfrac{1}{2}u_4+\tfrac{1}{2}b_0h-\tfrac{1}{6}b_2h^3-\tfrac{1}{6}f_0h^2-\tfrac{1}{12}f_4h^2. \tag{10}$$

It is possible to replace b_0 by $(u_x)_0$, $2b_2h^2$ by $(u_x)_1+(u_x)_x-2(u_x)_0$ in (10), thereby expressing all the terms in (10) in terms of the nodal values. Note that u_3 does not appear in the formula (9). If we set $b_2=0$ and write $f_4=f_0$ in (10), we get the classical formula for the boundary point which is of order h, while (10) is of order h^2. Other formulas may be obtained in a similar fashion. For instance, if we select points 1, 2, 3 and 5 in Fig. 2, we get

$$u_0=\tfrac{1}{2}(u_3+u_5)+b_0h+\tfrac{2}{3}b_2h^3-\tfrac{1}{2}c_{0,0}h^2+\tfrac{1}{6}c_{1,0}h^3 \tag{11}$$

which is of order h^2.

In general, the use of formulas (10) or (11) on the boundary in place of the classical formula along with the second order formula (8) in the interior are not likely to produce better results. This is due to the fact that a boundary formula is permitted to be one order less than interior formulas to maintain the same overall accuracy. In this sense, these formulas are not of any practical significance. We have given them here to demonstrate the method which has been used to derive higher order formulas.

It is not necessary to select only those mesh points which are symmetrically placed around the mesh point 0 shown in Fig. 1. Formulas with an arbitrary selection of mesh points can be obtained by the procedure described earlier. Such a distribution of points may be needed for points near the boundary or on a curved boundary or where the mesh is not uniform. In the case of curved boundaries it may not always be possible to write the constraints in the form (4). In this case each constraint is replaced by collocation of the boundary conditions at one additional boundary point. In fact, for any other linear differential equation, the only modification made is in the constraints which follow from the differential equations or the boundary conditions.

For symmetrically placed points in the Poisson equation, one can find formulas of high order with very few additional points. We now give some of the formulas of order h^4 and h^6 obtained by this method.

Following [2] we write

$$\begin{aligned} L_hu_0 &= 4(u_1+u_2+u_3+u_4)+(u_5+u_6+u_7+u_8)-20u_0 \\ &= 4\Diamond u_0+\Box u_0-20u_0. \end{aligned}$$

Then some of the formulas for the interior points are

(12a) $$L_h u_0 = 6c_{0,0}h^2 + (c_{2,0}+c_{0,2})h^4,$$

(12b) $$L_h u_0 = 6f_0 h^2 + \tfrac{1}{2}(\Diamond f_0 - 4f_0)h^2,$$

(12c) $$L_h u_0 = 6f_0 h^2 + \tfrac{1}{4}(\Box f_0 - 4f_0)h^2,$$

(13a) $$L_h u_0 = 6c_{0,0}h^2 + (c_{2,0}+c_{0,2})h^4 + [\tfrac{2}{5}(c_{4,0}+c_{0,4}) + \tfrac{4}{15}c_{2,2}]h^6,$$

(13b) $$L_h u_0 = [-\tfrac{8}{15}f_0 - \tfrac{1}{30}\Diamond f_0 + \tfrac{1}{15}\Box f_0 + \tfrac{8}{5}\underline{\Diamond} f_0]h^2,$$

(13c) $$L_h u_0 = [\tfrac{119}{30}f_0 + \tfrac{7}{15}\Diamond f_0 + \tfrac{1}{15}\Box f_0 - \tfrac{1}{40}\bar{\Diamond} f_0]h^2,$$

(13d) $$L_h u_0 = [\tfrac{37}{15}f_0 + \tfrac{1}{15}\Diamond f_0 + \tfrac{1}{60}\Box f_0 + \tfrac{4}{5}\underline{\Box} f_0]h^2,$$

where

$$\underline{\Diamond} f_0 = f_9 + f_{10} + f_{11} + f_{12},$$

$$\underline{\Box} f_0 = f_{13} + f_{14} + f_{15} + f_{16},$$

$$\bar{\Diamond} f_0 = f_{17} + f_{18} + f_{19} + f_{20} \quad \text{(see Fig. 1).}$$

Formulas (12) and (13) are of order h^4 and h^6 respectively. Observe that if the nodal values of f are used, then the 9 mesh points of the cell are not sufficient to give a formula of order h^6. Use of the additional values of f may be acceptable, if f is known at every point. However, there are problems where only the nodal values of f are known. In such cases it may be more convenient to use the method of order h^4 given by (12). Formula (13c) is not convenient to use since near the boundary special formulas would have to be used.

Some formulas for the boundary mesh point of Figure 2 are given by

(14a) $$L_h^1 u_0 = u_0 - \tfrac{2}{5}u_4 - \tfrac{1}{10}(u_3+u_5) - \tfrac{1}{5}(u_1+u_2)$$
$$= -\tfrac{3}{10}c_{0,0}h^2 + \tfrac{1}{10}c_{1,0}h^3 - \tfrac{1}{20}(c_{2,0}+c_{0,2})h^4 + \tfrac{3}{5}b_0 h$$

(14b) $$= -\tfrac{1}{40}h^2[f_1 + f_2 - f_6 + 6f_4 + 5f_0] + \tfrac{3}{5}b_0 h,$$

(15a) $$L_h^1 u_0 = -\tfrac{3}{10}c_{0,0}h^2 + \tfrac{1}{10}c_{1,0}h^3 - \tfrac{1}{20}(c_{2,0}+c_{0,2})h^4 + \tfrac{3}{100}c_{3,0}h^5$$
$$+ \tfrac{7}{300}c_{1,2}h^5 + \tfrac{3}{5}b_0 h - \tfrac{2}{25}b_4 h^5$$

(15b) $$= -\tfrac{1}{75}h^2[f_1 + f_2 + f_8] + \tfrac{1}{200}h^2[13f_6 - 27f_0] - \tfrac{1}{6}h^2 f_4$$
$$- \tfrac{7}{600}h^2[f_3 + f_5] + \tfrac{3}{5}b_0 h - \tfrac{2}{25}b_4 h^5.$$

Formulas (14) and (15) are of order h^4 and h^6 respectively. Note that the formulas (14) make use of the normal derivative at the point 0 only since $b_0 = [u_x]_0$. Some numerical results obtained using these formulas are given in the next section.

Finally, the formulas for the derivatives u_x and u_y corresponding to the formulas (12) are given by

(16a) $$hu_{x0} = \tfrac{1}{3}(u_1 - u_3) + \tfrac{1}{12}(u_5 - u_6 - u_7 + u_8) - \tfrac{1}{12}h^2(f_1 - f_3)$$

and

(16b) $$hu_{y0} = \tfrac{1}{3}(u_2 - u_4) + \tfrac{1}{12}(u_5 + u_6 - u_7 - u_8) - \tfrac{1}{12}h^2(f_2 - f_4).$$

Numerical results. In order to demonstrate the accuracy and usefulness of the higher order formulas we have solved (1) over the square $0 \leqq \xi, \eta \leqq 1$: The test problems considered have the exact solutions

$$u = F_1(\xi, \eta) = \xi^5 + \xi^3\eta^2 + 2\eta^5,$$

$$u = F_2(\xi, \eta) = \xi^8 + 2\xi^7 + 3\xi^5\eta^3 + 4\xi^2\eta^6 + 3\eta^8,$$

$$u = F_3(\xi, \eta) = e^{2\xi} \cos \eta.$$

In Table 1 we give the results for the Dirichlet problem when u is given on all four sides. The derivatives are computed using central differences when (8) is used

TABLE 1
Maximum errors for Dirichlet problems

	Formulas					
h	(8)	Derivs.	(12a)	Derivs.	(13a)	Derivs.
Problem F_1						
1/3	.1080	.4090	exact	.2881E-1	exact	.2881E-1
1/6	.3016E-1	.1535	exact	.1800E-2	exact	.1800E-2
1/12	.7893E-2	.5021E-1	exact	.1125E-3	exact	.1125E-3
1/24	.1991E-2	.1486E-1	exact	.7118E-5	exact	.7118E-5
r	1.99	1.76		3.98		3.98
Problem F_2						
1/3	.8293	1.738	.9032E-1	.8572	.8868E-3	.7693
1/6	.2418	1.132	.5820E-2	.1008	.1685E-4	.8842E-1
1/12	.6364E-1	.5026	.3721E-3	.8492E-2	.2606E-6	.7286E-2
1/24	.1617E-1	.1737	.2327E-4	.6109E-3	.1471E-7*	.5193E-3
r	1.98	1.53	4.0	3.80	6.01	3.81
Problem F_3						
1/3	.2680E-1	.5141	.1899E-4	.2699E-1	.4530E-7	.2698E-1
1/6	.7601E-2	.1795	.1266E-5	.2412E-2	.8254E-9	.2410E-2
1/12	.1941E-2	.5391E-1	.7777E-7	.1791E-3	exact	.1790E-3
1/24	.4922E-3	.1495E-1	.1771E-7	.1218E-4	exact	.1217E-4
r	1.98	1.85	4.02	3.88	5.78	3.88

and (16) when the high order formulas are used. In Table 2, we give results when u is given on three sides and the normal derivative is given on $\xi = 1$. In both tables we list the maximum absolute error in the computed approximation. The table also includes the experimentally observed rate of convergence r, given by

$$r = \log\left(\frac{e_2}{e_1}\right) \Big/ \log\left(\frac{h_2}{h_1}\right)$$

where e_2 and e_1 are the errors observed for $h_2 = 1/24$ and $h_1 = 1/12$ respectively. In the case when the results are affected by roundoff errors as shown by * or when they are exact, previous values are used. We consider the results exact if errors are less than 10^{-9}. The computations were carried out on a DEC-20 in double precision.

TABLE 2
Maximum errors for the mixed problem

	Formulas				
h	(8), classical	(8), (10)	(8), (11)	(12a), (14a)	(13a), (15a)
Problem F_1					
1/3	.1562	.1430	.1481	.5252E-2	exact
1/6	.4782E-1	.4240E-1	.4488E-1	.3754E-3	exact
1/12	.1268E-1	.1208E-1	.1245E-1	.2370E-4	exact
1/24	.3202E-2	.3221E-2	.3283E-2	.1520E-5	exact
r	1.98	1.91	1.92	3.96	
Problem F_2					
1/3	1.245	.9493	1.010	.2757	.9739E-2
1/6	.4115	.3431	.3787	.1912E-1	.1689E-3
1/12	.1078	.1082	.1164	.1225E-2	.2677E-5
1/24	.2722E-1	.3126E-1	.3255E-1	.7731E-4	.7755E-7*
r	1.99	1.79	1.84	3.99	5.98
Problem F_3					
1/3	.2241	.3052E-1	.8987E-1	.1854E-2	.4837E-6
1/6	.6658E-1	.1123E-1	.6595E-2	.1269E-3	.9462E-8
1/12	.1728E-1	.3498E-2	.2700E-2	.8072E-5	exact
1/24	.4349E-2	.9991E-3	.8709E-3	.5618E-6	exact
r	1.99	1.8	1.63	3.84	5.68

Because of storage limitations, matrices were stored in banded form and the solutions were obtained using S.O.R.

As expected, the results using formulas (10) and (11) give similar results to those obtained by the use of the classical formula. However, to illustrate that these formulas are an improvement on the classical formula, we give in Table 3 a comparison of the results obtained when we used the classical formula and the second order improved formulas (10) and (11) with the fourth order formula (12a) for interior points.

The truncation errors for the formulas in (12) and (13) are $O(h^4)$ and $O(h^6)$ respectively. Explicit expressions for the truncation error can be written down. In the case of the Poisson equation, for h sufficiently small, the descretizations are of positive type, hence the discretization error is of the same order as the truncation error. This result also holds for the Helmholtz equation, as shown by Boisvert [6].

Conclusions. A method for obtaining high order difference formulas for linear partial differential equations has been described. For equations with constant coefficients and also for equations in which the coefficients are polynomials of low degree, the difference equations are the same for every mesh point for which the mesh elements are similarly shaped. In the case of irregular mesh points, it is necessary to determine the difference formula for every mesh point by solving a system of equations. This involves extra computational effort which can be justified in view of the accuracy attained. Estimates of the number of operations required in deriving the difference equations and then solving them have been given by Lynch and Rice [4]. The operation count in our procedure is not the same, but we expect the order of magnitude to be the same.

The difference equations we obtain by our method are the same as those obtained by the procedure of Lynch and Rice for the Poisson equation but our method of

TABLE 3
Classical versus "improved" formulas

	h	Classical	(10)	(11)
Problem F_1	1/3	.2326	.9860E-2	.5751E-1
	1/6	.8538E-1	.2527E-2	.5840E-2
	1/12	.2405E-1	.3931E-3	.6599E-3
	1/24	.6320E-2	.5477E-4	.7953E-4
	r	1.93	2.84	3.05
Problem F_2	1/3	1.674	.4941	.4196
	1/6	.7181	.9912E-1	.6548E-1
	1/12	.2186	.1546E-1	.8187E-2
	1/24	.5922E-1	.2110E-2	.1096E-2
	r	1.88	2.87	2.90
Problem F_3	1/3	.2410	.1369	.1040
	1/6	.7715E-1	.2513E-2	.1302E-1
	1/12	.2084E-1	.3600E-3	.1693E-2
	1/24	.5378E-2	.4780E-4	.2166E-3
	r	1.95	2.91	2.97

derivation is quite different. It is possible to predict a priori the maximum number of nodal values required in the difference formula to attain a certain order of accuracy. Another important feature of our procedure is that we avoid the use of the nodal values of the coefficient functions at least when the difference formulas are derived. The option of using these values if necessary is exercised independently. All the other conclusions and generalization remain the same as given by Lynch and Rice.

A computer program for second order elliptic equations with constant coefficients has been developed using the technique described here. For a given degree, the program requires coordinates of a certain number of grid points for collocation. In the case when the system of equations for the determination of the difference formula is singular, a different selection of grid points is called for. The program also checks for the lowest degree for which the formula is not exact. In this way, questions of consistency can be answered.

REFERENCES

[1] R. MANOHAR AND J. W. STEPHENSON, *Optimal finite analytic methods*, Journal of Heat Transfer, 104 (1982). pp. 432–437.

[2] L. COLLATZ, *The Numerical Treatment of Differential Equations*, 3rd ed., Springer-Verlag, New York, 1960.

[3] D. M. YOUNG AND J. H. DAUWALDER, *Discrete representations of partial differential operators*, in Error in Digital Computation, L. Rall, ed., Academic Press, New York, 1965, pp. 181–207

[4] R. E. LYNCH AND J. R. RICE, *The Hodie method and its performance for solving elliptic partial differential equations*, in Recent Developments in Numerical Analysis, C. de Boor, ed, Academic Press, New York, 1978, pp. 143–175.

[5] R. MANOHAR AND J. W. STEPHENSON, *Single cell high order methods for the Helmholtz equation*, J. Comput. Phys., to appear.

[6] R. F. BOISVERT, *Families of high order accurate discretizations of some elliptic problems*, this Journal, 2 (1981), pp. 268–284.

[7] Y. KWON, R. MANOHAR AND J. W. STEPHENSON, *Single cell fourth order methods for the biharmonic equations*, Congressus Numerantium, 34 (1982), pp. 475–482.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March, 1984

A FINITE ELEMENT METHOD FOR GAS CENTRIFUGE FLOW PROBLEMS*

M. D. GUNZBURGER†, H. G. WOOD‡ AND J. A. JORDAN§

Abstract. A finite element method for the approximate solution of the flow in rapidly rotating gas centrifuges is presented. The model equations to be discretized are derived using, in particular, the Onsager pancake approximation. Various analytic and computational features of the finite element method are discussed and a series of illustrative numerical examples are given.

Key words. finite element method, gas centrifuges, Onsager equation

1. Introduction.

Worldwide, gas centrifugation is one of the most popular means by which to enrich uranium in its fissionable isotope. Naturally, the nature of the fluid flow within the centrifuge largely determines the efficiency with which the centrifuge separates the different isotopes. In the past, this flow has been calculated by a semi-numerical technique based on solutions by eigenfunction expansion methods of the Onsager "pancake" model equations [11], [12]. The need for a more flexible means of calculating the flow field results in the need for developing a fully numerical solution technique. In this paper, we describe a finite element method for solving the Onsager pancake equations in the case where the flow may be driven by sources of mass, momenta or energy, or by boundary phenomena such as temperature gradients or mass flows introduced at boundaries. A simpler version of this finite element method, which applies to flows without sources, has been presented in [6].

The plan of this paper is as follows. In § 2, we describe the Onsager pancake model, concentrating on the development of the model to render it suitable for discretization by a Galerkin finite element method. The latter is described in §§ 3 and 4. Numerical results are presented in § 5.

Before proceeding, we point out that the pancake model is a linear steady state approximation of the compressible viscous flow inside a gas centrifuge. Another numerical technique for approximating this flow field in the presence of sources is given in [2]. Their technique retains some of the nonlinear character of the governing equations and uses a time-dependent algorithm for marching to a steady state. A survey of source free models for the flow in gas centrifuges is found in [9], where other numerical techniques for such problems may also be found.

2. The Onsager pancake model with the Carrier–Maslen boundary conditions.

The derivation of the Onsager pancake equations and the Carrier–Maslen Ekman layer boundary conditions differ in some of their details from that found in [11]. The two derivations yield equivalent problems and the formal differences in the present derivation are motivated by the desire to make the resulting problem more immediately amenable to discretization by a Galerkin finite element method.

Let (r, θ, z) denote cylindrical coordinates with the origin located at the bottom of the centrifuge, which we take to be a right circular cylinder, and on the axis of rotation. The z-axis coincides with the axis of rotation while the r-coordinate measures

* Received by the editors May 11, 1982. This research was prepared for the U.S. Department of Energy under contracts W-7405 eng 26 and DE-AC05-760R01779.

† Department of Mathematics, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213 and Department of Mathematics, University of Tennessee, Knoxville, Tennessee 37996.

‡ Department of Mechanical and Aerospace Engineering, University of Virginia, Charlottesville, Virginia 22901.

§ Union Carbide Corporation, Nuclear Division, Oak Ridge, Tennessee 37830.

the perpendicular distance from this axis. If the fluid is isothermal and rotates as a solid body, then the velocity components in the (r, θ, z) direction are respectively given by [11]

$$U=0, \quad V=\Omega r, \quad W=0,$$

and the pressure distribution is given by

$$P=p_w \exp\left\{-A^2\left(1-\frac{r^2}{a^2}\right)\right\}$$

where Ω is the angular velocity of the cylinder, p_w is the pressure on the wall of the cylinder, a is the radius of the cylinder, $A=a\Omega/(2RT_0)^{1/2}$, R is the specific gas constant, and T_0 is the uniform gas temperature.

We assume that the flow in the centrifuge is a small perturbation from the above isothermal solid body rotation. This enables us to linearize the equations of viscous compressible flow. Furthermore, we assume that the rotational speed is high enough, i.e., $A^2 \gg 1$, so that most of the fluid is found very near the wall of the cylinder, i.e., $r=a$. This enables us to set $r=a$ whenever r appears algebraically in the governing equations. Third, we assume that away from the Ekman layers adjacent to the top and bottom of the cylinder that the axial diffusion terms are negligible. Finally, we assume that the flow is axially symmetric.

Using these four assumptions, the equations of viscous compressible flow reduce to (see [11] for details)

$$e^{-x}w_y-2A^2(e^{-x}u)_x=\mathcal{M}, \tag{2.1}$$

$$\phi=(e^xp)_x+e^x\mathcal{U} \tag{2.2}$$

$$\phi_{xx}=-\frac{\operatorname{Re} S}{A^4}e^{-x}u-(\mathcal{T}-2\mathcal{V}) \tag{2.3}$$

$$p_y=\frac{8A^6}{\operatorname{Re}}w_{xx}+\mathcal{W} \tag{2.4}$$

$$-4A^4H_{xx}-H_{yy}=4A^4[\mathcal{T}+2(S-1)\mathcal{V}] \tag{2.5}$$

where (u, ω, w) are the perturbation velocities in the (r, θ, z) direction, respectively, p is the pressure, $\phi=\theta-2\omega$ and $H=\theta+2(S-1)\omega$ where θ is the temperature, $y=z/a$ and $x=A^2(1-r^2/a^2)$. $\mathcal{M}$, $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$ and $\mathcal{T}$ are nondimensional sources of mass, the three momentum components, and energy, respectively. In (2.1)–(2.5) all variables have been nondimensionalized, using Ωa for the velocity components, p_w for the pressure, T_0 for the temperature and the wall density, ρ_w, for the density. The constants appearing in (2.1)–(2.5) are the Reynolds number $\operatorname{Re}=\rho_w\Omega a^2/\mu$ and $S=1+\operatorname{Pr}A^2(\gamma-1)/2\gamma$ where $\operatorname{Pr}=c_p\mu/k$ is the Prandtl number and μ and c_p are the viscosity and specific heat at constant pressure, respectively. We note that x measures distances from the wall of the cylinder in "scale heights", i.e., e-foldings, of the ambient density $\rho=\rho_w \exp(-x)$.

We now introduce a function ψ satisfying

$$e^{-x}u=-\psi_y+\frac{1}{2A^2}\int_x^{x_T}\mathcal{M}(\xi, y)\,d\xi \tag{2.6}$$

and

$$e^{-x}w=-2A^2\psi_x, \tag{2.7}$$

where x_T is chosen large enough to accurately simulate the flow as $x \to \infty$. Note that if $\mathcal{M} \equiv 0$, then ψ is the usual streamfunction, while if $\mathcal{M} \not\equiv 0$, we view ψ as being a "convenient" function. Eliminating the pressure between (2.2) and (2.4) yields that

$$\phi_y = \frac{8A^6}{\text{Re}}(e^x w_{xx})_x + (e^x \mathcal{W})_x + e^x \mathcal{U}_y. \tag{2.8}$$

Substituting (2.6) and (2.7) into (2.3) and (2.8) yields that

$$\phi_{xx} = \frac{\text{Re}\, S}{A^4}\psi_y - \frac{\text{Re}\, S}{2A^6}\int_x^{x_T} \mathcal{M}(\xi, y)\, d\xi - (\mathcal{T} - 2\mathcal{V}) \tag{2.9}$$

and

$$\phi_y = -\frac{16A^8}{\text{Re}}(e^x(e^x\psi_x)_{xx})_x + (e^x\mathcal{W})_x + e^x\mathcal{U}_y. \tag{2.10}$$

Eliminating ϕ between (2.9) and (2.10) yields that

$$\begin{aligned}(e^x(e^x\psi_{xx})_{xx})_{xxx} + B^2\psi_{yy} &= \frac{\text{Re}^2 S}{32A^{14}}\int_x^{x_T} \mathcal{M}_y(\xi, y)\, d\xi \\ &\quad + \frac{\text{Re}}{16A^8}[(e^x\mathcal{W})_{xxx} + (e^x\mathcal{U})_{yxx} + (\mathcal{T} - 2\mathcal{V})_y],\end{aligned} \tag{2.11}$$

where $B = \text{Re}\, S^{1/2}/4A^6$. We now introduce the "master potential" χ satisfying

$$\psi = -2A^2\chi_x \tag{2.12}$$

into (2.11) and integrate the result from x to x_T. Assuming that χ and all its derivatives are negligible as $x \to x_T$, we arrive at

$$(e^x(e^x\chi_{xx})_{xx})_{xx} + B^2\chi_{yy} = F(x, y) \quad \text{for } 0 < x < x_T \text{ and } 0 < y < y_T \tag{2.13}$$

where

$$\begin{aligned}F(x, y) \equiv &-\frac{\text{Re}}{32A^{10}}\left[(e^x\mathcal{W})_{xx} + (e^x\mathcal{U})_{xy} + \int_x^{x_T} (2\mathcal{V} - \mathcal{T})_y\, d\xi\right] \\ &+ \frac{B^2}{4A^4}\int_x^{x_T}\int_\xi^{x_T} \mathcal{M}_y(\xi', y)\, d\xi'\, d\xi,\end{aligned} \tag{2.14}$$

and where y_T is the location of the top of the cylinder. Equation (2.13) is the Onsager pancake equation and is equivalent in form to that used by [11].

We now examine the boundary conditions which the master potential χ should satisfy. First, at the "top of the atmosphere," we assume that $u = \omega_x = w_x = \theta_x = 0$, i.e., that the radial velocity and the normal derivative of the azimuthal and axial velocities and of the temperature vanish. Then, by (2.7) and (2.12), we have that

$$L_3\chi(x_T, y) = 0 \quad \text{for } 0 < y < y_T, \tag{2.15}$$

where

$$L_3\chi = (e^x\chi_{xx})_x. \tag{2.16}$$

Further, since $\phi = \theta - 2\omega$ we have that $\phi_x(x_T, y) = 0$. But (2.10), (2.12) and (2.13) combine into

$$\phi_{yx} = -\frac{32A^{10}B^2}{\text{Re}}\chi_{yy} + \frac{32A^{10}}{\text{Re}}F(x, y) + (e^x\mathcal{W})_{xx} + (e^x\mathcal{U}_y)_x.$$

Integrating with respect to y yields that

$$\phi_x(x,y)-\phi_x(x,0)=-\frac{32A^{10}B^2}{\mathrm{Re}}[\chi_y(x,y)-\chi_y(x,0)] \tag{2.17}$$
$$+\int_0^y\left\{\frac{32A^{10}}{\mathrm{Re}}F(x,\eta)+[e^x\mathcal{W}(x,\eta)]_{xx}+[e^x\mathcal{U}_y(x,\eta)]_x\right\}d\eta.$$

Evaluating at $x=x_T$ and assuming that all sources vanish identically in a neighborhood of $x=x_T$, yields that

$$\chi_y(x_T,y)=K_1$$

where K_1 is a constant. Integrating with respect to y yields that

$$\chi(x_T,y)=K_1y+K_2$$

where K_2 is also a constant. Without loss of generality, we may choose $K_1=K_2=0$. First, $K_2=0$ because the variables of physical interest only involve derivatives of the master potential χ. We may take $K_1=0$ since all variables of interest except ϕ_x involve an x derivative of χ and by (2.17), ϕ_x depends only on the difference $\chi_y(x,y)-\chi_y(x,0)$, which is again independent of K_1. Therefore, our second boundary condition at $x=x_T$ is that

$$\chi(x_T,y)=0 \quad \text{for } 0\leqq y\leqq y_T. \tag{2.18}$$

The condition $u=0$ at $x=x_T$ and (2.6) yields that

$$\psi_y(x_T,y)=0$$

or that

$$\psi(x_T,y)=K_3$$

where $K_3=$ constant. Setting the streamfunction to be zero at the top of the atmosphere then yields that $K_3=0$ and thus, through (2.12), that

$$\chi_x(x_T,y)=0 \quad \text{for } 0\leqq y\leqq y_T. \tag{2.19}$$

At the wall of the cylinder, i.e., $x=0$, we assume that the velocity components vanish and that the temperature gradient along the wall is prescribed. From (2.10) and (2.12) we have that

$$\phi_y(0,y)=\frac{32A^{10}}{\mathrm{Re}}L_5\chi(0,y)+(e^x\mathcal{W}_x)(0,y)+(e^x\mathcal{U}_y)(0,y) \tag{2.20}$$

where

$$L_5\chi=(e^x(e^x\chi_{xx})_{xx})_x.$$

Furthermore, since $\phi=\theta-2\omega$ and $\omega(0,y)=0$, we have that $\phi_y(0,y)=\theta_y(0,y)$. Assuming that the sources $\mathcal{W}$ and $\mathcal{U}$ vanish in a neighborhood of the wall $x=0$, we then have from (2.20) that

$$L_5\chi(0,y)=f(y)\equiv\frac{\mathrm{Re}}{32A^{10}}\frac{d\theta_w}{dy} \quad \text{for } 0<y<y_T \tag{2.21}$$

where $d\theta_w/dy$ is the prescribed wall temperature gradient. Now, since $w(0,y)=0$, (2.7) and (2.12) imply that

$$\chi_{xx}(0,y)=0 \quad \text{for } 0<y<y_T. \tag{2.22}$$

Finally, we have that $u(0, y)=0$, so that by (2.6) and (2.12),

$$\chi_{xy}(0, y)=-\frac{1}{4A^4}\int_0^{x_T}\mathcal{M}(\xi, y)\,d\xi. \tag{2.23}$$

Integrating with respect to y from 0 to y yields that

$$\chi_x(0, y)=\chi_x(0,0)-\frac{1}{4A^4}\int_0^{y}\int_0^{x_T}\mathcal{M}(\xi, \eta)\,d\xi\,d\eta. \tag{2.24}$$

If we integrate (2.23) from y to y_T, we then have that

$$\chi_x(0, y)=\chi_x(0, y_T)+\frac{1}{4A^4}\int_y^{y_T}\int_0^{x_T}\mathcal{M}(\xi, \eta)\,d\xi\,d\eta. \tag{2.25}$$

Combining (2.24) and (2.25) yields that

$$\frac{1}{4A^4}\int_0^{y_T}\int_0^{x_T}\mathcal{M}(\xi, \eta)\,d\xi\,d\eta=\chi_x(0,0)-\chi_x(0, y_T). \tag{2.26}$$

Since $\chi_{xx}=-\psi_x/2A^2=(e^{-x}w)/4A^4$ we have that

$$4A^4\chi_x(0,0)=-\int_0^{x_T}e^{-x}\bar{w}(x,0)\,dx+\chi_x(x_T, 0)$$

or, by (2.19)

$$4A^4\chi_x(0,0)=-\int_0^{x_T}e^{-x}\bar{w}(x,0)\,dx, \tag{2.27}$$

where $e^{-x}\bar{w}(x, 0)$ is the axial flow through the boundary $y=0$. Similarly

$$4A^4\chi_x(0, y_T)=-\int_0^{x_T}e^{-x}\bar{w}(x, y_T)\,dx. \tag{2.28}$$

Combining (2.26)–(2.28) yields that

$$\int_0^{y_T}\int_0^{x_T}\mathcal{M}(\xi, \eta)\,d\xi\,dy=\int_0^{x_T}e^{-x}[\bar{w}(x, y_T)-\bar{w}(x,0)]\,dx$$

which, since e^{-x} is the nondimensionalized ambient density, merely expresses the fact that the mass introduced into the flow field by sources equals the mass exiting from the flow through the boundary. Now, if we combine (2.24) and (2.27), we arrive at our last boundary condition at $x=0$, namely that

$$\chi_x(0, y)=G(y)\equiv-\frac{1}{4A^4}\int_0^{y}\int_0^{x_T}\mathcal{M}(\xi, \eta)\,d\xi\,d\eta-\frac{1}{4A^4}\int_0^{x_T}e^{-x}\bar{w}(x,0)\,dx \quad \text{for } 0<y<y_T. \tag{2.29}$$

To complete the specification of the problem, we need only determine appropriate boundary conditions at the top and bottom of the cylinder, i.e., $y=y_T$ and $y=0$, respectively. We recall that in deriving (2.13) that we have neglected all axial diffusion terms so that (2.13) does not adequately model the flow in the Ekman layers adjacent to the top and bottom of the cylinder. We would like to prescribe u, ω, w and θ at these locations, but due to the absence of axial diffusion in our model, we are unable to do so. However, Carrier and Maslen [1], [11] have developed through a series of consistent approximations, a relation among the flow variables which replaces the details of the flow within the Ekman layer. This relation holds outside the Ekman layer, but in a manner consistent with our linearization procedures, we may impose the Carrier–Maslen relation at a physical boundary, e.g., $y=0$.

At $y=0$, the Carrier–Maslen relation takes the form [11]

$$-4e^{x/2}S^{3/4}\,\mathrm{Re}^{1/2}\,[\psi(x,0)-\bar{\psi}(x,0)]=\phi(x,0)-\bar{\phi}(x,0)+2S^{1/2}\bar{u}(x,0)$$

where the overbars denote quantities specified at the wall $y=0$. Differentiating with respect to x, recalling that $\bar{\phi}=\bar{\theta}-2\bar{\omega}$ yields that

$$(2.30)\ -4S^{3/4}\,\mathrm{Re}^{1/2}\,[(e^{x/2}\psi(x,0))_x-(e^{x/2}\bar{\psi}(x,0))_x]=\phi_x(x,0)-\bar{\phi}_x(x,0)+2S^{1/2}\bar{u}_x(x,0).$$

Combining (2.9) and (2.12) and then integrating with respect to x yields that

$$\phi_x(x,y)-\phi_x(x_T,y)=-\frac{2\,\mathrm{Re}\,S}{A^2}[\chi_y(x,y)-\chi_y(x_T,y)]$$
$$+\frac{\mathrm{Re}\,S}{2A^6}\int_x^{x_T}\int_\xi^{x_T}\mathcal{M}(\xi',y)\,d\xi'\,d\xi+\int_x^{x_T}(\mathcal{T}-2\mathcal{V})(\xi,y)\,d\xi.$$

Evaluating at $y=0$, assuming that the sources vanish there, and recalling that $\phi_x(x_T,y)=0$ and $\chi_y(x_T,y)=0$, yields that

$$\phi_x(x,0)=-\frac{2\,\mathrm{Re}\,S}{A^2}\chi_y(x,0). \tag{2.31}$$

Now, from (2.7),

$$\bar{\psi}(x,0)=\bar{\psi}(x_T,0)+\frac{1}{2A^2}\int_x^{x_T}e^{-\xi}\bar{w}(\xi,0)\,d\xi=\frac{1}{2A^2}\int_x^{x_T}e^{-\xi}\bar{w}(\xi,0)\,d\xi \tag{2.32}$$

since we have set $\bar{\psi}(x_T,y)=0$. Combining (2.12), (2.30)–(2.32) then yields that

$$B^2\chi_y(x,0)+2AB^{3/2}[e^{x/2}\chi_x(x,0)]_x=g_0(x)$$
$$\equiv-\frac{\mathrm{Re}}{32A^{10}}\bar{\phi}_x(x,0)+\frac{B}{4A^4}\bar{u}_x(x,0)$$
$$-\frac{B^{3/2}}{2A^3}\left[e^{x/2}\int_x^{x_T}e^{-\xi}\bar{w}(\xi,0)\,d\xi\right]_x\quad\text{for } 0<x<x_T \tag{2.33}$$

where $\bar{\phi}$, $\bar{u}$ and $\bar{w}$ are the prescribed combination of temperature and angular velocity, radial and axial velocity components, respectively, at the bottom of the cylinder.

Similarly, at the top of the cylinder, we can derive the condition

$$B^2\chi_y(x,y_T)-2AB^{3/2}[e^{x/2}\chi_x(x,y_T)]_x=g_1(x)$$
$$\equiv-\frac{\mathrm{Re}}{32A^{10}}\bar{\phi}_x(x,y_T)+\frac{B}{4A^4}\bar{u}_x(x,y_T)$$
$$+\frac{B^{3/2}}{2A^3}\left[e^{x/2}\int_x^{x_T}e^{-\xi}\bar{w}(\xi,y_T)\,d\xi\right]_x\quad\text{for } 0<x<x_T. \tag{2.34}$$

The statement of the problem governing the master potential χ is now complete. χ satisfies the differential equation (2.13) and the boundary conditions (2.16), (2.18), (2.19), (2.21), (2.22), (2.29), (2.33) and (2.34). Once χ is determined, we may recover ψ, u, and w by differentiating χ, i.e.,

$$\psi = -2A^2\chi_x, \quad w = 4A^4\chi_{xx}, \quad u = 2A^2\chi_{xy} + \frac{1}{2A^2}\int_x^{x_T} \mathcal{M}(\xi, y)\,d\xi.$$

ω and θ are given by

$$\omega = \frac{H-\phi}{2S} \quad \text{and} \quad \theta = \frac{H+(S-1)\phi}{S}.$$

Now ϕ is determined from χ through integrating (2.8) and (2.9). To determine H we must solve the differential equation (2.5) with the following boundary conditions. At $x=0$, $\omega=0$ so that $H(0, y) = \theta_w(y)$ where θ_w is the prescribed wall temperature. At $x = x_T$, $\omega_x = \theta_x = 0$ so that $H_x(x_T, y) = 0$. At $y=0$ and $y = y_T$, $\bar{\omega}$ and $\bar{\theta}$ are prescribed so that $H(x, 0) = \bar{\theta}(x, 0) + 2(S-1)\bar{\omega}(x, 0)$ and $H(x, y_T) = \bar{\theta}(x, y_T) + 2(S-1)\bar{\omega}(x, y_T)$, where again the overbar denotes a prescribed quantity. This completes the specification of the problem for $H(x, y)$. We will not consider this problem any further here except to note that it is a standard second order elliptic boundary value problem whose solution may be approximated by the use of standard finite element methodology [10].

3. Weak formulation. The finite element method described in § 4 will be a discretization of a Galerkin formulation of the governing problem for χ. The latter is derived in this section.

We begin by multiplying (2.13) by a smooth function $\tilde{\chi}$ and then integrating the result over the domain $D = \{0 < x < x_T, 0 < y < y_T\}$. Then we have that

$$\int\!\!\int_D \tilde{\chi}\{(e^x(e^x\chi_{xx})_{xx})_{xx} + B^2\chi_{yy} - F\}\,dx\,dy = 0.$$

Integrating the first term by parts three times with respect to x and the second term once with respect to y then yields that

$$\begin{aligned}(3.1)\qquad &\int\!\!\int_D \{(L_3\tilde{\chi})(L_3\chi) + B^2\tilde{\chi}_y\chi_y + F\tilde{\chi}\}\,dx\,dy - \int_0^{x_T} B^2\{\tilde{\chi}\chi_y\}\big|_{y=0}^{y=y_T}\,dx \\ &\quad - \int_0^{y_T} \{\tilde{\chi}L_5\chi - e^x\tilde{\chi}_x L_4\chi + e^x\tilde{\chi}_{xx}L_3\chi\}\big|_{x=0}^{x=x_T}\,dy = 0\end{aligned}$$

where the differential operators L_3 and L_5 are defined in § 2 and where L_4 is defined by

$$L_4\chi = (e^x\chi_{xx})_{xx}.$$

We now substitute the boundary conditions (2.16), (2.21), (2.23) and (2.34) into (3.1) and require that

$$\tilde{\chi}_x(0, y) = \tilde{\chi}_{xx}(0, y) = \tilde{\chi}(x_T, y) = \tilde{\chi}_x(x_T, y) = 0.$$

This results in

$$\begin{aligned}\int\!\!\int_D &\{(L_3\tilde{\chi})(L_3\chi)+B^2\tilde{\chi}_y\chi_y+F\tilde{\chi}\}\,dx\,dy\\ &-2AB^{3/2}\int_0^{x_T}\tilde{\chi}(x,y_T)[e^{x/2}\chi_x(x,y_T)]_x\,dx\\ &-2AB^{3/2}\int_0^{x_T}\tilde{\chi}(x,0)[e^{x/2}\chi_x(x,0)]_x\,dx\\ &+\int_0^{y_T}\tilde{\chi}(0,y)f(y)\,dy-\int_0^{x_T}\{g_1(x)\tilde{\chi}(x,y_T)-g_0(x)\tilde{\chi}(x,0)\}\,dx=0.\end{aligned}$$

Finally, integrating the second and third integrals by parts once with respect to x, and recalling that $\tilde{\chi}(x_T,y)=0$ and (2.29) yields that

$$\begin{aligned}\int\!\!\int_D &\{(L_3\tilde{\chi})(L_3\chi)+B^2\tilde{\chi}_y\chi_y\}\,dx\,dy+2AB^{3/2}\int_0^{x_T}e^{x/2}\tilde{\chi}_x(x,y_T)\chi_x(x,y_T)\,dx\\ &+2AB^{3/2}\int_0^{x_T}e^{x/2}\tilde{\chi}_x(x,0)\chi_x(x,0)\,dx\\ &=-\int\!\!\int_D F\tilde{\chi}\,dx\,dy+2AB^{3/2}\int_0^{x_T}\{\tilde{\chi}(x,y_T)g_1(x)-\tilde{\chi}(x,0)g_0(x)\}\,dx\\ &-\int_0^{y_T}\tilde{\chi}(0,y)f(y)\,dy-2AB^{3/2}\{\tilde{\chi}(0,y_T)G(y_T)+\tilde{\chi}(0,0)G(0)\}.\end{aligned}\tag{3.2}$$

We have not made use of the boundary conditions

$$\chi(x_T,y)=\chi_x(x_T,y)=\chi_{xx}(0,y)=0\quad\text{and}\quad\chi_x(0,y)=G(y)\quad\text{for } 0<y<y_T.$$

These must be explicitly imposed on the trial functions $\chi(x,y)$ as the conditions $\tilde{\chi}_x(0,y)=\tilde{\chi}_{xx}(0,y)=\tilde{\chi}(x_T,y)=\tilde{\chi}_x(x_T,y)=0$ were imposed on the test functions $\tilde{\chi}(x,y)$. These are the *essential boundary conditions* for the problem, while the boundary conditions $L_3\chi(x_T,y)=0$, $L_5\chi(0,y)=f(y)$ as well as (2.33) and (2.34) are *natural boundary conditions.*

We next define function classes in which we will seek our solution χ and in which we will choose our test functions $\tilde{\chi}$. We let $\mathcal{H}(D)$ denote the space of functions with three square integrable derivatives in the x direction and one square integrable derivative in the y direction; i.e., if $\chi\in\mathcal{H}(D)$, then

$$\int\!\!\int_D\left(\frac{\partial^j\chi}{\partial x^j}\right)^2dx\,dy<\infty\quad\text{for } j=0,1,2,3$$

and

$$\int\!\!\int_D\left(\frac{\partial^k\chi}{\partial y^k}\right)^2dx\,dy<\infty\quad\text{for } k=0,1.$$

The space $\mathcal{H}(D)$ is an anisotropic Sobolev space in the sense of Nikol'skii [7]. We also introduce the subspaces of $\mathcal{H}(D)$ defined by

$$\mathcal{H}_1(D)=\{\chi\in\mathcal{H}(D):\chi_{xx}(0,y)=\chi(x_T,y)=\chi_x(x_T,y)=0\}$$

and

$$\mathcal{H}_2(D)=\{\chi\in\mathcal{H}_1(D):\chi_x(0,y)=0\},$$

the bilinear form

$$\begin{aligned}\mathcal{B}(\chi,\tilde{\chi}) &= \int_D\!\!\int \{(L_3\chi)(L_3\tilde{\chi}) + B^2\chi_y\tilde{\chi}_y\}\,dx\,dy \\ &\quad + 2AB^{3/2}\int_0^{x_T} e^{x/2}\{\chi_x(x,y_T)\tilde{\chi}_x(x,y_T) + \chi_x(x,0)\tilde{\chi}_x(x,0)\}\,dx\end{aligned} \tag{3.3}$$

and the linear functional

$$\begin{aligned}\mathcal{F}(\tilde{\chi}) &= -\int_D\!\!\int F\tilde{\chi}\,dx\,dy - \int_0^{y_T}\tilde{\chi}(0,y)f(y)\,dy \\ &\quad -2AB^{3/2}\{\tilde{\chi}(0,y_T)G(y_T) + \tilde{\chi}(0,0)G(0)\} \\ &\quad +2AB^{3/2}\int_0^{x_T}\{\tilde{\chi}(x,y_T)g_1(x) - \tilde{\chi}(x,0)g_0(x)\}\,dx.\end{aligned} \tag{3.4}$$

Then our weak formulation is given by the following problem. We seek a function $\chi \in \mathring{\mathcal{H}}_1(D)$ such that $\chi_x(0,y) = G(y)$ and (3.2) holds for all $\tilde{\chi} \in \mathring{\mathcal{H}}_2(D)$, i.e., we seek a $\chi \in \mathring{\mathcal{H}}_1(D)$ such that

$$\mathcal{B}(\chi,\tilde{\chi}) = \mathcal{F}(\tilde{\chi}) \quad \forall \tilde{\chi} \in \mathring{\mathcal{H}}_2(D) \tag{3.5}$$

and

$$\chi_x(0,y) = G(y) \quad \text{for } 0 < y < y_T. \tag{3.6}$$

By retracing the steps that led from (2.13) to (3.2), it is easy to see that if χ satisfies (3.5) and (3.6) and is *smooth* enough, then χ satisfies (2.13) and all boundary conditions, i.e., smooth solutions of (3.5) and (3.6) are *classical solutions.* On the other hand, (3.5) and (3.6) admit solutions which are not smooth enough to be classical solutions of (2.13). These *weak solutions* are important because it allows us to find solutions, in a generalized sense, of (2.13) when the data $f(y)$, $g_0(x)$, $g_1(x)$, $G(y)$ and $F(x,y)$ are not smooth enough for classical solutions to exist.

Eastham [5] has shown, for a problem similar to the one considered here, that the bilinear form (3.3) is bounded and coercive on $\mathring{\mathcal{H}}_2(D)\times\mathring{\mathcal{H}}_2(D)$. Furthermore, for smooth enough F, G, f, g_0 and g_1, the linear functional (3.4) is bounded on $\mathring{\mathcal{H}}_2(D)$. These guarantee, by the Lax–Milgram theorem, the existence and uniqueness of the solution $\chi \in \mathring{\mathcal{H}}_1(D)$ of (3.5) and (3.6).

It is the weak formulation of the problem, given by (3.5) and (3.6) whose solution we will approximate by a finite element method.

4. The finite element algorithm. In order to define our approximate solution, we need to define sequences of finite-dimensional subspaces of $\mathcal{H}(D)$, parametrized by a parameter h such that h tends to zero. For us h is simply a measure of the grid size. The requirements that χ possess three L^2-derivatives in x and one in y leads us to require that $\mathcal{H}(D)$ must consist of $C^2(0,x_T)$ functions in x and $C^0(0,y_T)$ functions in y. We therefore will approximate with cubic splines in x and linear splines in y. Specifically, let

$$0 = y_0 < y_1 < \cdots < y_N = y_T \tag{4.1}$$

and

$$0 = x_0 < x_1 < \cdots < x_M = x_T \tag{4.2}$$

be partitions of the intervals $[0, y_T]$ and $[0, x_T]$, respectively. A point (x_i, y_j), $0 \leqq i \leqq M$,

$0 \leqq j \leqq N$ is called a *node*. We define the space of linear splines $L(0, y_T)$, with respect to the partition (4.1), by

$$L(0, y_T) = \{l(y) \in C^0(0, y_T);\ l(y) \text{ is a linear polynomial in each subinterval } [y_j, y_{j+1}],\ j = 0, \cdots, N-1\}.$$

The basis set for $L(0, y_T)$ consisting of functions with minimal support is the set $l_j(y)$ of hat functions depicted in Fig. 1a. Similarly, we define the space of cubic splines $S(0, x_T)$, with respect to the partition (4.2), by

$$S(0, x_T) = \{s(x) \in C^2(0, x_T);\ s(x) \text{ is a cubic polynomial in each subinterval } [x_i, x_{i+1}],\ i = 0, \cdots, M-1\}.$$

A basis set for $S(0, x_T)$ consisting of functions of as small support as possible is the set of B-spline functions $s_i(x)$ depicted in Fig. 1b. To properly define this set, fictitious

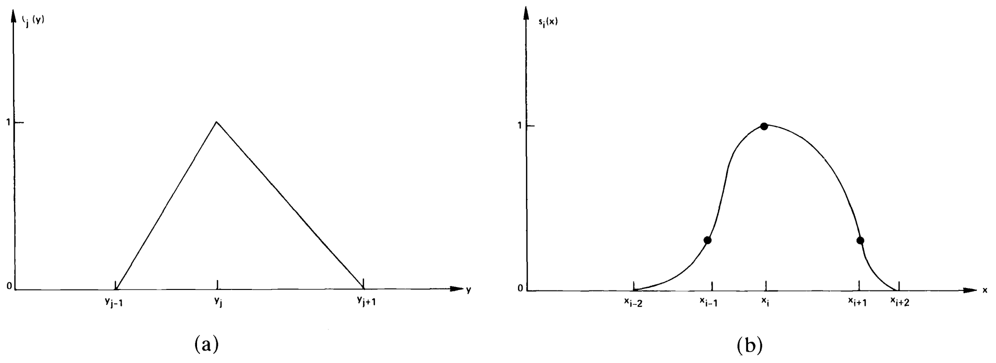


FIG. 1. *Basis functions for finite element spaces.* (a) *The hat function* $l_j(y) \in C^0(0, y_T)$. (b) *The B-spline* $s_i(x) \in C^2(0, x_T)$.

nodes should be added outside the interval $[0, x_T]$. For details of the construction of both the hat and B-spline basis functions, see [3], [6]. We note that dim $(L(0, y_T)) = N+1$ and dim $(S(0, x_T)) = M+3$.

The basic finite element space we will use to approximate (3.5) and (3.6) is the tensor product of the one-dimensional spaces $L(0, y_T)$ and $S(0, x_T)$. Specifically, we let

$$S^h = \left\{\chi^h(x, y);\ \chi^h(x, y) = \sum_{i=-1}^{M+1} \sum_{j=0}^{N} c_{ij} l_j(y) s_i(x)\right\}$$

where $l_j(y) \in L(0, y_T)$ is one of the hat functions, $s_i(x) \in S(0, x_T)$ is one of the B-spline functions, and c_{ij} are a set of coefficients. That S^h is a subspace of $\mathscr{H}(D)$ is shown in [5]. We then have that dim $(S^h) = (M+3)(N+1)$ and that the set

$$(4.3)\quad \chi_k^h(x, y) = \{l_j(y) s_i(x)\}, \qquad k = j(M+3) + (i+2) \quad \text{for } -1 \leqq i \leqq M+1,\ 0 \leqq j \leqq N$$

is a set of basis functions for S^h. Our parameter h is defined as

$$h = \max(h_1, h_2)$$

where

$$h_1 = \max_{0 \leqq i \leqq M-1} |x_{i+1} - x_i| \quad \text{and} \quad h_2 = \max_{0 \leqq j \leqq N-1} |y_{j+1} - y_j|.$$

We also define the subspaces

$$S_1^h = \{\chi^h \in S^h;\ \chi_{xx}^h(0, y) = \chi^h(x_T, y) = \chi_x^h(x_T, y) = 0\}$$

and

$$S_2^h = \{\chi^h \in S_1^h;\ \chi_x^h(0, y) = 0\}.$$

The basis functions (4.3) can be constructed so that the set $\{\chi_k^h\}$ for $k = j(M+3)+(i+2)$, $1 \leqq i \leqq M-1$ and $0 \leqq j \leqq N$, are a basis set for S_2^h. We have that $\dim(S_2^h) = (M-1)(N+1)$. (The space S_1^h will only be used when $G(y) \neq 0$, i.e., when one of the essential boundary conditions is inhomogeneous. We treat this case separately below.)

In the case of $G(y) = 0$, our approximate problem is given by: seek a $\chi^h \in S_2^h$ such that

$$\mathscr{B}(\chi^h, \tilde{\chi}^h) = \mathscr{F}(\tilde{\chi}^h) \quad \forall \tilde{\chi}^h \in S_2^h. \tag{4.4}$$

It suffices to choose $\tilde{\chi}^h = \chi_l^h$ for $l = j(M+3)+(i+2)$ for $1 \leqq i \leqq M-1$ and $0 \leqq j \leqq N$. Then (4.4) is equivalent to the matrix problem

$$A\mathbf{c} = \mathbf{b} \tag{4.5}$$

where

$$A_{rs} = \mathscr{B}(\chi_k^h, \chi_l^h), \qquad 1 \leqq r, s \leqq (M-1)(N+1) \tag{4.6}$$

where $k = j(M+3)+(i+2)$, $r = j(M-1)+i$, $l = j'(M+3)+(i'+2)$ and $s = j'(M-1)+i'$. Here i, j and i', j' are node counters, as are k and l, while r, s count only those nodes whose associated coefficient c_{ij} is not determined, by the essential boundary condition, to vanish. The $(M-1)(N+1)$ vector $\mathbf{c}$ has components $c_s = c_{i'j'}$, while the $(M-1)(N+1)$ vector $\mathbf{b}$ has components $b_r = \mathscr{F}(\chi_l^h)$. Since the bilinear form $\mathscr{B}(\chi, \tilde{\chi})$ is symmetric and coercive on $\mathring{\mathscr{H}}_2(D)$, the matrix A is symmetric and positive definite and thus the existence and uniqueness of the approximate solution is assured. In (4.3) we have used a node numbering system wherein we sweep in the x-direction first, then in the y-direction. Then the half bandwidth of the matrix A is given by $(M+2)$ while A has $(M-1)(N+1)$ rows and columns.

The case of $G(y) \neq 0$ is treated in an analogous manner. We note that $G(y) = 0$ unless we have mass sources and/or the net mass flow through the top or bottom of the cylinder is nonvanishing. When $G(y) \neq 0$, one of the essential boundary conditions is inhomogeneous. We first approximate $G(y)$ by its interpolant in $L(0, y_T)$, i.e., we let

$$G^h(y) = \sum_{j=0}^{N} G(y_j) l_j(y). \tag{4.7}$$

The approximate problem we solve is given by: seek a $\chi^h \in S_1^h$ such that $\chi_x^h(0, y) = G^h(y)$ and (4.4) hold. Now, the application of the former and the homogeneous boundary condition $\chi_{xx}^h(0, y) = 0$ enables us to determine the coefficients c_{ij} for $i = -1, 0$ and $0 \leqq j \leqq N$, in the expansion of χ^h in terms of the basis functions (4.3). In fact, our approximate problem is again equivalent to the matrix problem (4.5) with A given by (4.6) and the vector $\mathbf{c}$ defined as before. However, now the right-hand side vector $\mathbf{b}$ has components

$$b_r = \mathscr{F}(\chi_l^h) - \mathscr{B}(\hat{\chi}, \chi_l^h) \tag{4.8}$$

where

$$\hat{\chi}(x, y) = \sum_{j=0}^{N} \{c_{-1j} s_{-1}(x) + c_{0j} s_0(x)\} l_j(y) \tag{4.9}$$

and c_{-1j}, c_{0j} are the solution of

$$\hat{\chi}_x(0, y_j) = G^h(y_j) = G(y_j) = c_{-1j}s'_{-1}(0) + c_{0j}s'_0(0), \tag{4.10}$$

$$\hat{\chi}_{xx}(0, y_j) = 0 = c_{-1j}s''_{-1}(0) + c_{0j}s''_0(0) \tag{4.11}$$

for $0 \leqq j \leqq N$. Clearly, if $G(y) = 0$, then $c_{0j} = c_{-1j} = 0$ and the remaining problem for **c** is identical to the one prescribed before.

Using this finite element method, the expected rate of convergence for smooth solutions χ and its derivatives, which are related to physical quantities, is given by [6]

$$\|\chi - \chi^h\| = O(h^2), \tag{4.12}$$

$$\|\psi - (-2A^2\chi_x^h)\| = 2A^2\|\chi_x - \chi_x^h\| = O(h^2), \tag{4.13}$$

$$\|(e^{-x}w) - 4A^4\chi_{xx}^h\| = 4A^4\|\chi_{xx} - \chi_{xx}^h\| = O(h^2), \tag{4.14}$$

$$\|\chi_{xy} - \chi_{xy}^h\| = O(h) \tag{4.15}$$

where $2A^2\chi_{xy} = e^{-x}u$ when $\mathcal{M} = 0$, and is otherwise related to u by (2.6). In (4.12)–(4.15),

$$\|\chi\| = \left[\iint_D \chi^2\, dx\, dy\right]^{1/2}.$$

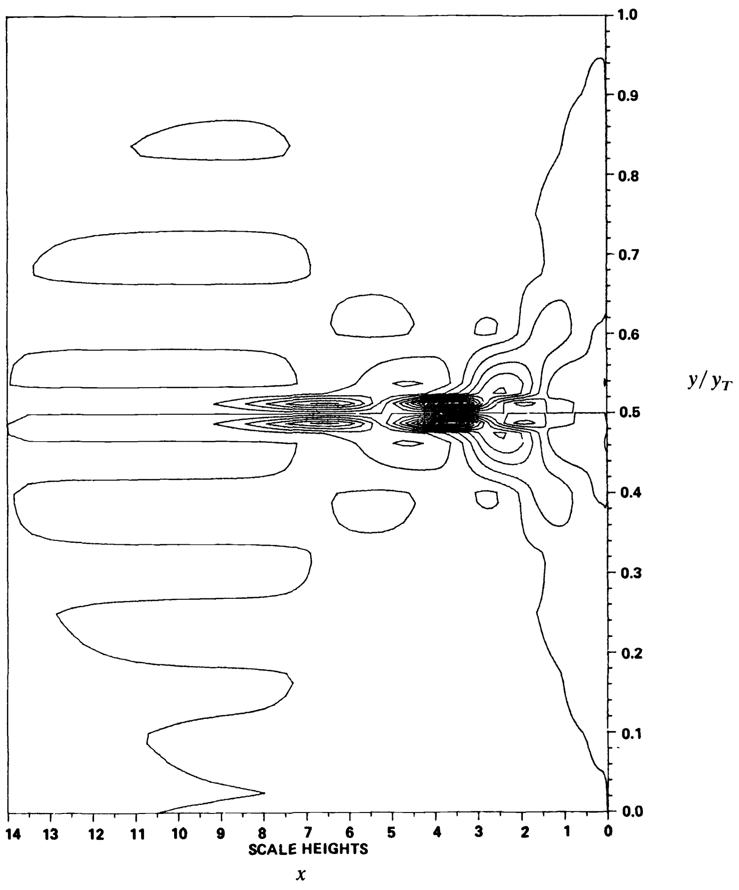


FIG. 2. *Lines of constant ψ^h for a source of radial momentum located at $x = 8$ and $y/y_T = 0.5$.*

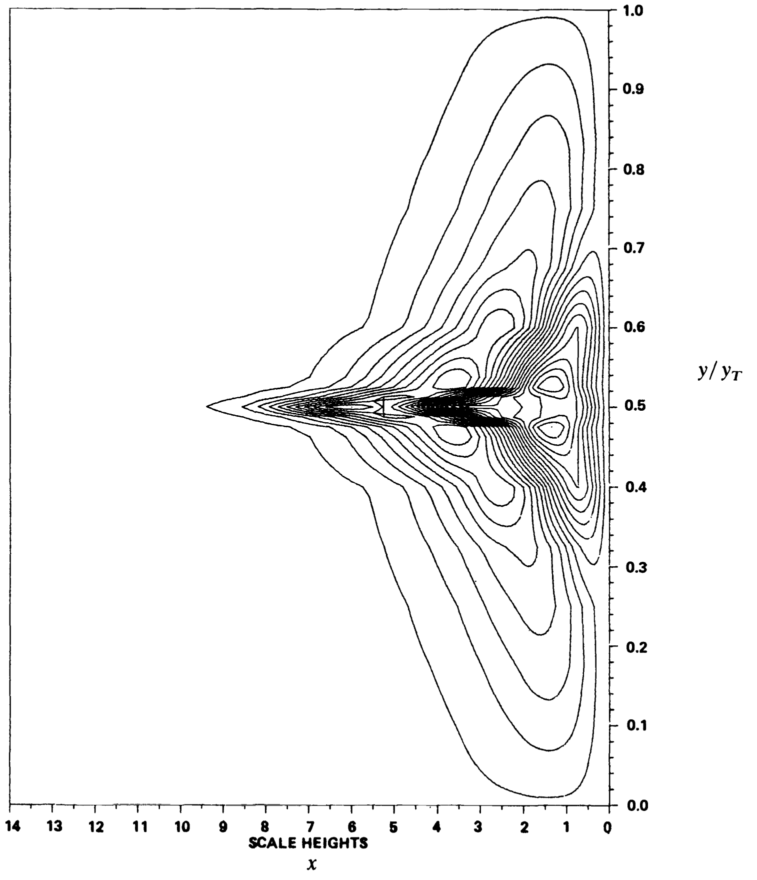


FIG. 3. *Lines of constant ψ^h for a source of axial momentum located at $x = 8$ and $y/y_T = 0.5$.*

Sample computations illustrating these rates for problems whose right-hand side F and inhomogeneities in the boundary conditions f, g_0, g_i and G are contrived so that a smooth exact solution χ is known were reported on in [6]. In § 5 we report the results of other computations, using a variety of data which induce secondary flows in a gas centrifuge.

5. Examples. In this section we wish to present the results of numerical computations for different choices of data F, G, f, g_0 and g_1. Since our problem for χ is a linear one, the principle of superposition may be applied, and thus complicated flows, driven by a variety of sources and boundary conditions, may be constructed by superimposing simple flows. Here we consider a variety of such simple flows, each driven by a different mechanism. For all cases, we present, in Figs. 2–6, level lines for $\psi^h(x, y) = -2A^2\chi_x^h(x, y)$. When $\mathcal{M} \equiv 0$, these level lines are identically streamlines. If $\mathcal{M}(x, y) \not\equiv 0$ these lines are essentially streamlines only for those values of y such that $\mathcal{M}(x, y)$ is negligible.

Two simple cases for which $F \equiv 0$, i.e., no sources, and $G = 0$ were reported on in [6]. The first case was $F = G = g_0 = g_1 = 0$ and f corresponding to a linear wall temperature distribution. Indeed, in this case f is a constant. The second case was

$F = G = f = 0$ and g_0 and g_1 corresponding to the introduction and removal of mass through the end caps of the centrifuge at $y = 0$ and $y = y_T$. The net mass flow through each end cap was zero, which implied that $G = 0$. In this work we confine ourselves to cases in which F and/or $G \neq 0$.

Figure 2 displays plots of the level lines of $\psi^h(x, y) = -2A^2\chi_x^h(x, y)$ for flows driven by a radial momentum source placed at $x_S = 8$, $y_S = y_T/2$ for a centrifuge having the following parameters:

$$\begin{aligned} &a = 9.145 \text{ cm}, && T_0 = 300°\text{K}, \\ &y_T = 36.66, && \Omega a = 700 \text{ m/s}, \\ &p_w = 13.3 \text{ kPa}. && \end{aligned} \tag{5.1}$$

These parameters were obtained from centrifuges described in [4], [8] and were the same as those in calculations reported in [6], [11], [12]. The value of $\chi_T = 14$ is large enough so that "top of the atmosphere" effects are essentially negligible. The momentum source $\mathcal{U}$ was chosen to be the Gaussian

$$S_0 \exp\{-\alpha[(x - x_S)^2 + (y - y_S)^2]\}. \tag{5.2}$$

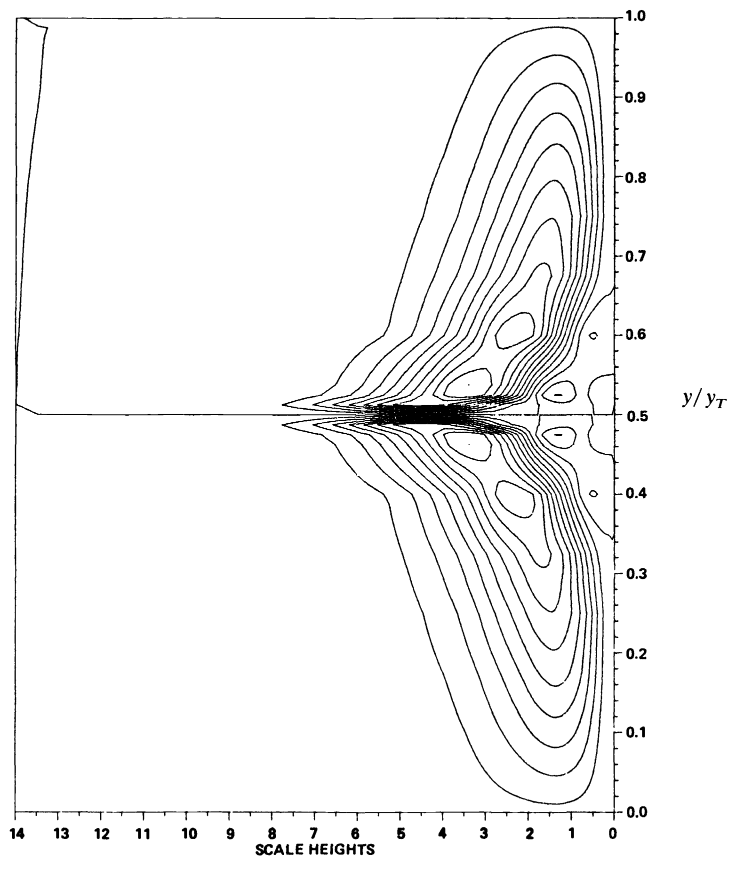


FIG. 4. *Lines of constant ψ^h for a source of azimuthal momentum located at $x = 8$ and $y/y_T = 0.5$.*

A value of α was chosen so that $\mathcal{U}(x, y) = 10^{-6} S_0$ at $(x - x_S)^2 + (y - y_S)^2 = 1$. Then S_0 is chosen so that

$$\iiint_V \mathcal{U}\, dV = 1.$$

In a similar manner, Figs. 3–4 display level lines of $\psi^h(x, y)$ for unit sources of axial momentum $\mathcal{W}$ and azimuthal momentum $\mathcal{V}$. Since in F, $\mathcal{T}$ and $\mathcal{V}$ appear in the combination $\mathcal{T} - 2\mathcal{V}$ only, a unit energy source would yield contours exactly like those of a unit source of axial momentum except, of course, with twice the amplitude and the opposite flow direction. For all these cases, $f = G = g_0 = g_1 = 0$ and $F(x, y)$ is easily computed from the source shape (5.2).

Figures 5 and 6 display level lines of ψ^h for two cases for which $G(y) \neq 0$. In the first of these, we take $\mathcal{M}$ to be the Gaussian spike (5.2), and all other sources to vanish. Half the introduced mass is removed, at each end of the machine, through orifices located at $5.5 \leqq x \leqq 6.5$. It is assumed that the outflow through these orifices is uniform and is in the axial direction. With this information, we may calculate F, G, g_0 and g_1. Here $f = 0$. As an example of what these functions look like in a typical

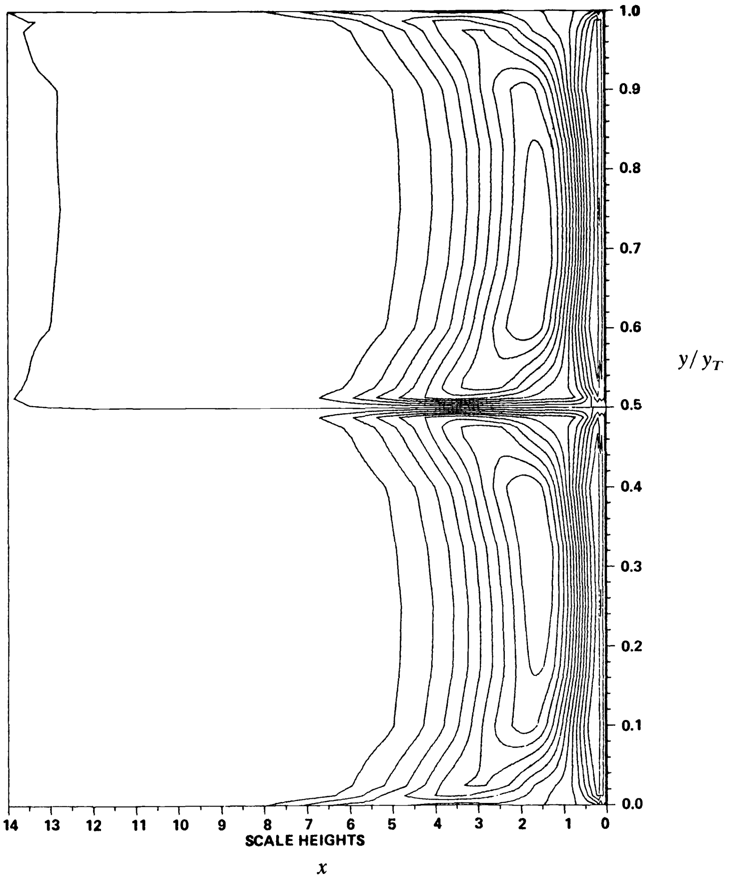


FIG. 5. *Lines of constant ψ^h for a source of mass located at $x = 8$ and $y/y_T = 0.5$. Half the mass is removed at each of the top and bottom ends of the centrifuge.*

case, we list them for this particular example:

$$F(x,y)=\frac{B^2\sqrt{\alpha\pi}S_0}{4A^4}(y-y_S)\exp[-\alpha(y-y_S)^2]$$
$$\cdot\{(x-x_S)E(x_T-x_S)$$
$$-(x-x_S)E(x-x_S)+\frac{1}{\sqrt{\alpha\pi}}[\exp[-\alpha(x_T-x_S)^2]-\exp[-\alpha(x-x_S)^2]]\},$$

$$G(y)=\frac{1}{8A^4}-\frac{S_0\pi}{16A^4\alpha}\{E(x_T-x_S)+E(x_S)\}\{E(y-y_S)+E(y_S)\},$$

$$g_0(x)=g_1(x)=-\frac{B^{3/2}}{8A^3}e^{x/2}\begin{cases}1, & 0\leqq x\leqq 5.5,\\ (x-8.5), & 5.5<x<6.5,\\ 0, & 6.5<x<x_T\end{cases}$$

where $E(x)=\operatorname{erf}(\sqrt{\alpha x})$. For Fig. 6, we place a Gaussian mass source at $(8, 3y_T/4)$ and a mass sink of equal strength at $(8, y_T/4)$ so that

$$\mathcal{M}(x,y)=S_0[\exp\{-\alpha[(x-8)^2+(y-3y_T/4)^2]\}-\exp\{-\alpha[(x-8)^2+(y-y_T/4)^2]\}].$$

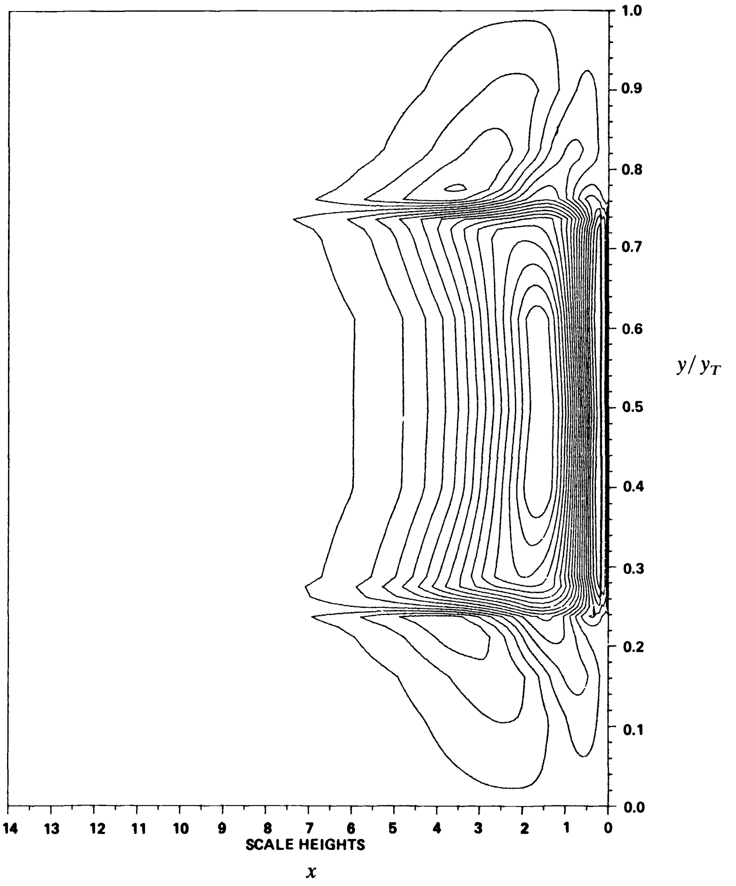


FIG. 6. *Lines of constant* ψ^h *for a source of mass located at* $x=8$ *and* $y/y_T=0.75$ *and a sink of mass, of equal strength, located at* $x=8$ *and* $y/y_T=0.25$.

In this case $f = g_0 = g_1 = 0$ but F and $G \neq 0$. We again caution that for those y values for which $\mathcal{M}(x, y)$ is not negligible, the level lines of Figs. 5 and 6 are *not* streamlines. We note that $|\mathcal{M}(x, y)| < 10^{-6}\alpha/\pi$ for $|y/y_T - 1/2| < 1/y_T$ in Fig. 5 and for $|y/y_T - 1/4| < 1/y_T$ and $|y/y_T - 3/4| < 1/y_T$ in Fig. 6.

Whenever comparisons are possible, the results of Figs. 2–6 are in close agreement with those calculated by eigenfunction expansion methods [11], [12]. The time and storage requirements for the finite element code were also similar to those required for the eigenfunction expansion method. All calculations were done using variable grids. These grids were chosen so that points were packed in regions where the data has steep gradients, e.g., near the center of a source (x_S, y_S), or where the solution is expected to have steep gradients, e.g., in the boundary layer adjacent to the wall $x = 0$. The particular calculations reported used $M = 40$ and $N = 21$.

REFERENCES

[1] G. F. Carrier and S. H. Maslen, *Flow phenomena in rapidly rotating systems*, USAEC Rep. TID-18065, 1962.

[2] L. D. Cloutman and R. A. Gentry, *Numerical simulation of the countercurrent flow in a gas centrifuge*, Los Alamos Scientific Laboratory Rep. LA-UR-81-1821, Los Alamos, NM, 1981.

[3] C. De Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[4] J. Durivault and P. Louvet, *Etude théorique de l'écoulement dans une centrifugeuse à contre courant thermique*, Centre d'Etudes Nucléaires de Saclary, Rapport CEA-R-4714, 1976.

[5] J. F. Eastham, *The Finite Element Method in Anisotropic Sobolev Spaces*. Ph.D. thesis, Univ. Tennessee, Knoxville, 1981.

[6] M. D. Gunzburger and H. G. Wood, III, *A finite element method for the Onsager pancake equation*, Comp. Meth. Appl. Mech. Eng., to appear.

[7] A. Kufner, O. John and S. Fucik, *Function Spaces*, Nordhoff, Leyden, 1977.

[8] W. G. May, *Separation parameters of gas centrifuges*, AIChE Symposium Series, 169, Vol. 73, AIChE, New York, 1977.

[9] Soubbaramayer, *Centrifugation*, in Uranium Enrichment, Topics in Applied Physics, 35, S. Villani, ed., Springer-Verlag, Berlin, 1979.

[10] G. Strang and G. Fix, *An Anaysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[11] H. G. Wood, III and J. B. Morton, *Onsager's pançake approximation for the fluid dynamics of a gas centrifuge*, J. Fluid Mech., 101 (1980), pp. 1–31.

[12] H. G. Wood, III and G. Sanders, *Rotating compressible flows with internal sources and sinks*, J. Fluid Mech., to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March, 1984

NUMERICALLY STABLE SOLUTION OF DENSE SYSTEMS OF LINEAR EQUATIONS USING MESH-CONNECTED PROCESSORS*

A. BOJANCZYK†, R. P. BRENT‡ AND H. T. KUNG§

Abstract. We propose a multiprocessor structure for solving a dense system of n linear equations. The solution is obtained in two stages. First, the matrix of coefficients is reduced to upper triangular form via Givens rotations. Second, a back substitution process is applied to the triangular system. A two-dimensional array of $\theta(n^2)$ processors is employed to implement the first step, and (using a previously known scheme) a one-dimensional array of $\theta(n)$ processors is employed to implement the second step. These processor arrays allow both stages to be carried out in time $O(n)$, and they are well suited for VLSI implementation as identical processors with a simple and regular interconnection pattern are required.

Key words. Givens method, least squares, linear systems, numerical stability, orthogonal factorization, parallel algorithms, *QR* method, special-purpose hardware, systolic arrays, VLSI

1. Introduction.

Recently, several algorithms have been proposed for solving a system of linear equations on a parallel computer. The algorithm of Csanky [1] solves a dense system of size n in $\theta(\log^2 n)$ time steps with $\theta(n^4)$ processors. This is the best known upper bound on the time complexity of the problem. Since $\Omega(\log n)$ is a lower bound we have a gap of order $\log n$. Unfortunately, Csanky's algorithm is numerically unstable [14] and uses too many processors to be useful in practice. Gaussian elimination without pivoting can trivially be carried out in parallel in $\theta(n)$ steps using n^2 processors [5]. If the matrix of the system is not special (e.g., diagonally dominant or symmetric positive definite) then pivoting is generally necessary to guarantee numerical stability. With pivoting we need $\theta(n \log n)$ steps and n^2 processors. To avoid the pivoting problem, Sameh and Kuck [12] (and also Kowalik et al. [7], [8], [11]) proposed the use of Givens transformations to triangularize the matrix of coefficients. The orthogonal factorization requires $\theta(n)$ steps with $\theta(n^2)$ processors. The factorized linear system can then be solved in $\theta(n)$ steps using $\theta(n)$ processors. Hence, the algorithm for solving dense system of linear equations requires $\theta(n)$ time steps and $\theta(n^2)$ processors, yielding a speed-up of order n^2 over the usual sequential algorithms, which require $\theta(n^2)$ time steps.

However, traditional operation counts do not adequately measure the cost of a parallel computation. There are many other factors which must be considered when evaluating the performance of parallel algorithms. One of the most important is the cost of data transmission. In many papers dealing with parallel algorithms, there is an explicit or implicit assumption that the time required to obtain a single datum is negligible. This is not true in practice as every data transfer between processors takes time. Interprocessor communication must be realized by a network that interconnects the processors. Any algorithm can be supported by different networks but, in general, the number of data transfers depends on the topology of the network. With different networks one can have different execution times for the same algorithm. Thus, one should decide what kind of network is to be employed and only then proceed to evaluate the performance of the algorithm. Bearing this in mind, Kant and Kimura

* Received by the editors June 10, 1981, and in revised form July 15, 1982.

† Institute for Informatics, University of Warsaw, PKiN, p. 850, 00-901 Warsaw, Poland.
‡ Department of Computer Science, Australian National University, Canberra, ACT 2600, Australia.
§ Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213. The research of this author was supported in part by the National Science Foundation under grant MCS78-236-76.

[6] showed that the solution of a dense system of linear equations can be obtained in $\theta(n)$ steps using n^2 mesh connected processors, but their algorithm requires that the matrix of the system be "strongly nonsingular". The assumption of strong nonsingularity is a severe one as it excludes many interesting nonsingular matrices (e.g. the identity matrix) and it appears to be no easier to verify the assumption than to solve the corresponding linear system. Thus, the result of Kant and Kimura [6] is mainly of theoretical interest.

Kung and Leiserson [10] introduced a new model of parallel computation. The model takes into account such issues as cost of I/O, control and data transfers. A point of their work is that one should fit a network to an algorithm in order to obtain good overall performance. Using a simple and regular network, called a "systolic array", of $\theta(n^2)$ hexagonally connected processors, Kung and Leiserson [10] improved the result of Kant and Kimura [6] by requiring only that the linear system be solvable by Gaussian elimination without pivoting. For example, the matrix of the linear system could be symmetric positive definite or irreducible and diagonally dominant. See Kung [9] for a general discussion of systolic architectures for various special-purpose computational devices.

Combining the ideas of Sameh and Kuck [12] and Kung and Leiserson [10], we introduce a systolic array of $\theta(n^2)$ processors which is capable of transforming any nonsingular matrix to triangular form in $\theta(n)$ units of time in a numerically stable manner. The resulting triangular system can be solved in $\theta(n)$ steps on an array of n linearly connected processors. Both processor arrays enjoy regular geometries, and all processors are similar. As a consequence, cost-effective special purpose hardware devices based on our scheme could conceivably be built using VLSI technology. For many applications each processor needs to perform floating-point computations on words of at least 32 bits. To achieve a throughput of one floating-point operation every microsecond, present technology would allow only one (or a small number) of processors per chip, but advances in technology should soon make it possible to put many processors on a chip.

2. Givens rotations. Our algorithm is based on the orthogonal factorization of a real nonsingular n by n matrix $A = (a_{ij})$,

$$QA = R,$$

where Q is an orthogonal matrix formed as the product of plane rotations, and R is upper triangular.

A plane rotation is defined by a matrix

$$\begin{array}{rcl} & \begin{array}{c}\text{col. } i\\ \downarrow\end{array} & \\ P_{i+1,j} = & \begin{bmatrix} 1 & & \vdots & & & \\ & \ddots & \vdots & & & \\ & & c_i & s_i & \cdots & \\ & & -s_i & c_i & & \\ & & & & \ddots & \\ & & & & & 1\end{bmatrix} & \leftarrow \text{row } i. \end{array}$$

The matrix $P_{i+1,j}$ applied on the left rotates rows i and $(i+1)$ of A so as to annihilate the off-diagonal element $a_{i+1,j}$. The parameters of $P_{i+j,j}$ are defined (except in degenerate cases, which are dealt with in § 4) by

$$\bar{a}_{i,j} = (a_{i,j}^2 + a_{i+1,j}^2)^{1/2},$$

$$c_i = a_{i,j}/\bar{a}_{i,j},$$

$$s_i = a_{i+1,j}/\bar{a}_{i,j}.$$

Rows i and $i+1$ of the product $\bar{A} = P_{i+1,j}A$ are given by

$$\bar{a}_{i,p} = c_i a_{i,p} + s_i a_{i+1,p},$$

$$\bar{a}_{i+1,j} = 0,$$

$$\bar{a}_{i+1,p} = -s_i a_{i,p} + c_i a_{i+1,p} \quad \text{for } p \neq j.$$

The orthogonal matrix Q is formed as the product of plane rotations $P_{i,j}$ such that the elements of A below the main diagonal are annihilated.

3. Parallel Givens rotations. As some of the rotations $P_{i,j}$ are independent it is possible to apply more than one at a time. There are many possibilities. We propose a scheme that requires $N = 3n - 5$ sweeps. Each sweep Q_k, $k = 1, 2, \cdots, N$, is a direct sum of plane rotations $P_{i,j}$, where $(i, j) \in L_k$ and sets of indices L_k, $k = 1, 2, \cdots, N$, are defined by

$$(3.1) \qquad (i, j) \in L_k \quad \text{iff} \quad 3(j-1) + n - (i-1) = k, \qquad 1 \leqq j < i \leqq n.$$

Note that Q_k is a product of commuting orthogonal matrices $P_{i,j}$. The orthogonal matrix Q is the product of sweeps Q_k,

$$Q = Q_N Q_{N-1} \cdots Q_1.$$

From (3.1) it follows that if $(i, j) \in L_k$ then $(i+3, j+1)$ and $(i-3, j-1)$ also belong to L_k provided they are in $\{(i, j) | 1 \leqq j < i \leqq n\}$. The rule of thumb is as follows. Starting from any element $a_{i,j}$, $i > j$, and moving like a "long" chess knight on the chessboard, one square left and three squares up or one square right and three squares down within the lower triangular part of the matrix A, we reach all elements which are annihilated at the same time as the element $a_{i,j}$. This is illustrated for $n = 8$ in Fig. 1, where all elements annihilated in the kth sweep are denoted by $\boxed{k}$.

x							
7	x						
6	9	x					
5	8	11	x				
4	7	10	13	x			
3	6	9	12	15	x		
2	5	8	11	14	17	x	
1	4	7	10	13	16	19	x

FIG 1. *Ordering of rotations* ($n = 8$).

4. The basic processing element. Two kinds of operations are required for the transformations $P_{i,j}$: determination of the rotation parameters c and s, and application of the rotation, which is equivalent to

$$\begin{bmatrix} x \\ y \end{bmatrix} := \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{4.1}$$

The operation (4.1) will be referred to as a rotation step.

Thus, we need a processor which is able to determine rotation parameters and execute rotation steps. In addition to its computing capabilities the processor should have four connections (two inputs and two outputs) and four internal registers (R_x, R_y, R_c and R_s). To perform the first operation, the processor shifts data x and y on its input lines (denoted by X and Y) into registers R_x and R_y (see Fig. 2). Then it

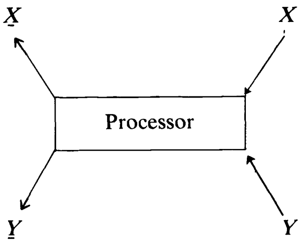


FIG. 2. *The processing element.*

computes parameters c and s by the following algorithm:

```
if $R_x = 0$ then
    begin
        $c := 0$;
        $s := 1$
    end
else if abs $(R_x) >$ abs $(R_y)$ then
    begin
        $\underline{x} :=$ abs $(R_x) \times$ sqrt $(1 + (R_y/R_x)^2)$;
        $c := R_x/\underline{x}$;
        $s := R_y/\underline{x}$
    end
else
    begin
        $\underline{x} :=$ abs $(R_y) \times$ sqrt $(1 + (R_x/R_y)^2)$;
        $c := R_x/\underline{x}$;
        $s := R_y/\underline{x}$
    end;
```

The computed values c and s are stored in registers R_c and R_s, and the new value $\underline{x}$ is made available as output on the output line $\underline{X}$. (The new value $\underline{y}$ on the output line $\underline{Y}$ is not calculated since c and s are chosen in such a way that $\underline{y}$ is known to be zero.) The processor determines the parameters c and s only once, so the contents of the R_c and R_s registers are not subsequently changed. Every subsequent operation performed by the processor is a rotation step. More precisely, the processor shifts data on its input lines X and Y into registers R_x and R_y, then executes the rotation step, i.e.,

$$\underset{\sim}{x} := R_x R_c + R_y R_s, \qquad \underset{\sim}{y} := -R_x R_s + R_y R_c$$

and makes new values $\underset{\sim}{x}$ and $\underset{\sim}{y}$ available as outputs on the output lines $\underset{\sim}{X}$, $\underset{\sim}{Y}$. There is a simple finite-state machine which controls switching the processor from one kind of operation to the other.

Knowing what operations the processor must perform, we define a *time unit* to be the maximal time that is necessary for a processor to determine parameters c and s *or* to perform a rotation step together with loading and unloading its registers.

It is possible that a rotation will take more or less time than determination of the rotation parameters. The rotation parameters are determined only once, so the processors may occasionally be idle. This is a price we pay to guarantee that the whole system works correctly while keeping the system control relatively simple.

We assume that there is a synchronization mechanism which latches input and output lines. When processors are connected together, the changing output of one processor during a time unit should not interfere with the input to another processor. Sometimes we shall refer to the operations executed by a processor within one time unit as a pulsation (see Kung and Leiserson [10]).

5. Network organization. The systolic array proposed here is made up of a network of $n(n-1)/2$ processors, where n is the problem dimension. The position of a processor in the network is fully determined by integers i and k, $1 \leqq k < i \leqq n$, so every processor will be specified by a pair (i, k). The processor (i, k) is assigned to perform the transformation $P_{i,k}$.

The network organization has the property that all connections from a processor are to at most four neighboring processors. More precisely, output line $\underset{\sim}{X}$ of processor (i, k) coincides with input line Y of processor $(i-1, k)$, and output line $\underset{\sim}{Y}$ of processor (i, k) coincides with input line X of processor $(i+1, k+1)$ (see Fig. 3). All connections form a rectangular grid on a triangle, as illustrated for $n=6$ in Fig. 4.

There are special "gray" processors or shift registers along the bottom of the network. A gray processor does not perform any arithmetic. It simply delays data

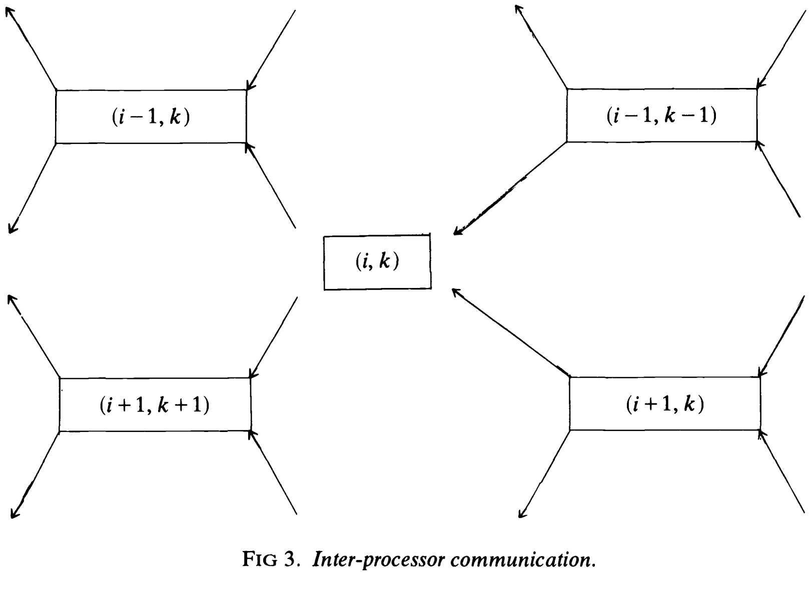


FIG 3. *Inter-processor communication.*

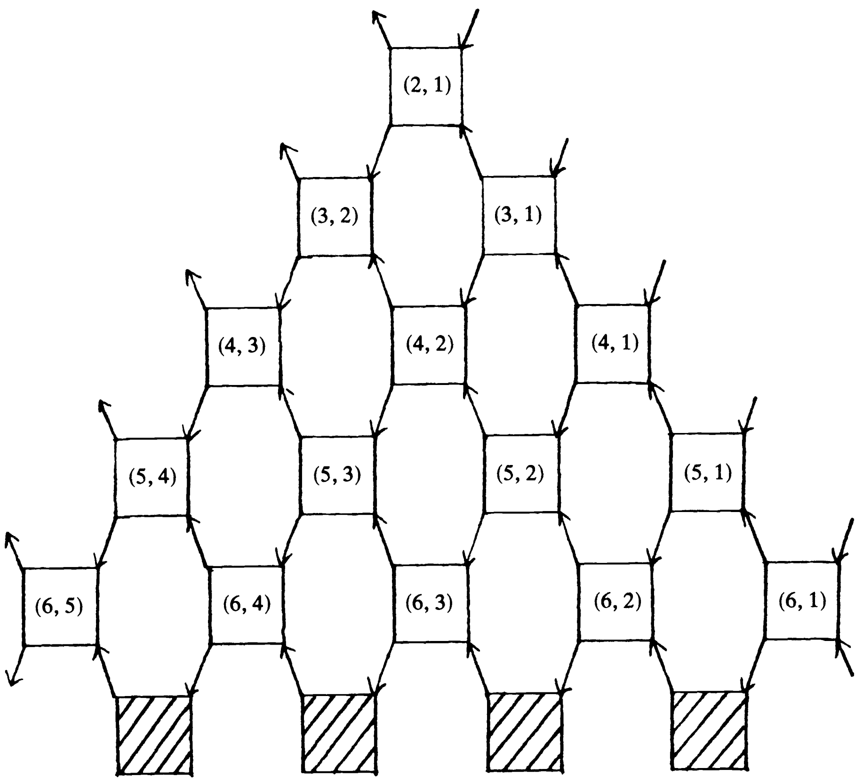


FIG. 4. *Layout of the processors* ($n = 6$).

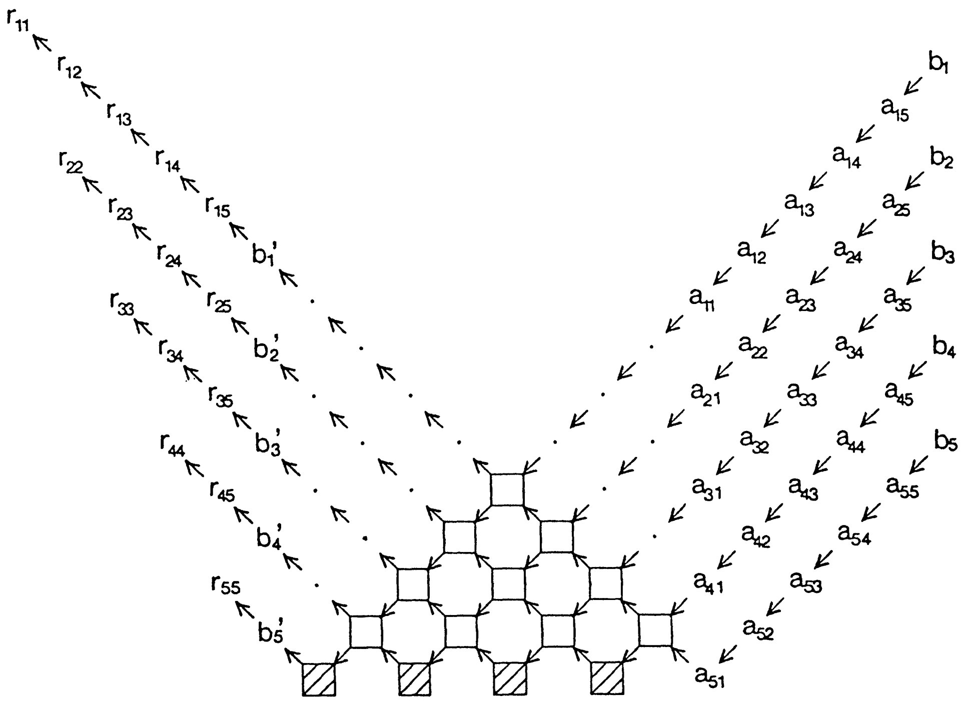


FIG. 5. *Data flow into the systolic array* ($n = 5$).

transmission by one time unit. This is necessary because only every third sweep introduces a zero in the last row.

Data enter the network through the right boundary, i.e. through the processors $(i, 1)$, $i = n, n-1, \cdots, 2$, and leave the network through the left boundary, i.e. through the processors $(i, i-1)$, $i = 2, 3, \cdots, n$ (see Figs. 4 and 5).

6. Operation of the systolic array. Our computational scheme is applied to the $n \times (n+1)$ matrix $\underset{\sim}{A} = [A, b]$, i.e., to the matrix A augmented by the vector b.

In the following description the superscript of a matrix coefficient will usually indicate the number of plane rotations $P_{i,k}$ in which this coefficient was involved. By convention, the coefficients of the original augmented matrix have superscript 0, with the exception of the coefficients of the last row, which for notational convenience have superscript 1.

The computation is initiated at time $\tau = 0$, when $a_{n-1,1}^{(0)}$ and $a_{n,1}^{(1)}$ enter the systolic array through the right bottom processor, i.e., processor $(n, 1)$. As its first operation, this processor determines rotation parameters c and s corresponding to transformation $P_{n,1}$ as well as computing $a_{n-1,1}^{(1)}$. At subsequent time steps processor $(n, 1)$ performs rotation steps. At time $\tau = 1$, processor $(n-1, 1)$ starts to work. Subsequently processors $(n-2, 1), (n-3, 1), \cdots, (2, 1)$ are activated at times $\tau = 2, 3, \cdots, n-2$. Figure 5 depicts how elements of the matrix are fed into the systolic array. In Fig. 6 we show four consecutive pulsations of the network.

We now specify the operation of the network precisely by giving the schedule of processor (i, k), $1 \leqq k < i \leqq n$. The processor (i, k) (assigned to perform plane rotation $P_{i,k}$) begins its activity at time $\tau = 3(k-1) + (n-i)$. Its first task is to annihilate the k-th element of row i. (The first $k-1$ elements of rows $i-1$ and i will already have been annihilated.) At time τ the processor determines rotation parameters c and s based on data $a_{i-1,k}^{(2k-2)}$ and $a_{i,k}^{(2k-1)}$, and computes the new value of the kth element of row $i-1$, i.e. element $a_{i-1,k}^{(2k-1)}$. Then $a_{i-1,k}^{(2k-1)}$ is made available as an output on the output line $\underset{\sim}{X}$. Every subsequent operation by the processor is a rotation step. More precisely, for $k < j \leqq n$, at time $3(k-1) + (n-i) + (j-k)$ processor (i, k) executes operations

$$a_{i-1,j}^{(2k-1)} := c \times a_{i-1,j}^{(2k-2)} + s \times a_{i,j}^{(2k-1)},$$

$$a_{i,j}^{(2k)} := -s \times a_{i-1,j}^{(2k-2)} + c \times a_{i,j}^{(2k-1)}$$

and makes $a_{i-1,j}^{(2k-1)}$ and $a_{i,j}^{(2k)}$ available as output on its output lines $\underset{\sim}{X}$ and $\underset{\sim}{Y}$ respectively. It is easy to check by induction that every processor gets its data at the right time.

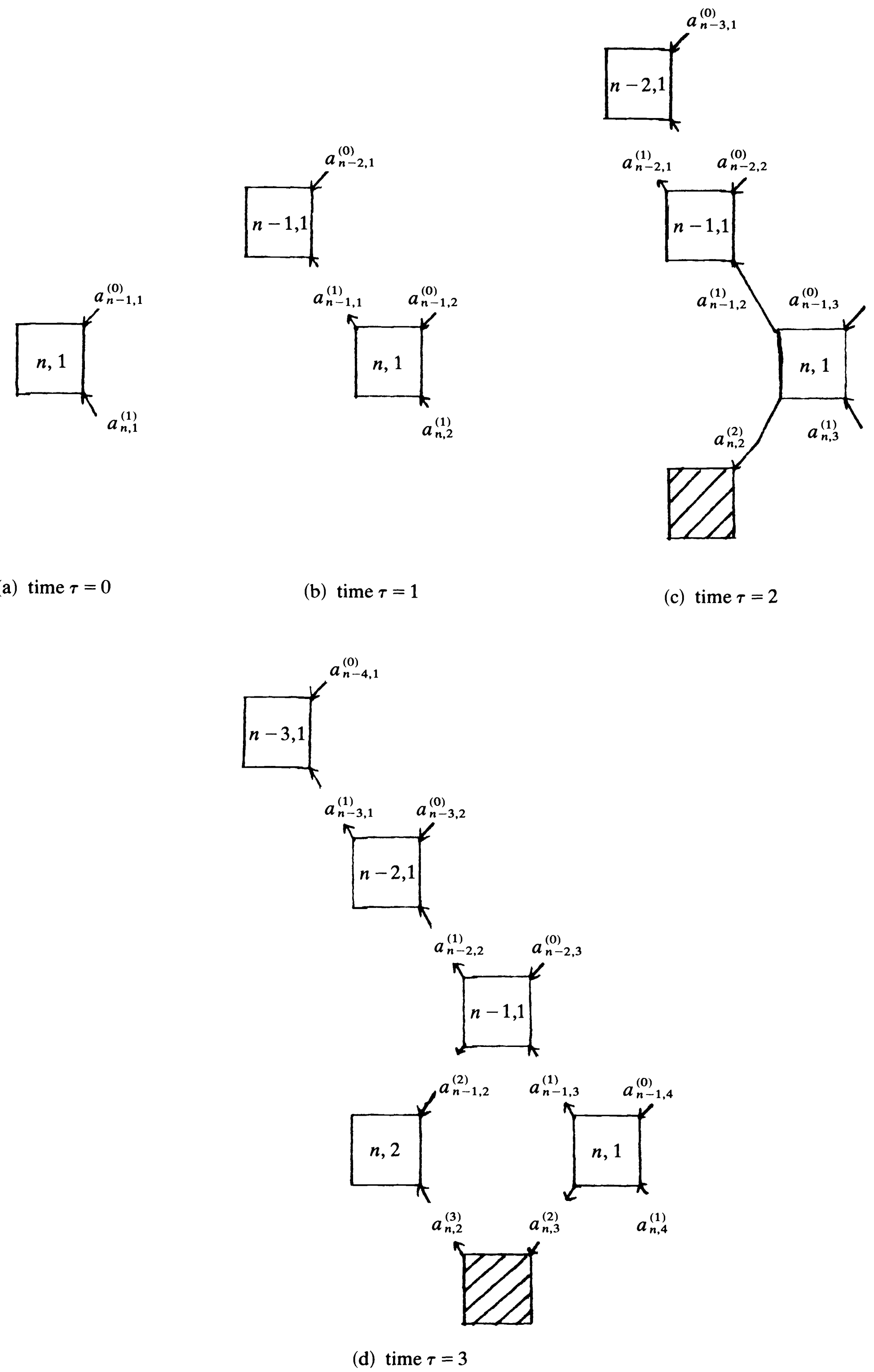
It follows from the schedule of the output processors, i.e. processors $(i, i-1)$, $i = 2, 3, \cdots, n$, that at time $n-1$ the coefficient $a_{1,1}^{(1)}$ leaves the network. The whole upper triangular matrix $R = QA$ and transformed right-hand side vector Qb are known at time $3n-3$. Thus we have:

THEOREM 6.1. *A dense nonsingular system of n linear equations can be orthogonally transformed to a triangular system in* $3n-3$ *time units using a systolic array consisting of* $n(n-1)/2$ *mesh-connected processors.*

We still have to solve a triangular linear system. This can be done in $3n$ time units, using a systolic array first introduced by Kung and Leiserson [10]. Thus we have:

THEOREM 6.2. *If A is an $n \times n$ nonsingular matrix, then a linear system of equations $Ax = b$ can be solved in $6n + O(1)$ time units using systolic arrays of $n(n-1)/2$ and n processors.*

Remarks. Note that several systems with the same matrix A and different right-hand side vectors $b_1, b_2, \cdots, b_m$ can be processed almost as easily as one. The



(a) time $\tau = 0$ (b) time $\tau = 1$ (c) time $\tau = 2$

(d) time $\tau = 3$

FIG. 6. *The first four steps.*

computational scheme is applied to the matrix $\underset{\sim}{A}=[A, b_1, \cdots, b_m]$ rather than to $[A, b]$, and $Q\underset{\sim}{A}$ is obtained in the time $3n-4+m$. Similarly, an obvious extension of our scheme may be used to solve linear least squares problems. Recently Gentleman and Kung [4] have proposed a different systolic scheme which has advantages over ours for solving linear least squares problems. In addition, for handling banded matrices they have proposed a scheme in which the number of processors needed in the systolic array depends on the band width of the matrix rather than its order.

The error analyses of our Givens process and back substitution process are as described in Gentleman [3] and Wilkinson [14] for the classical sequential processes. Thus, we have speeded up the process of solving systems of linear equations and maintained the numerical quality of the well-behaved sequential algorithm at the same time. (Singular or nearly singular A can be detected once the Givens triangularization of A has been computed, as in the sequential case.) A multiprocessor array structure equivalent to ours was independently proposed by Gannon [2].

It is worth noting that matrices which are too large for a given systolic array can be triangularized by first splitting them into blocks. The triangularization time is $\theta(n^3/p^2)$ where n is the matrix dimension and p^2 is the number of processor used, $p \leqq n$.

Sameh and Kuck [12] and Kowalik and Kamgnia [7] present schemes that require only $N=2n-3$ "sweeps" using the "short" chess knight move elimination order. However, the time taken by each of these sweeps is that required to generate the parameters in a rotation matrix *and* to perform a rotation. In addition, they do not consider the cost of data transfers. Suppose that generating a rotation matrix and performing a rotation each take a unit time, as assumed by the timing analysis of this paper. Then one can easily see from data dependency relations that our $N=3n-5$ sweeps with the "long" chess knight move elimination is the best one can do.

7. Application to the QR algorithm. One iteration of the QR algorithm can be expressed in the form

$$\text{factorization phase: } QA = R,$$

$$\text{multiplication phase: } \tilde{A} = RQ^T,$$

where Q is orthogonal and R is upper triangular. See, for example, Stewart [13].

Our systolic array is capable of performing the factorization phase. While the matrix A passes through the network, the orthogonal matrix Q is formed as a product of plane rotations $P_{i,j}$. Parameters defining the transformations $P_{i,j}$ are stored among the processors of the network. If we do not switch our network to process another factorization phase, the previously computed orthogonal matrix Q (in multiplicative form) is not destroyed and remains intact in the network. Now, by passing any other matrix B through the network, we obtain the product QB.

In the multiplication phase we have to know how to form a product $\tilde{A}=RQ^T$. By applying our systolic device to the matrix R^T we can easily get $\tilde{A}^T=QR^T$ instead. Now, to complete the multiplication phase it is enough to transpose the matrix $\tilde{A}^T$. Thus we need a fast method for matrix transposition. One way to achieve this is to use a buffer that supports fast two-dimensional addressing.

When we have a systolic array for matrix triangularization and a buffer to support matrix transposition, one iteration of the QR algorithm is easy to execute. First we produce the matrix R, then transpose it, form $\tilde{A}^T=QR^T$, and transpose the matrix $\tilde{A}^T$ to obtain $\tilde{A}$. The cost of one iteration of the QR algorithm performed in this way is Kn time units. We shall not specify the constant K as it depends on how fast we compute matrix transposition. Our treatment of the QR algorithm here is preliminary; future research is needed to study issues such as shift selection, convergence testing, etc.

REFERENCES

[1] L. CSANKY, *Fast parallel matrix inversion algorithms.* SIAM J. Comput, 5 (1976), pp. 618–623.

[2] D. GANNON, *A note on pipelining a mesh connected multiprocessor for finite element problems by nested dissection*, in *Proc. 1980 International Conference on Parallel Processing*, IEEE Computer Society, August, 1980, pp. 197–216.

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

[4] W. M. GENTLEMAN AND H. T. KUNG, *Matrix triangularization by systolic arrays*, in Proc. of SPIE Symposium, Vol. 298, Real-Time Signal Processing IV, The Society of Photo-optical Instrumentation Engineers, August, 1981.

[5] D. HELLER, *A survey of parallel algorithms in numerical linear algebra*, SIAM Rev., 20 (1978), pp. 740–777.

[6] R. M. KANT AND T. KIMURA, *Decentralized parallel algorithms for matrix computation*, in Proc.Fifth Annual ACM Symposium of Computer Architecture, Palo Alto, CA, 1978, pp. 96–100.

[7] J. S. KOWALIK AND E. R. KAMGNIA, *Parallel Givens transformations applied to matrix factorization and systems of linear equations*, Techn Rep. CS-79-050, Washington State University, Pullman, WA, 1979.

[8] J. S. KOWALIK, S. P. KUMAR AND E. R. KAMGNIA, *An implementation of fast Givens transformations on a* MIMD *computer*, Techn. Rep. Washington State University, Pullman, WA, November, 1980.

[9] H. T. KUNG, *Why systolic architectures?*, IEEE Computer Magazine, 15(1): (Jan 1982), pp. 37–46.

[10] H. T. KUNG AND C. E. LEISERSON, *Systolic arrays (for* VLSI), in Sparse Matrix Proceedings 1978, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1979, pp. 256–282. A slightly different version appears in Introduction to VLSI Systems by C. A. Mead and L. A. Conway, Addison-Wesley, Reading, MA, 1980, 8.3.

[11] R. E. LORD, J. S. KOWALIK AND S. P. KUMAR, *Solving linear algebraic equations on a* MIMD *computer*, Techn. Rep., Washington State University, Pullman, WA, August, 1980.

[12] A. H. SAMEH AND D. J. KUCK, *On stable parallel linear system solvers*, J. Assoc. Comput. Mach., 25 (1978), pp. 81–91.

[13] G. W. STEWART. *Introduction to Matrix Computations*, Academic Press, New York, 1973.

[14] J. H. WILKINSON, *The Algebraic Eigenvalue Problem.*, Oxford Univ. Press, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

CALCULATING THE ISOCHRONES OF VENTRICULAR DEPOLARIZATION*

J. J. M. CUPPEN†

Abstract. The inverse problem of electrocardiography is treated by a) the formulation of a suitable model and a powerful transformation of the time domain; b) regularization techniques using the singular value decomposition.

The solution calculated from the potentials at the body surface is expressed in terms of the activation times on the heart surface. Promising results of numerical experiments with simulated measurements errors are presented.

Key words. electrocardiography, ill-posed problems, regularization

AMS-MOS classification. 31B20; 65N99,R20

1. Introduction. In this paper an inverse problem from electrocardiography is considered. This problem can be stated as: calculate the course of the electrical depolarization process in the heart from the potentials which can be measured on the body surface. It is, like many inverse problems, of an ill-posed nature. It leads either to a Cauchy problem for the Laplace equation (cf. Colli Franzone et al. [1979]) or to an integral equation of the first kind with a smooth kernel. Both are ill-posed since small changes in the data may cause nonexistence or arbitrarily large perturbations of the solution.

In such a situation it is necessary to obtain and exploit additional information to make the solution better determined. Therefore in our research much effort has been put into the formulation of a model and a representation of the solution which incorporates as much physiological knowledge as possible.

Using this model a method is developed for the numerical treatment of the problem based on regularization techniques and the Singular Value Decomposition. Numerical experiments show that the method works remarkably well for a model set up where the heart is approximated by a sphere, and even quite well for a more realistic heart geometry including the ventricular cavities.

2. Problem description and model formulation. Electrocardiologists distinguish different phases in the heart beat cycle, of which we shall consider only one, namely the electrical activation of the ventricles which, in the electrocardiogram, is represented by a characteristic waveform known as the QRS complex. During the corresponding time interval, say $[0,T]$, in which the heart is mechanically at rest, the electrical activation takes place which initiates contraction of the ventricular muscle.

The activation takes place at a front which moves through this entire muscle (predominantly inward outward). The resulting electrical activity, as measured at a distance, can be modeled by a double layer of constant uniform strength situated at the boundary surface $S_r(t)$ between tissue that is already activated and tissue which is not yet reached by the depolarization wave front. In Fig. 2.1 a schematic representation is given of a cross section of the relevant part of the heart (the atria on top are not depicted). Marked are $S_r(t)$ and the already activated part of the ventricular muscle (dotted) at an early time instant t_1 and a later one t_2 in the depolarization sequence.

* Received by the editors December 14, 1981, and in revised form October 11, 1982.

† Department of Mathematics, University of Amsterdam, the Netherlands. Present address: Philips Medical Systems, NMR, HOI, NL5600MD Eindhoven, the Netherlands.

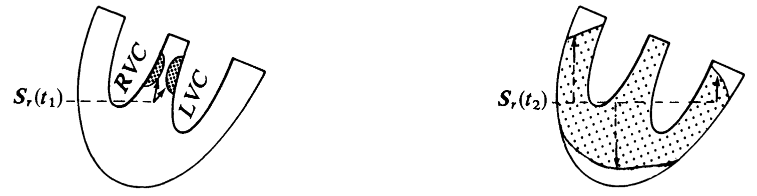


FIG. 2.1

From observation points outside the heart the double layer on $S_r(t)$ cannot be distinguished from a double layer $S'_r(t)$ with the same uniform strength and boundary curve in space (a "deformation" of $S_r(t)$, cf. Fig. 2.2), since the difference between them is a closed uniform double layer which generates no external potential (Courant and Hilbert [1961, IV.1.3]).

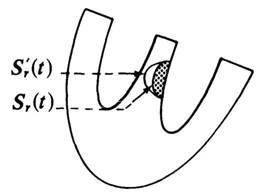


FIG. 2.2

We can however turn this indeterminacy to our advantage by considering instead of $S_r(t)$ its "outfolding" $S_h(t)$ to the heart surface X (the closed surface given by endocardium plus epicardium) as an equivalent source (cf. Fig. 2.3). This is of course equivalent to considering only the boundary curve of $S_r(t)$ as it moves over X. $S_h(t)$ is the part of X that borders already activated tissue at $t \in [0, T]$ (cf. Dotti [1974], Salu [1978]).

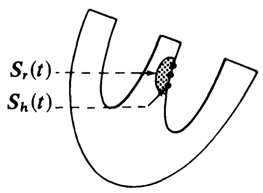


FIG. 2.3

Since the influence of the volume conductor surrounding the sources can be treated in a quasi-static approximation (propagational effects are negligible, Plonsey and Heppner [1967]), the relation between $S_h(t)$ and the potentials $v(y, t)$ on the body surface Y^1 can be expressed by means of a Green's function $\mathbf{A}(y, x)$. This function is defined as the potential at y on Y due to an elementary dipole in x on the heart surface X, directed along the outward normal on the surface. It also expresses the effects of the shape and conductivity of the body and of all regions of different conductivity that are being distinguished (such as lungs and cavities filled with blood).

^{1} The surfaces X and Y are closed, regular, orientable surfaces.

For each fixed x, the potentials $\mathbf{A}(y, x)$ on the body surface Y can be calculated as the solution of the so-called forward problem of electrocardiography, for example by finite element techniques for the Laplace equation (cf. Colli Franzone et al. [1979]) or a boundary integral approach (Barnard et al. [1967]).

Consider the case of a homogeneous body with conductivity σ. Let $v_\infty(y; x)$ be the potential that would be generated in an infinite homogeneous medium at y by an elementary dipole at x directed along the outward normal n_x to X:

$$v_\infty(y;x)=\frac{1}{4\pi\sigma}\frac{\partial}{\partial n_x}\frac{1}{r(x,y)}=\frac{1}{4\pi\sigma}\frac{(y-x)\cdot n_x}{|y-x|^3}.$$

$\mathbf{A}(y,x)$ is then equal to $v_\infty(y;x)+\chi(y)$, where χ satisfies the following Neumann problem for the Laplace equation:

$$\begin{aligned}&\Delta\chi=0 \quad \text{inside } Y,\\ &\frac{\partial}{\partial n_y}\chi=-\frac{\partial}{\partial n_y}v_\infty(y;x) \quad \text{on } Y,\end{aligned} \tag{2.2}$$

expressing that $(\partial/\partial n_y)\mathbf{A}(y,x)=0$ on Y. The integral equation used in the boundary integral approach follows from (2.2) as

$$\mathbf{A}(y,x)=2v_\infty(y;x)-\frac{1}{2\pi}⨍_Y \mathbf{A}(\eta,x)\,d\omega_\eta$$

where $d\omega_\eta$ denotes integration over the solid angle subtended in y by a surface element at η, and $⨍$ denotes the Cauchy principal value of the improper integral over Y. For more details see Cuppen and van Oosterom [1983].

Note that $\mathbf{A}(y,x)$ is smooth since x never comes close to y.

Taking into account the orientation of the double layer $S_r(t)$ corresponding to a negative, or inwardly oriented double layer at $S_h(t)$, it follows that $v(y,t)$ is equal to minus the integral of the contribution of all elementary dipoles contained in S_h (i.e. active) at time t:

$$v(y,t)=-\int_{S_h(t)}\mathbf{A}(y,x)\,dx, \qquad y\in Y, \quad t\in[0,T]. \tag{2.3}$$

The appearance of the unknown $S_h(t)$ in (2.3) as the range of integration is not very convenient for our purpose. We use the fact that for physiological reasons

$$S_h(t_1)\subset S_h(t_2) \quad \text{if } t_1<t_2 \tag{2.4}$$

which means that the direction of propagation of the depolarization front does not reverse. Therefore $S_h(t)$ can be characterized by a continuous function $\tau(x)$ which, for each point $x\in X$, yields the time at which the front reaches x:

$$\tau(x)=\inf\{t\in[0,T]\,|\,x\in S_h(t)\}, \qquad x\in X. \tag{2.5}$$

By means of the Heaviside step-function $\mathbf{H}$,

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

(2.3) can be written as

$$\int_X \mathbf{A}(y,x)\mathbf{H}(t-\tau(x))\,dx=-v(y,t), \qquad y\in Y, \quad t\in[0,T]. \tag{2.6}$$

Given the potentials $v(y,t)$ for $t\in[0,T]$ and y on the body surface Y or a part of it, (2.6) constitutes a nonlinear Fredholm integral equation of the first kind for the "depolarization time" $\tau(x)$ on the heart surface X.

3. Properties and transformation of the basic equation. The following integral equation is related to (2.6):

$$\int_X \mathbf{A}(y,x)\psi(x)dx=\phi(y),\qquad y\in Y. \tag{3.1}$$

This equation can be seen as an equation for a double-layer density ψ on X, given the potential distribution ϕ on Y. Whenever we encounter an equation of the form (3.1) or (2.6) we shall assume that a solution exists (a solution usually describes a source which we know to be physically present).

Let us denote (for convenience) by $\mathbf{A}$ also the integral operator defined by

$$(\mathbf{A}\psi)(y)=\int_X \mathbf{A}(y,x)\psi(x)\,dx,\qquad y\in Y. \tag{3.2}$$

The operator $\mathbf{A}$ can be written as the product of two operators $\mathbf{A}^{(1)}$ and $\mathbf{A}^{(2)}$, where $\mathbf{A}^{(1)}$ produces the potential ϕ_∞ generated by the double-layer source ψ on X in a homogeneous medium of infinite extent

$$\phi_\infty(y)=(\mathbf{A}^{(1)}\psi)(y)=\int_X \psi(x)\,d\omega_x$$

and $\mathbf{A}^{(2)}$ is the inverse of a deflated Fredholm integral operator of the second kind on the surfaces where the conductivity is discontinuous (cf. Lynn and Timlake [1970], Barnard et al. [1967]). Since the surfaces may be assumed to be smooth, the operator $\mathbf{A}$ behaves as a compact operator from $C[X]$ to $C[Y]$. See also the appendix where a sketch of proof is given of the following property:

(3.3) The null-space $\mathbf{N}(\mathbf{A})$ is the space of constant functions on X.

Note that this means that solutions to (3.1) are unique up to a constant, which implies uniqueness for the solution of (2.6) as expressed by the following lemma.

LEMMA. *If $v(y,t)$ is not identically zero then the solution $\tau(x)$ of* (2.6), *if it exists, is unique.*

Proof. If τ_1 and τ_2 are both solutions to (2.6), satisfying $0\leqq\tau(x)\leqq T$ for all x, then (3.3) yields that for each t there is a constant $c(t)$ such that for each x

$$\mathbf{H}(t-\tau_1(x))=\mathbf{H}(t-\tau_2(x))+c(t).$$

Hence, for each t, $\mathbf{H}(t-\tau_1(x))-\mathbf{H}(t-\tau_2(x))$ is independent of x. Therefore, it easily follows that if $\tau_1(x_1)\neq\tau_1(x_2)$ or $\tau_2(x_1)\neq\tau_2(x_2)$ then $\tau_1(x_1)=\tau_2(x_1)$ and $\tau_1(x_2)=\tau_2(x_2)$ (by varying t between the two values). Therefore if either τ_1 or τ_2 is not constant, then τ_1 and τ_2 must be equal. If both are constant, (3.3) yields that $v(y,t)$ is identically zero. □

The proof of this lemma suggests that $\tau(x)$ might be calculated in two phases, first approximating $f_t(x)=\mathbf{H}(t-\tau(x))$ for fixed values of t from equations of the form (3.1) and then calculating $\tau(x)$ from the obtained approximations. There is, however, a better way which is based on the observation that equation (2.6) behaves nicely under transformations in the domain of t. For instance integrating (2.6) over t from

0 to T yields

$$\int_0^T \int_X \mathbf{A}(y,x)\mathbf{H}(t-\tau(x))\,dx = -\int_0^T v(y,t)\,dt \tag{3.4}$$

so

$$\int_X \mathbf{A}(y,x)(T-\tau(x))\,dx = -\bar{v}(y),$$

where $\bar{v}(y)$ is a notation for $\int_0^T v(y,t)\,dt$. With (3.3) it follows that

$$\int_X \mathbf{A}(y,x)\tau(x)\,dx = \bar{v}(y), \qquad y \in Y. \tag{3.5}$$

Considering (3.5) it turns out that τ is determined, up to a constant, by $\bar{v}(y)$. This is remarkable because (3.5) employs only the *time-integrated* measurements $\bar{v}(y)$ from which one would not expect to be able to calculate the *development in time* $\tau(x)$ of the activation.

A physical explanation of this phenomenon is based on the fact that, in the model derived, the sources are highly restricted. The source in a point x on the heart surface is either active or inactive at a certain time, and if it is active, then it has a fixed strength c, and it stays active. Therefore the integrated signal contains a contribution from x equal to a static contribution corresponding to its fixed strength c, multiplied by the time it has been active, i.e. $(T-\tau(x))$. Consequently, the integrated signal can be viewed as the potentials one would obtain from a double-layer source on the heart surface X, with strength $c \times (T-\tau(x))$ in x for all x on X. It is however well known (cf. appendix) that each double-layer source is determined, up to a constant, from the body surface potentials; which shows that τ can indeed be determined from the integrated signal.

More general transformations can be applied to (2.6). Consider a continuously differentiable function $f(t)$ and define

$$\tilde{v}(y;f) = \int_0^T f'(t)v(y,t)\,dt. \tag{3.6}$$

Then integrating (2.6) as above yields

$$\int_X \mathbf{A}(y,x)\theta(x)\,dx = \tilde{v}(y;f), \qquad y \in Y, \tag{3.7}$$

$$\theta(x) = f(\tau(x)), \qquad x \in X, \tag{3.8}$$

which can be used to solve θ from (3.7) and subsequently τ from (3.8). Of course several functions f can be used simultaneously, each of them determining a function θ through (3.7). This gives a system of equations instead of (3.8). Examples of such sets of functions are the polynomials

$$p^{(i)}(t) = \left(\frac{2t}{T}-1\right)^i, \qquad i = 1, 2, \cdots, I \tag{3.9}$$

or, inspired on the Laplace transform,

$$f_s(t) = e^{-st}, \qquad s \in [-S, S]. \tag{3.10}$$

In the sequel two or three polynomials shall be used as given in (3.9) with right-hand sides

$$v^{(i)}(y)=\tilde{v}(y;p^{(i)})=\frac{2i}{T}\int_0^T\left(\frac{2t}{T}-1\right)^{i-1}v(y,t)\,dt. \tag{3.11}$$

This leads to a set of equations

$$\int_X \mathbf{A}(y,x)\theta^{(i)}(x)\,dx=v^{(i)}(y),\qquad y\in Y,\quad i=1,\cdots,I, \tag{3.12}$$

$$p^{(i)}(\tau(x))=\theta^{(i)}(x),\qquad x\in X,\quad i=1,\cdots,I. \tag{3.13}$$

4. Regularization and elimination of indeterminacies. The equations (3.12) and (3.13) from which $\tau(x)$ is to be solved can be written in operator form as

$$\mathbf{A}\theta^{(i)}=v^{(i)},\qquad p^{(i)}(\tau(x))=\theta^{(i)}(x),\qquad x\in X,\quad i=1,\cdots I. \tag{4.1}$$

If $\mathbf{A}\theta^{(i)}=v^{(i)}$ is to be solved for fixed i (1, 2 or 3 are used in the experiments) the problem arises that such equations are ill-posed (cf. § 1). Fortunately it is known from physiology that the unknown function τ is bounded and smooth (see also § 6, note 2), so standard techniques for the regularization of (4.1) can be applied (cf. Tikhonov and Arsenin [1977]). First (4.1) is written as a least squares problem and then a penalty function $\|\mathbf{C}\theta^{(i)}\|$ is added involving the norm of $\theta^{(i)}$ and/or some of its partial derivatives to suppress oscillations.[2] This leads to the following minimization problem:

Determine a function $\theta^{(i)}(x)$ that minimizes

$$\|\mathbf{A}\theta^{(i)}-v^{(i)}\|^2+\alpha_i\|\mathbf{C}\theta^{(i)}\|^2. \tag{4.2}$$

The parameter α_i, which must be chosen positive, controls the amount of regularization (smoothing, filtering). The choice of this so-called "regularization parameter" is discussed in § 5. For positive α_i, (4.2) is a well-posed problem if positive ε_1 and ε_2 exist such that $\|\mathbf{A}\Delta\theta\|<\varepsilon_1\|\Delta\theta\|$ implies that $\|\mathbf{C}\Delta\theta\|>\varepsilon_2\|\Delta\theta\|$. In practice $\mathbf{C}$ is formed as $(\mathbf{C}^*\mathbf{C})^{1/2}$ where $\mathbf{C}^*\mathbf{C}$ is composed as a positive linear combination of the identity $\mathbf{I}$ and components $\mathbf{C}_i^*\mathbf{C}_i$ where $\mathbf{C}_i$ is a differential operator. If the coefficient of $\mathbf{I}$ in this combination is positive, then $\mathbf{C}$ is positive definite and therefore (4.2) is well-posed for positive α_i. Probably, this is also true for combinations not containing $\mathbf{I}$ (cf. Locker and Prenter [1980], Colli Franzone and Magenes [1979] and Colli Franzone et al. [1979]), but an analysis of this problem is beyond the scope of this research.

Having obtained from (4.2) approximations $\tilde{\theta}^{(i)}$ for $\theta^{(i)}$ for one or more polynomials $p^{(i)}$, an approximation for τ has to be calculated. Assume that $p^{(i)}(t)=(2t/T-1)^i$, $i=1,2,3$. Since the null-space of $\mathbf{A}$ consists of the constant functions, $\tilde{\theta}^{(i)}$ approximates $\theta^{(i)}$ up to an unknown constant b_i. Ignoring further errors in $\tilde{\theta}^{(i)}$ leads to the following set of equations in τ and the constants b_i:

$$\left(\frac{2\tau(x)}{T}-1\right)^i=\tilde{\theta}^{(i)}(x)+b_i. \tag{4.3}$$

[2] For the treatment of the actual heart problem (cf. § 7) with its special geometry, derivatives on the surface are not sufficient. The lowest parts of the ventricular cavities are, in three-dimensional space, near to the lower parts of the outer surface. But measured over the surface the distance between these parts is large. To reflect the coupling between them which exists in reality, extra terms have to be incorporated in $\mathbf{C}$ reflecting upper bounds for the derivatives of the activation time throughout the ventricular muscle. In this way points which are close to each other in space are constrained to have activation times which do not differ too much.

A useful generalization of the method allows the kernel $\mathbf{A}(y, x)$ to contain an unknown multiplicative positive constant c (the double layer strength which may not be exactly known). This yields

$$c\left(\frac{2\tau(x)}{T}-1\right)^i = \tilde{\theta}^{(i)}(x)+b_i. \tag{4.4}$$

If the functions $\tilde{\theta}^{(i)}$ in (4.4) are given for $i = 1, 2, 3$ it follows that

$$c(\tilde{\theta}^{(2)}(x)+b_2) = (\tilde{\theta}^{(1)}(x)+b_1)^2 \tag{4.5}$$

and

$$(\tilde{\theta}^{(1)}(x)+b_1)(\tilde{\theta}^{(3)}(x)+b_3) = (\tilde{\theta}^{(2)}(x)+b_2)^2. \tag{4.6}$$

If $\tilde{\theta}^{(1)}(x)$, $\tilde{\theta}^{(2)}(x)$, $\tilde{\theta}^{(3)}(x)$ and the constant function are linearly independent, then (4.6) can have at most one solution for b_1, b_2 and b_3 (linear independence already follows if $\tau(x)$ attains at least four different values on X). A solution can be assumed to exist, from which c and subsequently $\tau(x)$ follow. In the presence of errors the equations are solved in the sense of least squares. If c is known (as in our experiments) $i = 1$ and 2 or $i = 1$ and 3 suffice for the calculation of τ.

5. The choice of the regularization parameters. The choice of the regularization parameters α_i balances the errors introduced by the regularization against those the regularization attempts to suppress. If the α_i are chosen too small, the resulting linear systems are still too ill-conditioned and regularization is ineffective: oscillating errors dominate the results. On the other hand, if the α_i are chosen too large, the smoothing terms are weighted so heavily that too much information is suppressed: the relevant characteristics of the solution are lost.

Presently, no satisfactory general solution to the problem of how to choose the α_i is available, although some progress has been made (cf. Golub, Heath and Wahba [1979], Köckler [1974]). The present problem formulation has the unusual property that a choice can be made on the basis of an additional equation. In practice namely only two of the independent equations given by (3.12) are used to calculate (for given values of α_1 and α_2) an approximate solution τ_{α_1,α_2}. Therefore the residual of the third equation

$$r_{\alpha_1,\alpha_2} = \left\| \int_X \mathbf{A}(y,x) p^{(3)}(\tau_{\alpha_1,\alpha_2}(x))dx - v^{(3)}(y) \right\|_Y$$

gives a measure for the quality of the regularization parameters α_1, α_2. In the experiments presented below, α_1 and α_2 are chosen by a minimization of r_{α_1,α_2}. This yields satisfactory results, although in some experiments r_{α_1,α_2} has one or more local minima which yield worse results than those obtained for the global minimum.

6. Discretization and implementation. The discretization of (4.2) is performed as follows. The surface X is approximated by a set of triangles with vertices x_j, and on the surface Y a number of collocation points y_k is chosen (the observation points where the electrodes are located in practice). For each x_j a basis function b_j on X is defined by the conditions that b_j be linear on each triangle, zero in all vertices x_l with $l \neq j$, and one in vertex x_j.

For each collocation point y_k and each basis function b_j the matrix element $A_{k,j}$ is defined by

$$A_{k,j} = \int_X \mathbf{A}(y_k, x) b_j(x)\, dx. \tag{6.1}$$

Actually the integral $A_{k,j}$ is directly calculated from the forward problem with the source $b_j(x)$ (cf. Barnard et al. [1967], Lynn and Timlake [1968a], [1968b], van Oosterom [1978]). $A_{k,j}$ is, of course, a weighted approximation to $\mathbf{A}(y_k, x_j)$.

A function $\theta(x)$ on X is approximated by a linear combination of the b_j:

$$\theta(x) \approx \sum_j \theta_j b_j(x), \qquad x \in X. \tag{6.2}$$

Consequently, the integral equation (3.12) is approximated by the following linear system of equations:

$$\sum_j A_{k,j}\theta_j = v^{(i)}(y_k), \qquad y_k \in Y. \tag{6.3}$$

Thus, an approximation of the first term in the minimization problem (4.2) is obtained.

Note 1. It is not necessary that the number of discretization points on Y is larger than or equal to that on X. Rather the discretization on X should be chosen such that the smooth part of the solution can be adequately described, and the number of points (measuring leads) on Y should be chosen according to the number of independent potential patterns which can at the given noise level be observed on the body surface. As in the example with the realistic geometry in § 7, this may lead to an underdetermined system of equations (6.3). However, the elements of the null-space of matrix A are discretizations of source distributions that yield a zero potential in all observation points on Y, so it follows from the smoothing properties of $\mathbf{A}$ that these distributions belong to the class of spurious oscillatory source components discussed above, which are filtered out by the regularization.

The second term in (4.2), the regularization operator, poses a problem in itself. To approximate this term, one would like to obtain a matrix C such that the C-norm of the vector $\theta = (\theta_j)_j$:

$$\|C\theta\|^2 = (\theta^T C^T C\theta)^{1/2} \tag{6.4}$$

approximates the norm of one of the derivatives of θ on X, say its Hessian,

$$\|\mathbf{H}(\theta)\|_X = \left(\int_X \left\{\left(\frac{\partial^2\theta}{\partial\xi^2}\right)^2 + 2\left(\frac{\partial^2\theta}{\partial\xi\partial\eta}\right)^2 + \left(\frac{\partial^2\theta}{\partial\eta^2}\right)^2\right\} dx\right)^{1/2}, \tag{6.5}$$

where (ξ, η) represent a local coordinate system on X around x.

Note 2. Strictly speaking, neither the gradient nor the Hessian of τ exist in points on the heart surface where the activation starts or where the wavefronts collide. However, since τ can be well approximated by functions for which these derivatives do exist, and the regularizing operator only plays a modifying role in the inverse calculation, it is still possible to use the Hessian for regularization. This is advantageous because higher derivatives in the regularizing operator tend to have more effect on the dangerous high frequency components in the solution, and less on the low frequency components which should rather not be modified at all.

Suppose that for each x_j and for each partial derivative occurring in (6.5) the matrix C contains a row giving a difference formula for the derivative on the basis of value of θ in x_j and a number of neighboring points x_i on X (this difference formula can be multiplied by the root of an integration weight for integration over X with knots x_j). Then it is clear that (6.4) is an approximation of (6.5).

Such difference formulae are obtained by first deriving a local coordinate system (ξ, η) around each point x_j and then performing a formal least squares fit to the truncated Taylor series around x_j. This means that for each discretization point x_j on

X a set of neighbors is chosen. A sphere[3] is fitted through x_j, as close as possible through its neighbors (close neighbors weighting more heavily than far ones). The neighbors of x_j are then projected onto the sphere and a local coordinate system with its origin in x_j is derived from the angular coordinates of a spherical coordinate system with its origin in the origin of the sphere. To obtain a local measure that conforms to the Cartesian system, these angles are multiplied by the radius of the sphere.

The next step is to consider the Taylor series expansion of θ around x_j in the coordinates ξ and η:

$$\theta(x_{\xi,\eta})-\theta(x_j)=\xi\frac{\partial\theta}{\partial\xi}+\eta\frac{\partial\theta}{\partial\eta}+2\xi\eta\frac{\partial^2\theta}{\partial\xi\partial\eta}+\xi^2\frac{\partial^2\theta}{\partial\xi^2}+\eta^2\frac{\partial^2\theta}{\partial\eta^2}+\cdots.$$

Truncating the expansion at some point, and substituting the known values of (ξ, η) for each of the neighbors x_i of x_j, yields a system of linear equations for the derivatives of θ, with the (yet unknown) differences $\theta(x_i)-\theta(x_j)$ as right hand sides. By means of the singular value decomposition a generalized inverse of the known coefficient matrix can be derived which approximates the partial derivatives of θ in x_j in terms of the function values of θ in x_j and its neighbors.

The difference formulae giving upper bounds for the derivatives in space of θ (in reality τ is defined throughout the muscle) are analogously derived. Here the set of neighbors included points across the muscle tissue and the formal Taylor series expansion is performed in terms of the normal Cartesian coordinates. Thus difference formulae follow for the partial space derivatives of θ in x_j. However, since the points across the muscle tissue cannot be considered as close discretization points in space, these difference formulae can only be seen as lower bounds for the derivatives of τ.

Having obtained difference formulae, matrices C_j^k are derived, such that $\theta^T C_j^{kT} C_j^k \theta$ approximates the square of the kth derivative at x_j. A weighted sum of squares of the norm of the derivatives on the surface is then obtained by

$$C^TC=\sum_{j,k}\lambda_j\mu_k C_j^{kT}C_j^k$$

where λ_j and μ_k are weighting coefficients.

One further manipulation is convenient which replaces the matrix C in (6.4) by the matrix L which is a lower triangular Choleski factor of the positive (semi-)definite matrix C^TC. Formula (6.4) is then replaced by

$$\|L^T\xi\|=(\theta^TLL^T\theta)^{1/2}. \tag{6.6}$$

Often L is a matrix with a relatively narrow profile, i.e. it has many rows starting with many zeros. This can be used to obtain greater computational efficiency.

If the null-spaces of A and C do not intersect then the discretized minimization problem approximating (4.2) has a solution given by

$$\theta_{\alpha_i}^{(i)}=(A^TA+\alpha_iLL^T)^{-1}A^Tv^{(i)}. \tag{6.7}$$

Assuming that L is nonsingular, which can be achieved in our case by adding $\|C^TC\|\cdot e_ne_n^T$ to C^TC (removing the constant vector from its null-space), (6.7) is equivalent to

$$\theta_{\alpha_i}^{(i)}=L^{-T}(\tilde{A}^T\tilde{A}+\alpha_iI)^{-1}\tilde{A}^Tv^{(i)}, \qquad \tilde{A}=AL^{-T},$$

[3] This is not a truly quadratic approximation since a sphere is not the most general quadratic surface. However, in the present application the obtained fit is considered acceptable. Moreover, the discretization of the regularizing operator is not critical.

and if the singular value decomposition of $\tilde{A}$ is given by

$$\tilde{A} = U\Sigma V^T = (U_1|U_2)\begin{pmatrix}\Sigma_1 & 0\\ 0 & 0\end{pmatrix}(V_1|V_2)^T = U_1\Sigma_1V_1^T, \tag{6.8}$$

U, V orthogonal, U_1, V_1 column-orthogonal, then (6.7) is further reduced to

$$\begin{aligned}\theta^{(i)}_{\alpha_i} &= L^{-T}(V\Sigma^T\Sigma V^T + \alpha_i I)^{-1}V\Sigma^TU^Tv^{(i)}\\ &= L^{-T}V(\Sigma^T\Sigma + \alpha_i I)^{-1}\Sigma^TU^Tv^{(i)}\\ &= L^{-T}V_1(\Sigma_1^2 + \alpha_i I)^{-1}\Sigma_1U_1^Tv^{(i)}.\end{aligned} \tag{6.9}$$

The decomposition (6.8) can be calculated by a new method described by Cuppen [1983]. This method calculates the decomposition at approximately the cost of a bidiagonalization of A. The subsequent calculation of $\theta^{(i)}_{\alpha_i}$ according to (6.9) for a sequence of values of α_i is comparatively inexpensive.

7. Numerical experiments. Two series of numerical experiments were performed. In the first a model geometry is used with two spheres, a smaller sphere as a "heart" lying eccentrically inside the larger one used as "torso" surface. In the second series calculations are performed using a realistic human heart-torso geometry. In the first case a reference solution was generated and for the realistic geometry the activation as given by the measurements of Durrer et al. [1970] was used as a reference solution. The right-hand sides $v^{(i)}(y)$ were calculated according to (3.12) and subsequently perturbed to represent measurement errors as are encountered in practice. The inverse problem was then tackled with the methods described in this paper and the results were compared with the reference solution.

The experiments were performed on a CDC Cyber 750 system. Setting up the equations required, for the realistic geometry, about 3 minutes CP time and 5 minutes IO time, whereas calculating inverse solutions for 100 choices of the regularization parameters required 1 minute CP time and 1 minute IO time.

Model geometry. The two spheres with radii $\frac{1}{3}$ and 1 are situated such that the center of the larger sphere is on the smaller sphere (cf. Fig. 7.1). The conductivities are chosen to be 0 outside the larger sphere and 1 inside. The spheres are discretized using 192 triangles and 98 points on the inner sphere, and 128 triangles and 66 points on the outer sphere.

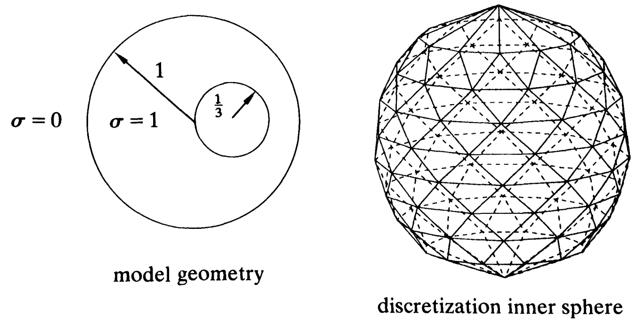


FIG. 7.1

The 128 centers of the triangles on the outer sphere are used as observation points y_i (cf. § 6). The reference solutions generated and the inverse solutions are given by the activation times $\tau(x_j)$ at each of the 98 vertices x_j on the inner sphere.

TABLE 7.1
Reference solution, model problem

					9.6						
15.6	17.4	17.4	17.4	15.6	12.3	8.7	4.5	0.0	4.8	8.7	12.3
20.4	21.9	21.9	20.1	15.3	9.9	4.8	5.1	5.1	9.0	12.6	15.9
19.8	24.0	24.9	24.0	19.5	14.1	9.0	9.0	9.3	9.3	13.2	16.2
23.4	27.3	27.3	23.4	17.7	12.9	12.9	12.9	13.5	13.5	16.8	20.1
23.7	27.0	30.0	27.0	22.8	16.8	16.8	16.8	16.8	17.1	17.1	20.1
25.5	28.8	29.7	25.8	21.0	21.0	21.0	21.0	21.0	15.9	15.9	21.0
22.2	26.1	29.4	28.2	24.3	24.3	24.3	24.3	22.2	17.1	12.3	17.1
24.0	27.0	29.7	27.6	27.6	27.6	27.0	24.0	20.4	16.2	16.2	20.4
					24.6						

By folding out the small sphere these can be represented as is done in Table 7.1. Note that this reference solution consists of a primary activation initiated at the grid point with $t = 0$ and a secondary activation starting at $t = 12$ in the lower right-hand corner. The point activated last at $t = 30$, is in the middle left. The secondary activation simulates a phenomenon called a "breakthrough" in electrocardiology. Plots of isochrones on X for this reference solution were generated by orthogonal projection and linear interpolation and are given in Fig. 7.2 The left-hand plot is a frontal view, and the right-hand plot gives a rear view.

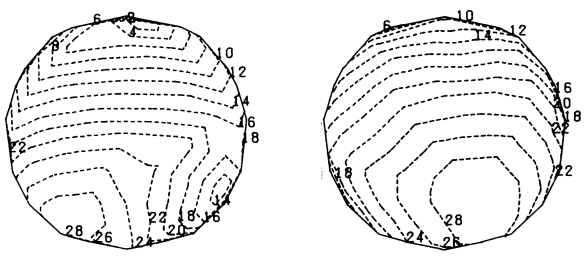


FIG. 7.2. *Reference solution, model problem.*

In this experiment (3.6) is applied for the polynomials $p_1(\tau) = (2\tau/T - 1)$, $p_2(t) = (2\tau/T - 1)^2$ and $p_3(\tau) = (2\tau/T - 1)^3$. After calculation of $v^{(1)}$, $v^{(2)}$ and $v^{(3)}$ from (3.11) these integrated potential distributions are randomly perturbed at the level of 1% of their respective maxima. The inverse solution is calculated from $v^{(1)}$ and $v^{(2)}$ by regularization with the Hessian on X. The necessary regularization parameters α_1 and α_2 are chosen by a crude minimization of the norm of the residual $r^{(3)} = v^{(3)} - \mathbf{A}p_3(\tau)$. Table 7.2 gives the value of α_1 and α_2 used in this minimization, the norm of the error in τ, relative to the norm of the reference solution, and the norm of $r^{(3)}$, relative to $\|v^{(3)}\|$. The calculated solution for the optimal values of α_1 and α_2 with respect to $\|r^{(3)}\|$ is given in Fig. 7.3 and Table 7.3.

Comparing the calculated solution with the reference solution shows that the activation pattern and the activation times are well represented in the results and that most difficulties lie with the starting points of the activation. These minima are slightly "flattened out" but are very well recognizable.

It must be stressed that this loose way of evaluating the calculated solution is justifiable since for medical purposes a qualitative correctness, in particular of the

TABLE 7.2

α_1	α_2	rel. err.	residual	no. of elements with 5, 10% err. etc.						
.100E-8	.100E-8	.131	.361082	15	48	20	11	3	1	
.250E-7	.100E-6	.465E-1	.552631E-1	61	35	2				
.250E-7	.250E-7	.463E-1	.376170E-1	58	38	2				
.250E-7	.251E-7	.463E-1	.376012E-1	58	38	2				
.100E-6	.100E-6	.421E-1	.566200E-1	66	30	2				
.100E-4	.100E-4	.804E-1	.226406	44	44	6	3	1		
.100E-2	.100E-2	.151	.335900	17	37	21	16	5	1	1

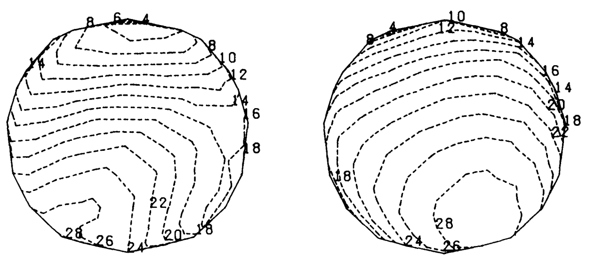


FIG. 7.3. *Calculated solution, model problem.*

TABLE 7.3
Calculated solution, model problem.

					10.2						
14.4	17.1	18.6	18.0	15.6	12.3	7.8	4.5	3.3	4.2	3.9	6.9
19.5	22.2	22.5	19.5	15.0	9.6	5.4	4.2	4.5	7.5	12.6	16.5
20.1	23.4	26.1	24.3	18.3	13.5	9.0	8.1	8.1	9.3	13.8	17.7
23.4	26.4	27.9	23.4	17.7	13.8	12.6	13.2	12.9	14.4	17.7	20.7
22.8	26.1	28.8	27.6	22.2	18.3	16.5	18.0	18.0	16.2	16.8	19.5
25.5	28.2	29.1	26.1	22.2	20.4	20.7	22.2	19.8	16.5	17.1	20.7
22.8	27.6	28.8	28.2	25.5	24.0	23.7	24.0	22.8	17.7	14.7	16.5
24.9	28.2	28.8	27.6	27.0	27.3	26.7	24.6	19.8	14.7	14.1	18.6
					25.2						

points of initial activation and of the overall pattern of propagation, would already be valuable.

Realistic geometry. Based on the cross sections and the activation times of the heart provided in Durrer et al. [1970] a discretization of the heart geometry was generated (van Oosterom [1978a]) with activation times as displayed in Fig. 7.4. This was placed in a realistic torso geometry as described in Oosterom [1978b]. The body is considered to be a homogeneous medium. As above, transformed body surface potentials $v^{(i)}(y)$ are calculated for a number of observation points y on Y, perturbed with 1% of the maximum signal and an inverse solution is calculated by the methods described in this paper. The discretization of the heart consists of 283 points and 562 triangles. The discretization of the torso consists of 320 triangles. For the inverse calculation the potentials at the 162 vertices of these triangles are used.

The results of the inverse calculation are given in Fig. 7.5. These show a good qualitative correspondence for the outer surface of the heart (epicardium) but a poor correspondence for the ventricular surfaces (endocardium).

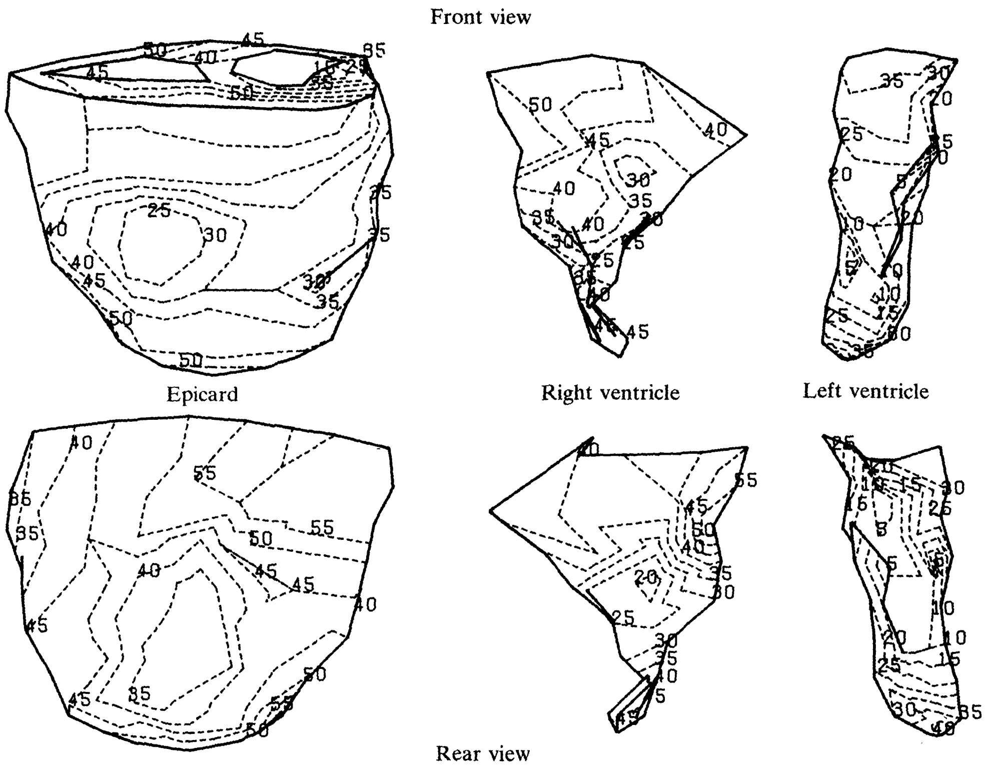


FIG. 7.4. *Reference solution.*

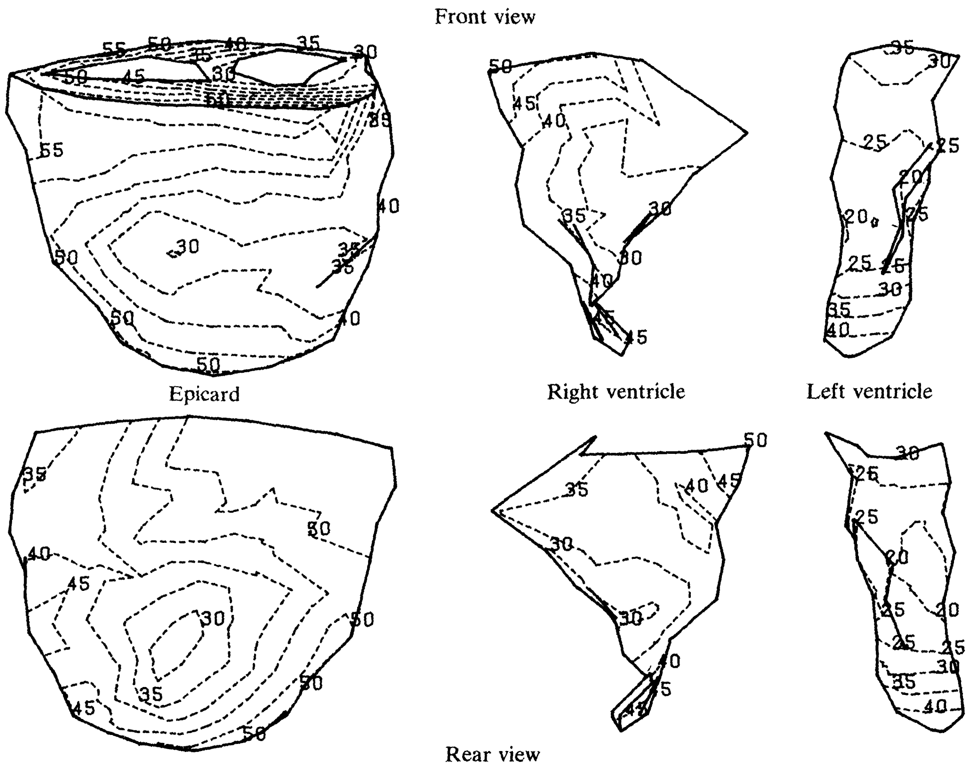


FIG. 7.5. *Calculated solution.*

Two remarks can be made. Firstly, it is not surprising that the activation times on the epicardial surface are better determined than those on the endocardial surface since the epicardial surface lies closer to the body surface and the activation as given is smoother on the epicardial surface. The reference solution which is used here is in many places rather irregular, especially on the ventricular surfaces. It is, however, the only measured set of human heart activation data available. Secondly, the calculations given assume a homogeneous body; the influence of inhomogeneities, as well as of the discretization still have to be investigated.

The results given are not expected to be the best that can be obtained in the present approach. The reason for this is that the application and the implementation of the method requires various choices some of which are known not to be optimal yet. These choices include the discretization of the heart surface, the relative weight of the various components in the regularizing operator, and the choice of the polynomials $p^{(i)}$ for calculating the inverse solution and for determining the regularization parameters. Investigations will proceed on these points.

8. Conclusion. By considering the activation time on the heart surface the inverse problem of electrocardiography can be stated (for the QRS complex) in a form which allows both effective regularization and relatively efficient calculations. A transformation is possible which eliminates the time dimension, so only a few weighted integrals of the potential at each observation point are needed for the calculation of the inverse solution. Numerical experiments, in the presence of 1% measurement errors, but without modeling errors, show that good results can thus be obtained for a model geometry consisting of two spheres approximating the heart and body surface respectively, and that promising results can be obtained for a realistic heart-torso geometry and a realistic activation.

9. Appendix. In this appendix the question of uniqueness of a double layer $S_h \subset X$ which induces a given potential distribution on the body surface Y is discussed.

Let V denote the (conducting) body, and the surface S_0 denote the boundary of V (so $Y \subset S_0$). It is assumed that V is composed of a finite number of subregions, namely i) the heart, which is not subdivided, bounded by the surface $S_1 = X$; ii) regions with a different conductivity (such as lungs and cavities filled with blood), bounded by surfaces $S_j, j > 1$; and (iii) the rest of V (cf. Fig. 9.1). The surfaces $S_j, j \geqq 0$ do not intersect. Further it is assumed that each subregion is isotropic and homogeneous (constant, scalar, nonzero conductivity), so for each $j \geqq 0$ σ_j^+ can be defined as the conductivity just outside S_j and σ_j^- as the conductivity just inside S_j.

The electrical potential u in V satisfies

$$\Delta u = 0 \quad \text{in } V \text{ except on } S_j, j = 0, 1, \cdots, \tag{9.1}$$

$$\frac{\partial u}{\partial n} = 0 \quad \text{on } S_0, \tag{9.2}$$

$$\sigma^+ \frac{\partial u}{\partial n^+} = -\sigma^- \frac{\partial u}{\partial n^-} \quad \text{on } S_j, \quad j > 0, \tag{9.3}$$

$$u^+ = u^- \quad \text{on } S_j, \quad j > 1, \tag{9.4}$$

$$u^+ - u^- = \text{double-layer strength on } S_1. \tag{9.5}$$

For physical reasons u must be bounded in V, therefore formula 5.4 from Payne [1975] can be applied. This yields that if u is given on an open subset Σ of S_0 then u is uniquely determined in any sphere S with its center outside S_0 and $S \cap S_j \subset \Sigma$ for all j (cf. Fig. 9.1).

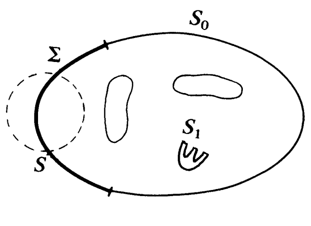


FIG. 9.1

Note that by giving u on Σ and $\partial u/\partial n$ on S_0 (9.2) the problem of determining u in V has the form of a Cauchy problem for the Laplace equation in $V \subset \mathbf{R}^3$.

Since u is uniquely determined, Σ can be deformed inside S into the volume conductor. The system with deformed surfaces S_0' and Σ_0' (cf. Fig. 9.2) also satisfies (9.1) to (9.5) and u is determined on Σ'. Continuing in this manner it follows that u is determined in V outside S_j, $j \geqq 1$.

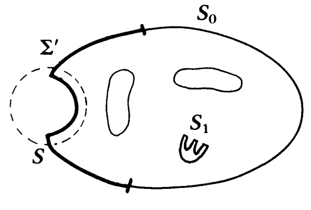


FIG. 9.2

The statement that u is determined in V outside the surfaces $S_j, j \geqq 1$, implies that u is determined in an outside neighborhood of at least one of the S_j. Now if such a surface S_j is not the heart surface ($j > 1$) then (9.3) and (9.4) give that u and $\partial u/\partial n$ are determined on the inner side of S_j. This yields a Cauchy problem for u inside S_j for which the same approach is valid (there may be surfaces S_k inside S_j). This gives unicity for u inside S_j and an analogous consistency criterion for existence of the overall solution.

If u is determined in an outside neighborhood of S_1 (as can be achieved in a finite number of applications of the reasoning given above) $u^+$and $\partial u/\partial n^+$ are determined on S_1. Equation (9.3), possibly with $\sigma_1^+ = \sigma_1^-$, yields that $\partial u/\partial n^-$ is determined on S_1. This gives a Neumann problem for u inside S_1 so u is determined up to a constant inside S_1, i.e. in the heart. Therefore u^- is determined on S_1 up to a constant, so (9.5) yields that the double-layer strength on S_1 is determined up to a constant. A condition for the existence of a solution S_h to the problem is that $u^+ - u^-$ takes only two values on $S_1 = X$. S_h will then be that part of X where $u^+ - u^-$ has the lower value of the two.

Acknowledgments. The discussions the author enjoyed having with Prof. A. van Oosterom were of great importance for the development of this research. Thanks are due to Jan Lankelma for a remark which led to the transformation applied in the time domain.

REFERENCES

A. C. L. Barnard, J. M. Duck, M. S. Lynn and W. P. Timlake, *The application of electromagnetic theory to electrocardiography* II, Biophys. J., 7 (1967), pp. 463–491.

P. Colli Franzone, B. Taccardi and C. Viganotti, *An approach to inverse calculation of epicardial potentials from body surface maps*, Adv. Cardiol., Karger Basel, 21 (1977), pp. 167–170.

P. Colli Franzone, L. Guerri, B. Taccardi and C. Viganotti, *The direct and inverse potential problem in electrocardiology. Numerical aspects of some regularization methods and application to data collected in isolated dog heart experiments*, Publ N. 222 dell'I.A.N.-C.N.R., Pavia 1979.

P. Colli Franzone and E. Magenes, *On the inverse potential problem of electrocardiology*, Calcolo, 16 (1979), pp 459–538.

R. Courant and D. Hilbert, *Methods of Mathematical Physics, vol.* II, Interscience, New York, 1961.

J. J. M. Cuppen, *The singular value decomposition in product form*, this Journal, 4 (1983), pp. 216–222.

J. J. M. Cuppen and A. van Oosterom, *Model studies with the inversely calculated isochrones of ventricular depolarization*, 1983, submitted.

D. Dotti, *A space-time solution of the inverse problem* in Body Surface Mapping of Cardiac Fields, Adv. Cardiol., Karger Basel, 10 (1974), pp. 231–238.

D. Durrer, R. Th. van Dam, G. E. Freud, M. J. Janse, F. L. Meyler and R. C. Arzbaecher, *Total excitation of the isolated human heart*, Circulation, 41 (1970), pp. 899–912.

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.

N. Köckler, *Parameterwahl und Fehlerabschätzung bei der regularisierten Lösung von inkorrekt gestellten Problemen*, Thesis, Mainz, 1974.

J. Locker and P. M. Prenter, *Regularization with differential operators* I: *general theory*, Math. Anal. Appl., 74 (1980), pp. 504–529.

M. S. Lynn and W. P. Timlake, *The numerical solution of singular integral equations of potential theory*, Numer. Math., 11 (1968), pp. 77–98.

———, *The use of multiple deflations in the numerical solution of singular systems of equations, with applications to potential theory*, SIAM J. Numer. Anal., 5 (1968), pp. 303–322.

———, *On the eigenvalues and eigenvectors of the integral equation formulation of the multi-interface Neumann problem*, J. Inst. Maths Applics, 6 (1970), pp. 391–399.

A. van Oosterom, *Cardiac potential distributions*, Thesis, Amsterdam University, 1978.

———, *Triangulating the human torso*, Comput. J., 21 (1978), pp. 253–258.

L. E. Payne, *Improperly Posed Problems in Partial Differential Equations,* CBMS Regional Conference Series in Applied Mathematics, 22, Society for Industrial and Applied Mathematics, Philadelphia, 1975.

R. Plonsey and D. Heppner, *Considerations of quasistationarity in electrophysiological systems*, Bull. Math. Biophys., 29 (1967), pp. 657–664.

Y. Salu, *Relating the multipole moments of the heart to activated parts of the epicardium and endocardium*, An. Biom. Engng., 6 (1978), pp. 492–505.

A. N. Tikhonov and V. Ja. Arsenin, *Solution of Ill-Posed Problems*, John Wiley, New York, 1977.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

DEFLATION TECHNIQUES AND BLOCK-ELIMINATION ALGORITHMS FOR SOLVING BORDERED SINGULAR SYSTEMS*

TONY F. CHAN†

Abstract. In numerical continuation methods for solving parametrized nonlinear systems, one often has to solve linear systems with matrices of the following form:

$$M=\begin{bmatrix} A & b \\ c^T & d \end{bmatrix}$$

where A may become singular but M is well conditioned. If A has special structures (e.g. sparseness, special data structure, special solver), then direct Gaussian elimination on M with pivoting will destroy the structures in A. An often used method that does exploit structures in A is the block-elimination (BE) algorithm which involves solving two systems with A for each system with M. In this paper, we show that the BE algorithm may become unstable and inaccurate when A is nearly singular. We then propose a stable variant which employs deflation techniques for solving the two systems with A. The deflation techniques can be viewed as working in coordinate systems orthogonal to the approximate null vectors of A, enabling an accurate representation of the solution to be computed. The extra work amounts to a few (e.g. 2) more backsolves with A. Backward error bounds and numerical results are presented.

Key words. singular systems, bordered systems, deflation, continuation methods

1. Introduction.

In this paper, we shall be concerned with computational techniques for solving linear systems of the form:

$$M\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} A & b \\ c^T & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} f \\ g \end{bmatrix} \tag{1}$$

where the $n\times n$ matrix A may become singular,[1] but the vectors b and c are chosen so that M remains nonsingular and well conditioned. The following lemma gives necessary and sufficient conditions for M to be nonsingular.

LEMMA 1. (a) *If A is nonsingular, then M is nonsingular if and only if*

$$d-c^TA^{-1}b\neq 0. \tag{2}$$

(b) *If A is singular and has a one-dimensional null space represented by a left null vector ψ and a right null vector ϕ, then M is nonsingular if and only if*

$$\psi^Tb\neq 0 \tag{3}$$

and

$$c^T\phi\neq 0. \tag{4}$$

Proof. Straightforward. A more general version can be found in [10]. The version given above is more suitable for our discussion.

Systems such as (1) arise, for example, in numerical continuation methods for solving parametrized nonlinear systems [10], [16], [17], [18] and in homotopy continuation methods for solving general nonlinear systems [2], [9], and the solution of such systems often constitutes the most time-consuming part of the overall computation. Since M is assumed to be well conditioned, the use of Gaussian elimination on M

* Received by the editors March 26, 1982, and in revised form October 20, 1982. This research was supported by the Department of Energy under contract DE-ACO2-81ER10996.

† Computer Science Department, Yale University, New Haven, Connecticut 06520.

[1] We shall assume the nullity of A to be one, which is the most common case in applications. The algorithms generalize easily to highest dimensional null spaces, but we shall not discuss that here.

with some form of pivoting is guaranteed to be stable. However, this approach is only suitable when n is small or when A is dense, since the whole matrix M has to be stored to allow for fill-ins. When A is large but has special structures (e.g. sparseness, band or profile structures) or when a special solver is available for A (e.g. fast elliptic solvers, sparse matrix solvers, band or profile solvers), or when a solver for A is needed for some other purposes anyway, it is natural to consider algorithms for solving systems with M which only involve solving systems with A. The following block-elimination algorithm has this desirable property:

ALGORITHM BE [4], [10].

Step 1: Solve

$$Av = b, \tag{5}$$

$$Aw = f. \tag{6}$$

Step 2: Compute

$$y = \frac{g - c^T w}{d - c^T v}. \tag{7}$$

Step 3: Compute

$$x = w - yv. \tag{8}$$

The work consists mainly of one factorization of A and two backsolves with the LU-factors of A. If there are many right-hand sides with the same matrix M, then the factorization of A and the vector v can be computed once, and the work reduces to only one backsolve for each right-hand side, which makes Algorithm BE extremely attractive in such cases. These situations arise, for example, in continuation methods where chord-Newton type methods are used [6] (this issue, pp. 135–148), [10], [15], [18].

Algorithm BE is well defined if A and M are nonsingular, because the denominator in (7) is nonzero by Lemma 1. However, in § 2, we show that Algorithm BE maybe unstable *numerically* when A is nearly singular and can produce completely inaccurate solutions (x, y) in those situations. The main source of instability is in Step 1 of Algorithm BE where the vectors v and w are computed inaccurately when A is nearly singular. In § 3, we review *implicit deflation* techniques developed in [3], [20] which can be used to compute accurate representations for the solutions v and w. These deflation techniques can be viewed as working in subspaces orthogonal to approximate null vectors of A and are implicit in the sense that they only involve solving systems with A. In § 4, we show how to use these *deflated decompositions* of v and w to obtain a stable variant of the BE algorithm. Further, we show that the new algorithm can be used to obtain a stable deflated decomposition of the solution (x, y) when M itself is nearly singular, for example, in applications to continuation around bifurcation points [1], [10], [17], [18]. We present a backward error analysis in § 5 that shows that the stability of the new algorithm is *independent* of the singularity of A. This means that in practice only one technique is needed to solve the system with the matrix M, independent of whether A is singular or not. Numerical tests demonstrating the accuracy and stability of the new algorithm will be presented in § 6.

Rheinboldt [19] has considered a related algorithm for solving the system (1) in the special case when A is *banded.* In his applications, the vector c is always equal to a unit vector; d is always equal to zero but the vector b is general. He considered splittings of M of the form $M = M_0 + uz^T$ where u and z are chosen so that M_0 is nonsingular and so that it is easy to obtain a factorization for M_0 (actually for a reduced banded matrix of smaller dimension than M_0). The solutions (x, y) can then be obtained through the use of the Sherman–Morrison formula [7], [19] by two backsolves with M_0. However, his approach requires explicitly working with the storage structure of A in order to obtain the factorization of the reduced matrix, whereas our approach is completely implicit in that it only requires the ability to solve systems with A. Moreover, our approach works for general b, c and d as long as they satisfy the conditions in Lemma 1 so that M is nonsingular. However, it should be pointed out that Rheinboldt's algorithm can be generalized to handle this more general case. A rank-two modification is required and one more backsolve is involved, which makes it about as efficient as Algorithm BE.

2. Stability of Algorithm BE. In this section, we show that Algorithm BE may produce inaccurate solutions when A is nearly singular even though M is well conditioned.

LEMMA 2. *If we use vectors $\tilde{v}$ and $\tilde{w}$ satisfying $A\tilde{v}-b=r_1$ and $A\tilde{w}-f=r_2$ in Steps* 2 *and* 3 *of Algorithm* BE, *then the solutions $(\tilde{x}, \tilde{y})$ satisfy*:

$$A\tilde{x}+\tilde{y}b-f=r_2-\tilde{y}r_1, \qquad c^T\tilde{x}+\tilde{y}d-g=0. \tag{9}$$

Proof. Straightforward.

In other words, the computed solutions $(\tilde{x}, \tilde{y})$ always satisfy the last equation of (1) independent of their accuracy, whereas the residual for the first n equations in (1) depends on the accuracy of $\tilde{v}$ and $\tilde{w}$.

Next we show that when A is nearly singular, r_1 and r_2 are generally large. From (3), we see that $b\notin \text{Range}(A)$, and therefore the computed $\tilde{v}$ will be large when A is nearly singular. Similarly, $\tilde{w}$ will also be large unless $f\in\text{Range}(A)$. From standard round-off error analysis [7], it follows that the residuals r_1 and r_2 will be large. Moreover, these residuals will not cancel out in (9). Specifically, consider solving the system $Az=p$ where A is nearly singular but p is not consistent with A. Let $\{\sigma_1, \cdots, \sigma_n\}$ be the singular values of A arranged in descending magnitude, $\{u_1, \cdots, u_n\}$ be the corresponding left singular vectors and $\{v_1, \cdots, v_n\}$ be the corresponding right singular vectors. Then the solution z can be written as:

$$z=\sum_{i=1}^{n} c_i v_i, \tag{10}$$

where

$$c_i=\left(\frac{u_i^T p}{\sigma_i}\right). \tag{11}$$

Since p is not consistent with A, $u_n^T p$ is not small, and therefore the last term in the sum in (10) will be large if σ_n is small. In finite precision arithmetic, this last term will dominate the rest of the sum and the *computed* $\tilde{z}$ will have an expansion similar to (10) but where the first $(n-1)$ coefficients $\tilde{c}_i$ are inaccurate. Since the residual for $\tilde{z}$ can be written as

$$r(\tilde{z})\equiv A\tilde{z}-p=\sum_{i=1}^{n}(c_i-\tilde{c}_i)u_i, \tag{12}$$

we see that the residual corresponding to the first $(n-1)$ terms in (12) will be large. Furthermore, this part of the residual depends on the particular values of the coefficients c_i's. Therefore, we cannot expect these parts of the residuals r_1 and r_2 in

(9) to cancel out. It follows from Lemma 2 that the computed solutions $(\tilde{x}, \tilde{y})$ will give large residuals for the system (1). Since M is assumed to be well conditioned, it follows that the errors in $(\tilde{x}, \tilde{y})$ will generally be large. As we shall see in § 6, this actually happens numerically.

3. Deflation. The implicit deflation techniques that we are going to discuss here were developed in [3], [20]. They can be viewed as techniques for separating the subspaces corresponding to σ_n from its orthogonal complement, and only need the ability to solve systems with A. Referring to (10), they correspond to computing the first $n-1$ terms *separately* from the last term so that the accuracy will not be adversely affected by the last term. Basically, the idea is to "project" the right-hand side p, using approximate singular vectors, such that for the projected p, $u_i^T p$ is as small as σ_n. Explicit deflation techniques [3], [11], [12], [13], [14], which achieve the same goal by working with parts of the LU-factorization of A explicitly, can also be used when they are applicable.

The ideas mentioned above can be made more rigorous, as is done in [3]. We shall now give a brief description of the main results. We consider computing a *deflated decomposition* of the solution z of the system $Az=p$ of the form:

$$z = z_D + \left(\frac{c_p}{\delta}\right)\phi, \tag{13}$$

where z_D is the *unique* solution to the following system:

$$A_S z_D \equiv SAz_D = Rp, \tag{14}$$

$$Nz_D = z_D, \tag{15}$$

where $A_S \equiv SA$ is a singular matrix "close" to A with a left null vector ψ and a right null vector ϕ. The matrices R and N are defined in terms of ψ and ϕ and are chosen so that the system (14) and (15) is consistent and has a unique solution that remains bounded as A tends to being exactly singular. The coefficient c_p is a scalar that depends on p, and the scalar δ tends to zero as A tends to being singular. In [3], we considered two different classes of such deflated decompositions, one corresponding to choosing A_S to be the nearest singular matrix to A in the Frobenius norm, and the other corresponding to choosing A_S to be a singular matrix obtained by perturbing some elements of A by amounts bounded by the smallest pivot, say ε, in an LU-factorization of A. For the LU-based approach to be successful, we need to make the assumption that $\varepsilon = O(\sigma_n)$, which is definitely not valid in general, but which we show empirically here and in [3] and theoretically in [5] to be valid in practice.

In order to carry out the deflation techniques, we need to compute some approximate null vectors and define some projectors based on them.

DEFINITION 3. (a) Let $P_u = 1 - uu^T$, with $\|u\| = 1$, be the *orthogonal projector*.[2]

(b) For any u with $u_j \neq 0$ and $1 \leqq j \leqq n$, define the *oblique projector*:

$$E_u^j = I - \frac{u e_j^T}{u_j}. \tag{16}$$

(c) Let ψ_{SV}, with $\|\psi_{SV}\| = 1$, be an approximation to the left singular vector corresponding to the smallest singular value σ_n of A. Define $\phi_{SV} = \sigma A^{-1}\psi_{SV}$, where $\sigma = 1/\|A^{-1}\psi_{SV}\|$.

[2] All norms used are the Euclidean norms.

(d) Let k be the index of the smallest pivot in the LU-factorization of A. Define:

(17) $$\psi_{LU} = \alpha A^{-T} e_k, \quad \text{where } \alpha = \frac{1}{\|A^{-T} e_k\|}.$$

(e) Define:

(18) $$\phi_P = \gamma A^{-1} \psi_{LU}, \quad \text{where } \gamma = \frac{1}{\|A^{-1} \psi_{LU}\|}.$$

(f) Let j be chosen so that[3] $|(\psi_{LU})_j| = O(1)$. Define:

(19) $$\phi_E = \beta A^{-1} e_j, \quad \text{where } \beta = \frac{1}{\|A^{-1} e_j\|}.$$

In Definition 3, all the ϕ's and ψ's are approximations to the right and left null vectors of A, respectively. The computation of ψ_{LU}, ϕ_{SV}, ϕ_E and ϕ_P costs one back-substitution each. For ψ_{SV}, a variant of the inverse power method can be used, which is fast when the smallest singular value of A is well isolated. It is well known that the inverse power method may have convergence difficulty for clustered eigenvalues.

In [3], we discussed eight deflated decompositions based on the LU-factorization and three based on the singular value decomposition (SVD). We shall only use a subset of these here since there exist simple relationships among the deflated solutions, and the ones used here are representative and can be computed most efficiently. The algorithms developed here can be applied to the other deflated decompositions as well. These deflations are denoted by z_S, z_E and z_P, standing for deflation by the SVD (S), by LU-factorization with an oblique projector (E), and by LU-factorization with the orthogonal projector (P). The projectors (orthogonal or oblique) are used to project the right-hand side p onto the range space of the nearby singular matrix.

We are now ready to define the specific deflated decompositions by specifying the matrices S, R and N in (14) and (15) and the coefficients c_p and δ in (13) in terms of these approximate null vectors and projectors. In Table 3.1, we give the expressions for ψ, ϕ, S, R, N and δ for each of the three deflations (each corresponding to a row). Note that $R = S$ for all three deflations. It was shown in [3] that for the z_S deflation, the coefficient c_p reduces to $\psi_{SV}^T p$ if ψ_{SV} is exactly equal to the left singular vector corresponding to σ_n. The more general form in the table has to be used when the

TABLE 3.1
Deflated decompositions for $Az = p$. $z = z_D + (c_p/\delta)\phi$. k is the index of the smallest pivot.

z_D	ψ	ϕ	$S = R$	N	δ	c_p
z_S	ψ_{SV}	ϕ_{SV}	$P_{\psi_{SV}}$	$P_{\phi_{SV}}$	σ	$\psi_{SV}^T(p - Az_S)$
z_E	ψ_{LU}	ϕ_E	$(E^j_{\psi_{LU}})^T$	$E^k_{\phi_E}$	$\beta(\psi_{LU})_j$	$\psi_{LU}^T p$
z_P	ψ_{LU}	ϕ_P	$P_{\psi_{LU}}$	$E^k_{\phi_P}$	γ	$\psi_{LU}^T p$

[3] We shall use the notation $(u)_k$ to denote the kth component of the vector u.

inverse iteration fails to converge or converges to the wrong singular value. We also gave the following *iterative improvement* algorithm for stably computing the deflated solutions, derived from one first proposed by Stewart [20]:

```
ALGORITHM IIA. Given $Az = p$, compute the deflated solution $z_D$ satisfying (14) and (15).
Start with an initial guess $z_D$ such that $Nz_D = z_D$ (e.g. $z_D = 0$).
Loop until convergence.
    Step 1. Form $r = Rp - SAz_D$.
    Step 2. Form $d = A^{-1}r$.
    Step 3. $z_D \leftarrow -z_D + Nd$.
```

In [3], we analyzed the convergence and stability of Algorithm IIA, and showed that for the z_S, z_E and z_P deflations, the above iteration converges in exactly *one* step for any initial guess satisfying the stated condition. By taking zero as initial guess for z_D, we then arrived at the following *noniterative* algorithm for computing z_D:

```
ALGORITHM NIA.
Step 1. $r = Rp$.
Step 2. $d = A^{-1}r$.
Step 3. $z_D = Nd$.
```

Since the vector r in Step 1 of both Algorithms IIA and NIA are consistent with A_S which is "close" to A, the first $n-1$ coefficients in the singular vector expansion of z_D will not be dominated by the last coefficient, and therefore they can be computed accurately. Therefore, the deflated decomposition (13) can be viewed as an accurate *representation* of the solution z. It was shown in [3] that the scalar σ (Definition 3c, above) tends to the smallest singular value σ_n as A tends to being singular and that α, β and γ are $O(\varepsilon)$. Thus we see from Table 3.1 that δ goes to zero as A becomes singular for all three deflations. If we were to compute z in (13) directly, then the last term would dominate the first and the accuracy in z_D, and consequently z, would be lost.

4. Deflated block elimination. Recall that the reason Algorithm BE becomes unstable when A is nearly singular is that the computed $\tilde{v}$ and $\tilde{w}$ have large relative errors in the subspace orthogonal to ϕ. The deflation techniques discussed in § 3 overcome this problem by computing z_D with low relative errors in the same subspace. In this section, we show how to use the deflation techniques to obtain a stable variant of Algorithm BE.

The main idea is to compute the deflated decompositions of v and w instead of computing them directly from (5) and (6). By using Algorithm NIA (or Algorithm IIA if necessary), we can obtain the following deflated decompositions for v and w:

$$v = v_D + \left(\frac{c_b}{\delta}\right)\phi \tag{20}$$

and

$$w = w_D + \left(\frac{c_f}{\delta}\right)\phi. \tag{21}$$

When A is nearly singular, one wants to avoid actually carrying out the division by δ and the addition in the above formulas, because the second terms will be large and will overwhelm the first terms, causing a loss of accuracy of the solution (x, y). It turns out that it is not difficult to derive a stable variant of Algorithm BE that uses the representations of v and w in (20) and (21), but which does not involve adding large vectors to the accurate deflated solutions v_D and w_D.

LEMMA 4. *If v and w are represented by* (20) *and* (21), *then the solutions* (x, y) *of* (1) *can be expressed as*:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} w_D \\ 0 \end{bmatrix} + \frac{1}{D}\begin{bmatrix} h_3\phi - h_4 v_D \\ h_4 \end{bmatrix}, \tag{22}$$

where

$$h_1 = g - c^T w_D, \qquad h_3 = h_1 c_b - h_2 c_f,$$
$$h_2 = d - c^T v_D, \qquad h_4 = (c^T\phi)c_f - \delta h_1,$$
$$D = (c^T\phi)c_b - \delta h_2.$$

Moreover, D is nonzero if M is nonsingular.

Proof. The proof that (x, y) given by (22) satisfy (1) can be derived by direct substitution. We shall only show that D is nonzero if M in nonsingular. When A is singular $\delta = 0$, and from Lemma 1 both $c^T\phi$ and $\psi^T b$ are nonzero. For the z_E and z_P deflations, $c_b = \psi^T b$. For the z_S deflation, $c_b = \psi^T(b - Az_S) = \psi^T b$, because ψ is then a left null vector of A. Therefore, for all three deflations, we have $D = (c^T\phi)(\psi^T b) \neq 0$. When A is not singular $(\delta \neq 0)$, it can be easily shown from (5) and (20) that $D = -\delta(d - c^T A^{-1} b)$, and therefore by Lemma 1, $D \neq 0$.

We shall call the algorithm represented by (22) Algorithm DBE.

The expressions defining h_1, h_2, h_3 and h_4 are all stable formulas in the sense that no large vectors are involved. Note also from (22) that the vectors v_D and w_D generally have as much weight in the solutions (x, y) as the vector ϕ, and therefore the accuracy of (x, y) depends directly on the accuracy of ϕ, v_D and w_D. Moreover, one can also see from the formulas involved that when δ is small (i.e. when A is nearly singular), it is enough to control the *absolute error in δ*. This is important because in general we cannot hope to be able to do better than this in computing δ. Furthermore, the stability of Algorithm DBE depends only on the singularity of M in the sense that $|D|$ is as small as M is singular, and is independent of the singularity of A. This means that in practice only one algorithm is needed for dealing with solving (1). We shall prove all the above assertions rigorously in the next section.

There is a reason why we use the particular form (22) for expressing the solutions (x, y). The reason is that as M itself tends to being singular, D tends to zero, and (22) automatically becomes a deflated decomposition of (x, y). Moreover, the vector multiplying $(1/D)$ in (22) is then a null vector of M. In practice, one can monitor the size of D and avoid performing the division by D and the addition to the first vector when $|D|$ becomes too small. The form (22) will then remain an accurate representation of the solutions (x, y) and can be used in further computations just as we have done here for the deflated decompositions of v and w. Such situations arise, for example, in applying continuation methods around bifurcation points [10], where A is singular but $\psi^T b = 0$ and therefore M is also singular.

Compared to Algorithm BE, the extra overhead involved in Algorithm DBE, with Algorithm NIA for computing the deflated solutions, amounts to a few more backsolves for computing the two null vectors and storage for them. For the SVD-based deflated decompositions, the number of extra backsolves depends on the convergence of the inverse iteration. If the inverse iteration fails to converge, then an extra copy of A has to be stored for computing c_p. For the LU-based deflated decompositions, only two more backsolves are needed and the matrix A does not have to be stored. When there are multiple right-hand sides for the M-system, the null vectors and the deflated decomposition for v have to be computed only once, and the cost per extra right-hand side is then no more than that of Algorithm BE.

5. Error analysis. We prove in this section that Algorithm DBE is stable by exhibiting a backward round-off error bound for (x, y). We shall show that the (x, y) produced by Algorithm DBE give small residual to the system (1) if the computed solution is not large. If M is well conditioned, then it follows that the errors in (x, y) are also small. If M is ill conditioned, a small residual is all we can hope for.

We shall use $\tilde{}$ to denote computed quantities. To simplify the error analysis, we shall make the following assumption: *We shall assume that the only source of errors in Algorithm* DBE *is in solving systems with A (i.e. errors in $\tilde{v}_D$, $\tilde{w}_D$, $\tilde{\psi}$ and $\tilde{\phi}$) and that no round-off errors are made in carrying out the operations in* (22). This is a reasonable assumption because (22) represents a stable algorithm. The actual round-off errors made in (22) can be bounded by a small constant times the machine precision times quantities like $\tilde{v}_D$, $\tilde{w}_D$, $\tilde{\delta}$, b, c, d, f, g. These can all be absorbed into our final bounds.

The following lemma shows that the residual for (1) depends on the accuracy of the computed $\tilde{v}_D$, $\tilde{w}_D$, $\tilde{\psi}$, $\tilde{\phi}$ and $\tilde{\delta}$.

LEMMA 5. *If the computed $\tilde{v}_D$, $\tilde{w}_D$, $\tilde{\psi}$, $\tilde{\phi}$ and $\tilde{\delta}$ used in Algorithm* DBE *satisfy*

$$SA\tilde{v}_D - Rb = r_b, \quad SA\tilde{w}_D - Rf = r_f, \quad A\tilde{\phi} = \tilde{\delta}\tilde{\psi} = r_\psi,$$

then the computed solutions $(\tilde{x}, \tilde{y})$ satisfy

$$A\tilde{x} + \tilde{y}b - f = r_f - \tilde{y}r_b + \left(\frac{\tilde{h}_3}{\tilde{D}}\right) r_\psi,$$

$$c^T\tilde{x} + \tilde{y}d - g = 0. \tag{23}$$

Proof. The proof is rather straightforward, albeit a bit tedious, and follows from a direct substitution of (22) into the expression for the residuals.

The actual residuals r_b, r_f and r_ψ depend on the way $\tilde{v}_D$, $\tilde{w}_D$ and $\tilde{\phi}$ are computed. Next, we consider the use of Gaussian elimination with some form of pivoting.

LEMMA 6. *If we use Gaussian elimination with some form of pivoting for solving systems with A, then the residual r_ψ satisfy*:

$$r_\psi \leqq p(n)\varepsilon_M \|A\|, \tag{24}$$

where ε_M is the machine precision and $p(n)$ is a polynomial in n that depends on the form of pivoting used. Further, if Algorithm NIA <u>*without*</u> *Step* 3 *is used to compute $\tilde{v}_D$ and $\tilde{w}_D$, then*

$$r_b \leqq p(n)\varepsilon_M \|\tilde{v}_D\| \, \|A\|, \tag{25}$$

$$r_f \leqq p(n)\varepsilon_M \|\tilde{w}_D\| \, \|A\|. \tag{26}$$

Proof. The standard backward error bounds for Gaussian elimination with pivoting (e.g. [7, p. 181]) for solving a general linear system $Az = p$ has the form

$$\|p - A\tilde{z}\| \leqq p(n)\varepsilon_M \|\tilde{z}\| \, \|A\|, \tag{27}$$

where $k(n)$ is a slowly growing polynomial that depends on the pivoting strategy. The bound for r_ψ follows directly from this. For the other two bounds, observe that the vector d produced in Step 2 of Algorithm NIA when solving for v_D, satisfies $Ad = Rb + B$, where the vector B satisfies a bound similar to (27). Therefore, the residual in Step 2, denoted by r_d, is given by $r_d = SAd - Rb = S(Rb + B) - Rb = (S - I)Rb + SB$. It can easily be shown from Table 3.1 that $(S - I)R = 0$ for all the choices of S and R. For $S = P_{\psi_{SV}}$ or $P_{\psi_{LU}}$, $\|SB\| \leqq \|B\|$. For $S = (E^j_{\psi_{LU}})^T$, $\|SB\| \leqq (1 + 1/(\psi_{LU})_j)\|B\|$. Since $(\psi_{LU})_j$ is chosen to be $O(1)$, we have $\|SB\| \leqq C\|B\|$ where C is a small constant. The bounds for r_b and r_f follow since C can be absorbed into $k(n)$. This completes the proof of Lemma 6.

Notice that we have used Algorithm NIA without Step 3 in order to obtain the above bounds. If Step 3 of Algorithm NIA is used, then we can show, by using the bound (24), that $SANd - SAd = B_1$, where $\|B_1\| \leqq 2\varepsilon_M p_1(n)\|A\|\,\|d\|$. It follows that $\|r_z\| = \|(SANd - SAd) + (SAd - Rb)\| \leqq \varepsilon_M p_2(n)\|A\|\,\|d\|$, where again the constants are absorbed into $p_1(n)$ and $p_2(n)$. However, we cannot in general obtain a bound in terms of $\|z\|$ rather than $\|d\|$. If ψ is not close to the left null vector of A in the z_S deflation, or if $\varepsilon \gg O(\sigma_n)$ in the LU-based deflation, then $\|z\|$ can be much smaller than $\|d\|$. However, we believe that in practice the bounds for r_b and r_f will still be satisfied if either Algorithm NIA or Algorithm IIA is used for computing $\tilde{v}_D$ and $\tilde{w}_D$.

Using Lemmas 5 and 6, we obtain our main result:

THEOREM 7. *If the computed quantities $\tilde{v}_D$, $\tilde{w}_D$, $\tilde{\phi}$, $\tilde{\psi}$ and $\tilde{\delta}$ satisfy the bounds in Lemma* 6, *and no further round-off errors are made in Algorithm* DBE *in the sense of Assumption* 1, *then the computed solutions $(\tilde{x}, \tilde{y})$ satisfy*:

$$\|A\tilde{x} + \tilde{y}b - f\| \leqq \{2(\|\tilde{w}_D\| + |\tilde{y}|\,\|\tilde{v}_D\|) + \|\tilde{x}\|\}p(n)\|A\|\varepsilon_M,$$
$$\|c^T\tilde{x} + \tilde{y}d - g\| = 0.$$

Proof. The proof follows directly from Lemmas 5 and 6, and by observing that, from (22),

$$\left|\frac{\tilde{h}_3}{\tilde{D}}\right| \leqq \|\tilde{x}\| + \|\tilde{w}_D\| + |\tilde{y}|\,\|\tilde{v}_D\|.$$

From Theorem 7, we see that the key to the success of Algorithm DBE is to control the size of $\tilde{v}_D$, $\tilde{w}_D$, $\tilde{y}$ and $\tilde{x}$. When A is not nearly singular, this is no problem. When A is nearly singular, Algorithm DBE achieves this by the deflation techniques. Thus, the stability of Algorithm DBE is *independent* of the singularity of A. In practice, this means that, with only a little overhead, the same algorithm can be used to solve systems with M accurately independent of whether A is nearly singular or not.

6. Numerical results. We performed some numerical tests to verify the accuracy and stability of Algorithm DBE with the various deflation techniques. We considered two classes of matrices for A:

A_1: $(1 - 2uu^T)\,\mathrm{Diag}\,(\sigma_n, n-1, n-2, \cdots, 1)(1 - 2vv^T)$ where u and v are chosen randomly and scaled to have norm 1, and σ_n varies from 1 to 10^{-8}.

A_2: $T - \lambda_{\min}(T)I - \sigma_n I$ where $T = \mathrm{Tridiagonal}\,(1, -2, 1)$ and σ_n again varies from 1 to 10^{-8}.

Note that σ_n is equal to the smallest singular value of A_1 and A_2. For A_1, the smallest singular value has multiplicity 2 when $\sigma_n = 1$. The dimension n of A is chosen to be 19, so that the dimension of M is 20. The vectors b and c are chosen randomly in (0, 1) and d is set to 1. The solutions (x, y) are also generated randomly and the corresponding right-hand sides (f, g) are then computed by multiplying (x, y) by M.

In the inverse iteration for determining the approximate singular vector ψ_{SV}, we start with the vector with all components equal to 1 and always take 5 iterations. When A is highly singular, one or two iterations is enough for full accuracy. However, we have some convergence difficulties when A does not have a well-isolated small singular value.

For the deflation, we use Algorithm NIA *without* the correction step 3. This is the case covered by the error bounds in § 5. We have also performed the same tests using Algorithms NIA and IIA with qualitatively the same results. Moreover, Algorithm IIA always converged in one iteration. For comparison, we will also use Algorithm BE without deflation and direct Gaussian elimination (GE) on M itself. All LU-factorizations are performed by the routine SGECO of LINPACK [8] which uses the row partial pivoting strategy. The computations were performed on a DEC-20 with 27-bit mantissas corresponding to a machine precision of about $.4\times 10^{-8}$.

The first set of tests is to see how well the LU-factorization identifies a small pivot ε that is $O(\sigma_n)$. The computed ε, its position k, the computed σ and the reciprocal of the estimated condition number of M (the parameter RCOND in routine SGECO) are given in Table 6.1 for A_1 and A_2. We see that, at least for these two classes of matrices, the smallest pivot ε is indeed roughly $O(\sigma_n)$. Moreover, the smallest pivot always appears at the (n, n)th position. Note also that, when σ_n is well isolated, the computed σ is rather accurate and has low *absolute* error. However, when the smallest singular value is not well isolated, the inverse iteration is not successful at all. This is

TABLE 6.1
Table of ε, k and RCOND *as a function of $\sigma_n = 10^{-I}$.*

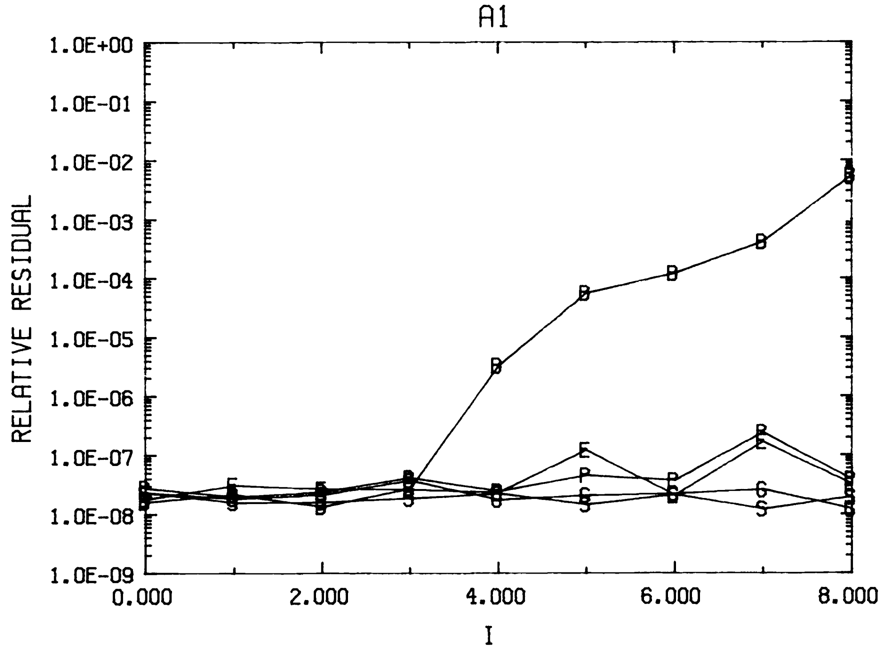
A_1	I	computed σ	ε	k	RCOND
	0	0.1000007E+01	0.1183648E+01	19	0.1065610E−01*
	1	0.9999996E−01	0.9543458E+00	19	0.1099506E−01
	2	0.1000000E−01	−0.6528494E−01	19	0.1687405E−02
	3	0.1000016E−02	0.7956855E−01	19	0.1218959E−03
	4	0.9999102E−04	−0.1555381E−02	19	0.2518016E−02
	5	0.1001859E−04	−0.9973533E−04	19	0.1758764E−02
	6	0.9929115E−06	0.4390627E−04	19	0.6221786E−02
	7	0.9942711E−07	−0.7852563E−04	19	0.2147673E−02
	8	0.1375734E−07	0.8866191E−06	19	0.6173249E−03
A_2	I	computed σ	ε	k	RCOND
	0	0.6739568E−01	−0.7148705E+00	19	0.9330396E−02†
	1	0.9806986E−01	0.7308936E+00	19	0.4726424E−02‡
	2	0.1000000E−01	0.5541958E+00	19	0.1737031E−03
	3	0.1000000E−02	0.6328177E−01	19	0.3318758E−02
	4	0.1000006E−03	0.6386045E−02	19	0.3319462E−02
	5	0.9999954E−05	0.6391779E−03	19	0.1049820E−02
	6	0.9980513E−06	0.6379932E−04	19	0.2220898E−03
	7	0.1047228E−06	0.6694347E−05	19	0.5903182E−02
	8	0.1462737E−07	0.9350479E−06	19	0.7973481E−03

* Singular vectors had not converged after 5 iterations.
† Singular value converged to the wrong value.
‡ Inverse iteration has not converged after 5 iterations.

especially true for A_2 because its lowest eigenvalues are rather close to each other. Since this only occurs when A is well conditioned, the accuracy of Algorithm DBE with the SVD-based deflations is not affected.

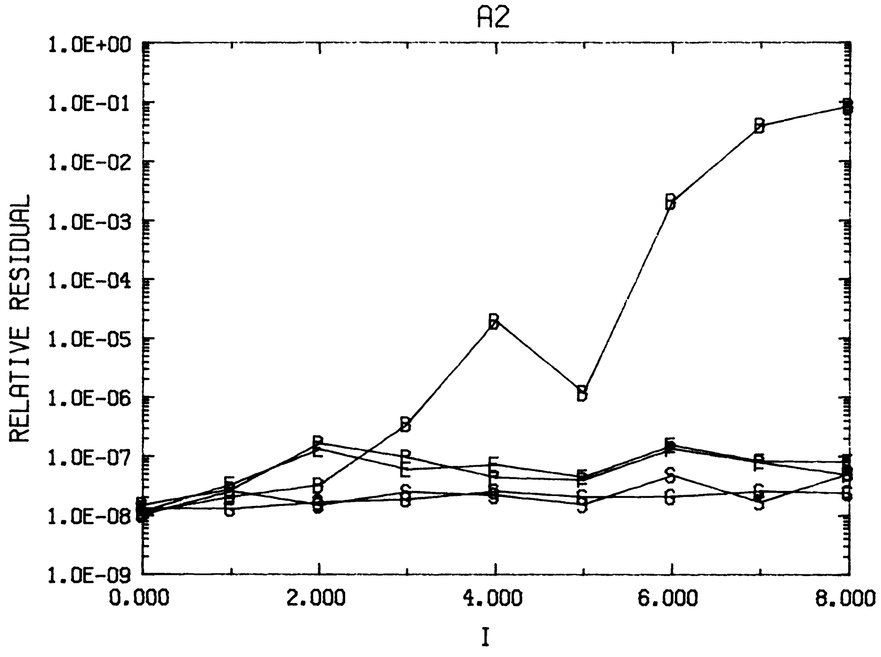
The relative residuals of the M equation (normalized by the right-hand sides of the M system) as computed by the various algorithms are displayed in Figs. 6.1 and 6.2. The relative errors in the solution (x, y) are displayed in Figs. 6.3 and 6.4. It is seen that Algorithm DBE with any of the three deflation techniques achieves about the same accuracy as does GE on M, whereas Algorithm BE with no deflation loses accuracy as A tends to being singular. Note also that we do not try to control the singularity of M itself and it can become rather ill conditioned. However, by the backward error bounds in § 5, the *relative residuals* to the M equation should still be small in these situations because the solutions (x, y) are small by construction. The *relative errors*, however, will be large if M is ill conditioned. This is reflected in the numerical results, too.

Theoretically, the (S) deflation is more robust as it is based on the SVD, whereas the (E) and the (P) deflations depend on the identification of a small pivot in the LU-factorization. However, based on the numerical results presented here, it seems that in practice the (E) and (P) deflations are almost as accurate as the (S) deflation. Since the latter two are both cheaper and simpler to implement (e.g. no inverse iteration, no need to store A), we recommend using them in practice, possibly with a check on the size of the smallest pivot as a safety precaution.



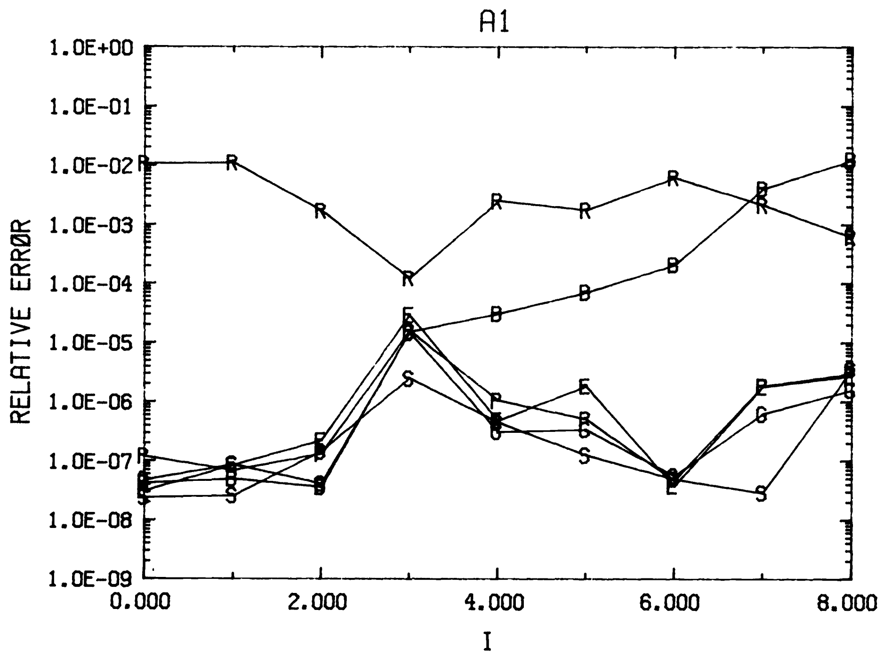
G	G.E. ØN M
B	B.E. WITH NØ DEFLATIØN
E	B.E. WITH E-DEFLATIØN
P	B.E. WITH P-DEFLATIØN
S	B.E. WITH S-DEFLATIØN

FIG. 6.1. *Relative residual as a function of* $\sigma_n = 10^{-I}$ *for* A_1.



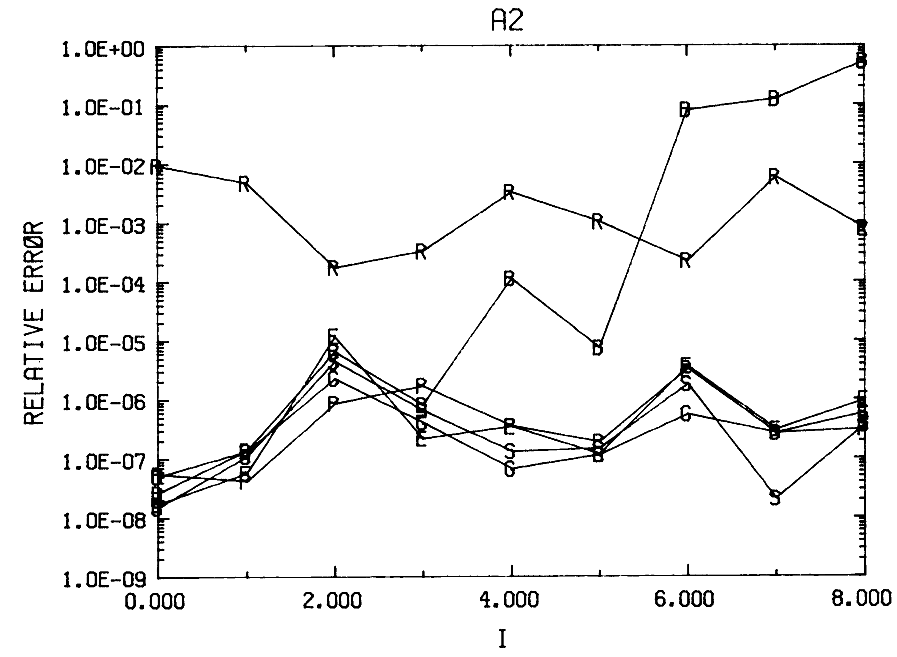
G G.E. ØN M P B.E. WITH P-DEFLATIØN
B B.E. WITH NØ DEFLATIØN S B.E. WITH S-DEFLATIØN
E B.E. WITH E-DEFLATIØN

FIG. 6.2. *Relative residual as a function of* $\sigma_n = 10^{-1}$ *for* A_2.



G G.E. ØN M P B.E. WITH P-DEFLATIØN
B B.E. WITH NØ DEFLATIØN S B.E. WITH S-DEFLATIØN
E B.E. WITH E-DEFLATIØN R RCØND ØF M (LINPACK)

FIG. 6.3. *Relative error as a function of* $\sigma_n = 10^{-1}$ *for* A_1.



G	G.E. ØN M	P	B.E. WITH P-DEFLATIØN
B	B.E. WITH NØ DEFLATIØN	S	B.E. WITH S-DEFLATIØN
E	B.E. WITH E-DEFLATIØN	R	RCØND ØF M (LINPACK)

FIG. 6.4. *Relative error as a function of* $\sigma_n = 10^{-1}$ *for* A_2.

7. Conclusion. In this paper, we have presented a stable version of the block-elimination algorithm for solving bordered nearly singular systems that fully exploits structures in A by only calling A^{-1} as a black box for solving linear systems. To summarize briefly, Algorithm DBE goes like this.

ALGORITHM DBE.

Step 1. Select one of the three deflated decompositions in Table 3.1.

Step 2. Compute the corresponding pair of approximate left and right null vectors and the constant δ using Definition 3.

Step 3. Compute the deflated solutions v_D and w_D in (20) and (21) by using Algorithm NIA with the appropriate right-hand sides b and f respectively. Each of the two solutions involves solving one system with A with a "projected" right-hand side.

Step 4. Compute the corresponding coefficients c_b and c_f by using Table 3.1.

Step 5. Compute the solution (x, y) from the formulas in Lemma 4.

From the backward error bounds derived in § 5, we see that the stability of Algorithm DBE derives from *using deflation techniques to control the size of the computed quantities in the algorithm.* Numerical experiments reported in § 6 confirm this. Thus, Algorithm DBE solves the dilemma of trying to exploit structures in A by treating A^{-1} as a black box while being able to control the numerical stability caused by the near singularity of A.

REFERENCES

[1] J. P. ABBOTT, *An efficient algorithm for the determination of certain bifurcation points*, J. Comput. Appl. Math., 4 (1978), pp. 19–27.

[2] E. ALLGOWER AND K. GEORG, *Simplicial and continuation methods for approximating fixed points and solutions to systems of equations*, SIAM Rev., 22 (1980), pp. 28–85.

[3] T. F. CHAN, *Deflated decompositions of solutions of nearly singular systems*, Technical Report 225, Computer Science Department, Yale Univ., New Haven, CT, 1982; SIAM J. Numer. Anal., 21 (1984), to appear.

[4] T. F. CHAN AND H. B. KELLER, *Arclength continuation and multigrid techniques for nonlinear eigenvalue problems*, this Journal, 3 (1982), pp. 173–194.

[5] T. F. CHAN, *On the existence and computation of LU-factorizations with small pivots*, Tech. Rep. 227, Computer Science Dept., Yale Univ., New Haven, CT, 1982.

[6] ———, *Newton-like pseudo-arclength methods for computing simple turning points*, Tech. Rep. 233, Computer Science Dept., Yale Univ., New Haven, CT, 1982; this Journal, this issue, pp. 135–148.

[7] G. DAHLQUIST AND A. BJORCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[8] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[9] G. B. GARCIA AND W. I. ZANGWILL, *Pathways to Solutions, Fixed Points and Equilibria*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[10] 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.

[11] ———, *Singular systems, inverse iteration and least squares*, unpublished manuscript, Applied Mathematics Dept., California Institute of Technology, Pasadena, CA, 1978.

[12] ———, *Numerical continuation methods*, Short Course Lecture Notes, Center for Applied Mathematics, National Bureau of Standards, Washington, DC, 1981.

[13] ———, *Practical procedures in path following near limit points*, in Computing Methods in Applied Sciences and Engineering, Glowinski and Lions, eds., North-Holland, Amsterdam, 1982.

[14] ———, *The bordering algorithm and path following near singular points of higher nullity*, this Journal, 4 (1983), pp. 573–582.

[15] R. G. MELHAM AND W. C. RHEINBOLDT, *A comparison of methods for determining turning points of nonlinear equations*, Computing, 29 (1982), pp. 201–226.

[16] H. D. MITTELMANN AND H. WEBER, *Numerical methods for bifurcation problems—A survey and classification*, in Bifurcation Problems and their Numerical Solution, H. D. Mittelmann and H. Weber, eds., Workshop on Bifurcation Problems and their Numerical Solution, January 15–17, 1980, Birkhauser, Dortmund, 1980, pp. 1–45.

[17] W. C. RHEINBOLDT, *Numerical methods for a class of finite dimensional bifurcation problems*, SIAM J. Numer. Anal., 15 (1978), pp. 1–11.

[18] ———, *Solution fields of nonlinear equations and continuation methods*, SIAM J. Numer. Anal., 17 (1980), pp. 221–237.

[19] ———, *Numerical analysis of continuation methods for nonlinear structural problems*, Comput. & Structures, 13 (1981), pp. 103–113.

[20] G. W. STEWART, *On the implicit deflation of nearly singular systems of linear equations*, this Journal, 2 (1981), pp. 136–140.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

NEWTON-LIKE PSEUDO-ARCLENGTH METHODS FOR COMPUTING SIMPLE TURNING POINTS*

TONY F. CHAN†

Abstract. We present a new method for computing *simple turning points* of nonlinear equations of the form $G(u, \lambda) = 0$ which is based on applying Newton's method to the characterization $d\lambda(\sigma)/d\sigma = 0$, where σ is a pseudo-arclength parameter used in a continuation method for following the solution paths. The method is quadratically convergent and needs only one starting point on the solution path. Second derivatives of G (or difference approximations of them) have to be computed but the method is relatively insensitive to their values and they also give rise to a more accurate second order predictor in the continuation method. We present a chord-Newton variant for improving the efficiency of the algorithm which requires only one factorization of a Jacobian matrix. We also present a damped-Newton variant for improving the robustness and the global convergence of the algorithm. Results of numerical experiments on two standard nonlinear elliptic problems of Simpson's [SIAM J. Numer. Anal., 12 (1975), pp. 439–451] show that the new algorithm compares favorably with the best of the existing methods in terms of efficiency and robustness.

Key words. turning points, arclength continuation, Newton's method, nonlinear systems

1. Introduction.

Many problems in computational physics can be formulated as *nonlinear eigenvalue problems of* the form

$$G(u, \lambda) = 0, \tag{1}$$

where $u \in B$ (a real Banach space), $\lambda \in R$, and G is a continuously differentiable operator mapping $B \times R$ into B. Usually, u represents the "solution" to the physical problem (e.g. flow field, structural displacement) and λ is related to a physical parameter (e.g. Reynolds number, load on a structure). Often, one is interested in the dependence of the solution $u(\lambda)$ on the parameter λ, i.e. in tracing the *solution branches* $[u(\lambda), \lambda]$ of (1). When the operator G is nonlinear in u and λ, this is usually accomplished numerically by some version of Newton's method applied to (1) for a fixed value of λ, which makes use of the Jacobian matrix $G_u(u, \lambda)$. However, the solution branches often possess very interesting but complicated nonlinear bifurcation behavior, among which are existence of multiple solutions and singular points (where $G_u(u, \lambda)$ is singular) known as *turning points* (where the solution branch bends back on itself) and *bifurcation points* (where two or more solution branches cross). Straightforward application of Newton's method to (1) encounters difficulties near these singular points. To overcome these difficulties, some kind of path following continuation method [2], [11], [15], [19] is usually employed. These continuation methods are designed to trace past turning points and can be modified to switch branches at bifurcation points.

In many applications, in addition to tracing the solution branches, one is also interested in locating the singular points themselves, because they are often related to the stability of the solution. Due to their special physical significance, many algorithms have been proposed for determining these singular points accurately. In this paper, we shall only deal with the determination of *simple* turning points, which can be characterized as points on the solution curve where

$$\begin{aligned} &G_u \text{ is singular, with a one-dimensional null space,} \\ &G_u \text{ boundedly invertible on its range,} \end{aligned} \tag{2}$$

* Received by the editors April 29, 1982, and in revised form October 20, 1982. This research was supported by the Department of Energy under contract DE-ACO2-81ER10996.

† Computer Science Department, Yale University, Box 2158, Yale Station, New Haven, Connecticut 06520.

and

$$G_\lambda \notin \text{Range}\,(G_u). \tag{3}$$

For an excellent survey of existing methods for computing turning points, we refer the reader to the report by Melhem and Rheinboldt [14] in which they compare the performances of algorithms proposed by Abbott [1], Moore and Spence [16], Seydel [24], Paumier [17], Pönisch and Schwetlick [18], Rheinboldt [20], [21], Schwetlick [23] and Simpson [25]. These methods can be classified into two general classes. The first consists of local iterative algorithms based on an inflated system consisting of (1) augmented by a characterization similar to (2), constructed so that the turning point is a unique and isolated solution of the inflated system. The other class of algorithms consists of methods based on a path tracing continuation method by successively using it to compute points on the solution curve that approach the turning point. These algorithms can further be categorized by whether one or more points on the solution curve are needed as initial guess and whether second derivatives of G are needed. In Table 1.1, we tabulated the properties of the best methods as found by Melhem and Rheinboldt, judged by an overall measure of excellency in terms of a combination of efficiency, robustness and generality. For the purpose of comparison, we have included the method that we are proposing in this paper.

TABLE 1.1
Properties of some methods for computing turning points.

	Initial points	Rate of convergence	Needs 2nd derivative?	Class of method	Characterization of turning point
Abbott	1	2	yes/no	I	$\dot\lambda(s)=0$
Moore & Spence	1	2	yes/no	I	$G_u w = 0, w \neq 0$
Pönisch & Schwetlick	1	2	yes/no	I	$\dot\lambda(s)=0$
Rheinboldt	2	1.618	no	C	$\dot\lambda(s)=0$
Schwetlick	2	1.618	no	C	$\lambda(\tau)=$ extremum
Chan	1	2	yes/no	C	$\lambda'(\sigma)=0$

Notation. I = inflated system, C = continuation, s is the arclength parameter, τ and σ are arclength-like parameters, "yes/no" means that the basic method requires second derivatives of G but difference approximations are also used by the authors.

As can be seen from the table, the method that we are proposing is quadratically convergent, needs only one initial guess on the solution curve, and is based on an underlying continuation method for branch tracing. Methods based on continuation have certain desirable properties. First, they can build upon the curve tracing capabilities that are already in the continuation procedure. For example, very often the same linear equation solver can be used without having to refactor any Jacobian matrix. Second, they naturally provide more details of the solution curve around the turning points. Lastly, as we shall demonstrate later, requiring the iterates to lie on the solution curve tends to make the algorithms more robust than methods based on using augmented systems. The property of requiring only one initial point is also desirable because most continuation methods have to be slowed down near turning points and it may be relatively expensive to obtain two points on the solution curve where $\dot\lambda$ changes sign, as is needed by some methods. We shall review briefly the formulation of typical continuation methods in § 2.

Many of the methods in Table 1.1 use the characterization $\dot{\lambda}(s)=0$ for turning points. Our method is based on an alternative characterization of simple turning points, namely, that

$$\lambda'(\sigma) \equiv \frac{d\lambda(\sigma)}{d\sigma} = 0 \tag{4}$$

where σ is the *pseudo-arclength parameter* used in the continuation method. This characterization has been suggested by Keller [12], although he only considered secant methods and no numerical results were given. Our method is based on applying Newton's method to (4). We show in § 3 that the second derivatives $[u''(\sigma), \lambda''(\sigma)]$ can be computed rather inexpensively if the second derivatives of G are available. We note that none of the methods based on continuation cited in [14] is quadratically convergent. This is obviously because the authors tried to avoid computing second derivatives, which in some applications are very difficult to obtain. However, since a function evaluation in these methods involves an inner Newton iteration which could be costly, we believe that in many applications where the second derivatives (or approximations of them) are available, the use of a method with a faster convergence rate may be beneficial. As we shall show in § 4.1, the availability of second derivatives also leads to a much more accurate predictor in the underlying continuation method, which in turn improves the efficiency of the overall algorithm.

Another desirable property for an algorithm is that of requiring only a solver for G_u (rather than a matrix derived from G_u) since such a solver may already be available in the application discipline and it can also exploit special solution techniques (e.g. fast elliptic solvers). We show in § 4.2 how this can be arranged in our algorithm. In § 5, we present a chord-Newton variant for improving the efficiency and a damped-Newton variant for improving the robustness and global convergence of the basic algorithm. In § 6, we discuss briefly the work and storage requirements of the new algorithm, which are comparable to most of the existing methods. Extensive numerical experiments have been performed on applying the algorithm and its variants to two standard nonlinear elliptic problems of Simpson's [25] and the results are presented in § 7. They demonstrate that the new algorithm is both efficient and robust and compares favorably with the best of the existing methods on these two problems.

2. Pseudo-arclength continuation.

In this section, we review the essential features of some common path-following continuation methods.

The key idea is to *parametrize* the solutions $[u(\sigma), \lambda(\sigma)]$ in terms of a new parameter σ that approximates the arclength parameter s, instead of parametrizing $u(\lambda)$ in terms of the natural parameter λ. This is usually achieved by augmenting the equation (1) by an auxiliary equation that approximates the arclength condition:

$$\|\dot{u}(s)\|^2 + |\dot{\lambda}(s)|^2 = 1, \tag{5}$$

to give an inflated system with unknowns $u(\sigma)$ *and* $\lambda(\sigma)$:

$$G(u(\sigma), \lambda(\sigma)) = 0, \qquad N(u(\sigma), \lambda(\sigma), \sigma) = 0. \tag{6}$$

Instead of solving for $u(\lambda)$ for a given value of λ, we solve for $u(\sigma)$ and $\lambda(\sigma)$ for a given value of σ. Newton's method and its variants are usually used to solve (6), in which case we need to solve linear systems with the following inflated matrix:

$$M = \begin{bmatrix} G_u & G_\lambda \\ N_u & N_\lambda \end{bmatrix}. \tag{7}$$

The auxiliary function N is constructed so that the matrix M is *nonsingular* on the solution branch, *even near or at simple turning points*. Thus, Newton's method encounters no difficulties with this inflated system and quadratic convergence is achievable.

Another major component of a continuation method is the computation of the unit tangent $[\dot{u}(s), \dot{\lambda}(s)]$ to the solution curve at a point $[u, \lambda]$ on the solution curve, which can be computed relatively inexpensively from its definition,

$$G_u(u,\lambda)\dot{u} + G_\lambda(u,\lambda)\dot{\lambda} = 0, \qquad \|\dot{u}\|^2 + |\dot{\lambda}|^2 = 1, \tag{8}$$

by solving only *one* linear system with G_u. The system (8) determines $[\dot{u}, \dot{\lambda}]$ up to a directional orientation, which can be fixed by some convention. The tangent is usually used in a first order predictor to obtain an initial guess for the Newton iteration applied to the system (6).

We summarize the essential features in the following general algorithm.

ALGORITHM PAC$[u_0, \lambda_0, \sigma, u(\sigma), \lambda(\sigma)]$

Pseudo-arclength continuation. Given $[u_0, \lambda_0]$ on the solution curve, and a step length σ (for step length algorithms, see [8]), compute the new solution $[u(\sigma), \lambda(\sigma)]$ satisfying (6).

1. Compute the unit tangent $[\dot{u}_0, \dot{\lambda}_0]$ at $[u_0, \lambda_0]$ by (8).
2. Compute the predicted solution $[u_p, \lambda_p]$ given by

$$u_p = u_0 + \sigma\dot{u}_0, \qquad \lambda_p = \lambda_0 + \sigma\dot{\lambda}_0. \tag{9}$$

3. Use $[u_p, \lambda_p]$ as initial guess in a Newton-like iteration for solving the system (6) to obtain $[u(\sigma), \lambda(\sigma)]$.

A few typical N's that have been used in the literature are:

1. $N_1(u, \lambda, \sigma) \equiv \dot{u}_0^T(u - u_0) + \dot{\lambda}_0(\lambda - \lambda_0) - \sigma$ (introduced by Keller [11]),

2. $N_2(u, \lambda, \sigma) \equiv e_i^T(y - y_0) - \sigma$, where $y = (u, \lambda)^T$, e_i is the ith unit vector and the index i is chosen so that the matrix M is as well-conditioned as possible (introduced by Abbott [1], Kubicek [13] and Rheinboldt [19]).

3. Newton on $\lambda'(\sigma) = 0$. We shall consider only *simple* turning points where the nullity of G_u is one. Consider the situation where we have an approximation $[u_0, \lambda_0]$ to a turning point $[u_*, \lambda_*]$. The method that we are proposing works by *estimating the step length σ* to use in applying one step of the pseudo-arclength continuation procedure PAC$[u_0, \lambda_0, \sigma, u(\sigma), \lambda(\sigma)]$ so that $u(\sigma) \equiv u_*$ and $\lambda(\sigma) \equiv \lambda_*$. The basis for estimating σ is derived from the following characterization of simple turning points:

DEFINITION 1. Define $\lambda'(\sigma) \equiv d\lambda(\sigma)/d\sigma$ and $u'(\sigma) \equiv du(\sigma)/d\sigma$.

THEOREM 2. *Assume $N_\sigma(\sigma) \neq 0$. Then $\lambda'(\sigma) = 0$ if and only if $[u(\sigma), \lambda(\sigma)] \equiv [u_*, \lambda_*]$.*

Proof. First note, by differentiating (6) by σ, that $[u'(\sigma), \lambda'(\sigma)]$ satisfies:

$$\begin{pmatrix} G_u(\sigma) & G_\lambda(\sigma) \\ N_u(\sigma) & N_\lambda(\sigma) \end{pmatrix} \begin{pmatrix} u'(\sigma) \\ \lambda'(\sigma) \end{pmatrix} = \begin{pmatrix} 0 \\ -N_\sigma(\sigma) \end{pmatrix}, \tag{10}$$

where the coefficient matrix in (10) is *nonsingular* by construction. Thus $[u'(\sigma), \lambda(\sigma)]$ is well defined even near or at a turning point. Now first assume $\lambda'(\sigma) = 0$. The second equation in (10) implies that $N_u u' = -N_\sigma \neq 0$. Thus u' is nontrivial. The first equation in (10) reduces to $G_u u' = 0$. Since u' is nontrivial, G_u must be singular. Next assume that $[u(\sigma), \lambda(\sigma)] \equiv [u_*, \lambda_*]$. Then by (2) $G_u(\sigma)$ is singular. If $\lambda'(\sigma) \neq 0$, then the first equation in (10) implies that $G_\lambda(\sigma) \in \text{Range}\,(G_u(\sigma))$, which contradicts (3). □

We note that both N_1 and N_2 satisfy the hypothesis of Theorem 2.

Theorem 2 provides a basis for estimating the step length σ because it reduces the problem to one of finding a root of $\lambda'(\sigma)=0$. Our method is based on applying Newton's method for doing this, for which we need to compute $\lambda''(\sigma)$. It is not surprising that this requires computing the second derivatives of G. By differentiating (10) with respect to σ, we obtain the following system for computing $[u''(\sigma), \lambda''(\sigma)]$:

$$\begin{pmatrix} G_u(\sigma) & G_\lambda(\sigma) \\ N_u(\sigma) & N_\lambda(\sigma) \end{pmatrix} \begin{pmatrix} u''(\sigma) \\ \lambda''(\sigma) \end{pmatrix} \tag{11}$$
$$= \begin{pmatrix} -(G_{uu}(\sigma)u'(\sigma)u'(\sigma)+2G_{u\lambda}(\sigma)u'(\sigma)\lambda'(\sigma)+G_{\lambda\lambda}(\sigma)\lambda'(\sigma)\lambda'(\sigma)) \\ -(N_{uu}(\sigma)u'(\sigma)u'(\sigma)+2N_{u\lambda}(\sigma)u'(\sigma)\lambda'(\sigma)+N_{\lambda\lambda}(\sigma)\lambda'(\sigma)\lambda'(\sigma))-N_{\sigma\sigma}(\sigma) \end{pmatrix}.$$

We note that the systems governing $[u', \lambda']$ and $[u'', \lambda'']$ have identical coefficient matrices, which are almost exactly the same as that used in the last step of the Newton iteration in the pseudo-arclength procedure. Thus, the same factorization of these matrices can be used to compute $[u', \lambda']$ and $[u'', \lambda'']$ and they can be obtained essentially free.

We outline the basic version of our method in algorithmic form.

ALGORITHM NTP$[u_0, \lambda_0, u_*, \lambda_*]$.

Newton's method for locating turning points. Starting with an initial guess $[u_0, \lambda_0]$ on the solution curve, compute an approximation $[u_*, \lambda_*]$ to a turning point.

```
Initialize $\sigma = 0$.
Loop until convergence:
    1. Compute $[u'(\sigma), \lambda'(\sigma)]$ and $[u''(\sigma), \lambda''(\sigma)]$ by (10) and (11).
    2. Compute the *change* in the step length $\delta\sigma = -\lambda'(\sigma)/\lambda''(\sigma)$.
    3. Update the new step length $\sigma \Leftarrow \sigma + \delta\sigma$.
    4. Call PAC $[u_0, \lambda_0, \sigma, u(\sigma), \lambda(\sigma)]$.
Set $u_* \Leftarrow u(\sigma)$, $\lambda_* \Leftarrow \lambda(\sigma)$.
```

Note that in Step 4 of Algorithm NTP, we use u_0 and λ_0 instead of the most current iterate. The reason is that the local parametrizations (N_1 or N_2) usually depend on $[u_0, \lambda_0]$ and Algorithm NTP is trying to find a pseudo-arclength step σ within this local parametrization that will correspond to the turning point. Using the most current iterate in Step 4 here will change the local parametrization.

Under mild conditions on the smoothness of G, it is not difficult to prove local quadratic convergence for Algorithm NTP for simple *quadratic* (i.e. $\lambda''(\sigma)\neq 0$) turning points. It can be shown from evaluating the first equation in (11) at the turning point that the condition $\lambda''(\sigma)\neq 0$ is equivalent to $G_{uu}(\sigma)u'(\sigma)u'(\sigma)\notin$ Range $(G_u(\sigma))$. We shall not pursue the convergence analysis here.

We note that $[u', \lambda']$ is not equal to the unit tangent $[\dot{u}, \dot{\lambda}]$ but is a scaled version of it. As can be seen from Table 1.1, the characterization $\dot{\lambda}(s)=0$ for turning points have been used by many authors but Keller [12] seems to be the only one who has considered the use of the pseudo-arclength parameter σ of a continuation procedure, together with the characterization $\lambda'(\sigma)=0$, in the context of an algorithm for finding turning points. A system similar to (11) has been derived by Pönisch and Schwetlick [18] but their method is not based on a continuation procedure.

4. Implementation. In this section, we discuss some of the implementation details for Algorithm NTP. We address three issues: the construction of a more accurate predictor, algorithms for solving linear systems with the inflated matrix of the form (7), and the use of difference approximations for second derivatives.

4.1. Second order predictor. Note that in Algorithm PAC, the solution $[u(\sigma), \lambda(\sigma)]$ is uniquely defined by the parametrization N and the step length σ, and one can use any reasonable predictor in Step 2 instead of the commonly used one given in (9). We want to emphasize that the unit tangent at $[u_0, \lambda_0]$ is usually used both to define the local parametrization and to obtain a predictor solution although these two processes can be separated. For example, instead of the first order predictor used in Step 2 of Algorithm PAC, we can use the following more accurate predictor:

$$\begin{aligned} u_p &= u(\sigma)+(\delta\sigma)u'(\sigma)+(\delta\sigma)^2u''(\sigma)/2, \\ \lambda_p &= \lambda(\sigma)+(\delta\sigma)\lambda'(\sigma)+(\delta\sigma)^2\lambda''(\sigma)/2. \end{aligned} \tag{12}$$

This new predictor is more accurate for two reasons. First, the current approximation $[u(\sigma), \lambda(\sigma)]$ to the turning point is used instead of $[u_0, \lambda_0]$. It corresponds to using the Taylor series expansions of $[u(\sigma+\delta\sigma), \lambda(\sigma+\delta\sigma)]$ around $[u(\sigma), \lambda(\sigma)]$ instead of around $[u_0, \lambda_0]$ which is (9). Second, it has *second* order accuracy. Note that this more accurate predictor is essentially *free*, since all the quantities in (12) have already been computed in Algorithm NTP before the call to Algorithm PAC. Its higher accuracy greatly reduces the cost of the inner Newton iteration in Step 4 of Algorithm PAC.

4.2. Block elimination and deflation techniques. All the linear systems that arise in our algorithm are of the form:

$$M\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A & b \\ c^T & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \tag{13}$$

where the $n\times n$ matrix A may become singular near a turning point but the vectors b and c are chosen so that M remains nonsingular and well conditioned. The algorithm that we have chosen to use for solving the linear systems of the form (13) is the following *block-elimination* algorithm:

ALGORITHM BE [5], [11].

1. Solve

$$Av = b, \tag{14}$$

$$Aw = f. \tag{15}$$

2. Compute

$$y = \frac{g - c^T w}{d - c^T v}. \tag{16}$$

3. Compute

$$x = w - yv. \tag{17}$$

The work consists mainly of one factorization of A and two backsolves with the LU-factors of A. If there are many right-hand sides with the same matrix M, then the factorization of A and the vector v need only be computed once, and the work reduces to only one backsolve for each right-hand side, which makes Algorithm BE extremely attractive in such cases. These situations arise in the chord-Newton variant of Algorithm NTP (see § 5.1). Note also that only a solver for A is needed, and therefore any special structures (e.g. sparsity, bandedness, special data structures) in A can be exploited and special solvers for A can be used (e.g. fast direct solvers, multi-grid solvers). However, as we have shown in [3], Algorithm BE may be unstable numerically when A is nearly singular, as is the case in the present application. The main source of instability is in Step 1 of Algorithm BE where the vectors v and w are computed inaccurately when A is nearly singular. In [3], we proposed using *implicit deflation* techniques developed in [4], [26] to compute accurate representations for the solutions v and w. These deflation techniques can be viewed as working in subspaces orthogonal to approximate null vectors of A and are implicit in the sense that they only involve solving systems with A. We then use these *deflated decompositions* of v and w to obtain a stable variant of the BE algorithm, which we called Algorithm DBE. The only overhead involved for performing the deflation in this algorithm is the computation of two approximate left and right null vectors for A. These can be obtained either by an inverse power method or by a technique based on the existence of a small pivot in the LU-factorization of A [6]. In any case, the extra work amounts to only a few backsolves, which is usually negligible in comparison with the work involved in computing the factorization. We refer the details of Algorithm DBE to [3], where we also presented a backward error analysis that shows that it is numerically stable.

We have assumed that direct elimination methods are used for solving the linear systems that arise. For the use of iterative methods, which might be more attractive for large and sparse problems, we refer the reader to [7]. For another method for solving the inflated systems, see [22].

4.3. Difference approximations for second derivatives. In the context of algorithms for computing turning points, for *any* method to achieve *quadratic* convergence, second derivatives of G are required in general. Unfortunately, in many applications, second derivatives are difficult to compute or not available at all. For this reason, many algorithms avoid using second derivatives explicitly. In the specific context of using the characterization $\lambda'(\sigma)=0$ for locating turning points, there are at least three ways to achieve this. The first is to use a secant-like method for finding a zero of $\lambda'(\sigma)$, as is the case in the methods of Keller [12] and Rheinboldt [20], [21]. However, the convergence rate will then not be quadratic. In order to retain quadratic convergence, at least approximately, we choose to work with a Newton-like method similar to Algorithm NTP. Within this context, there are at least two ways to avoid second derivatives. This first is to use a difference approximation for $\lambda''(\sigma)$, by evaluating $\lambda'(\sigma)$ at two adjacent points. This is essentially the approach taken by the method of Abbott in Table 1.1. Note that each evaluation of $\lambda'(\sigma)$ may be rather costly as it involves calling Algorithm PAC with a few different values of σ and consequently involves solving a few linear systems with the inflated matrix M inside the Newton iteration in Step 3 of Algorithm PAC. The last approach, which is the one we have adopted in this paper, is to use a difference approximation for computing the second derivatives of G. Note that these appear only on the right-hand side of (11) rather than in the coefficient matrix, as is the case in the method of Moore and Spence [16]. We believe that this property of the algorithm leads to better numerical stability. For the numerical experiments in this paper, we have used a simple centered difference approximation. For example, $G_{\lambda\lambda}(\sigma)$ is approximated by

$$G_{\lambda\lambda}(\sigma)\approx(G_\lambda(u(\sigma),\lambda(\sigma)+\varepsilon)-G_\lambda(u(\sigma),\lambda(\sigma)-\varepsilon))/2\varepsilon. \tag{18}$$

In practice, one can use better techniques; see for example [10].

5. Variants. In this section, we present variants of the basic Algorithm NTP designed to improve its efficiency and robustness.

5.1. Chord-Newton variant. With direct methods for solving the linear systems, the most expensive part of the computation is usually in factoring the Jacobian matrix G_u. Therefore, one can save a great deal of computation by reusing the factors of a nearby matrix. There are two Newton iterations involved in Algorithm NTP, both of which allow chord-Newton variants. For the outer Newton iteration, it does not pay to use the chord-Newton variant because the coefficient matrix governing $[u''(\sigma), \lambda''(\sigma)]$ in (11) is the *same* as the one governing $[u'(\sigma), \lambda'(\sigma)]$ in (10). Thus, the second derivatives $[u''(\sigma), \lambda''(\sigma)]$ can be computed very inexpensively by performing only one back-substitution. For the inner iteration in Algorithm PAC, however, one can obtain a chord-Newton variant by using the same LU-factors of a Jacobian matrix G_u in all the Newton steps, for example, by using the LU-factors of the matrix $G_u(\sigma)$ used in computing $\lambda'(\sigma)$ and $\lambda''(\sigma)$ in Step 1 of Algorithm NTP.

We note that with Algorithm DBE, G_λ can be updated in $M(\sigma)$ in each step of the chord-Newton iteration without incurring a factorization of G_u, which in general gives a better approximation for $M(\sigma)$ than if an old copy of G_λ, say $G_\lambda(0)$, were used. However, if we choose not to update G_λ, then the vector v in Algorithm DBE can be computed once for all and each solve with M then involves only one, rather than two, solves with G_u. Therefore, if G_λ does not change very much around the turning point, it might be more efficient *not* to update G_λ at every step in the chord-Newton iteration. This is the strategy that we have used in our numerical experiments.

The above chord-Newton variant requires one factorization of $G_u(\sigma)$ *per outer iteration step*. This is similar to the treatment of the chord version of Rheinboldt's method. However, if the initial guess $[u_0, \lambda_0]$ is close enough to the turning point, one can reduce the work further. We can factor $G_u(0)$ once only at the initial guess $[u_0, \lambda_0]$ and reuse these factors of $G_u(u_0, \lambda_0)$ in all subsequent iteration steps, *both outer and inner*. However, for the convergence of the outer iteration, we have to ensure that the function values in the *outer* iteration, i.e. $\lambda'(\sigma)$, are evaluated accurately. Since the system (10) governing $[u'(\sigma), \lambda'(\sigma)]$ is linear, we can use the following *iterative improvement* algorithm for doing this:

Starting with an initial guess for $t(\sigma)$, iterate until convergence:

$$t(\sigma) \Leftarrow t(\sigma) + M(0)^{-1}(r(\sigma) - M(\sigma)t(\sigma)), \tag{19}$$

where

$$M(0) = \begin{pmatrix} G_u(0) & G_\lambda(0) \\ N_u(\sigma) & N_\lambda(\sigma) \end{pmatrix}, \qquad M(\sigma) = \begin{pmatrix} G_u(\sigma) & G_\lambda(\sigma) \\ N_u(\sigma) & N_\lambda(\sigma) \end{pmatrix},$$

$$r(\sigma) = (0, -N_\sigma(\sigma))^T, \qquad t(\sigma) = (u'(\sigma), \lambda'(\sigma))^T.$$

A similar algorithm can be applied to the $[u''(\sigma), \lambda''(\sigma)]$ system (11) as well. Moreover, since the second derivatives $[u''(\sigma), \lambda''(\sigma)]$ are available from the last outer Newton iteration, one can use a first order predictor for $[u'(\sigma), \lambda'(\sigma)]$ in (19), similar to the one used in (12). Furthermore, although we have not pursued it here, the iteration (19) can also be accelerated, for example, by a conjugate gradient type method. No predictor for $[u'', \lambda'']$ is available, however, unless one stores previous values and uses extrapolation.

5.2. Damped-Newton variant. It is well known that Newton's method is only locally convergent. In the context of Algorithm NTP, if the initial guess $[u_0, \lambda_0]$ is far away from a turning point, then the step $\delta\sigma$ generated at Step 2 of Algorithm NTP may be so large that either there is no solution for $[u(\sigma), \lambda(\sigma)]$ or the inner Newton iteration in Algorithm PAC fails to converge. To improve the robustness of the algorithm, we consider the use of a *damped*-Newton variant. Since our algorithm is based on a continuation method, this can be arranged naturally by replacing Steps 3 and 4 of Algorithm NTP by

```
3'. **If** the previous $\delta\sigma$, say $\delta\sigma_p$, was *damped* **and** $|\delta\sigma| > |\delta\sigma_p|$ **then**
    $\delta\sigma \Leftarrow \text{sign}(\delta\sigma)\,|\delta\sigma_p|$
4'. **Repeat until** convergence:
    4.1 $\sigma \Leftarrow \sigma + \delta\sigma$.
    4.2 **Call** PAC $[u_0, \lambda_0, \sigma, u(\sigma), \lambda(\sigma)]$.
    4.3 **If** no convergence in PAC, **then** $\delta\sigma \Leftarrow \delta\sigma/\gamma$.
```

Here γ is a scalar damping factor (we used $\gamma = 2$). To reduce the work wasted in the damped steps, we declare that Algorithm PAC has "failed" if either the number of iterations exceeds a maximum (we used a value of 5) or if the norm of the residuals $\|G\|$ is not less than that at the previous iteration. This is similar to the treatment in [19], [20]. Since the methods are based on a continuation procedure, it can be shown [19] that the loop in Step (4') above will terminate with a *nonzero* step length $\delta\sigma$. For methods based on inflated systems, no natural damped-Newton variant exists.

6. Work and storage. In Table 6.1, we summarize the work and storage requirements of Algorithm NTP and its chord variant, assuming that a direct factorization-solve method is used for solving the linear systems. The storage for the damped Newton version is the same as that for the nondamped version. Its work is more difficult to estimate since it depends on exactly how the damped steps are taken. We have therefore not included it in the table.

TABLE 6.1
Work and storage per step.

	Work/step					Storage	
			Evaluations				
Algor.	Factors	Solves	Function	Jacobian	2nd deriv.	*LU* Factors	Vectors
True Newton	$N+1$	$N+2$	$N+1$	$N+1$	1	1	9
Chord-Newton	1*	$N+I$	$N+1$	$N+1$	1	1	10

Notation. N = number of iterations in Algorithm PAC, I = number of iterative improvement iterations for computing $[u', \lambda']$ and $[u'', \lambda'']$.

* For the chord version, no factorization is needed after the first iteration.

Storage is needed for the vectors: u, δu (in the inner Newton iteration), G_λ, N_u, G, u', u'' and the two approximate null vectors. For the chord version, one more vector is needed to store the old G_λ or v. We have ignored the work involved in computing the approximate null vectors needed for deflation in Algorithm DBE since they have to be computed only once per factorization and the work is thus negligible in comparison to the factorization cost for G_u.

We note that the storage is comparable to those of methods of similar type surveyed in [14], except that a few more vectors are required. The work is also similar, except that for the chord version, no other author seems to have used the potentially more efficient iterative improvement algorithm (19) for computing $[u', \lambda']$ and $[u'', \lambda'']$ with only one factorization of G_u.

For a general, dense n by n problem, the work required for evaluating the second derivatives of G in our algorithm is $O(n^3)$. However, for many problems with sparsity (e.g. see § 7), the work is usually much less.

7. Numerical experiments. We have performed extensive experiments on applying our algorithm and its variants to the following nonlinear elliptic eigenvalue problem [14], [16], [25]:

$$G(u,\lambda)\equiv\Delta u+F(u,\lambda)=0, \tag{20}$$

on the unit square with zero Dirichlet boundary conditions. Following previous authors, we use a *fourth*-order finite difference discretization of (20) on a uniform mesh of size $h=1/m$, which results in a system of $n\equiv(m-1)^2$ nonlinear equations. Two choices for the function F have been considered:

$$F_1=\lambda e^u, \tag{21}$$

$$F_2=\lambda(1+(u+u^2/2)/(1+u^2/100)). \tag{22}$$

For $m=8$, the turning points that we are interested in are given in Table 7.1.

TABLE 7.1
Turning points for $m=8$.

F	λ	$u(0.5, 0.5)$
F_1	6.807504	1.391598
F_2	7.980356	2.272364

All computations have been performed on a DEC-20, with 27-bit mantissas, corresponding to a relative machine precision of about 0.4×10^{-8}. The matrices corresponding to G_u are banded and are factored and solved by the LINPACK routines SGBCO and SGBSL [9]. The work for the factorization is $O(m^4)$, for the solve is $O(m^3)$ and for the evaluation of second derivatives is $O(m^2)$. Thus, for problems of this kind (generally differential equations with a local stencil), the cost of evaluating second derivatives is smaller than the cost of the solve phase.

We use the pseudo-arclength function N_1 in all our computations. We note that N_1 is linear in all its arguments and hence all its second derivatives vanish and $N_u(\sigma)$ and $N_\lambda(\sigma)$ are constants.

For the convergence of the Newton iteration in Algorithm PAC, we use the criterion: $\|G\|<10^{-5}$ and $\|N\|<10^{-5}$, which is adequate for the scale of our problems. For the iterative improvement algorithm (19), we stop if the relative change in the iterate is less than 10^{-5}. For the difference approximations of second derivatives, we use a value of $\varepsilon=10^{-4}$ in (18). For computing the approximate null vectors needed in Algorithm DBE, we always use 3 steps of inverse iteration, the details of which can be found in [3] (this issue, pp. 121–134). The damped version is always used. We shall use the switch/FD to denote the use of difference approximations of second derivatives.

Following Melhem and Rheinboldt [14], we considered two starting points for F_2: $\lambda_0=7.96754$ and $\lambda_1=7.94617$. We also considered two other starting points: $\lambda_3=7.5$ and $\lambda_4=7.0$. All are on the lower branch of the solution curve.

We first tested the Newton version. In Tables 7.2, 7.3 and 7.4, we tabulate the results of applying Algorithm NTP to F_2, starting from λ_0, λ_1 and λ_2 respectively. In Table 7.5, we tabulate the results for F_1 starting from $\lambda = 6.8$ on the lower branch. The notation is as follows:

- I: number of outer Newton iterations.
- I1: number of iterative improvement iterations in computing λ', chord version only,
- I2: number of iterative improvement iterations in computing λ'', chord version only,
- D: number of damped-Newton steps,
- N: number of inner Newton iteration in Algorithm PAC.

For comparison, we have included in the tables the values of $\dot{\lambda}(s)$, which are not needed in the algorithm.

TABLE 7.2
Results for F_2, initial guess $\lambda_0 = 7.96754$, true Newton.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	2.9E−01	0	−3.3E+00	0	8.8E−02	0	1	7.9803556E+00	2.2727977E+00	−6.2E−04
2	−6.5E−04	0	−3.3E+00	0	−2.0E−04	0	0	7.9803557E+00	2.2723642E+00	3.6E−08
3	3.8E−08	0	−3.3E+00	0	1.2E−08	0	0	7.9803557E+00	2.2723642E+00	−1.4E−07

TABLE 7.3
Results for F_2, initial guess $\lambda_1 = 7.94617$, true Newton.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	4.7E−01	0	−2.8E+00	0	1.6E−01	0	1	7.9791579E+00	2.3324510E+00	−8.2E−02
2	−9.7E−02	0	−4.0E+00	0	−2.4E−02	0	1	7.9803553E+00	2.2735657E+00	−1.7E−08
3	−1.9E−03	0	−3.8E+00	0	−5.1E−04	0	0	7.9803558E+00	2.2723647E+00	−7.5E−07
4	−8.5E−07	0	−3.8E+00	0	−2.2E−07	0	0	7.9803558E+00	2.2723642E+00	−1.6E−07

TABLE 7.4
Results for F_2, initial guess $\lambda_2 = 7.5$, true Newton.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	4.8E−01	0	−1.9E−01	0	2.6E+00	0	3	7.8877703E+00	2.8753642E+00	−4.9E−01
2	−2.2E−01	0	−2.1E−01	0	−1.1E+00	0	2	7.9699776E+00	2.1055603E+00	2.6E−01
3	8.1E−02	0	−3.2E−01	0	2.6E−01	0	1	7.9803556E+00	2.2724285E+00	−9.2E−05
4	−3.0E−05	0	−3.2E−01	0	−9.4E−05	0	0	7.9803556E+00	2.2723643E+00	−1.6E−07
5	−5.2E−08	0	−3.2E−01	0	−1.6E−07	0	0	7.9803556E+00	2.2723642E+00	9.3E−10

TABLE 7.5
Results for F_1, initial guess $\lambda = 6.8$, true Newton.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	4.5E−01	0	−1.1E+01	0	4.1E−02	0	2	6.8062598E+00	1.4189429E+00	−1.9E−01
2	−2.5E−01	0	−2.7E+01	0	−9.2E−03	0	1	6.8074830E+00	1.3951085E+00	−2.6E−02
3	−2.9E−02	0	−2.1E+01	0	−1.4E−03	0	0	6.8075035E+00	1.3916595E+00	−4.5E−04
4	−5.1E−04	0	−2.0E+01	0	−2.5E−05	0	0	6.8075035E+00	1.3915978E+00	−3.3E−07
5	−3.6E−07	0	−2.0E+01	0	−1.8E−08	0	0	6.8075035E+00	1.3915977E+00	−1.3E−07

We observe from these results that:

- The computed turning points are accurate to within machine precision.
- The convergence is quadratic.
- The number of inner Newton iterations decreases rapidly as the turning point is approached. In fact, as the turning point is approached, the predictor is often so good that no Newton iteration is needed to satisfy the convergence criteria.
- No damped-Newton step is taken.

When compared to the methods surveyed in [14], our method seems to be more efficient. For example, for the cases corresponding to Tables 7.2 and 7.3, all of the methods in [14] took 4 *outer* iterations or more, whereas our method has converged after 2 and 3 iterations respectively, as judged by the magnitude of $\dot{\lambda}$.

Next, we tested the *chord* version on F_2, with difference approximations for second derivatives, which is the most efficient and most general version. The results are presented in Tables 7.6 and 7.7.

TABLE 7.6
Results for F_2, initial guess $\lambda_0 = 7.96754$, chord/FD.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	2.9E−01	0	−3.3E+00	0	8.8E−02	0	1	7.9803587E+00	2.2727739E+00	—
2	−6.2E−04	2	−3.3E+00	3	−1.9E−04	0	0	7.9803588E+00	2.2723626E+00	—
3	−3.3E−07	1	−3.3E+00	2	−9.9E−08	0	0	7.9803588E+00	2.2723624E+00	−9.1E−08

TABLE 7.7
Results for F_2, initial guess $\lambda_1 = 7.94617$, chord/FD.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	4.7E−01	0	−2.8E+00	0	1.6E−01	0	5	7.9791640E+00	2.3324056E+00	—
2	−9.7E−02	6	−4.0E+00	6	−2.4E−02	0	1	7.9803528E+00	2.2735747E+00	—
3	−2.0E−03	4	−3.8E+00	5	−5.1E−04	0	0	7.9803533E+00	2.2723666E+00	—
4	−7.2E−07	1	−3.8E+00	3	−1.9E−07	0	0	7.9803533E+00	2.2723661E+00	−7.0E−07

These results show that the *outer* iteration is very similar to the results of the basic algorithm. The *inner* iteration took a few more iterations because of the chord-Newton strategy, but due to the more accurate predictor, the number of inner iterations also decreases rapidly as the turning point is approached. As expected, both the inner Newton iterations and the iterative improvement took more iterations when the starting guess (λ_1) is farther away from the turning point. But the total number of solves is still reasonably small considering only one factorization was performed. Note also that the number of iterative improvement iterations is less for $[u', \lambda']$ (which have a better initial guess from a first order predictor) than for $[u'', \lambda'']$.

To test the robustness of the *damped* version, we applied the true Newton version on F_2, starting at $\lambda_4 = 7.0$. The results are given in Table 7.8. Notice that the starting point is quite far away from the turning point and, as a consequence, many damped-Newton steps had to be taken in the beginning. As the turning point is approached, however, no damping is needed and quadratic convergence is regained.

The next test we did was designed to show the effectiveness of the more accurate *second-order predictor.* We repeated exactly the case corresponding to Table 7.4, except that the first-order predictor was used instead. The results are presented in Table 7.9. They are very similar to the results in Table 7.4, except as expected, the number of

TABLE 7.8
Results for F_2, initial guess $\lambda_4 = 7.0$, true Newton.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	9.8E-01	0	-3.3E-02	0	9.3E-01	5	3	7.8777199E+00	1.7912264E+00	7.1E-01
2	8.4E-01	0	-7.5E-01	0	1.2E-01	3	2	7.9665943E+00	2.0815067E+00	3.0E-01
3	6.1E-01	0	-5.9E+00	0	2.5E-02	2	2	7.9790850E+00	2.2123320E+00	9.0E-02
4	3.0E-01	0	-2.6E+01	0	1.2E-02	0	2	7.9794920E+00	2.3233010E+00	-7.0E-02
5	-4.9E-01	0	-2.0E+02	0	-2.5E-03	0	1	7.9802267E+00	2.2919466E+00	-2.8E-02
6	-1.5E-01	0	-9.7E+01	0	-1.5E-03	0	1	7.9803523E+00	2.2755943E+00	-4.6E-03
7	-2.2E-02	0	-7.1E+01	0	-3.1E-04	0	0	7.9803558E+00	2.2724587E+00	-1.4E-04
8	-6.4E-04	0	-6.7E-01	0	-9.6E-06	0	0	7.9803558E+00	2.2723642E+00	-1.6E-07

TABLE 7.9
Results for F_2, initial guess $\lambda_2 = 7.5$, true Newton, first-order predictor.

I	λ'	I1	λ''	I2	$\delta\sigma$	D	N	λ	$u(0.5, 0.5)$	$\dot{\lambda}$
1	4.8E-01	0	-1.9E-01	0	2.6E+00	0	4	7.8877699E+00	2.8753644E+00	-4.9E-01
2	-2.2E-01	0	-2.1E-01	0	-1.1E+00	0	4	7.9699775E+00	2.1055603E+00	2.6E-01
3	8.1E-02	0	-3.2E-01	0	2.6E-01	0	4	7.9803549E+00	2.2724289E+00	-9.2E-05
4	-3.0E-05	0	-3.2E-01	0	-9.4E-05	0	4	7.9803550E+00	2.2723647E+00	-1.6E-07
5	-5.2E-08	0	-3.2E-01	0	-1.6E-07	0	4	7.9803548E+00	2.2723647E+00	3.6E-08

inner Newton iterations does not decrease as the turning point is approached. Comparing the two tables shows the dramatic increase in efficiency made possible by the more accurate predictor.

The last test we performed was designed to test the effect of the *deflation techniques* used in Algorithm DBE, by running some tests using Algorithm BE instead. Without going into details, we shall just report that Algorithm BE is fairly reliable in practice, producing results that are practically the same as if Algorithm DBE had been used. A plausible explanation for the unexpected reliability of Algorithm BE is that it only fails when G_u is *very* singular, at which point the accuracy is usually high enough that the iterations can be terminated. The only kind of problems that we have encountered with Algorithm BE occur when an iterate happens to be very close to the turning point, then $\|G\|$ can actually increase in the inner Newton iteration, causing a damped-Newton step to be taken. On the other hand, we have had no problem with Algorithm DBE at all, and we believe that it is to be preferred because of its higher reliability and minimal extra cost.

8. Conclusions. We have presented and tested a new algorithm for computing simple turning points of nonlinear equations. It possesses quadratic convergence, which, together with the more accurate second order predictor, makes it extremely fast when applied close to a turning point. We have also demonstrated that, through the use of a chord-Newton variant, the efficiency can be increased dramatically in such cases. On the other hand, when started far away from a turning point, its use of a natural damped-Newton strategy makes it reasonably reliable and robust. The use of the block-elimination algorithm with implicit deflation makes it possible to exploit special structures and solvers for the problem. Although second derivatives of G are required, the experimental results show that difference approximations for them can be used safely. Although more tests on different and larger problems are needed to more completely validate the new algorithm, our limited experimental results show rather convincingly that it is both efficient and reliable and compares favourably with the best of the existing methods.

REFERENCES

[1] J. P. ABBOTT, *An efficient algorithm for the determination of certain bifurcation points*, J. Comp. Appl. Math., 4 (1978), pp. 19–27.

[2] E. ALLGOWER AND K. GEORG, *Simplicial and continuation methods for approximating fixed points and solutions to systems of equations*, SIAM Rev., 22 (1980), pp. 28–85.

[3] T. F. CHAN, *Deflation techniques and block-elimination algorithms for solving bordered singular systems*, Tech. Rep. 226, Computer Science Dept., Yale Univ., New Haven, CT 06520, 1982; this Journal, this issue, pp. 121–134.

[4] ———, *Deflated decompositions of solutions of nearly singular systems*, Tech. Rep. 225, Computer Science Dept., Yale Univ., New Haven, CT, 1982; SIAM J. Numer. Anal., 21 (1984), to appear.

[5] T. F. CHAN AND H. B. KELLER, *Arclength continuation and multi-grid techniques for nonlinear eigenvalue problems*, SIAM J. Sci. Stat. Comp., 3 (1982), pp. 173–194.

[6] T. F. CHAN, *On the existence and computation of LU-factorizations with small pivots*, Tech. Rep. 227, Computer Science Dept., Yale Univ., New Haven, CT, 1982.

[7] T. F. CHAN AND Y. SAAD, *Iterative methods for solving bordered systems with applications to continuation methods*, Tech. Rep. 235, Computer Science Dept., Yale Univ., New Haven, CT, 1982.

[8] C. DEN HEIJER AND W. C. RHEINBOLDT, *On steplength algorithms for a class of continuation methods*, SIAM J. Numer. Anal., 18 (1981), pp. 925–947.

[9] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[10] P. E. GILL, W. MURRAY, M. A. SAUNDERS AND M. H. WRIGHT, *Computing forward difference intervals for numerical optimization*, this Journal, 4 (1983), pp. 310–321.

[11] H. B. KELLER, *Numerical solution of bifurcation and nonlinear eigenvalue problems*, Applications of Bifurcation Theory, P. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359–384.

[12] ———, *Global homotopies and Newton methods*, in Recent Advances in Numerical Analysis, Carl de Boor and Gene Golub, eds., Academic Press, New York, 1978, pp. 73–94.

[13] M. KUBICEK, *Dependence of solution of nonlinear systems on a parameter*, ACM Trans. Math. Software, 2 (1976), pp. 98–107.

[14] R. G. MELHEM AND W. C. RHEINBOLDT, *A comparison of methods for determining turning points of nonlinear equations*, Computing, 29 (1982), pp. 201–226.

[15] H. D. MITTELMANN AND H. WEBER, *Numerical methods for bifurcation problems—A survey and classification*, Bifurcation Problems and their Numerical Solution, H. D. Mittelmann and H. Weber, eds., Workshop on Bifurcation Problems and their Numerical Solution, January 15–17, Birkhauser, Dortmund, 1980, pp. 1–45.

[16] G. MOORE AND A. SPENCE, *The calculation of turning points of nonlinear equations*, SIAM J. Numer. Anal., 17 (1980), pp. 567–576.

[17] J. C. PAUMIER, *Une methode numérique pour le calcul des points de retournement. Application a un probleme aux limites non-lineaires*, Numer. Math., 37 (1981), pp. 433–444.

[18] G. PÖNISCH AND H. SCHWETLICK, *Computing turning points of curves implicitly defined by nonlinear equations depending on a parameter*, Computing, 26 (1981), pp. 107–121.

[19] W. C. RHEINBOLDT, *Solution fields of nonlinear equations and continuation methods.*, SIAM J. Numer. Anal., 17 (1980), pp. 221–237.

[20] W. C. RHEINBOLDT AND J. V. BURKARDT, *A program for a locally parametrized continuation process*, Tech. Rep. ICMA-81-30, Institute for Computational Mathematics and Applications, Dept. Mathematics and Statistics, University of Pittsburgh, Pittsburgh, 1981.

[21] W. C. RHEINBOLDT, *Computation of critical boundaries on equilibrium manifolds*, Tech. Rep. ICMA-81-20, Institute for Computational Mathematics and Applications, Dept. Mathematics and Statistics, Univ. of Pittsburgh, Pittsburgh, 1980.

[22] ———, *Numerical analysis of continuation methods for nonlinear structural problems*, Computers and Structures, 13 (1981), pp. 103–113.

[23] H. SCHWETLICK, *Effective methods for computing turning points of curves implicitly defined by nonlinear equations*, Tech. Rep. 46, Sektion Mathem., Martin Luther Univ., Halle Wittenberg, 1981.

[24] R. SEYDEL, *Numerical computation of branch points in nonlinear equations*, Numer. Math., 33 (1979), pp. 339–352.

[25] R. B. SIMPSON, *A method for the numerical determination of bifurcation states of nonlinear systems of equations*, SIAM J. Numer. Anal., 12 (1975), pp. 439–451.

[26] G. W. STEWART, *On the implicit deflation of nearly singular systems of linear equations*, this Journal, 2 (1981), pp. 136–140.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

A NUMERICAL METHOD FOR THE INVERSE STURM-LIOUVILLE PROBLEM*

JOHN PAINE†

Abstract. In this paper we present a method for solving a form of the inverse Sturm–Liouville problem. The basis of the method is to modify the given differential eigenvalues so that the $N\times N$ tridiagonal matrix eigenvalue problem recovered from the first N eigenvalues can (after a suitable transformation) be identified with the finite difference approximation of the required differential eigenvalue problem. Numerical results are presented to illustrate the effectiveness of this method.

Key words. inverse eigenvalue problems, eigenvalues, finite difference approximation

1. Introduction. The inverse eigenvalue problem for the Sturm–Liouville problem

$$-\ddot{u}+qu=\lambda u, \qquad x\in[0,\pi], \tag{1.1}$$

$$\alpha u(0)+\beta\dot{u}(0)=0, \tag{1.2}$$

$$\gamma u(\pi)+\delta\dot{u}(\pi)=0 \tag{1.3}$$

consists of recovering the potential q from one or more of the spectra $\{\lambda_k\}_{k=1}^{\infty}$ corresponding to distinct sets of the boundary conditions (1.2)–(1.3). It is well known [2] that when the spectrum for the essential boundary conditions

$$u(0)=0=u(\pi) \tag{1.4}$$

only is available, the inverse problem only has a unique solution in the class of square integrable functions which satisfy the symmetry condition

$$q(\pi-x)=q(x) \tag{1.5}$$

almost everywhere in $[0,\pi]$. In the remainder of this paper we will assume that the potential satisfies this condition.

The method we present here uses the algorithm given in de Boor and Golub [3] to recover a persymmetric tridiagonal matrix from N eigenvalues (we refer to this algorithm as ALGC), the new feature of our approach being that a correction is added to the differential eigenvalues before recovering the matrix. This correction compensates for the differing asymptotic behaviour of the given differential eigenvalues and the required finite difference eigenvalues.

In what follows we use the vector pair $\mathbf{a}^T=(a_1,\cdots,a_N)$ and $\mathbf{b}^T=(b_2,\cdots,b_N)$ to represent the class of $N\times N$ tridiagonal matrices

$$T=\begin{bmatrix} t_{1,1} & t_{1,2} & & \\ t_{2,1} & t_{2,2} & t_{2,3} & \\ & \cdot & \cdot & \cdot \\ & & t_{N,N-1} & t_{N,N} \end{bmatrix}$$

which satify the conditions $t_{i,i}=a_i$, $i=1,\cdots,N$ and $t_{i,i+1}*t_{i+1,i}=b_{i+1}$, $i=1,\cdots,N-1$. This class contains a unique (up to the signs of the off-diagonal elements) symmetric matrix, and all matrices in this class are similar.

* Received by the editors June 28, 1982.

† Department of Mathematics, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6.

2. Development of the eigenvalue correction. There has been a great deal of interest in developing an adequate theoretical grounding and useful numerical methods for inverse eigenvalue problems (see e.g. [1], [4]). This work has usually been in two distinct but related directions. The first is in establishing the existence and uniqueness of solutions to the differential inverse eigenvalue problems (DIEVP) and in developing numerical methods for recovering the potential; and the second is a study of the same problems in the context of reconstructing a matrix eigenvalue problem with a specified form from its eigenvalues i.e. the matrix inverse eigenvalue problem (MIEVP). The two approaches are frequently linked in the development of numerical methods in the sense that the DIEVP is often solved by converting it to a related MIEVP for which a method of solution exists. The motivation usually given for the connection between the MIEVP and the DIEVP is that the recovered matrix has the same structure as the matrix eigenvalue problem obtained by applying a particular numerical method to solve (1.1)–(1.3). Examples of this approach can be found in Hald [5] where a Rayleigh–Ritz method with trigonometric basis functions is used and in Morel [6] where the central finite difference approximation of (1.1)–(1.3) is used. The identification used by Morel is particularly common since the finite difference approximation of (1.1)–(1.3) yields a matrix eigenvalue problem of the form

$$-A\mathbf{u}+h^2Q\mathbf{u}=h^2\mu\mathbf{u}, \tag{2.1}$$

where $Q=\operatorname{diag}\{q_1,\cdots,q_N\}$, $\mathbf{u}^T=(u_1,\cdots,u_N)$, and $-A+h^2Q$ is the symmetric tridiagonal matrix given by $a_i=2+h^2q_i$, $i=1,\cdots,N$ and $b_i=1$, $i=2,\cdots,N$. Here we adopt the usual notation $q_i=q(x_i)$, where $x_i=ih$, and $h=\pi/(N+1)$, and it is assumed that $q\in C^2[0,\pi]$.

As the finite difference approximation of (1.1), (1.4) yields a tridiagonal matrix eigenvalue problem, if we can recover a tridiagonal matrix from N given eigenvalues it may be possible to identify the recovered matrix with the finite difference form (2.1) and hence recover the potential. This objective has been given as the motivation for a number of methods aimed at solving the tridiagonal MIEVP, but to the author's knowledge there is no published work which examines its validity. (Note that Morel's numerical results use the exact finite difference eigenvalues rather than the differential eigenvalues.)

It is clear that this identification should be examined more closely if we compare the behaviour of the eigenvalues λ_k of (1.1), (1.4) with $q(x)=0$ with that of eigenvalues μ_k of (2.1) with $Q=\operatorname{diag}\{0,\cdots,0\}$. Clearly $\lambda_k=k^2$ while $\mu_k=2h^{-2}(1-\cos kh)$ and so $\mu_k/\lambda_k\to 4/\pi^2+0(h)$ as $k\to N$. In other words the asymptotic behaviour of the differential and finite difference eigenvalues is substantially different and so the matrix recovered using the differential eigenvalues cannot necessarily be directly identified with the finite difference form. (It should be noted here that in Hald's method [5] the matrix and differential eigenvalues have the same asymptotic behaviour and so the matrix he recovers from the differential eigenvalues can be directly related to the matrix obtained by the Rayleigh–Ritz method.)

To investigate the effect that the differing asymptotic behaviour of the differential and finite difference eigenvalues has on the matrix recovered using ALGC, we recovered the tridiagonal matrix **a**, **b** from h^2k^2, $k=1,\cdots,N$ for $N=25$.

The elements of the recovered matrix given in Table 1 for $i=1,\cdots,(N+1)/2$, when compared with the results given in Morel and in de Boor and Golub, clearly show that the difference in asymptotic behaviour severely affects the recovered matrix since it obviously deviates substantially from the expected form.

TABLE 1
Matrix recovered from uncorrected differential eigenvalues.

i	a_i	b_i
1	0.560363	—
2	1.227185	0.189143
3	1.836272	0.578852
4	2.387198	1.112919
5	2.879860	1.740226
6	3.314264	2.414698
7	3.690462	3.095297
8	4.008537	3.746015
9	4.268588	4.335891
10	4.470715	4.839035
11	4.615014	5.234659
12	4.701559	5.507123
13	4.730402	5.645974

The question now arises as to whether we can improve the agreement of the recovered matrix with the form (2.1). Since ALGC would yield the exact finite difference matrix if we had the exact finite difference eigenvalues, and since the matrix obtained by ALGC depends continuously on the given eigenvalues, we would expect ALGC to give a matrix close to the finite difference matrix if we could supply eigenvalues which are close to the finite difference eigenvalues. This observation would not at first appear to be useful since the finite difference eigenvalues are not known; however, a method has been given in Paine, de Hoog and Anderssen [7] for improving the accuracy of finite difference eigenvalues. This improvement is achieved by adding a correction to each of the finite difference eigenvalues which gives them the same asymptotic behaviour as the differential eigenvalues. Thus if we subtract this correction from the (exact) differential eigenvalues we will obtain estimates of the desired finite difference eigenvalues which will now have the correct asymptotic behaviour. What we propose to do then is:

(i) correct the differential eigenvalues using

$$\hat{\lambda}_k = \lambda_k - (k^2 - 2h^{-2}(1-\cos kh)), \qquad k = 1, \cdots, N \tag{2.2}$$

which is the reverse of the correction given in [7],

(ii) then use ALGC with these corrected eigenvalues to recover the matrix **a**, **b**.

If we use this method in the above example we will recover the exact finite difference matrix since the corrected eigenvalues in this case are precisely the finite difference eigenvalues. So to test this method on a nontrivial problem we applied it to recover the potential $q(x) = 6\cos 2x$ from its eigenvalues. The results given in Table 2 for $N = 25$, represent a dramatic improvement in the agreement of the recovered matrix with the form (2.1) since the b_i are now almost unit entries.

If we ignore the discrepancy between the expected and recovered **b** and recover the discrete potential using the equations $a_i = 2 + h^2 q_i$ the values obtained, also given in Table 2, indicate that we have recovered some information on the potential since the recovered potential has the same qualitative structure as the true potential. However the discrete values are only approximately half those of the true potential at the points x_i. Therefore, although correcting the differential eigenvalues does yield a matrix which is approximately of the required form, the discrepancy with respect to the form (2.1) cannot be ignored, and so this approach does not in itself give us a means for recovering the potential.

TABLE 2
Matrix recovered from corrected differential eigenvalues.

i	a_i	b_i	$h^{-2}(a_i-2)$
1	2.042623	—	2.919384
2	2.038761	0.959640	2.654865
3	2.032704	0.964465	2.239976
4	2.024749	0.971388	1.695160
5	2.015352	0.980012	1.051529
6	2.005061	0.989850	0.346661
7	1.994482	1.000334	−0.377927
8	1.984239	1.010851	−1.079494
9	1.974933	1.020774	−1.716917
10	1.967100	1.029513	−2.253420
11	1.961181	1.036551	−2.658824
12	1.957499	1.041477	−2.910994
13	1.956251	1.044011	−2.996534

3. Development of the inverse method.

3. Development of the inverse method. Although the matrix recovered in this example clearly cannot be identified with the finite difference matrix in (2.1) since the off-diagonal elements are not unit entries, it has been noted in Hald [4] that the matrix can be identified with the finite difference approximation of the more general Sturm–Liouville eigenvalue problem

$$-(p\dot{v})^{\cdot}+rv=\lambda v, \qquad s\in[0,\pi] \tag{3.1}$$

with the boundary conditions (1.4), where p and r satisfy (1.5), and $p\in C^4[0,\pi]$, $r\in C^2[0,\pi]$. For this problem the finite difference eigenvalue problem is of the form

$$-D\mathbf{v}+h^2R\mathbf{v}=h^2\lambda\mathbf{v}, \tag{3.2}$$

where $R=\operatorname{diag}\{r_1,\cdots,r_N\}$ and $-D+h^2R$ is the persymmetric tridiagonal matrix given by

$$a_i=(p_i+p_{i+1})+h^2r_i, \qquad i=1,\cdots,N, \tag{3.3}$$

$$b_i=p_i^2, \qquad i=2,\cdots,N. \tag{3.4}$$

Here $r_i=r(s_i)$, $s_i=ih$ and $p_i=p(s_{i-1/2})$ where $s_{i-1/2}=(i-1/2)*h$, $i=1,\cdots,N+1$.

If we attempt to identify the recovered matrix with (3.2) we immediately discover that p_1 is undetermined. However on observing that (3.1) and (1.1) are connected via the Liouville transform

$$x=x(s), \qquad x(s)=\int_0^s p^{-1/2}\,ds, \tag{3.5}$$

$$u=wv, \qquad w=p^{-1/4}, \tag{3.6}$$

$$q(x)=r(x)-\left(\frac{\dot{w}}{w}\right)'(x)+\left(\frac{\dot{w}}{w}\right)^2(x) \tag{3.7}$$

if $x(\pi)=\pi$, it seems reasonable to attempt to find a discrete version of the Liouville transform to take (3.2) to the form (2.1). Unfortunately there is no discrete analogue of (3.5)–(3.6) which will transform (3.2) exactly to (2.1), so we will instead discretize (3.5)–(3.6) in the same fashion as (3.1) to obtain an approximate transformation.

The first step in implementing this approximate transform is to observe that the values of the function p in (3.1) are known (via ALGC) at the midpoints of the subintervals of a uniform partition of $[0, \pi]$ except on the first and last intervals. This immediately suggests that we can use midpoint quadrature to evaluate the integral in (3.5) once we have fixed p_1. Since we are assuming that p is symmetric about $\pi/2$ the obvious choice for p_1 (and hence p_{N+1}) is that value which yields $x(\pi)=\pi$. This requirement is satisfied if we set

$$p_1=4h^2\bigg/\left(\pi-\sum_{i=2}^{N} hp_i^{-1/2}\right)^2 \tag{3.8}$$

where the p_i, $i=2,\cdots,N$ are obtained from **b** using (3.4), and it is assumed that the term in parentheses in (3.8) is positive. If this assumption is not satisfied, we have $x(\pi)>\pi$ for the approximate transform and so it will not be possible to transform (3.2) to (2.1) on the interval $[0, \pi]$. Once p_1 is determined in this way, the discrete analogues of (3.5)–(3.6) are

$$x_i=x_{i-1}+hp_i^{-1/2}, \qquad x_0=0, \qquad i=1,\cdots,N+1, \tag{3.9}$$

$$u_i=w_iv_i, \qquad w_i=p_i^{-1/4}, \qquad i=1,\cdots,N. \tag{3.10}$$

This identification of the recovered matrix with the finite difference form (3.2) already provides a much better approximation of the potential as can be seen if we recover the matrices with $N=15$, 31, and 63, evaluate p_i, $i=2,\cdots,N$ from (3.4), p_1 from (3.8) and then recover the r_i using (3.3). The errors in the discrete potential at the (uniform) points s_i of the s interval are displayed in Fig. 1 for $i=1,\cdots,(N+1)/2$. In fact the recovered potential is very accurate and the convergence to the exact potential appears to be second order for points away from the boundary. For points near to the boundary, convergence is still obtained but is more complex in that the order of convergence for the point closest to the origin is only one.

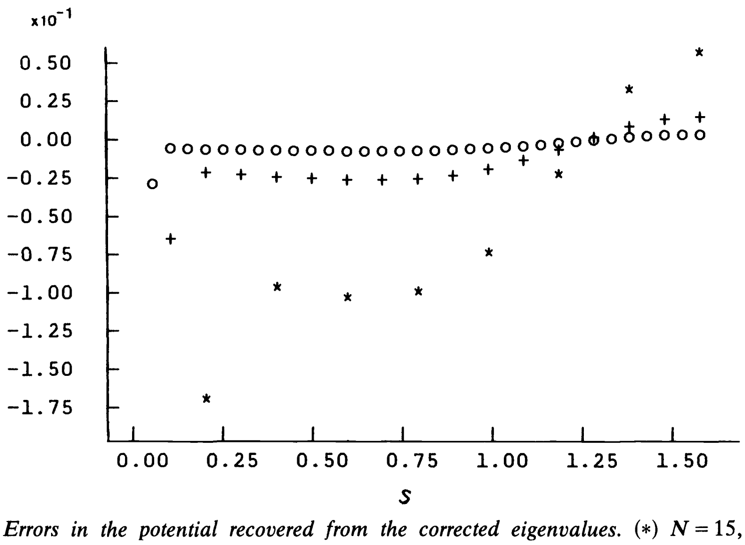


FIG. 1. *Errors in the potential recovered from the corrected eigenvalues.* (*) $N=15$, (+) $N=31$, (O) $N=63$.

The remaining step in implementing the transform is to evaluate $q(x_i), i = 1, \cdots, N$ from (3.7), which will then be the required discrete potential of the Liouville normal form, but will in this case be known on the nonuniform partition of $[0, \pi]$ given by the x_i. The natural choice for evaluating (3.6) at the points x_i is to replace the derivatives with respect to x with differences evaluated on the x partition. However the nonuniformity of the partition of the x interval, coupled with the realization that $q(x) = q(x(s))$ and that the partition of the s interval is uniform, suggested that (3.7) be restated in terms of derivatives with respect to s, which can then be readily and accurately approximated by differences in the s interval. Thus, on noting that

$$q(s) = r(s) + \frac{\ddot{p}(s)}{4} - \frac{(\dot{p})^2(s)}{16p(s)}$$

we evaluate the discrete potential using the formula

$$q_i = r_i + .25\ddot{p}_i - .0625\frac{\dot{p}_i^2}{\hat{p}_i} \tag{3.11}$$

where

$$\dot{p}_i = \frac{p_{i+1} - p_i}{h}, \qquad i = 1, \cdots, N,$$

and

$$\ddot{p}_i = \begin{cases} \dfrac{-3\dot{p}_1 + 4\dot{p}_2 - \dot{p}_3}{2h}, & i = 1, \\ \dfrac{\dot{p}_{i+1} - \dot{p}_{i-1}}{2h}, & i = 2, \cdots, N-1, \\ \dfrac{3\dot{p}_N - 4\dot{p}_{N-1} + \dot{p}_{N-2}}{2h} & i = N. \end{cases}$$

Since the values of p are known at the midpoints of the subintervals of a uniform partition of $[0, \pi]$, the difference approximations of $\dot{p}_i$ and $\ddot{p}_i$ will be second order accurate, and so the values of q_i will also be second order accurate provided we use $\hat{p}_i = (p_{i+1} + p_i)/2$ in (3.11).

We implemented this method for recovering the potential from the matrices recovered in the previous example and give the errors in the discrete potential at the points x_i in Fig. 2.

The results obtained show that the q_i are generally better approximations of the exact potential than are the r_i, and that the convergence of the discrete potential has the same characteristics. Though it should be noted that there appears to be a Gibbs type phenomenon for the points near the origin which was not observed in the previous results.

As a final point we observe that, since this approximate Liouville transform is aimed at producing a finite difference eigenvalue problem on a nonuniform partition of $[0, \pi]$ which has the form (2.1) where $-A + h^2Q$ is given by

$$a_i = 2h^2/h_{i-1}h_i + h^2 q_i, \qquad i = 1, \cdots, N. \tag{3.12}$$

$$b_{i+1} = 4h^4/h_i^2(h_{i-1} + h_i)(h_i + h_{i+1}), \quad i = 1, \cdots, N-1 \tag{3.13}$$

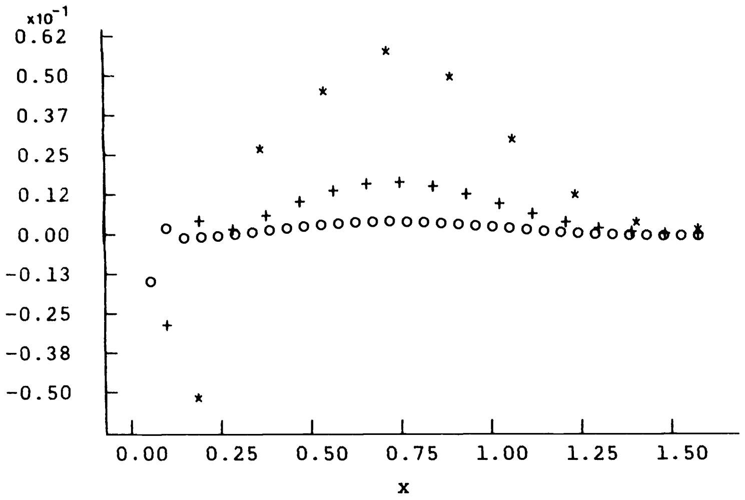


FIG. 2. *Errors in the potential recovered using* (3.11). (*) $N=15$, (+) $N=31$, (O) $N=63$.

where $h=\pi/(N+1)$ and h_i, $i=0,\cdots,N$ is defined as $h_i := (x_{i+1}-x_i)$ $i=0,\cdots,N$, an alternative method for recovering the q_i is to recover the matrix $\mathbf{a}$ and $\mathbf{b}$ as before and then determine the h_i by solving the system of nonlinear equations (3.13) with the given b_i, subject to the constraint $h_0+h_1+\cdots+h_N=\pi$. Once the h_i are known the q_i can be obtained directly from (3.12).

We implemented this approach by solving the system of nonlinear equations using Newton's method, and found that the partition defined by the h_i and the recovered potential agreed closely with those given by the approximate transform method. There therefore appears to be little justification for using this alternative formulation as it requires the iterative solution of a (possibly large) system of nonlinear equations.

4. Conclusions and discussion.

In this paper we have presented what appears to be an efficient method for recovering symmetric potentials from the spectrum associated with essential boundary conditions. The main deficiencies are that no theoretical justification has as yet been obtained, and that the method as it stands can only be applied to recover potentials $q \in C^2[0, \pi]$ which satisfy (1.5).

Since a method for correcting the finite difference eigenvalues for the general boundary conditions (1.2)–(1.3) is given in [7], and a method is given in [3] for recovering a general Jacobi matrix from its eigenvalues together with the eigenvalues of its left principal submatrix of order $N-1$, it would seem likely that the symmetry constraint could be removed by combining these two methods. It would also seem likely that potentials which are only piecewise twice differentiable could be recovered from their spectra by using the method given in [8] for obtaining finite difference approximations for such potentials. However the means for overcoming the lack of theoretical justification of the methods given here are not obvious.

REFERENCES

[1] V. BARCILON, *Well-posed inverse eigenvalue problems*, Geophys. J. Roy. Astr. Soc., 42 (1975), pp. 375–383.

[2] G. BORG, *Eine Umkehrung der Sturm-Liouvilleschen Eigenwertaufgabe*, Acta. Math., 78 (1946), pp. 1–96.

[3] C. DE BOOR AND G. H. GOLUB, *The numerically stable reconstruction of a Jacobi matrix from spectral data*, Linear Algebra Appl., 21 (1978), pp. 245–260.

[4] O. H. HALD, *On discrete and numerical inverse Sturm–Liouville problems*, Ph.D. thesis, New York Univ., New York, 1972.

[5] ———, *The inverse Sturm–Liouville problem and the Rayleigh–Ritz method*, Math. Comp., 32 (1978), pp. 687–705.

[6] P. MOREL, *Des algorithmes pour le problème inverse des valeurs propres*. Linear Algebra Appl., 13 (1976), pp. 251–273.

[7] J. W. PAINE, F. DE HOOG AND R. S. ANDERSSEN, *On the correction of finite difference eigenvalue approximations for Sturm–Liouville problems*, Computing, 26 (1981), pp. 123–139.

[8] N. A. TIKHONOV AND A. A. SAMARSKII, *The Sturm–Liouville difference problem*, USSR Comput. Math. and Math. Phys., 1 (1961), pp. 784–805.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

COMPUTATIONAL METHODS FOR MINIMUM SPANNING TREE ALGORITHMS*

R. E. HAYMOND†, J. P. JARVIS† AND D. R. SHIER†

Abstract. The well-known algorithms of Kruskal, Prim, and Sollin for constructing a minimum spanning tree are surveyed in this paper. We discuss various data structures that can facilitate the search and update operations of these algorithms. In particular, certain novel data structures for carrying out the Kruskal and Prim algorithms are presented. Computational experience using four implementations of the Kruskal algorithm and two implementations of the Prim algorithm on a class of moderately large networks is also given. Overall, the best performance is attained by our new implementation of Prim's algorithm which incorporates an address calculation sort.

Key words. minimum spanning trees, computational experience, data structures, networks

1. Introduction.

Minimum spanning trees, which connect together nodes of a network at minimum total cost (or length), find application in the planning of efficient distribution systems (pipelines, transmission lines) and in the designing of layouts for circuit boards [28], [31]. In addition, minimum spanning trees provide useful information for problems arising in clustering and taxonomy [15], [41], pattern recognition [9], [30], [42], minimax control processes [20], [24], network reliability [26], and chemical physics [37]. Variants of minimum spanning trees also occur in the study of multiterminal network flows [21] and in the approximate solution of travelling salesman problems [16], [17].

Three distinct procedures for calculating minimum spanning trees have been developed [3], [27], [31] as well as several variants of these basic procedures. An excellent survey of a number of solution procedures is provided by Kershenbaum and Van Slyke [26]. The present paper extends this survey of solution procedures and examines various data structures that can facilitate the search and update operations of these procedures. Computational experience with existing and alternative implementations is also presented.

2. Basic solution procedures.

In this section, the minimum spanning tree problem is defined more precisely and the three basic solution procedures are presented. Detailed discussion of specific data structures and implementation issues is taken up in § 3.

Suppose that $G=(N,E)$ is a loop-free connected network with node set N and (undirected) edge set E. With each edge $e=(i,j)$ in E is associated a cost $c(e)=c(i,j)$, unrestricted in sign. Let $n=|N|$ and $m=|E|$ signify the number of nodes and edges in G respectively. A *spanning tree* T of G is a subgraph (N', E') of G with $N'=N$, $E'\subseteq E$, such that $|E'|=n-1$ and E' contains no cycles. The *cost* $c(T)$ of a spanning tree $T=(N,E')$ is given by

$$c(T)=\sum_{e\in E'} c(e). \tag{1}$$

A *minimum spanning tree* (MST) for G is then a spanning tree T' such that $c(T')\leqq c(T)$ for all spanning trees T of G.

* Received by the editors April 9, 1981. This research was supported by the Department of Mathematical Sciences, Clemson University; the Operations Research Division, National Bureau of Standards; and the National Science Foundation under grant ISP-8011451 (EPSCOR). All computations were performed at the Clemson University Computer Center.

† Department of Mathematical Sciences, Clemson University, Clemson, South Carolina 29631.

2.1. Kruskal's algorithm. Suppose e is an edge of G not in some given spanning tree T. Since there is a unique path $P_T(e)$ in T between the two end nodes of e, then $P_T(c) \cup \{e\}$ forms a cycle in G. It follows that if T is a *MST* then $c(e) \geqq c(u)$ for all edges u in $P_T(e)$, since otherwise the cost of T could be reduced by exchanging e for some edge u in $P_T(e)$. This necessary condition is also sufficient [12], [34], and so the following optimality condition characterizes minimum spanning trees.

Optimality condition 1. Suppose T is a spanning tree of G. T is a MST iff for all edges e not in T, $c(e) \geqq c(u)$ for all edges u in $P_T(e)$.

This optimality condition immediately suggests one method for obtaining a MST. Namely, construct an arbitrary spanning tree for G and test whether the optimality condition above holds. If so, the current tree is optimal. Otherwise, there are edges e and u such that the condition is not satisfied, and it is therefore advantageous to exchange e for u in the current tree. By repeating this test/exchange procedure, we will obtain a minimum spanning tree after a finite number of steps.

An alternative, and more systematic, method for obtaining a minimum spanning tree also derives from the above optimality condition. Namely, suppose that edges are considered for inclusion in T in order of nondecreasing cost and only chosen for inclusion when they do not form a *cycle* with edges already chosen. Because of this manner of selection, an edge forming a cycle with the chosen edges must have a cost at least as large as the cost of any chosen edge in $P_T(e)$. Thus, the optimality condition is guaranteed to hold and so a minimum spanning tree is indeed produced by this method. Although its origin has been attributed to Boruvka [4], this method is generally referred to as *Kruskal's algorithm* [27]. It was also independently discovered and amplified upon by Loberman and Weinberger [28]. A statement of the general algorithm is provided by the following.

KRUSKAL'S ALGORITHM. Given a connected, n-node network $G=(N, E)$, this algorithm produces a MST for G.

```
Step 1. Set $S = E$, $T = \phi$, $i = 0$.
Step 2. Let $e'$ have minimum cost among edges in $S$.
Step 3. If $T \cup \{e'\}$ does not contain a cycle then
            Set $i = i + 1$, $T = T \cup \{e'\}$.
            If $i = n - 1$ then STOP.
Step 4. Set $S = S - \{e'\}$. Return to Step 2.
```

In the above algorithm, set S contains the candidate edges, T contains the edges selected so far, and $i = |T|$. At any stage of Kruskal's algorithm, the selected edges form a forest (a node-disjoint collection of trees). Ultimately, after $n-1$ edges have been added, a spanning tree is obtained.

There are two major tasks involved in implementing the Kruskal algorithm: selecting a minimum cost edge from S (Step 2) and checking for the creation of a cycle (Step 3). These tasks have been carried out in a variety of ways by various investigators. Namely, selection of a minimum cost edge can be done by searching the set S each time for a smallest element [29], by presorting the edges according to cost [5], [26], [36], or by keeping the edges in a heap [1], [6], [19], [26], [33]. Checking for cycle creation can be accomplished by labeling and updating connected components [5], [26], [36], by the use of linked lists to maintain connected components [1], [26], by storing components as subtrees and using the subtree root as the component label [1], [29], or by using tree structures and path compression together with efficient "find" and "set union" operations [1], [6], [19], [33]. Depending on the specific implementation used, the worst-case complexity of Kruskal's algorithm ranges from $O(m^2)$ to $O(m \log m)$ where $m = |E|$. (In this paper, all logarithms are base 2).

2.2. Prim's algorithm. The optimality condition of § 2.1 allows one to decide whether a given spanning tree is a MST by examining certain (fundamental) cycles in G. Alternatively, there is another optimality condition, dual to the first, that involves *cutsets* of G [1], [8], [14], [34]. Recall that the cutset (X, X^c) is the set of all edges in E joining a node in X to a node in $X^c = N - X$.

Optimality condition 2. T is a MST iff for all X such that $\phi \subset X \subset N$, there is an edge e' of T such that $e' \in (X, X^c)$ and $c(e') = \min\{c(e): e \in (X, X^c)\}$.

Thus, a MST for G contains a minimum cost edge in every cutset of G. In particular, suppose that $T' = (N', E')$ is a subtree of a MST. Note that T' is a minimum spanning tree on the subnetwork of G induced by the node set N'. Taking $X = N'$ in the second optimality condition shows that T' can be extended to a larger subtree of a MST by adding to it a minimum cost edge $e' \in (N', N'^c)$. This observation provides motivation for a second basic solution method for constructing a MST. Namely, at each stage we have a set of nodes N' and a subtree T' on these nodes. A minimum cost edge $e' = (i, j)$, with $i \in N'$ and $j \notin N'$, is then selected from the cutset (N', N'^c), whereupon j is added to N' and e' is added to T'. Continuing in this fashion, the current subtree is extended until it contains all nodes of G, at which point a MST for G has been found. This method was first proposed by Prim [31] and later independently by Dijkstra [11]. In contrast to the Kruskal procedure, each stage of the Prim procedure generates a tree (rather than a forest). Moreover, a MST can be grown in the Prim procedure starting from any node in N. A general version of *Prim's algorithm* is embodied in the following.

PRIM'S ALGORITHM. Given a connected, n-node network $G = (N, E)$, this algorithm produces a MST for G.

```
Step 1. Set $X = \phi$, $T = \phi$. Choose any $j' \in N$.
Step 2. Set $X = X \cup \{j'\}$. If $X = N$ then STOP.
Step 3. Choose $e' = (i, j)$ with $i \in X, j \in X^c$, such that $c(e') =$
        $\min\{c(e): e \in (X, X^c)\}$.
Step 4. Set $T = T \cup \{e'\}$, $j' = j$. Return to Step 2.
```

Several variants of this algorithm have been proposed to carry out the cutset minimization in Step 3. Although it is possible to form the entire cutset (X, X^c) and find the minimum cost edge by searching through the whole cutset of potentially $O(n^2)$ edges [5], [18], it is more efficient to restrict the search to an $O(n)$ subset of (X, X^c). A useful device is to keep a temporary label $d(j)$ that represents at any stage either the minimum cost of an edge between node $j \in X$ and set X^c, or the minimum cost of an edge between set X and node $j \in X^c$. The former approach has been employed, together with a fast nearest-neighbor procedure, for calculating MST's of k-dimensional point sets [2]. More commonly, however, the latter approach has been used. Thus, the node j selected in Step 3 is simply the node of X^c having minimum temporary label. When set X is enlarged in Step 2 by adding node j', the remaining nodes $j \in X^c$ need to have their temporary labels updated via: $d(j) = \min\{d(j), c(j', j)\}$.

Accordingly, two tasks necessary for implementing this form of Prim's algorithm are: updating the temporary node labels and selecting a node with minimum temporary label. Generally, the updating is done in a fairly standard way, while the selection of a minimum-label node can be carried out in various ways. For example, the selection step can be implemented by a direct search of all node labels, possibly using a flag to indicate membership in X^c [19], [32], [33], [35] or using an array to index nodes in X^c [14], [39]. Alternatively, the temporary labels (or their corresponding edges) can be stored and updated using a heap [26], or a β-heap [22]. These different implementations all have worst-case complexity $O(n^2)$. However, if G is sparse, an

$O(m \log n)$ implementation is possible (see § 3.2); if G is sufficiently dense, an $O(m)$ algorithm can be constructed [22].

2.3. Sollin's algorithm. Another procedure for constructing a MST can be based on optimality condition 2. This procedure appears to have been first proposed in 1938 by Choquet [7] for points embedded in a metric space, and then independently in 1961 by Sollin [3], [25] for arbitrary networks. Notice that by using $X=\{i\}$ in the second optimality condition, it follows that a MST must contain a minimum cost edge out of each node i. Therefore, in this construction method, a minimum cost edge out of each node i is first selected. If these selected edges form a tree , then this tree is a MST. More generally, these edges will form a forest of node-disjoint subtrees $T_1, \cdots, T_k$. If the node set of each subtree is "shrunk" to a single node, then the procedure can be reapplied to the subnetwork defined by these k nodes; namely, a minimum cost edge out of each new node is again selected. The procedure is repeated until the original network has been shrunk to a single node. A general statement of what is known as *Sollin's algorithm* is provided by the following.

SOLLIN'S ALGORITHM. Given a connected, n-node network $G=(N, E)$, this algorithm produces a MST for G.

```
Step 1. Initialize the list $L$ to consist of the single-node sets $\{1\}, \{2\}, \cdots, \{n\}$. Set
        $T=\phi$.
Step 2. If $|L|=1$ then STOP.
Step 3. Select a set $S$ from $L$.
Step 4. Find a minimum cost edge $e'=(i, j)$ with $i$ in $S$, $j$ not in $S$. Set $T=T\cup\{e'\}$.
Step 5. Merge the set $S$ with the set $S'$ in $L$ containing node $j$. Return to Step 2.
```

Sollin's algorithm has the advantage that Steps 3–5 above can be carried out using parallel processors [14]. However, if the edge costs are not distinct then some care must be exercised to avoid the creation of cycles. Goodman and Hedetniemi [14] provide a tie-breaking rule for Step 4 that avoids this difficulty.

In carrying out this algorithm, several choices are available for selecting the set S from list L in Step 3: e.g., randomly, using a stack, or using a (priority) queue. Major tasks in executing the algorithm are identifying a minimum cost edge e' (Step 4), finding the node set S' (Step 5), and merging two sets of L into a single set (Step 5). Cheriton and Tarjan [6] have investigated several variants of the Sollin algorithm, using tree structures for $T_1, \cdots, T_k$ together with efficient "find" and "set union" operations for Step 5; also, several types of priority queues for edges are employed in carrying out the minimization of Step 4. Yao [40] has modified the basic Sollin algorithm by partitioning the edges incident to each node into "levels" based on edge cost. Another Sollin-based algorithm that uses 2–3 trees as the underlying data structure is outlined in [1].

The worst-case complexity of the basic Sollin algorithm is $O(m \log n)$, where $m=|E|$ and $n=|N|$. This complexity can be reduced to $O(m \log \log n)$ using several of the above-mentioned modifications to the Sollin algorithm [6], [40].

3. Alternative data structures. In this section, algorithms for the basic Kruskal and Prim approaches are presented in conjunction with data structures useful in their implementation. Four distinct implementations of the Kruskal procedure and two implementations of the Prim procedure are described. We have chosen not to investigate the Sollin-type algorithms in the present paper. While such algorithms have a better theoretical worst-case complexity than the Kruskal or Prim approaches, the more complicated data structures and consideration of tie-breaking in the Sollin algorithm may adversely affect its actual empirical performance. Moreover, computational experience with the algorithms developed here shows that fairly large problems can be solved very efficiently by properly implementing the Prim and Kruskal approaches: problems with up to 500 nodes and 24,000 edges were solved in no more than 0.6 seconds CPU time.

3.1. Implementations of Kruskal's algorithm. As noted in § 2.1, there are two primary tasks in implementing Kruskal's algorithm. The first, finding a minimum cost edge, is frequently accomplished via a heap and that approach is taken here. The second, determining whether a candidate edge creates a cycle, will be examined in more detail.

Four approaches, leading to Algorithms K1, K2, K3, and K4, are considered for detecting cycles. Recall that at any stage of Kruskal's algorithm the edges selected so far define a forest of disjoint subtrees $\{T_1, T_2, \cdots, T_k\}$. Algorithm K1, the most straightforward implementation, employs a node-length array A of subtree labels [38]. That is, $A(i)$ provides for each node i the identity (or label) of the unique subtree T_r containing i. A candidate edge forms a cycle with the current forest if and only if the labels of its end nodes are equal. When these labels are different, the candidate edge joins distinct subtrees T_r and T_s; in this case, the subtree labels can be updated by scanning the array A and changing every occurrence of the label for T_r to the label for T_s. As noted in [26], this cycle-detection approach requires $O(n^2)$ operations in the worst case. In addition, it may be necessary to examine m edges in order to construct the MST, in which case finding the minimum cost edges (via a heap) requires $O(m \log m)$ computations. Hence the effort required to generate the MST can be dominated by either the cycle detection procedure or the examination of edges.

An improved procedure, Algorithm K2, maintains a linked list structure for each subtree T_r. In other words, the nodes comprising each such subtree are chained together using a successor function. Therefore, the subtree labels of all nodes in T_r can be accessed and updated in a direct manner. Furthermore, it is advantageous to update the subtree labels only for nodes of the smaller subtree. Algorithm K2 then requires at most $O(m \log m)$ operations to generate the candidate edges, plus $O(n \log n)+O(m)$ operations to process and update the linked list structure [26].

Algorithm K3 employs a rooted tree as a data structure for determining the existence of cycles [1], [29]. The root of each subtree serves as the subtree label. Using path compression and information as to the size of subtrees to maintain a balanced tree data structure results in a cycle detection procedure which is almost linear in m [1]. Note, however, that generating the candidate edges is still an $O(m \log m)$ operation.

The new algorithm proposed here, Algorithm K4, is similar to Algorithm K3 in the use of a tree structure for identifying subtree components but it does not require complete information as to the root of each subtree. The data structure employed in Algorithm K3 maintains the nodes of each subtree T_r as a rooted tree on those nodes. A predecessor function, defined for each node in T_r, gives the predecessor node of node i along the unique path from the root to node i in the rooted tree. By convention, the root node is used to label each subtree. Thus, the subtree label for any node is obtained by tracing back (using the predecessor function) until the root of its subtree is obtained. In Algorithm K3, this traceback cannot be terminated prior to reaching the root.

An alternative approach, used in Algorithm K4, is to maintain subtree information in a tree data structure where successive unions of subtrees are referenced by new *component* names. This will require a maximum of $(n-1)$ additional storage locations but allows a parallel traceback through successive components for any pair of nodes to determine subtree membership. In determining whether two nodes are in the same subtree, the parallel traceback can be terminated once the traceback for either node is completed since two nodes are in the same subtree if and only if they share a component at a higher *level*. This structure is best illustrated by example.

Consider the six node graph shown in Fig. 1. Suppose that edges (1, 2), (6, 5), (3, 5), (2, 6), and (5, 4) are successively added to the spanning forest in constructing a MST. Start initially with a component at *level one* for each node (Fig. 2a). For edge (1, 2), nodes 1 and 2 share no higher level component and hence are in different subtrees. Create a higher level component at *level two* labeled 7 (Fig. 2b). Similarly create component 8 from edge (6, 5) and nodes 5 and 6 (Fig. 2b).

Edge (3, 5) is next to be added to the spanning forest. The predecessor component of 3 at level two is 0 (none exists); for node 5, the predecessor component is 8. The traceback can be terminated at level two and the level two component of node 3 is set equal to 8 (Fig. 2c). For edge (2, 6), the level two components are 7 and 8, which are nonzero but unequal. The traceback continues to *level three* where the predecessors

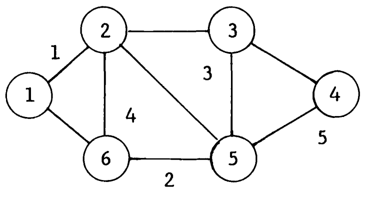


FIG. 1. *Six-node graph to illustrate component level data structure.*

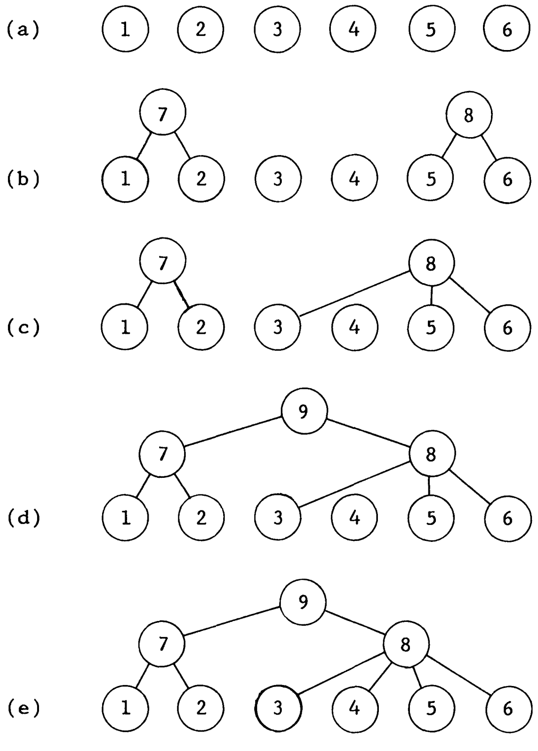


FIG. 2. *Successive development in component level data structure.*

of components 7 and 8 are both 0 and so the original nodes (2 and 6) are in different subtrees. A new component, 9, at level three is created as the predecessor of components 7 and 8 of level two (Fig. 2d).

If edge (2, 5) were considered at this juncture, its two endpoints would be found to have different components at level two (7 and 8), but the same component at level three (9). Thus, the edge would be rejected for inclusion in the MST since nodes 2 and 5 are in the same subtree. Finally, consideration of edge (5, 4) yields the predecessor component 0 for node 4 and the predecessor component 8 for node 5. The nodes are in different subtrees and so the edge is added to the current forest; the level two component of 4 is set to 8 (Fig. 2e). The algorithm terminates having found the five edges required for a minimum spanning tree in this six node network (Fig. 3).

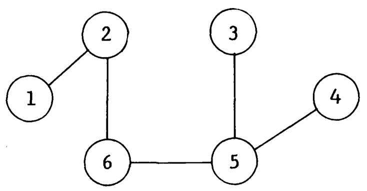


FIG. 3. *Minimum spanning tree for the network of Fig.* 1.

In general, the traceback will yield predecessor components for each node at each level. If the components are nonzero and equal, the original nodes are in the same subtree. If only one of the components is zero, that component is set to the other (nonzero) component number. If both components are zero, a new component at the next higher level is generated. In each of these latter two cases, the edge is added to the spanning forest. If both components are nonzero but unequal, the traceback continues to the next level. A property of this construction technique is that the rooted trees so generated have a depth which is at most that of a balanced binary tree on the same number of nodes; i.e. the smallest integer greater than or equal to $(\log n)$.

As with Algorithms K1, K2, and K3, the effort required in Algorithm K4 will depend on the topology of the particular graph to be analyzed. Note that the maximum depth of the component levels is $O(\log n)$ and this bound is attained if the associated data structure is a binary tree. In the worst case, it would be necessary to scan m edges in order to complete the construction of the MST. Since the maximum depth of the rooted tree is $O(\log n)$, then at most $O(m \log n)$ operations are required. Again, the generation of candidate edges by nondecreasing cost will require $O(m \log m)$ operations in the worst case.

Computational experience with these four variations on Kruskal's algorithm is presented in § 4.

3.2. Implementations of Prim's algorithm. Two variations, denoted P1 and P2, of Prim's algorithm are presented here. As noted in § 2.2, maintaining temporary labels for nodes $j \in X^c$ requires the ability to update these labels and to find their minimum. Algorithm P1 employs a heap [23], [26] to keep a partial order on the temporary labels and a node index list to facilitate label updating. Algorithm P2, a new variation on Prim's algorithm, maintains the temporary labels via an address calculation. The approach is similar to that employed by Dial [10] for solving shortest route problems. The method for maintaining temporary node labels is now briefly described.

Let the maximum edge cost be denoted $c_{\max}$. The edge costs can be assumed strictly positive and integral since all edge costs can be monotonically translated and scaled without changing the minimum spanning tree. The address space consists of a list L of $c_{\max}$ elements (node indices) initialized to zero. If a new temporary label v is obtained for node j, examine the element of L in the vth position, $L(v)$. If $L(v)$ is zero, then set $L(v)=j$. If $L(v)$ is nonzero, j is added to a linked list LL (of maximum length n) containing all such duplications. If node j had been labeled previously and is not currently in the set X, there will be an entry j in either L or LL which must be deleted. The entry can be found via the previous temporary label of node j. Note that the address calculation and data structure update take place only if the new label for j is smaller than its previous label.

To find the minimum label (corresponding to the next node to be added to the set X), simply scan successive elements of L searching for the first nonzero entry. That entry will be the index of the node having minimum label. Note that the scan must start at the top of the list L as opposed to the rotating scan employed in the corresponding shortest route algorithm. However, some savings can be achieved by starting the scan at the minimum of the last node label (previous minimum) and the smallest label updated when that node was added to the set X. When a smaller updated label is generated, the first nonzero element of L is available without scanning if the minimum of such labels is retained.

It should be noted that this algorithm is the only one of the six presented here which is sensitive to the actual values of the edge costs and not just their rank. If $c_{\max}$ is large relative to the number of edges, one might expect relatively few duplications and hence less effort in updating labels since the linked list is rarely required. However, a low duplication rate could also mean that many elements of L are zero (empty) and more effort is required to find the minimum label. Conversely, if nonzeros are dense in L, it is expected that fewer operations are needed to obtain the minimum label, but more effort is required to update labels since duplications are more likely.

Finally, note that the length of the linked list will never exceed n if previous label entries for nodes are eliminated when such labels are updated. Otherwise, LL could grow considerably larger and when a minimum label is determined, the corresponding node must be checked for inclusion in X.

To illustrate the address calculation data structure, consider the network of Fig. 4. With $X=\{1,2,6\}$, $(X,X^C)=\{(2,3),(2,4),(1,4),(6,5)\}$, nodes $\{3,4,5\}$ have temporary labels $d(3)=4$, $d(4)=4$, and $d(5)=3$. The list L and associated linked list LL for duplicate labels are shown in Fig. 5. Notice that $c_{\max}$ is five.

The worst-case complexity of the Prim-type algorithms can be derived by noting that every edge in the graph is eventually checked as a candidate to update a temporary node label. In the case of sparse networks, Algorithm P1 requires at most $O(m \log n)$ operations for updating the heap of temporary node labels and $O(n \log n)$ operations

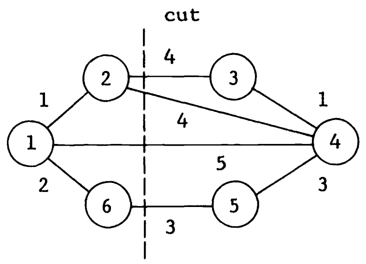


FIG. 4. *Network illustrating the address calculation data structure.*

```
         L (address list)         LL (linked list)

           label:node             node --> successor

              1:0                    1 --> *

     TOP --> 2:0                    2 --> *

              3:5                    3 --> *

              4:4                    4 --> 3

              5:0                    5 --> *

                                     6 --> *
```

FIG. 5. *Illustration of address calculation data structure. The * denotes no following node in the linked list.* TOP *is the starting position of the scan for the next minimum.* TOP *is* 2 *since node* 6 *with label* 2 *was the last node added to the set X. The address space length is* 5 (*the maximum edge cost*).

to reform the heap after removing the minimum elements, yielding $O(m \log n)$ worst-case complexity. For dense networks, updating the heap of temporary node labels can be done in $O(n^2)$ operations, resulting in an $O(n^2)$ algorithm overall. Algorithm P2 requires in the worst-case $O(m\, n)$ operations to update labels in the address space and $O(n\, c_{\max})$ operations to find the next minimum element.

Empirical computational evidence concerning the four Kruskal-type and two Prim-type algorithms is presented in the next section.

4. Computational experience. The algorithms presented above were tested on sets of randomly generated networks of various sizes. Before discussing the results, some explanation of the test conditions is necessary.

4.1. Implementation of algorithms. The various algorithms were coded in FORTRAN IV using structured programming and modular design. In particular, all versions of the Kruskal-type algorithms were exactly the same except for minor differences in the initialization routine and the particular section of code that determined whether a candidate edge was to be added to the spanning forest. This last section of code included the details of updating the data structure specific to each algorithm. A similar approach was used for the Prim-type algorithms. In that case, the codes differed only in the manner of finding the node of minimum temporary label and in the method of updating node labels. This coding procedure permitted analysis of the specific differences between algorithms by separating those tasks common to all approaches.

The measure of computational effort reported below is CPU time (excluding compilation time and I/O processing). Times were obtained by running the various codes as load modules produced by the IBM Extended H level compiler on an IBM 370/3033 computer.

4.2. Sample problems. A set of connected networks was generated with the number of nodes n ranging from 25 to 500 and with the edge density d ranging from 10 to 100%. The edge density is the total number of edges m divided by the maximum number of possible edges, $n(n-1)/2$. In addition, for each value of n, a problem having n edges was also considered. Table 1 gives the number of nodes and edges for the problems considered. For each problem size (fixed n, m), a sample of 5 networks was randomly generated. Restrictions imposed by the amount of readily available core storage limited the number of edges to 24,000 in this study.

The graphs were generated at random in the following manner. Given a spanning subtree on nodes $\{1, 2, \cdots, j\}$, a node k was chosen at random from that set and the edge $(k, j+1)$ was added to the spanning subtree. Additional edges were successively

TABLE 1
Number of edges for each test problem. (*) denotes number of edges equal to number of nodes. (—) denotes problem not considered.

Nodes	Edge density						
	*	0.1	0.3	0.5	0.7	0.9	1.0
25	25	30	90	150	210	270	300
50	50	123	368	613	858	1,103	1,225
75	75	278	833	1,388	1,943	2,498	2,775
100	100	495	1,485	2,475	3,465	4,455	4,950
150	150	1,118	3,353	5,588	7,823	10,058	11,175
200	200	1,990	5,970	9,950	13,930	17,910	19,900
300	300	4,485	13,455	22,425	—	—	—
400	400	7,980	23,940	—	—	—	—
500	500	12,475	—	—	—	—	—

generated to obtain a spanning tree for the graph. Finally, edges were generated randomly from the set of all remaining edges and added to the graph until the specified number of edges was obtained. Edge costs were generated from a uniform distribution over a range 1 to 10,000. These costs were subsequently scaled down for successive runs using the address calculation sort (Algorithm P2). The application of Algorithm P2 to these data will be referred to as Algorithms P2A, P2B, P2C, and P2D for $c_{\max}$ equal to 10,000, 2,000, 400, and 80 respectively. In fact, the graphs and algorithms are the same, with only the size of the address space changing; the designations P2A, P2B, P2C, and P2D are used simply as a matter of convenience.

The data for each problem were saved and accessed in turn by each implementation of the six algorithms. The information required by the Kruskal-type algorithms was the number of nodes n, the number of edges m, and a list of edge triples $(i, j, c(i, j))$. The Prim-type algorithms required n, m, and a list of nodes adjacent to each node in the network and the associated edge costs (i.e., a forward star representation [13] of the edge set). The processing order of edges for the Kruskal algorithms is randomized by virtue of the random generation of edge costs. For the Prim algorithms, the starting node j' was arbitrarily set to the first node of the graph. Table 2 provides a summary of the data storage requirements of each algorithm for a network having n nodes and m edges. It should be noted that because m is generally the dominant term, the Prim algorithms are more economical of space than the Kruskal algorithms.

TABLE 2
Data storage requirements for algorithms, expressed in terms of m (number of edges), n (number of nodes), and $c_{\max}$ (maximum edge cost).

Algorithm	Storage (words)
K1	$5m+2n$
K2	$5m+4n$
K3	$5m+3n$
K4	$5m+3n$
P1	$4m+5n$
P2	$4m+4n+c_{\max}$

4.3. Execution times. Table 3 gives the average execution times (in hundredths of a second) for codes K1–K4 on the collection of test problems summarized in Table 1. A number of observations are apparent from this table. First, Algorithms K2 and K3 are fairly comparable in terms of average execution time, and both produce consistently smaller times than K4, which in turn performs better than K1. A linear model of the form

$$T = b_1 m + b_2 n \log m \tag{2}$$

TABLE 3
Average execution times for the Kruskal algorithms in hundredths of seconds. The number of edges in each problem is given in Table 1. Edge density ranges from 0.1 *to* 1.0; *the number of nodes, from* 25 *to* 500. (*) *indicates n = m*; (—) *indicates problems not considered.*

Nodes	Alg.	Edge density						
		*	0.1	0.3	0.5	0.7	0.9	1.0
25	K1	2.19	2.27	3.14	2.95	3.78	3.36	3.86
	K2	2.03	2.28	3.11	2.92	3.51	3.30	3.78
	K3	2.02	2.23	3.33	3.09	3.71	3.26	3.73
	K4	2.13	2.29	3.34	3.06	3.69	3.36	3.84
50	K1	3.52	5.68	7.97	6.17	7.45	6.81	8.93
	K2	3.43	5.50	7.75	6.03	7.15	6.56	8.56
	K3	3.38	5.44	7.85	5.93	7.13	6.56	8.58
	K4	3.40	5.53	8.02	6.05	7.30	6.84	8.87
75	K1	5.07	9.14	13.58	12.37	12.31	12.92	12.97
	K2	4.73	8.85	13.07	11.77	11.66	12.30	12.25
	K3	4.83	8.91	13.00	11.83	11.67	12.29	12.39
	K4	4.77	8.93	13.37	12.43	12.28	12.77	12.90
100	K1	6.77	14.52	15.15	17.96	15.60	20.65	19.36
	K2	6.23	13.93	14.25	17.06	14.73	19.51	18.31
	K3	6.28	13.90	14.21	17.08	14.68	19.32	18.28
	K4	6.26	14.31	14.69	17.63	15.39	20.36	19.10
150	K1	10.11	28.04	28.71	28.74	33.50	34.46	37.58
	K2	8.86	26.29	27.15	26.80	31.07	32.07	35.00
	K3	8.92	26.39	26.97	26.94	31.00	31.82	34.79
	K4	8.89	26.86	28.05	28.19	32.03	33.87	37.10
200	K1	13.74	39.15	40.72	44.61	44.05	50.53	56.45
	K2	11.53	36.47	37.64	41.10	41.96	46.84	51.76
	K3	11.55	36.76	37.87	41.67	39.82	46.30	50.64
	K4	11.60	37.86	39.87	43.55	42.66	49.92	55.10
300	K1	23.45	62.10	62.50	76.50	—	—	—
	K2	17.40	55.08	55.70	68.11	—	—	—
	K3	17.81	55.33	55.68	68.25	—	—	—
	K4	17.52	56.90	58.85	72.84	—	—	—
400	K1	32.21	86.92	102.07	—	—	—	—
	K2	22.71	75.41	88.27	—	—	—	—
	K3	22.77	80.73	88.26	—	—	—	—
	K4	23.07	78.18	94.07	—	—	—	—
500	K1	42.97	120.50	—	—	—	—	—
	K2	29.22	97.37	—	—	—	—	—
	K3	28.36	97.36	—	—	—	—	—
	K4	28.46	107.89	—	—	—	—	—

was found to provide a good fit of execution time T as a function of n (number of nodes) and m (number of edges). Least squares estimates for b_1 and b_2 as well as the multiple correlation coefficient R^2 are summarized in Table 4 for each of the Kruskal algorithms. The virtual equality of the least squares estimates for K2 and K3 in Table 4 confirms the observation that these two algorithms are comparable in terms of execution time. The high values obtained for R^2 indicate an excellent fit for model (2).

TABLE 4
Summary statistics for least squares fit of execution times for the Kruskal algorithms versus $(b_1m + b_2n \log m)$.

Algorithm	Regression statistics		
	b_1	b_2	R^2
K1	0.000974	0.01333	0.960
K2	0.001166	0.01067	0.946
K3	0.001103	0.01090	0.939
K4	0.001288	0.01113	0.941

As an example, Fig. 6 shows the average execution time for Algorithm K2, plotted against the value predicted by (2). The near-linearity of this plot indicates that the empirical complexity of K2 is $O(m+n \log m)$. A similar observation applies to the other Kruskal-type algorithms.

This empirical $O(m+n \log m)$ behavior can be explained as follows. The computational effort involved in implementing the Kruskal algorithms is dominated by the heap operations. Forming the heap of edge costs is an $O(m)$ operation and each reforming of the heap requires $O(\log m)$ operations. Since the heap is reformed once for every new candidate edge, the total effort to establish and maintain the heap is $O(m+k \log m)$, where k is the number of candidate edges examined during the progress of Kruskal's algorithm. Clearly, the value k depends on the network topology and edge costs, but not on the particular computer implementation.

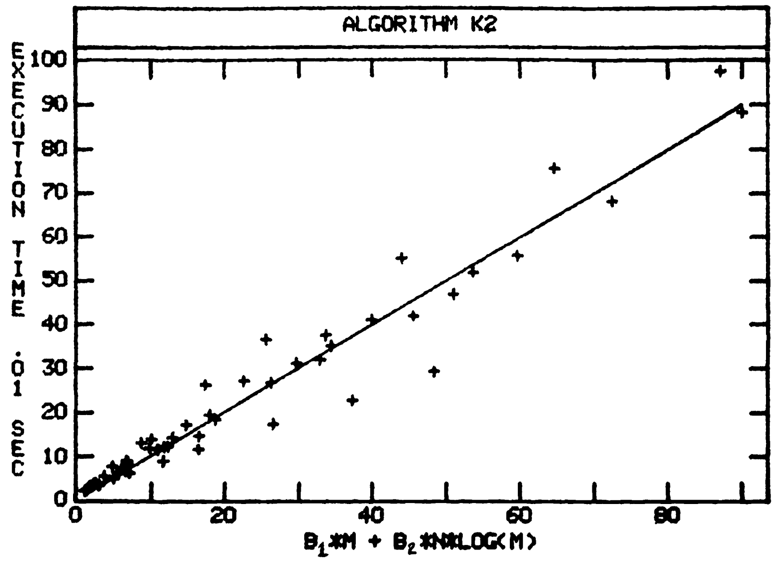


FIG. 6. *Average execution time (hundredths of seconds) versus* $(b_1m + b_2n \log m)$ *for Algorithm* K2.

The observed dependence of the number of candidate edges on the number of nodes n and the density d is shown in Table 5, which displays the mean number of candidate edges $\bar{k}$ (averaged over the five networks at each size). In addition, for each value of n, the maximum and minimum values of $\bar{k}/n$ are shown. It is seen that the value $\bar{k}/n$ lies in the fairly restricted range of 1.1–3.2 for all problems considered here. In other words, $\bar{k}$ is approximately 1 to 3 times the number of nodes. This implies, in turn, that the computational effort for the Kruskal algorithms should be dominated by $O(m+n \log m)$ operations, as we have already observed. It is worth emphasizing that for the random networks generated here, the number of edges that need to be examined by the Kruskal algorithm is never more than about three times the *minimum* number of edges $(n-1)$ required to build a *spanning* tree.

TABLE 5
Average number of edges, $\bar{k}$, examined by the Kruskal algorithms; also, the minimum and maximum values of $\bar{k}/n$ for edge densities ranging from 0.1 *to* 1.0.

	Edge density						Min	Max
Nodes	0.1	0.3	0.5	0.7	0.9	1.0	$\bar{k}/n$	$\bar{k}/n$
25	29	45	39	50	43	52	1.14	2.07
50	87	125	87	107	94	127	1.74	2.54
75	148	214	186	176	176	171	1.97	2.85
100	235	229	270	213	282	254	2.13	2.82
150	457	438	402	437	435	476	2.68	3.17
200	637	596	599	522	586	643	2.61	3.21
300	914	805	910	—	—	—	2.68	3.05
400	1237	1243	—	—	—	—	3.09	3.11
500	1572	—	—	—	—	—	3.14	3.14

Table 6 displays the average execution times (in hundredths of a second) for codes P1 and P2A–P2D on the test problems. It is seen that Algorithm P2C consistently produces smaller execution times than P1, except in the very low density case when $m=n$. However, in this case P2D is always superior to P1. In fact, Algorithm P2D is generally the best performer for networks with up to approximately 5,000 edges. With networks having some 5,000 to 15,000 edges, Algorithm P2C performs very well, and for networks with more than 15,000 edges Algorithm P2B becomes competitive.

Recall that the maximum edge costs $c_{\max}$ used in Algorithms P2A–P2D are 10,000, 2,000, 400, and 80 respectively. Thus, it is not surprising that P2D performs well for networks having relatively few edges, since in this case there will be relatively few node labels in the address space and relatively few duplications. As the number of edges is increased, more effort is required to manage duplicate labels but less effort is needed to find the minimum node label. The empirical evidence shows that even with up to 5,000 edges an address space with $c_{\max}=80$ is fairly effective, indicating that the burden of duplicated labels is handled efficiently over a broad range for m. Beyond 5,000 edges, an address space with $c_{\max}=400$ is quite effective. A rough rule of thumb suggested by these empirical findings is that Algorithm P2 can be quite efficient so long as the number of edges is not more than 40 times the value of $c_{\max}$ (at least when edge costs are uniformly distributed over the range 1 to $c_{\max}$).

Execution times T for the Prim algorithms were fit quite well by a model of the form

$$T=b_1n+b_2m. \tag{3}$$

TABLE 6

Average execution times for the Prim algorithms using heap sort (P1) *or address calculation sort* (P2). *Algorithms* P2A, P2B, P2C, *and* P2D *refer to applications of Algorithm* P2 *with* $c_{\max}$ *equal to* 10,000, 2,000, 400, *and* 80, *respectively.* (*) *indicates* $n = m$; (—) *indicates problems not considered.*

		Edge density						
Nodes	Alg.	*	0.1	0.3	0.5	0.7	0.9	1.0
	P1	3.31	3.33	3.57	3.46	3.54	3.65	3.66
	P2A	5.16	4.80	4.03	4.08	3.73	3.80	3.81
25	P2B	3.63	3.57	3.48	3.45	3.68	3.59	3.59
	P2C	3.34	3.34	3.32	3.43	3.64	3.58	3.61
	P2D	3.27	3.28	3.52	3.38	3.48	3.58	3.57
	P1	5.89	6.19	6.25	6.64	6.81	7.04	7.06
	P2A	9.53	7.92	6.91	6.79	6.79	7.04	7.12
50	P2B	6.59	6.32	6.36	6.41	6.50	6.82	6.95
	P2C	5.95	6.05	6.23	6.46	6.59	6.79	6.96
	P2D	5.85	6.00	6.18	6.46	6.58	6.76	6.88
	P1	8.54	8.82	9.39	9.83	10.43	10.93	11.25
	P2A	14.86	10.34	9.81	10.00	10.26	10.82	11.17
75	P2B	9.75	8.94	9.33	9.64	10.13	10.68	10.90
	P2C	8.64	8.83	9.30	9.58	10.20	10.61	10.94
	P2D	8.45	8.65	9.10	9.52	10.13	10.52	10.89
	P1	11.28	11.82	12.58	13.32	14.14	15.43	15.51
	P2A	19.48	13.18	12.65	13.28	13.90	14.80	15.10
100	P2B	12.92	11.84	12.31	13.02	13.64	14.59	15.12
	P2C	11.49	11.71	12.15	12.99	13.69	14.65	15.00
	P2D	11.27	11.51	12.23	12.87	13.71	14.68	15.04
	P1	16.20	17.50	19.67	21.54	23.60	25.13	25.92
	P2A	28.16	18.46	19.59	21.10	22.66	24.38	25.16
150	P2B	18.55	17.43	18.95	20.69	22.81	24.16	24.91
	P2C	16.63	16.94	19.04	20.59	22.93	24.09	24.93
	P2D	16.13	16.90	18.96	20.53	22.92	24.19	24.97
	P1	21.59	23.50	27.15	31.00	34.30	39.64	39.81
	P2A	36.10	24.12	26.50	29.78	32.66	37.78	37.55
200	P2B	24.13	23.10	26.18	29.44	32.58	35.58	39.06
	P2C	21.46	22.70	26.17	29.28	34.03	35.92	38.42
	P2D	21.16	22.65	26.02	29.67	32.64	36.78	38.52
	P1	32.54	38.49	44.77	52.36	—	—	—
	P2A	55.56	37.08	43.52	50.53	—	—	—
300	P2B	36.30	36.16	42.79	52.34	—	—	—
	P2C	33.48	36.05	42.60	50.07	—	—	—
	P2D	32.19	36.85	43.18	51.00	—	—	—
	P1	43.13	51.07	66.70	—	—	—	—
	P2A	72.14	49.99	63.98	—	—	—	—
400	P2B	48.15	51.56	61.98	—	—	—	—
	P2C	43.24	48.47	65.06	—	—	—	—
	P2D	42.11	49.18	64.94	—	—	—	—
	P1	53.80	65.77	—	—	—	—	—
	P2A	89.06	63.58	—	—	—	—	—
500	P2B	61.43	63.84	—	—	—	—	—
	P2C	53.58	62.87	—	—	—	—	—
	P2D	52.84	63.47	—	—	—	—	—

Table 7 shows the least squares estimates for the parameters b_1 and b_2 together with the multiple correlation coefficient R^2. Table 7 indicates that as $c_{\max}$ is increased for the P2 algorithms, the value of b_1 increases while the value of b_2 decreases. This finding supports the observation that as $c_{\max}$ increases, the effort to select the minimum label increases while the effort to manage duplicate labels decreases.

TABLE 7
Summary statistics for least squares fit of execution times for the Prim algorithms versus $(b_1n + b_2m)$.

Algorithm	Regression statistics		
	b_1	b_2	R^2
P1	0.10866	0.0009322	0.999
P2A	0.15430	0.000008431	0.945
P2B	0.11685	0.0006672	0.997
P2C	0.10723	0.0008379	0.998
P2D	0.10608	0.0008730	0.999

The high values obtained for R^2 indicate an excellent fit for model (3). As an example, Fig. 7 shows average execution time for Algorithm P2C plotted against the value predicted by (3). Notice that the empirical $O(m+n)$ complexity of the Prim algorithms is in decided contrast to the worst-case estimates of at least $O(m \log n)$ for these algorithms.

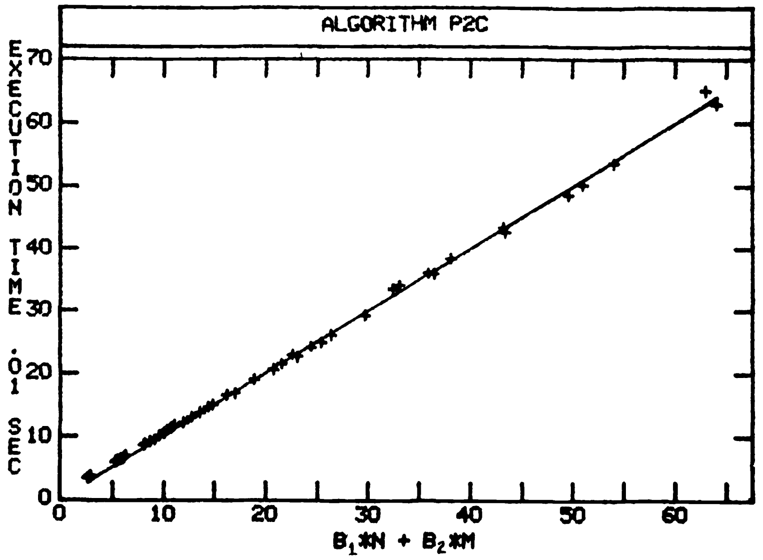


FIG. 7. *Average execution time (hundredths of seconds) versus* $(b_1n + b_2m)$ *for Algorithm* P2C.

Table 8 compares one of the best Kruskal algorithms (K2) with one of the best Prim algorithms (P2C). It is seen that the Kruskal algorithm is generally faster for $n \leqq 50$ and the "ultra-sparse" case $m = n$. Otherwise, the Prim algorithm is superior, even in the cases of low densities ($d = 0.1, d = 0.3$). In fact, the Prim algorithm is some 35% faster than the Kruskal algorithm for the cases $d = 0.1$ and $n \geqq 150$.

More disaggregated data than that given in Tables 3 and 6 show that the individual execution times are more variable among the five sample networks at each size for the Kruskal algorithms than for the Prim algorithms. This greater variability appears

TABLE 8
Comparison of average execution time (hundredths of seconds) for Algorithms K2 *and* P2C *(address calculation sort for temporary labels,* $c_{\max} = 400$*). (*) indicates $n = m$; (—) indicates problems not considered.*

Nodes	Alg.	Edge density *	0.1	0.3	0.5	0.7	0.9	1.0
25	K2	2.03	2.28	3.11	2.92	3.51	3.30	3.78
	P2C	3.34	3.34	3.32	3.43	3.64	3.58	3.61
50	K2	3.43	5.50	7.75	6.03	7.15	6.56	8.56
	P2C	5.95	6.05	6.23	6.46	6.59	6.79	6.96
75	K2	4.73	8.85	13.07	11.77	11.66	12.30	12.25
	P2C	8.64	8.83	9.30	9.58	10.20	10.61	10.94
100	K2	6.23	13.93	14.25	17.06	14.73	19.51	18.31
	P2C	11.49	11.71	12.15	12.99	13.69	14.65	15.00
150	K2	8.86	26.29	27.15	26.80	31.07	32.07	35.00
	P2C	16.63	16.94	19.04	20.59	22.93	24.09	24.93
200	K2	11.53	36.47	37.64	41.10	41.96	46.84	51.76
	P2C	21.46	22.70	26.17	29.28	34.03	35.92	38.42
300	K2	17.40	55.08	55.70	68.11	—	—	—
	P2C	33.48	36.05	42.60	50.07	—	—	—
400	K2	22.71	75.41	88.27	—	—	—	—
	P2C	43.24	48.47	65.06	—	—	—	—
500	K2	29.22	97.37	—	—	—	—	—
	P2C	53.58	62.87	—	—	—	—	—

to be the result of variations in k (the number of candidate edges considered by the Kruskal algorithm) with network topology and edge costs. The resulting "robustness" of the Prim algorithms, coupled with their smaller computation times and reduced storage requirements (compared to the Kruskal algorithms), make them especially well-suited for large-scale networks of the type considered here. In particular, Algorithm P2 (adjusted as necessary for the value $c_{\max}$) can be especially effective in solving large minimum spanning tree problems.

5. Conclusions. There appears to be little overall difference in the efficiency of several sophisticated approaches for constructing a minimum spanning tree via Kruskal's algorithm. While maintaining component flags in a linked list or representing each subtree by a rooted tree exhibited better behavior than the component level approach, computational effort was still dominated by the task of sorting the edge list. Although expected for high density networks, the Prim-type algorithms generally outperformed the Kruskal-type algorithms over all problems considered. In addition, the Prim-type algorithms require less storage than the Kruskal-type algorithms.

Finally, the use of an address calculation sort for node labels in conjunction with Prim's algorithm provided the best overall performance, particularly when the edge costs could be scaled to adjust the size of the address space for a particular problem.

REFERENCES

[1] A. V. AHO, J. E. HOPCRAFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] J. L. BENTLEY AND J. H. FRIEDMAN, *Fast algorithms for constructing minimal spanning trees in coordinate spaces*, IEEE Trans. Comput., C27 (1978), pp. 97–105.

[3] C. BERGE AND A. GHOUILA-HOURI, *Programming, Games and Transportation Networks*, Methuen, London, 1965, pp. 179–180.

[4] O. BORUVKA, *On a minimal problem*, Práce Moravské Prirodovedecké Spolecnosti, 3 (1926), pp. 37–58.

[5] V. CHACHRA, P. M. GHARE AND J. M. MOORE, *Applications of Graph Theory Algorithms*, North-Holland, New York, 1979, pp. 67–73.

[6] D. CHERITON AND R. E. TARJAN, *Finding minimum spanning trees*, SIAM J. Comput., 5 (1976), pp. 724–742.

[7] G. CHOQUET, *Etude de certains réseaux de routes*, C. R. Acad. Sci., 206 (1938), pp. 310–313.

[8] N. CHRISTOFIDES, *Graph Theory: An Algorithmic Approach*, Academic Press, New York, 1975, pp. 135–142.

[9] R. CLARK AND W. F. MILLER, *Computer-based data analysis systems*, in Methods in Computational Physics, Vol. 5, B. Alder et al., eds, Academic Press, New York, 1966, pp. 47–98.

[10] R. B. DIAL, *Algorithm 360: shortest-path forest with topological ordering*, Comm. ACM, 12 (1969), pp. 632–633.

[11] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[12] H. GABOW, *Two algorithms for generating weighted spanning trees in order*, SIAM J. Comput., 6 (1977), pp. 139–150.

[13] J. GILSINN AND C. WITZGALL, *A performance comparison of labeling algorithms for calculating shortest path trees*, Technical Note 772, National Bureau of Standards, Washington, DC, May 1973.

[14] S. E. GOODMAN AND S. T. HEDETNIEMI, *Introduction to the Design and Analysis of Algorithms*, McGraw-Hill, New York, 1977.

[15] J. C. GOWER AND G. J. S. ROSS, *Minimum spanning trees and single linkage cluster analysis*, Appl. Statistics, 18 (1964), pp. 54–64.

[16] J. HELD AND R. M. KARP, *The traveling-salesman problem and minimum spanning trees*, Oper. Res., 18 (1970), pp. 1138–1162.

[17] ———, *The traveling-salesman problem and minimum spanning trees: Part* II, Math. Prog., 1 (1971), pp. 6–25.

[18] F. S. HILLIER AND G. J. LIEBERMAN, *Operations Research*, 2nd Edition, Holden-Day, San Francisco, 1974, pp. 220–224.

[19] E. HOROWITZ AND S. SAHNI, *Fundamentals of Computer Algorithms*, Computer Science Press, Potomac, MD, 1978, pp. 174–183.

[20] T. C. HU, *The maximum capacity route problem*, Oper. Res., 9 (1961), pp. 898–900.

[21] ———, *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA, 1969, pp. 129–131.

[22] D. B. JOHNSON, *Priority queues with update and finding minimum spanning trees*, Inform. Proc. Lett., 4 (1975), pp. 53–57.

[23] E. L. JOHNSON, *On shortest paths and sorting*, Proc. ACM National Conference, 1972, pp. 510–517.

[24] R. KALABA, *Graph theory and automatic control*, in Applied Combinatorial Mathematics, E. F. Beckenbach, ed., John Wiley, New York, 1964, pp. 237–252.

[25] A. KAUFMANN, *Graphs, Dynamic Programming, and Finite Games*, Academic Press, New York, 1967, pp. 31–33.

[26] A. KERSHENBAUM AND R. VAN SLYKE, *Computing minimum spanning trees efficiently*, Proc. ACM National Conference, 1972, pp. 518–527.

[27] J. B. KRUSKAL, JR., *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc., 7 (1956), pp. 48–50.

[28] H. LOBERMAN AND A. WEINBERGER, *Formal procedures for connecting terminals with a minimum total wire length*, J. ACM, 4 (1957), pp. 428–437.

[29] A. OBRUCA, *Algorithm* 1: MINTREE, Comp. Bull. 8:2, 67 (1964).

[30] R. E. OSTEEN AND P. P. LIN, *Picture skeletons based on eccentricities of points of minimum spanning trees*, SIAM J. Comput., 3 (1974), pp. 23–40.

[31] R. C. PRIM, *Shortest connection networks and some generalizations*, Bell Systems Tech. J., 36 (1957), pp. 1389–1401.

[32] C. PYNN AND J. H. WARREN, *Improved algorithm for the construction of minimal spanning trees*, Electronics Letters, 8 (1972), pp. 143–144.

[33] E. M. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[34] P. ROSENSTIEHL, *L'arbre minimum d'un graphe*, in Theory of Graphs, P. Rosenstiehl, ed., Gordon and Breach, New York, 1967.

[35] G. J. S. ROSS, *Algorithm* AS13: *minimum spanning tree*, Appl. Statistics, 18 (1969), pp. 103–104.

[36] J. J. SEPPANEN, *Algorithm* 399: *spanning tree*, Comm. ACM, 13 (1970), pp. 621–622.

[37] F. H. STILLINGER, JR., *Physical clustering, surface tension, and critical phenomena*, J. Chem. Phys., 47 (1967), pp. 2513–2533.

[38] R. VAN SLYKE AND H. FRANK, *Network reliability analysis: Part* I, Networks, 1 (1972), pp. 279–290.

[39] V. K. M. WHITNEY, *Algorithm* 422: *minimal spanning tree*, Comm. ACM, 15 (1972), pp. 273–274.

[40] A. C. YAO, *An* $O(|E| \log \log |V|)$ *algorithm for finding minimum spanning trees*, Inform. Proc. Lett., 4 (1975), pp. 21–23.

[41] C. T. ZAHN, *Graph-theoretical methods for detecting and describing gestalt clusters*, IEEE Trans. Comput., C20 (1971), pp. 68–86.

[42] ———, *Using the minimum spanning tree to recognize dotted and dashed curves*, in International Computing Symposium 1973, A. Gunther et al., eds., North-Holland, Amsterdam, 1974, pp. 381–387.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

ON NONLINEAR FUNCTIONS OF LINEAR COMBINATIONS*

PERSI DIACONIS† AND MEHRDAD SHAHSHAHANI‡

Abstract. Projection pursuit algorithms approximate a function of p variables by a sum of nonlinear functions of linear combinations:

$$f(x_1, \cdots, x_p) \doteq \sum_{i=1}^{n} g_i(a_{i1}x_1 + \cdots + a_{ip}x_p). \tag{1}$$

We develop some approximation theory, give a necessary and sufficient condition for equality in (1), and discuss nonuniqueness of the representation.

Key words. approximation theory, nonlinear high-dimensional nonparametric regression, polynomials, Schwartz distributions

1. Introduction and statement of main results.

We present some mathematical analysis for a class of curve fitting algorithms labeled "projection pursuit" algorithms by Friedman and Stuetzle (1981a, b). These algorithms approximate a general function of p variables by a sum of nonlinear functions of linear combinations:

$$f(x_1, \cdots, x_p) \doteq \sum_{i=1}^{n} g_i(a_{i1}x_1 + \cdots + a_{ip}x_p). \tag{1.1}$$

In (1.1), f is a given function and univariate, nonlinear functions g_i and linear combinations $a_{i1}x_1 + \cdots + a_{ip}x_p$ are sought so that a reasonable approximation is attained. Such approximation is computationally feasible and performs well in examples of nonparametric regression with noisy data, high-dimensional density estimation, and multidimensional spline approximation. In addition to the articles of Friedman and Stuetzle cited above, see Friedman and Tukey (1974), and Friedman, Grosse and Stuetzle (1983) for examples and computational details. Huber (1981a, b) begins to connect the algorithms to statistical theory. This note treats the algorithms from the point of view of approximation theory.

It is easy to show that approximation is always possible.

THEOREM 1. *Functions of the form* $\sum \alpha_i e^{\mathbf{a}^i \cdot \mathbf{x}}$, *with* α_i *real,* $\mathbf{a}^i$ *a vector of nonnegative integers, and* $\mathbf{x} = (x_1, \cdots, x_p)$ *are dense in the continuous real valued functions on* $[0, 1]^p$ *under the maximum deviation norm.*

Proof. The functions $e^{\mathbf{a} \cdot \mathbf{x}}$ separate points of $[0, 1]^p$ and are closed under multiplication. Finite linear combinations of such functions form a point separating algebra which is dense because of the Stone–Weierstrass theorem. □

THEOREM 2. *Functions of the form*

$$\sum \alpha_i \cos(2\pi \mathbf{a}^i \cdot x) + \beta_i \sin(2\pi b^i \cdot x)$$

are dense in $L^2[0, 1]^p$.

Proof. Any function in $L^2[0, 1]^p$ can be well approximated by its Fourier expansion. See Zygmund (1959, Vol. 2) and the survey article by Ash (1976) for further details and refinements. □

* Received by the editors September 26, 1981.

† Stanford University, Stanford, California 94305 and Harvard University, Cambridge, Massachusetts 02138.

‡ University of Virginia, Charlottesville, Virginia 22903 and Stanford University, Stanford, California 94305.

Sometimes equality is possible in (1.1). For example,

$$xy = \tfrac{1}{4}(x+y)^2 - \tfrac{1}{4}(x-y)^2,$$

$$\max(x, y) = \tfrac{1}{2}|x+y| + \tfrac{1}{2}|x-y|,$$

$$(xy)^2 = \tfrac{1}{4}(x+y)^4 + \frac{7}{4\cdot 3^3}(x-y)^4 - \frac{1}{2\cdot 3^3}(x+2y)^4 - \frac{2^3}{3^3}(x+\tfrac{1}{2}y)^4.$$

In what follows we will focus on conditions for equality in (1.1) as a method of determining examples to test, compare, and evaluate algorithms. Consider first a smooth function of 2 variables of the special form,

$$f(x, y) = g(ax+by).$$

Clearly,

$$\left(b\frac{\partial}{\partial x} - a\frac{\partial}{\partial y}\right)f \equiv 0.$$

If f has the form

$$f(x, y) = \sum_{i=1}^{n} g_i(a_i x + b_i y), \tag{1.2}$$

then the differential operator

$$\prod_{i=1}^{n}\left(b_i\frac{\partial}{\partial x} - a_i\frac{\partial}{\partial y}\right) = \sum_{i=0}^{n} c_i \frac{\partial^n}{\partial x^i\, \partial y^{n-i}}$$

applied to f is identically zero. The next theorem gives a converse.

THEOREM 3. *Let $f \in C^n[0, 1]^2$. Suppose that for some real numbers $c_0, \cdots, c_n$, the operator $\sum_{i=0}^{n} c_i\, \partial^n/\partial x^i\, \partial y^{n-i}$ applied to f is identically zero. If the polynomial $\sum_{i=0}^{n} c_i z^i$ has distinct real zeros then* (1.2) *holds for some* (a_i, b_i). *The lines $a_i x + b_i y$ are all distinct.*

Theorem 3 is proved in § 2 which also contains a discussion of techniques for finding directions (a_i, b_i) given f. Some applications of Theorem 3 are contained in the following examples.

Application 1. The functions e^{xy} and $\sin xy$ cannot be written in the form (1.1) for any finite n. Indeed, the equation $\sum c_i(\partial^n/\partial x^i\, \partial y^{n-i})\{e^{xy}\} \equiv 0$ implies $c_i \equiv 0$ and the associated polynomial has complex roots.

Application 2. Let $f(x, y)$ be a polynomial of degree m. Then

$$f(x, y) = \sum_{i=1}^{m} g_i(a_i x + b_i y),$$

where each g_i is a polynomial of degree at most m. This follows by elementary manipulations from Theorem 3. Thus, any polynomial in two variables can be represented exactly. Since polynomials are dense in $C[0, 1]^2$, this gives another proof of denseness of projection pursuit approximations. A different proof of this result is in Logan and Shepp (1975). An extension to more than two variables is in Proposition 1 of § 2.

Application 3. Representations of the form (1.1) are not necessarily unique. For example,

$$xy = c(ax+by)^2 - c(ax-by)^2$$

for any a and b satisfying $ab \neq 0$, $a^2+b^2=1$ with $c=1/4ab$. Writing $a=\cos\theta$, $b=\sin\theta$, any noncoordinate direction can be chosen for the quadratic g_1. The second direction is forced as orthogonal to this. This suggests that substantive interpretation of the linear combinations (a_i, b_i) is difficult. For a more ambitious example, consider the function $(xy)^2$. This is of 4th degree. Use of Theorem 3 as outlined in § 2, shows that $(xy)^2$ cannot be expressed as a sum of $n=3$ or fewer terms in (1.1). Four terms of 4th degree suffice:

$$(xy)^2=\alpha_1(x+b_1y)^4+\alpha_2(x+b_2y)^4+\alpha_3(x+b_3y)^4+\alpha_4(x+b_4y)^4,$$

where b_1, b_2, b_3, b_4 are chosen as distinct, and satisfying

$$b_1b_2+b_1b_3+b_1b_4+b_2b_3+b_2b_4+b_3b_4=0.$$

Then α_1, α_2, α_3, α_4 are determined by

$$\alpha_i=\frac{1}{6}\frac{\sum^* b_j}{\prod^*(b_j-b_i)},$$

where the sum and product are over $j\neq i$. This clearly defines a three-dimensional family of solutions.

In thinking about nonuniqueness, we observed that the only examples of nonunique representation we could find are polynomials. Indeed, polynomials have the following strong nonuniqueness property.

DEFINITION. A function $f(x, y)$ has *strongly nonunique representations* if there are two sets of directions $\{(a_i, b_i)\}_{i=1}^n$, $\{(\alpha_i, \beta_i)\}_{i=1}^m$, all distinct from each other, such that

$$f(x,y)=\sum_{i=1}^n g_i(a_ix+b_iy)=\sum_{i=1}^m h_i(\alpha_ix+\beta_iy)$$

for some g_i and h_i.

Polynomials have strongly nonunique representations: if $\deg P(x, y)=n$, and $(a_i, b_i)_{i=1}^{n+1}$ are any distinct directions, Theorem 1 implies that $P(x, y)$ can be represented in these directions. It turns out that *only* polynomials have this property. This is a consequence of Theorem 4.

THEOREM 4. *Let $f(x, y)\in C^{n+m}[0, 1]^2$. Suppose that for some directions $\{a_i, b_i\}_{i=1}^n$ and $\{(\alpha_i, \beta_i)\}_{i=1}^m$,*

$$f(x,y)=\sum_{i=1}^n g_i(a_ix+b_iy)=\sum_{i=1}^m h_i(\alpha_ix+\beta_iy).$$

If, for some i, (α_i, β_i) is distinct from (a_j, b_j), $1\leq j\leq n$, then h_i is a polynomial of degree at most $n+m-2$.

Proof. Let

$$A=\prod_{j\neq i}\left(\beta_j\frac{\partial}{\partial x}-\alpha_j\frac{\partial}{\partial y}\right),$$

and

$$B=\prod_{j=i}^n\left(a_j\frac{\partial}{\partial x}-b_j\frac{\partial}{\partial y}\right).$$

Then,

$Af=\sum Ag_i$ is a sum of functions in directions (a_i, b_i);

so

$$0 = BAF = Ch_i^{(m+n-1)}(\alpha_i x + \beta_i y)$$

for

$$C = \left\{ \prod_{j=1}^{n} (\alpha_i b_j - \beta_i a_j) \right\} \cdot \left\{ \prod_{j \neq i} (\alpha_i \beta_j - \beta_i \alpha_j) \right\} \neq 0.$$

Thus, h_i is a polynomial of degree at most $m+n-2$. □

COROLLARY. *A function $f(x, y)$ has strongly nonunique representations if and only if f is a polynomial.*

Remark. The statement and proof of Theorem 4 carry over to functions of more than two variables in a straightforward way. See Lemma 1 of § 2.

How are the curve fitting algorithms affected by nonuniqueness? To understand this, we performed the following experiment. On each trial 200 independent, mean zero, variance 1, normal points (x_i, y_i) were generated. The algorithm of Friedman and Stuetzle was given x_i, y_i, and $x_i y_i + \varepsilon_i$, with ε_i normally distributed, mean zero, variance .1 errors. We expected the directions fit to change a great deal. In each of 100 trials the algorithm fit univariate functions in directions (1, 1) and (1, −1) (to two decimal places).

To understand this, it is important to consider the nature of the algorithm. At each stage, it chooses the direction which minimizes the residual sum of squares when the best fitting function in that direction is subtracted off. See Friedman and Stuetzle (1981) for a careful description. If the sample size is large, the algorithm will behave in the same way as the infinite population analogue. Thus, let X, Y be independent Gaussian variables with mean zero and variance 1. Consider approximating XY by the best linear combination of the form $aX + bY$. For fixed a and b, the L^2 norm is minimized by the function $E(XY|aX+bY)$. Which values of a and b, subject to $a^2 + b^2 = 1$, minimize

$$E\{XY - E(XY|aX+bY)\}^2?$$

Let us show that the minimum is achieved at $a = \pm b = \pm 1/\sqrt{2}$. Let $U = aX + bY$, $V = aX - bY$. Then U and V are independent standard normal and

$$XY = \frac{1}{4ab}\{U^2 - V^2\}, \qquad E(XY|aX+bY) = \frac{1}{4ab}\{U^2 - 1\}.$$

Then,

$$E\{XY - E(XY|aX+bY)\}^2 = \frac{1}{(4ab)^2} E\{(U^2 - V^2) - (U^2 - 1)\}^2$$

$$= \frac{1}{(4ab)^2} E\{V^2 - 1\}^2.$$

Since the distribution of V does not depend on a and b, the right side is minimized when $a^2 = b^2 = \frac{1}{2}$. When the best fitting function is subtracted off, the second stage of the algorithm subtracts off a quadratic in the orthogonal direction and the algorithm terminates after two steps. The same result can be shown to hold when X and Y are chosen uniformly in $[-1, 1]^2$.

Similar computations can be instructively carried out for functions other than XY. For example, consider $(XY)^2$. David Donoho has shown that if X and Y are normally distributed the algorithm chooses the four directions (1, 1), (1, −1), (1, 0), (0, 1). Moreover, Donoho can prove that the approximation does not terminate after four steps, even though the function can be expressed as a sum of four 4th degree polynomials. Infinitely many steps are required—successive terms being cyclically added in each of the four directions.

Donoho and Ian Johnstone have independently proved that for normally distributed X and Y, the greedy approximation, which at each stage finds the a and b to minimize

$$E\{f_n(X, Y)-E\{f_n(X, Y)|aX+bY\}\}^2,$$

converges in L^2.

These results underscore a property of the projection pursuit algorithm: the directions it chooses are the directions that minimize the L^2 error. The situation is somewhat like finding the principal components of a covariance matrix. There are many possible bases, but the directions chosen have a well-defined interpretation in terms of maximum reduction of variance.

Application 4. Even if the directions (a_i, b_i) are fixed, the representation need not be unique. Suppose that n is the smallest integer such that

$$f(x, y)=\sum_{i=1}^{n} g_i(a_ix+b_iy).$$

If also

$$f(x, y)=\sum_{i=1}^{n} h_i(a_ix+b_iy),$$

then

$$f_i(t)-h_i(t)=p_i(t), \qquad 1\leqq i\leqq n,$$

with p_i a polynomial of degree at most $n-1$. The polynomials p_i can be chosen in an arbitrary way subject to the constraint $\sum p_i\equiv 0$. In particular, any $n-1$ of the p_i can be chosen arbitrarily and a final polynomial can be found to satisfy the constraint. These results all follow easily from Theorem 3; indeed the operator $L_i=\prod_{j\neq i}[b_j\,\partial/\partial x-a_j\,\partial/\partial y]$ applied to $f(x, y)$ gives

$$h_i^{(n-1)}(a_ix+b_iy)\prod_{j\neq i}(b_ja_i-a_jb_i)=g_i^{(n-1)}(a_ix+b_iy)\prod_{j\neq i}(b_ja_i-a_jb_i).$$

The products are nonvanishing because the directions are distinct. It follows that h_i differs from g_i by at most a polynomial of degree $n-1$, and that an arbitrary polynomial may be added subject to the constraint.

In the special case $n=2$, Theorem 3 was given by Dotson [4] who suggests further application to factoring probability densities and separation of variables.

The generalization to dimension greater than two is not as neat. We give a result for three-dimensions which generalizes to p-dimensions. Suppose that for n distinct directions $a^i\in\mathbb{R}^3$, a function f can be represented, for $x\in\mathbb{R}^3$, as

$$f(x)=\sum_{i=1}^{n} g_i(a^i\cdot x). \tag{1.3}$$

Let $\Pi^i=\{\rho\in\mathbb{R}^3: \rho\cdot a^i=0\}$. Let $\nabla=(\partial/\partial x_1, \partial/\partial x_2, \partial/\partial x_3)$. Clearly,

$$\prod_{i=1}^{n}(\rho^i\cdot\nabla)f\equiv 0 \quad \text{for all } \rho^i\in\Pi^i.$$

This condition is not sufficient. To see this, consider the function $f(x_1, x_2, x_3) = x_1x_2$ and the three directions $a^1 = (1, 0, 0)$, $a^2 = (0, 1, 0)$ and $a^3 = (0, 0, 1)$. For any nonzero $\rho^i \in \Pi^i$, the operator $\Pi(\rho^i \cdot \nabla)$ applied to f is zero. Yet, f cannot be written as $f(x) = f_1(x_1) + f_2(x_2) + f_3(x_3)$.

The condition is sufficient "up to polynomials":

THEOREM 5. *Let a^r be distinct, nonzero, directions in $\mathbb{R}^3$. Let Π^r be the plane $\{\rho \in \mathbb{R}^3: a^r \cdot \rho = 0\}$. A function $f \in C^n(\mathbb{R}^3)$ has the form*

$$f(x) = \sum_{r=1}^{n} g_r(a^r \cdot x) + P(x)$$

for a polynomial P of degree less than n, if and only if

$$\prod_{i=1}^{n} (\rho^i \cdot \nabla) f = 0 \quad \textit{for all } \rho^r \in \Pi^r. \tag{1.4}$$

Remark. As noted above, condition (1.4) is not sufficient to ensure that representation (1.3) holds. If (1.3) holds, there are other obvious necessary conditions: if $\rho^{ij} \in \Pi^i \cap \Pi^j$, then

$$(\rho^{ij} \cdot \nabla) g_i(a^i \cdot x) = (\rho^{ij} \cdot \nabla) g_j(a^i \cdot x) = 0.$$

Thus, f is annihilated by differential operators of degree $[(n+1)/2], \cdots, n-1$. Unfortunately, even these conditions are not sufficient. H. Royden, in unpublished work, has determined necessary and sufficient conditions of a rather different type. These are stated at the end of this paper. In Proposition 1 of § 2 we show that any polynomial P can be written as a sum of univariate polynomials of linear combinations. If deg $P = k$, then $(k+1)(k+2)/2$ terms may be required.

Thus far we have been assuming sufficient differentiability. Versions of all theorems are valid if derivatives are interpreted in the sense of distributions. This is discussed in some detail in § 3.

Our theorems are related to Hilbert's 13th problem. In modern notation, Hilbert asked if there are genuine multivariate functions. Of course, $x + y$ is a function of two variables but $xy = e^{\log x + \log y}$ is a superposition of univariate functions and $+$. Kolmogorov and Arnold showed that, in this sense, $+$ is the only function of two variables. They constructed five monotone functions $\phi_i: [0, 1] \to \mathbb{R}$, which satisfy $|\phi_i(x) - \phi(y)| \leq |x - y|$. These functions have the following remarkable property: for each $f \in C[0, 1]^2$ there is a $g \in C[0, 1]$ such that for all (x, y),

$$f(x, y) = \sum_{i=1}^{n} g(\phi_i(x) + \tfrac{1}{2}\phi_i(y)).$$

Thus ϕ_i are a "universal change of variables" which allows exact equality. A nice discussion of this result and its refinements can be found in Lorentz (1966), (1976) and Vitushkin (1977). While the functions ϕ_i and g are given in a constructive fashion, it does not seem that this result is used to approximate functions in an applied context. This is probably because the functions ϕ_i are fairly "wild". For example, it is known that it is not possible to choose ϕ_i to be C^1 functions, so fixed linear combinations of x and y are ruled out. It is known that $f(x, y) = \sum_{i=1}^{n} g_i(a_ix + b_iy)$ for all polynomials $f(x, y)$ is not possible with a_i, b_i fixed independent of f. In the projection pursuit approach to approximation, a_i and b_i are allowed to depend on f and Example 2 shows that now any polynomial can be written in required form. Example 1 shows that not all functions can be so expressed.

This paper has characterized functions that can be represented *exactly* as a sum of nonlinear functions of linear combinations. It is important to be able to recognize functions that can be well approximated by such a sum. Some important work on this problem is in the papers by Logan and Shepp (1975) and Logan (1975). These papers work with prespecified directions, but the main results of Logan (1975) do not depend on the directions. Roughly, Logan shows that a function on the unit disc can be well approximated, in L^2, by a sum of n univariate functions if and only if the function has bandwidth n, in the sense that its Fourier transform is essentially supported on a disc of radius n.

2. Proof and discussion of Theorems 3 and 5. Let L be the differential operator: $\sum_{i=0}^{n} c_i\, \partial^n/\partial x^i\, \partial y^{n-i}$. By hypothesis, the polynomial

$$\sum_{i=0}^{n} c_i x^i y^{n-i} = y^n \sum_{i=0}^{n} c_i\left(\frac{x}{y}\right)^i$$

splits into distinct linear factors. Thus, L can be written as $\prod [b_i\, \partial/\partial x - a_i\, \partial/\partial y]$, with the lines $a_i x + b_i y$ distinct. It must be shown that f can be represented as $\sum_{i=1}^{n} g_i(a_i x + b_i y)$. The proof is by induction on n. For $n = 1$, suppose without real loss that $a_1 \neq 0$. Then $f(x, y) = g(a_1 x + b_1 y)$ with $g(z) = f(z/a_1, 0)$. One way to show this is to fix (x, y) and define $h(t) = f(x + (b_1/a_1)y - (b_1/a_1)yt, ty)$. Then $h(0) = f(x + (b_1/a_q)y, 0) = g(a_1 x + b_1 y)$; $h(1) = f(x, y)$ and $h'(t) \equiv 0$, for $0 \leqq t \leqq 1$. The fundamental theorem of calculus gives $h(1) = \int_0^1 h' + h(0)$. Suppose the result is true for operators of degree $\leqq n - 1$. To prove it for degree n, write

$$\prod_{i=1}^{n}\left(b_i \frac{\partial}{\partial x} - a_i \frac{\partial}{\partial y}\right) f = \left\{\prod_{i=1}^{n-1}\left(b_i \frac{\partial}{\partial x} - a_i \frac{\partial}{\partial y}\right)\right\}\left(b_n \frac{\partial}{\partial x} - a_n \frac{\partial}{\partial y}\right) f \equiv 0.$$

By the induction hypotheses, there are functions g_i, $1 \leqq i \leqq n - 1$ satisfying

$$\left(b_n \frac{\partial}{\partial x} - a_n \frac{\partial}{\partial y}\right) f = \sum_{i=1}^{n-1} g_i(a_i x + b_i y). \tag{2.1}$$

A solution f^* of (2.1) of the form

$$f^*(x, y) = \sum_{i=1}^{n-1} h_i(a_i x + b_i y)$$

is found by choosing $h_i(t) = (b_n a_i - a_n b_i)^{-1} \int_0^t g_i(s)\, ds$. This is well-defined because the lines are distinct. Now $\{b_n\, \partial/\partial x - a_n\, \partial/\partial y\}(f - f^*) \equiv 0$ can be solved explicitly with $(f - f^*)(x, y) = h_n(a_n x + b_n y)$ by the argument for $n = 1$. It follows that $f = f^* + h_n$ can be written in the required form. □

Remarks on explicit computations. If f is of the form (1.2) then Theorem 3 gives the existence of numbers $c_0, \cdots, c_n$ such that $\sum c_j(\partial^n/\partial x^i\, \partial y^{n-i})(f) \equiv 0$. The c_i can be found by fixing $n + 1$ distinct pairs (x_i, y_i), calculating $\partial^n/\partial x^i\, \partial y^{n-i}|_{(x_i, y_i)}$ and solving the resulting system of equations for c_i. It is feasible to check if the polynomial $c_0 + \cdots + c_n z^n$ has distinct real roots using techniques in Henrici (1977, Chap. 6). Each stage of the procedure is feasible by a finite algorithm. If the procedure fails at any stage, then equality is impossible. Given feasible $c_0, \cdots, c_n$, it may be possible to find the roots of the associated polynomial. This determines directions (a_i, b_i).

In simple examples there is often enough freedom of choice to make determination of (a_i, b_i) possible. Consider $f(x, y) = xy$ for $n = 2$,

$$\prod_{i=1}^{2}\left(b_i\frac{\partial f}{\partial x} - a_i\frac{\partial f}{\partial y}\right) = b_1b_2\frac{\partial^2 f}{\partial x^2} - (b_1a_2 + b_2a_1)\frac{\partial^2 f}{\partial x\,\partial y} + a_1a_2\frac{\partial^2 f}{\partial y^2}.$$

Since $\partial^2 f/\partial x^2 = \partial^2 f/\partial y^2 = 0$, $\partial^2 f/\partial x\,\partial y = 1$; any distinct choice of a_i and b_i with $b_1a_2 = -b_2a_1$ works. Taking $a_1 = b_1 = 1$, $a_2 = -b_2 = 1$, we are led to solve

$$f(x, y) = g_1(x + y) + g_2(x - y).$$

Applying $\partial/\partial x - \partial/\partial y$ to both sides leads to $y - x = 2g_2'(x - y)$; setting $y = 0$; $g_2'(x) = -x/2$, $g_2 = -x^2/4 + c_2$. Similarly, $g_1(x) = x^2/4 + c_1$ and the result is $xy = \frac{1}{4}(x + y)^2 + c_1 - \frac{1}{4}(x - y)^2 + c_2$ where $c_1 + c_2 = 0$ is forced. In general, if $f = \sum_{i=1}^{n} g_i(a_ix + b_iy)$; $\prod_{j\neq i}(b_i\,\partial/\partial x - a_i\,\partial/\partial y)f = c_ig_i^{(n-1)}(a_ix + b_iy)$, for an explicit c_i. This determines g_i up to an essentially free choice of an $n - 1$ degree polynomial.

In the case of a polynomial f, some additional tricks become available. For a multinomial $x^a y^b$ let $a + b = n$; only sums of the form $\sum_{i=1}^{n}\alpha_i(x + \beta_j y)^n$ need be considered. Expanding out and equating coefficients gives

$$\sum\alpha_i = 0, \qquad \sum\alpha_i\beta_i = 0 \cdots \sum\alpha_i\beta_j^a = \frac{1}{\binom{n}{j}} \cdots \sum\alpha_i\beta_j^n = 0.$$

This gives $n + 1$ equations in $2n$ unknowns. These are linear in the α's for given β's and may be solved explicitly because the matrix is a Vandermonde with a well-known inverse. See Gautschi (1963).

The proof of Theorem 5 was outlined by H. Royden. The proof follows from three lemmas. Throughout a^r are distinct nonzero directions in $\mathbb{R}^3$.

LEMMA 1. *If* $\sum_{r=1}^{l} g_r(a^r \cdot x) \equiv 0$ *then* $g_r^{(l-1)} \equiv 0$ *and* g_r *is a polynomial of degree at most* $l - 2$.

Proof. Fix r. For each $j \neq r$ there is $\rho^j \in \Pi^j$ but $\rho^j \cdot a^r \neq 0$. Apply the operator $\prod_{j\neq r}(\rho^j \cdot \nabla)$ to the sum to conclude $\prod_{j\neq r}(\rho^j \cdot a^j)g^{(l-1)}(\rho^j \cdot x) \equiv 0$. The coefficient is nonzero, so the conclusion follows. □

In the next two lemmas, the notation f_j means $\partial f/\partial x_j$.

LEMMA 2. *Let P and Q be polynomials of degree $\leqq k$ in (x_1, x_2, x_3). Suppose that in some open $\mathcal{O} \subset \mathbb{R}^3$*

$$P_3 = Q_2.$$

Then there is polynomial H of degree at most $k + 1$ such that

$$P = H_2 \quad \textit{and} \quad Q = H_3.$$

Proof. Argue in a cube $\{a \leqq x_1 \leqq \alpha,\ b \leqq x_2 \leqq \beta,\ c \leqq x_3 \leqq \gamma\}$ contained in $\mathcal{O}$. Let Γ be a path connecting (x_1, a, b) to (x_1, x_2, x_3) which lies entirely in the plane of constant x_1. The line integral

$$\int_\Gamma P(x_1, y, z)\,dy + Q(x, y, z)\,dz = H(x_1, x_2, x_3)$$

is independent of Γ in view of the hypothesis and Green's theorem. Furthermore,

$$dH = P\,dx_2 + Q\,dx_3.$$

Here d is exterior differentiation in the plane $x_1=\text{constant}$. Therefore

$$\frac{\partial H}{\partial x_2}=P \quad\text{and}\quad \frac{\partial H}{\partial x_3}=Q.$$

In particular, let Γ be the path

$$\begin{aligned} &t\to(x_1, a+t(x_2-a), b), && 0\leqq t\leqq 1,\\ &t\to(x_1, x_2, b+(t-1)(x_3-b)), && 1\leqq t\leqq 2.\end{aligned}$$

Then,

$$\begin{aligned} H(x_1,x_2,x_3)=(x_2-a)&\int_0^1 P(x_1, a+t(x_2-a), b)\,dt\\ +(x_3-b)&\int_1^2 P(x_1, x_2, (t-1)(x_3-b))\,dt\end{aligned}$$

which is clearly a polynomial of degree $\leqq k+1$. □

LEMMA 3. *Let $f\in C^{n+2}(\mathbb{R}^3)$ have the following properties: for n distinct directions a^r, with a^r distinct from* $(1,0,0)$,

(2.2a) $$f_2(x)=\sum_{r=1}^n g_r(a^r\cdot x)+P(x),$$

(2.2b) $$f_3(x)=\sum_{r=1}^n h_r(a^r\cdot x)+Q(x).$$

With P and Q polynomials of degree at most $n-1$; then there are univariate functions G_r, $1\leqq r\leqq n+1$, and a polynomial H of degree at most n such that

$$f(x)=\sum_{r=1}^n G_r(a^r\cdot x)+G_{n+1}(x_1)+H(x).$$

Proof. Because $f_{23}=f_{32}$, conditions (2.2a) and (2.2b) translate into

$$0=\sum_{r=1}^n (a_3^r g_r^1-a_2^r h_r^1)(a^r\cdot x)+P_3(x)-Q_2(x).$$

By hypothesis, for $i\neq r$ there are vectors $\rho^i\in\Pi^i$ with $\rho^i\cdot a^r\neq 0$. Let $A=\prod_{i\neq r}(\rho^i\cdot\nabla)$. Applying A to $P_3(x)-Q_2(x)$ gives zero because this polynomial is of degree at most $n-2$. Thus,

$$0\equiv c\{a_3^r g_r^{(n)}-a_2^r h_r^{(n)}\},$$

where

$$c=\prod_{i\neq r}(\rho^i\cdot a^r)\neq 0.$$

It follows that

(2.3) $$a_3^r g_r(a^r\cdot x)-a_2^r h_r(a^r\cdot x)=P_r(a^r\cdot x)$$

for P a polynomial of degree at most $n-1$. Because the a^r are distinct from $(1,0,0)$, either $a_2^r\neq 0$ or $a_3^r\neq 0$. Define the n functions G_r by

(2.4a) $$G_r'=g_r/a_2^r \quad\text{if } a_2^r\neq 0,$$

(2.4b) $$G_r'=h_r/a_3^r \quad\text{if } a_2^r=0.$$

Consider

$$\phi(x) = f(x) - \sum_{r=1}^{n} G_r(a^r \cdot x).$$

From the hypothesis, (2.3) and (2.4),

$$\phi_2(x) = f_2(x) - \sum a_2^r G_r'(a^r \cdot x) = \sum_{a_2^r = 0} g_r(a^r \cdot x) + P(x) = P^*(x)$$

with P^* a polynomial of degree at most $n-1$. Further,

$$\phi_3(x) = f_3(x) - \sum a_3^r G'(a^r \cdot x) = \sum \{h_r(a^r \cdot x) - a_3^r G'(a^r \cdot x)\} + Q.$$

Each term in the sum is a polynomial. If $a_2^r \neq 0$, the rth term equals

$$h_r(a^r \cdot x) - a_3^r g_r(a^r \cdot x)/a_2^r$$

a polynomial from (2.3). If $a_2^r = 0$, the rth term is zero. It follows that

$$\phi_3 = Q^*(x)$$

with Q^* a polynomial of degree at most $n-1$. To finish off, observe that $\phi_{23} = \phi_{32}$ gives $P_3^* = Q_2^*$. From Lemma 2, there is a polynomial H of degree at most n, such that $H_2 = P^*$ and $H_3 = Q^*$. The function $\psi = \phi - H$ has $\psi_2 = \psi_3 = 0$. Thus, ψ is only a function of x_1, as required. □

Proof of Theorem 5. Clearly, if f can be represented as a sum of n univariate functions plus a polynomial the differential operator kills f. The proof of the converse is by induction on n. For $n = 1$, we know that if $f_2 = f_3 = 0$ then f is a function of x_1 only. Rotating to bring the plane Π^1 into $\{\rho : \rho_2 = \rho_3 = 0\}$ proves the general case. Suppose that the result is true for $n-1$. Let $a^1, a^2, \cdots, a^{n-1}, a^n$ be n distinct nonzero directions. By rotating, we may assume that $a^n = (1\ 0\ 0)$. Then, for any $\rho^i \in \Pi^i$, $1 \leqq i \leqq n-1$,

$$\Pi(\rho^i \cdot \nabla) f_2 \equiv \Pi(\rho^i \cdot \nabla) f_3 \equiv 0.$$

The induction hypothesis yields that f satisfies conditions (2.2) of Lemma 3. The theorem follows. □

PROPOSITION 1. *Let p and k be positive integers. Let* $r = \binom{m+p-1}{m}$. *There are r distinct directions* $a^1, a^2, \cdots, a^r$ *in* $\mathbb{R}^p$ *such that any homogeneous polynomial f of degree m can be written as*

$$f(x) = \sum_{j=1}^{r} \alpha_j (a^j \cdot x)^m \textit{ for some real numbers } \alpha_r.$$

Proof. The space of homogeneous polynomials of degree m is an r-dimensional vector space over the real numbers. Let $m_i(x)$, $1 \leqq i \leqq r$ be an enumeration of the monomials. For each monomial, let D_i be the associated differential operator (e.g., if $m_i(x) = x_1^2 x_2 x_3$, $D_i = \partial^4/\partial x_1^2\, \partial x_2\, \partial x_3$). Observe that $D_i(a^r \cdot x)^m = m!\, m_i(a^r)$. For dimension reasons, to prove the proposition it suffices to show that directions a^i can be chosen so that the polynomials $(a^j \cdot x)^m$, $j = 1, \cdots, r$, are linearly independent. Suppose

$$\sum c_j (a^j \cdot x)^m = 0.$$

Applying D_i we get

$$\sum_{j=1}^{r} m_i(a^j) c_j = 0 \quad \text{for all } i = 1, \cdots, r.$$

For this system to have a nontrivial solution $(c_1, \cdots, c_r)$, we must have

$$(*) \qquad \det(m_i(a^j)) = 0.$$

We write $a^j = (a_1^j, \cdots, a_p^j)$. Since $\det(m_i(a^j))$ is a nontrivial algebraic expression with rational coefficients, if we choose the pr real numbers a^j in such a way that they are algebraically independent over Q, then

$$\det(m_i(a^j)) \neq 0,$$

contradicting $(*)$. □

3. Some generalizations. The theory presented so far does not apply to identities between nondifferentiable functions. Most of the results remain valid if differentiation is interpreted in the sense of distributions. Consider the identity

$$\max(x, y) = \tfrac{1}{2}(|x+y| + |x-y|).$$

For fixed y the function $\max(x, y)$ is constant for $x \leqq y$ and equal to x for larger x. Thus

$$\frac{\partial}{\partial x}\max(x, y) = \begin{cases} 0, & x < y, \\ 1, & x > y. \end{cases}$$

This function is also called the Heavyside function shifted to y. Its derivative is well known to be the delta function concentrated on the line $x = y$. This acts on $\phi \in C_0^\infty(\mathbb{R}^2)$ by $\delta(\phi) = \int \phi(t, t)\,dt$. Similarly, $\partial^2/\partial y^2 \max(x, y) = \delta$, so $\max(x, y)$ is a solution of the wave equation

$$\frac{\partial^2}{\partial x^2}U - \frac{\partial^2}{\partial y^2}U = 0.$$

The only solutions of this equation are of the form $U = f_1(x+y) + f_2(x-y)$ where f_1 and f_2 are distributions (see Schwartz (1966, p. 9) for some history). We further show, here and more generally, that if the solution U is a sufficiently well-behaved function, then the f_i are functions.

Any undefined terms in the following discussion can be found in Barros-Neto (1973) or Schwartz (1966). Let $\mathcal{D}(\mathbb{R}^2)$ be the space of test functions—compactly supported C^∞ functions. The dual space $\mathcal{D}'(\mathbb{R}^2)$ is the space of distributions on $\mathbb{R}^2$. For $\gamma = (a, b)$, the translate of $T \in \mathcal{D}'$ by γ is written T_γ. This acts on $\phi \in \mathcal{D}$ by $T_\gamma\{\phi(x)\} = T\{\phi(x - \gamma)\}$. The distribution $T \in D'(\mathbb{R}^2)$ *depends only on* $ax + by$ if for all real t, $T_{(bt, -at)} = T$. The following theorem collects together several results in Schwartz (1966, § II.5). It is the case $m = 1$ of the theorem at which we are aiming.

THEOREM 6. *Let* $T \in \mathcal{D}'(\mathbb{R}^2)$. *For* (a, b) *nonzero, the following conditions are equivalent*:

(a) *T depends only on $ax + by$.*

(b) $(b\,\partial/\partial x - a\,\partial/\partial y)T = 0$.

(c) *There is a distribution* $g \in \mathcal{D}'(\mathbb{R}^2)$ *such that for all* $\phi \in \mathcal{D}(\mathbb{R}^2)$,

$$T(\phi) = g\left\{\int \phi(au + bv, bu - av)\,dv\right\}$$

where g operates on the function of u inside the brackets.

Remark. If T and g are functions with $T(x, y) = g(ax + by)$ and $a^2 + b^2 = 1$, then part (c) becomes

$$T(\phi) = \iint T(x, y)\phi(x, y)\, dx\, dy = \iint g(ax + by)\phi(x, y)\, dx\, dy$$

$$= \iint g(u)\phi(au + bv, bu - av)\, du\, dv = g\left\{\int \phi(au + bv, bu - av)\, dv\right\}.$$

Notation. If T satisfies any of the three conditions of Theorem 6 we write $T = q_\gamma^*(g)$ where $\gamma = (a, b)$ and q_γ is the linear map from $\mathbb{R}^2 \to \mathbb{R}^1$ given by $q_\gamma(x, y) = ax + by$. We can now state the distribution version of Theorem 3.

THEOREM 7. *Let $T \in \mathscr{D}'(\mathbb{R}^2)$. Suppose that for some real numbers $c_0, c_1, \cdots, c_m$, the operator $\sum_{i=0}^m c_i\, \partial^m/\partial x^i\, \partial y^{m-i}$ applied to T is zero. If the polynomial $\sum c_i z^i$ has distinct real zeros then there are distinct nonzero $\gamma_1, \gamma_2, \cdots, \gamma_m$; $\gamma_i = (a_i, b_i)$ such that, writing q_i for q_{γ_i},*

$$T = \sum_{i=1}^m q_i^*(g_i) \quad \text{with } g_i \in \mathscr{D}'(\mathbb{R}). \tag{3.1}$$

Conversely, if (3.1) *holds, then $\prod (b_i\, \partial/\partial x - a_i\, \partial/\partial y)$ applied to f is zero.*

Proof. One direction is clear; the argument for the other direction is by induction on m. The case $m = 1$ follows from Theorem 6. Thus, assume the result for $m - 1$. Without loss of generality, assume $a_i^2 + b_i^2 = 1$ for $1 \leqq i \leqq m$. Then

$$\prod_{i=2}^m \left(b_i \frac{\partial}{\partial x} - a_i \frac{\partial}{\partial y}\right)\left(b_1 \frac{\partial}{\partial x} - a_1 \frac{\partial}{\partial y}\right) f = 0.$$

This implies

$$\left(b_1 \frac{\partial}{\partial x} - a_1 \frac{\partial}{\partial y}\right) f = \sum_{i=2}^m q_i^*(g_i). \tag{3.2}$$

We will show that (3.2) has a solution f^* of form

$$f^* = \sum_{i=2}^m q_i^*(h_i).$$

Supposing this, $(b_1\, \partial/\partial x - a_1\, \partial/\partial y)(f - f^*) = 0$, so by the result for $m = 1, f - f^* = q_1^*(g_1)$ for $g_1 \in \mathscr{D}'(\mathbb{R})$ giving the theorem. To complete the proof, let h_i be a distribution solution to

$$\frac{dh_i}{dt} = \frac{1}{b_1 a_i - a_1 b_i} g_i.$$

A solution exists by Schwartz (1966, Thm. IX, p. 130). We claim

$$\sum_{i=2}^m q_i^*(h_i) = f^*$$

is a solution to (3.2). To show this, we need the following relation:

$$aq^*(h') = \frac{\partial}{\partial x} q^*(h).$$

To prove this, consider $\phi \in C_0^\infty(\mathbb{R}^2)$. Now

$$\begin{aligned} aq^*(h')(\phi) &= ah'\left(\int \phi(au+bv, bu-av)\,dv\right) \\ &= -h\left(\int [a\phi_1(au+bv, bu-av)+b\phi_2(au+bv, bu-av)]\,dv\right). \end{aligned}$$

Since ϕ is compactly supported

$$\int \frac{d}{dv}(\phi(au+bv, bu-av))\,dv = 0.$$

Thus,

$$\int [b\phi_1(au+bv, bu-av)-a\phi_2(au+bv, bu-av)]\,dv = 0.$$

Using this gives

$$\begin{aligned} aq^*(h')(\phi) &= -h\left(\int [a^2\phi_1(au+bv, bu-av)+b^2\phi_1(au+bv, bu-av)]\,dv\right), \\ &= -h\left(\int \phi_1(au+bv, bu-av)\,dv\right) \\ &= \frac{\partial}{\partial x} q^*(h)(\phi). \end{aligned}$$

Thus, $(b_1\,\partial/\partial x - a_1\,\partial/\partial y)q_i^*(h_i) = q_i^*(g_i)$. The claim regarding f^* follows. □

The next theorem shows that if the equation

$$f(x, y) = \sum_{i=1}^n q_i^*(g_i)$$

holds in the sense that the two sides are equal as distributions, and if $f(x, y)$ is a sufficiently regular function, then each of the distributions g_i can be realized as a function on $\mathbb{R}$. Theorems of this sort may be described as results on propagation of singularities of partial differential equations.

The notion of "sufficiently regular" which we adopt involves the Sobolev spaces H^s; the definitions involve Fourier transforms, and so the space $\mathcal{S}$ of C^∞ functions that, together with all derivatives, tend to zero at infinity faster than any polynomial. The dual of $\mathcal{S}$, denoted $\mathcal{S}'$ is the space of tempered distributions. The Fourier transform of $\phi \in \mathcal{S}(\mathbb{R}^2)$ is

$$\hat{\phi}(\lambda) = \iint e^{-i\lambda\cdot x}\phi(x)\,dx,$$

where dx is $1/2\pi$ times Lebesgue measure. The Fourier inversion theorem becomes

$$\phi(x) = \iint e^{i\lambda\cdot x}\hat{\phi}(\lambda)\,d\lambda.$$

The Fourier transform of a tempered distribution $\theta \in \mathcal{S}'$ is defined by

$$\hat{\theta}(\phi) = \theta(\hat{\phi}).$$

For real s, $-\infty < s < \infty$, the Sobolev space $H^s(\mathbb{R}^2)$ is the set of tempered distributions $\theta \in S'(\mathbb{R}^2)$ such that $(1+|\rho|^2+|\eta|^2)^{s/2}\hat{f}(\rho, \eta) \in L^2(\mathbb{R}^2)$. There are various embedding theorems that say when a distribution is a function. For example, Taylor (1980, Chap. 1, § 3) gives:

(a) If $s > n/2$, then each $\theta \in H^s(\mathbb{R}^n)$ is a bounded continuous function that vanishes at infinity.

(b) If $s > n/2 + k$, then $H^s(\mathbb{R}^n) \subset C^k(\mathbb{R}^n)$.

(c) For $0 < \alpha < 1$, define C^α as the set of bounded functions u such that $|u(x+y) - u(y)| < C|y|^\alpha$ for $|y| \leqq 1$. If $s = n/2 + \alpha$, $0 < \alpha < 1$, then $H^2(\mathbb{R}^n) \subset C^\alpha(\mathbb{R}^n)$.

(d) If $0 \leqq s < n/2$, $H^s(\mathbb{R}^n) \subset L^q(\mathbb{R}^n)$, $q = 2n/(n-2s)$.

We have chosen the route of interpolating between integer values of s by means of the Fourier transform. There are other routes. See Adams (1975) for discussion.

For U open in $\mathbb{R}^2$, $H^s_{\text{loc}}(U)$ is the set of distributions $\theta \in \mathcal{D}'(U)$ such that for each compactly supported $\phi \in C_0^\infty(U)$, $\phi \cdot \theta \in H^s(\mathbb{R}^2)$. For example, $\max(x, y) \in H^1_{\text{loc}}(\mathbb{R}^2)$. With this notation, we can state the main result.

THEOREM 8. *Let $\gamma_i = (a_i, b_i)$, $1 \leqq i \leqq m$ be distinct nonzero directions in $\mathbb{R}^2$. Let q_i denote projection in the direction γ_i, so $q_i(x, y) = a_i x + b_i y$. Let U be open in $\mathbb{R}^2$. Let U_i be open sets in $\mathbb{R}$ with $q_i^{-1}(U_i) \supset U$ for all i. Suppose $f \in H^s_{\text{loc}}(U)$ can be written*

$$f = \sum_{i=1}^m q_i^*(g_i),$$

where $g_i \in \mathcal{D}'(U_i)$. Then $g_i \in H^s_{\text{loc}}(U_i)$.

The proof of Theorem 8 will be given following two preliminary lemmas. Let (a, b) be a unit vector in $\mathbb{R}^2$. Let $q(x, y) = ax + by$ denote the projection. For $g \in \mathcal{S}'(\mathbb{R})$, the distribution q^*g acts on $\psi \in \mathcal{S}(\mathbb{R}^2)$ as $g(\int \psi(au + bv, bu - av)\, dv)$. The distribution p_*g acts on functions $\psi \in \mathcal{S}(\mathbb{R}^2)$ by $g(\psi(at, bt))$. We have the next lemma.

LEMMA 4. $\widehat{(q^*g)} = p_*(\hat{g})$.

Proof. $\widehat{q^*g}(\psi) = q^*g(\hat{\psi}) = g\{\int \hat{\psi}(au + bv, bu - av)\, dv\}$. Now the integral equals

$$\begin{aligned}\int e^{-i(au+bv)x - i(bu-av)y}\phi(x, y)\, dx\, dy\, dv &= \int e^{-i(ax+by)u - i(bx-ay)v}\phi(x, y)\, dx\, dy\, dv \\ &= \int e^{-isu-itv}\phi(as+bt, bs-at)\, ds\, dt\, dv \\ &= \int e^{-isu}\left\{\int e^{-itv}\phi(as+bt, bs-at)\, dt\, dv\right\} ds.\end{aligned}$$

The inner integral equals $\phi(as, bt)$; indeed for any function $g \in C_0^\infty(\mathbb{R})$, $\int e^{itv}g(t)\, dt\, dv = \hat{\delta}(\hat{g}) = \delta(g) = g(0)$. Making this substitution, proves the result. □

The next lemma is the case $m = 1$ of Theorem 8.

LEMMA 5. *Let $\pi: \mathbb{R}^2 \to \mathbb{R}$ be the projection $\pi(x, y) = x$, $U_1 \subset \mathbb{R}$ and $U \supset \pi^{-1}(U_1)$. Let $T \in \mathcal{D}'(U)$ and assume there is $g \in \mathcal{D}'(U_1)$ such that*

$$T = \pi^*(g).$$

If $T \in H^s_{\text{loc}}(U)$, then $g \in H^s_{\text{loc}}(U_1)$.

Proof. Without loss of generality we may assume $U = \pi^{-1}(U_1)$ since $T = \pi^*(g)$. Let $V_1 \subset \bar{V}_1 \subset U_1$ and V_1 open, $\bar{V}_1$ compact, and $\chi \in C_0^\infty(U_1)$ with $\chi \equiv 1$ on $\bar{V}_1$. Then it suffices to show $\chi g \in H^s(\mathbb{R})$. Let $f = \chi g$ and $\theta = \pi^*(\chi g)$. Then the hypothesis on T implies

$$\theta \in H^s_{\text{loc}}(\mathbb{R}^2).$$

Let $\phi(x, y) = \phi_1(y)$ with $\phi_1 \in C_0^\infty(\mathbb{R})$ such that

$$\phi_1(y) = \begin{cases} 1, & |y| \leqq r, \\ 0, & |y| \geqq R, \end{cases}$$

for some constants $0 < r < R$. Then $\phi\chi \in C_0^\infty(U)$, so

$$\phi\theta \in H^s(\mathbb{R}^2).$$

This means $\widehat{\phi\theta}$ is a function and

$$\iint |(\widehat{\phi\theta})(\rho, \eta)|^2(1+|\rho|^2+|\eta|^2)^{s/2}\, d\rho\, d\eta < \infty. \tag{3.3}$$

We will argue that (3.3) implies $f \in H^s(\mathbb{R})$. Lemma 4 implies that $\hat{\theta} = p_*(\hat{f})$. Moreover, $\hat{f}$ is an analytic function of one variable, being the Fourier transform of a distribution of compact support (see Barros-Neto (1973, § 4.5)). Thus

$$\hat{\theta}(\rho, \eta) = \hat{f}(\rho)\delta_0(\eta). \tag{3.4}$$

Also,

$$\hat{\phi}(\rho, \eta) = \delta_0(\rho)\hat{\phi}_1(\eta). \tag{3.5}$$

Using $(\widehat{\phi\theta}) = \hat{\phi} * \hat{\theta}$ with (3.4) and (3.5),

$$(\widehat{\phi\theta})(\rho, \eta) = \hat{\phi}_1(\eta) \cdot \hat{f}(\rho).$$

Thus, (3.3) becomes

$$\iint |\hat{\phi}_1(\eta)|^2|\hat{f}(\rho)|^2(1+|\rho|^2+|\eta|^2)^{s/2} < \infty. \tag{3.6}$$

Now elementary arguments show that for any real s there are positive constants ρ_1, ρ_2 such that

$$\rho_1|\rho|^s < \int |\hat{\phi}_1(\eta)|^2(1+|\rho|^2+|\eta|^2)^{s/2}\, d\eta < \rho_2|\rho|^s \quad \text{for } |\rho| \geqq 1.$$

Using this and (3.6) gives

$$\int_{-\infty}^{-1} |\hat{f}(\rho)|^2|\rho|^s\, d\rho + \int_1^\infty |\hat{f}(\rho)|^2|\rho|^s\, d\rho < \infty.$$

Hence, the desired result:

$$\int |\hat{f}(\rho)|^2(1+|\rho|^2)^{s/2}\, d\rho < \infty. \qquad \square$$

Proof of Theorem 8. We may assume γ_i are unit vectors. The case $m = 1$ follows from Lemma 5 via linear transformation. For the general case, suppose

$$f = \sum_{i=1}^m q_i^*(g_i).$$

Let $D_i = \prod_{j \neq i}(b_j\, \partial/\partial x - a_j\, \partial/\partial y)$. Then, for a nonzero constant C_i,

$$D_i f = D_i q_i^*(g_i) = C_i q_i^*(g_i^{(m-1)}).$$

Since $f \in H^s_{\mathrm{loc}}(U)$, $D_i f \in H^{s-m+1}_{\mathrm{loc}}(U)$. By Lemma 5,

$$g_i^{(m-1)} \in H^{s-m+1}_{\mathrm{loc}}(U_1).$$

This implies that $g_i \in H^s_{\mathrm{loc}}(U_1)$, see for example Treves (1966, Thm. 7.6).

Acknowledgment. We thank David Donoho, Jerry Friedman, Bob Hulquist, Winni Li and Bruce Reznick for helpful discussions. Two extremely helpful referees found a gap in the original version of Theorem 5. We are grateful to Halsey Royden for allowing us to use his elegant proof of the corrected version.

Note added in proof. Halsey Royden has communicated the following conditions necessary and sufficient for a function f of m variables to be representable as a sum of N univariate functions: Let $\alpha = (\alpha_1, \cdots, \alpha_m)$ denote a multi-index of weight $\sum \alpha_j$. Let f_α denote the appropriate partial derivative and $f_{\alpha j} = (\partial/\partial x_j) f_\alpha$.

THEOREM (H. Royden). *Let $N = \binom{m+l-1}{l}$. Then a smooth function $f(x_1, \cdots, x_m)$ can be written*

$$f = \sum_{\nu=1}^{N} g_\nu\left(\sum a_k^\nu x_k\right)$$

where the N vectors $a^\nu = (a_k^\nu)$ do not lie on a hypersurface of degree l in projective $m-1$ space, if and only if there are functions h_ν, an invertible $N \times N$ constant matrix $C = [C_\alpha^\nu]$ where α runs over the N multi-indices of weight l) and N $m \times m$ constant invertible matrices $B_\nu = [B_{k\nu}^j]$ such that

$$\sum_{\alpha, j} c_\nu^\alpha f_{\alpha j} B_{k\nu}^j = \delta_k^1 h_\nu(x_1, \cdots, x_m).$$

Here δ_k^1 is Kronecker's delta function. The functions $g_\nu^{(l)}$ are then uniquely defined (so the representation is unique up to polynomials of degree $l-1$) and the directions a_ν are unique for those ν's with $g_\nu^{(l)} \not\equiv 0$.

REFERENCES

R. A. ADAMS (1975), *Sobolev Spaces*, Academic Press, New York.

J. M. ASH (1976), *Studies in Harmonic Analysis*, American Mathematical Society, Providence, RI.

J. BARROS-NETO (1973), *An Introduction to the Theory of Distributions*, Marcel Dekker, New York.

W. G. DOTSON (1968), *Decomposability of positive functions on* $\mathbb{R}^n$, Amer. Math. Monthly, pp. 350–357.

J. FRIEDMAN AND J. TUKEY (1974), *A projection pursuit algorithm for exploratory data analysis*, IEEE Transactions Computers, C-23, pp. 881–889.

J. FRIEDMAN AND W. STUETZLE (1981a), *Projection pursuit regression*, J. Amer. Statist. Assoc., 76, pp. 817–823.

——— (1981b), *Projection pursuit methods for data analysis*, Technical Report, Stanford Linear Accelerator, Stanford, CA.

J. FRIEDMAN, E. GROSSE AND W. STUETZLE (1983), *Multi-dimensional additive spline approximation*, this Journal, 4 (1983), pp. 291–301.

W. A. GAUTSCHI (1963), *On inverses of Vandermonde and confluent Vandermonde matrices*, Numer. Math., 5, pp. 425–430.

P. HENRICI (1977), *Applied and Computational Complex Analysis*, Vol. I, John Wiley, New York.

P. HUBER (1981a), *Density estimation and projection pursuit methods*, Tech. Rep. PJH-7, Dept. Statistics, Harvard Univ., Cambridge, MA.

—— (1981b), *Projection pursuit*, Tech. Rep. Dept. Statistics, Harvard, Univ., Cambridge, MA.

B. F. LOGAN (1975), *The uncertainty principle in reconstructing functions from projections*, Duke Math. J., 42, pp. 661–706.

B. F. LOGAN AND L. A. SHEPP (1975), *Optimal reconstruction of a function from its projections*, Duke Math. J., 42, pp. 645–660.

G. G. LORENTZ (1966), *Approximation of Functions*, Holt, Rinehart, and Winston, New York.

——— (1976), *The 13th problem of Hilbert*, in Mathematical Developments Arising from Hilbert Problems, Proc. Symp. Pure Math. XXVIII, American Mathematical Society, Providence, RI.

L. SCHWARTZ (1966), *Théorie des distributions*, 2nd edition, Hermann, Paris.

M. E. TAYLOR (1981), *Pseudo-differential Operators*, Princeton Univ. Press, Princeton, NJ.

F. TREVES (1966), *Linear Partial Differential Equations with Constant Coefficients*, Gordon-Breach, New York.

D. G. VITUSHKIN (1977), *On representation of functions by means of superpositions and related topics*, L'Enseignement Math., July–Dec. 1977, pp. 256–319.

K. ZYGMUND (1959), *Trigonometric Series*, 2nd ed., Cambridge Univ., Cambridge.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

A MIXED VARIATIONAL INEQUALITY BOUNDARY ITERATION METHOD FOR SOME FREE BOUNDARY PROBLEMS*

D. R. WESTBROOK†

Abstract. A combination of variational inequalities and free boundary iterations is used to obtain approximate finite element solutions to some elliptic free boundary problems in an attempt to take advantage of the best aspects of both methods.

Variational inequalities are used together with a fixed mesh to first obtain a good approximation to the free boundary. This approximation is then improved iteratively by the movement of the free boundary nodes leaving the rest of the mesh unchanged.

Numerical results are given for two membrane contact problems and for a dam seepage problem.

Key words. free boundary problems, variational inequalities, finite elements

1. Introduction.

The numerical solution of free boundary problems for second order differential equations, such as the dam seepage problem, has a lengthy history (see Cryer [5] for an extensive review). Many of the earlier workers, e.g. Shaw and Southwell [10], Taylor and Brown [11], use trial free boundary methods. In these methods an initial guess at the free boundary and corresponding domain is made. The partial differential equation is solved approximately in the domain using one of the boundary conditions on the free boundary. The second condition on the free boundary is then used to move the boundary to a new position. The process then proceeds iteratively until practical convergence is attained. The mesh is usually changed as the boundary is moved.

More recently many such free boundary problems have been reformulated as variational inequalities (see for example Duvaut and Lions [6] where many problems in mechanics are considered). A principal reason for such a formulation is that it provides a means to prove existence and uniqueness theorems, but the formulation also suggests an alternative numerical method in which the mesh remains fixed and the approximate boundary is found without iteration.

The solution of the variational inequality does require iterations, but as already stated it is done on a fixed mesh and according to Baiocchi, Comincoli, Guerri and Volpi [2] requires much less work. Kikuchi [8] compares the solution of the isotropic dam seepage problem by variational inequalities and by Taylor's method and states:

> (1) The method of variational inequalities gives a very rough free surface if the number of meshes is small. In order to get a suitably smooth surface it needs more than ten times the number of Taylor's method. However, in Taylor's method the system of linear equations has to be solved several times to get convergence. In the method of variational inequalities, it is not necessary to solve the system of linear equations more than one time.
>
> (2) The method of variational inequalities can determine the seepage point without special considerations. However, Taylor's method needs some special considerations.
>
> (3) Taylor's method has no guarantee of convergence, but the method of variational inequalities does.

The present work is an attempt to combine the best aspects of the two methods for problems which can be formulated as variational inequalities. The principal idea is to use finite elements with the direct methods and fixed mesh of the variational inequality formulation to obtain a good first guess for an iterative trial free boundary method.

* Received by the editors July 14, 1982.

† Department of Mathematics and Statistics, University of Calgary, Calgary, Alberta, Canada T2N 1N4.

A main ingredient in the success of the method used here is that the nodes which give the approximate free boundary in the variational inequality method are within one element of the actual boundary (at least for a reasonably fine mesh). There is to the author's knowledge no proof of this conjecture, but no cases where it is not true have been encountered. If this conjecture holds, then only those nodes on the approximate boundary need be moved, and no special precautions need be taken to see that elements do not become too distorted. Here a simple check is made after each movement of the boundary.

In § 2 the problems considered are stated along with the formulation as variational inequalities. The numerical algorithm is in § 3 and the numerical results are given in § 4.

2. Description of problems.

2.1. The membrane contact problem. If a membrane has a given boundary displacement and is stretched over a fixed surface, a contact problem is obtained in which the membrane is in contact with the surface over an area which is a priori unknown. The boundary of the contact region is the free boundary in this case.

The displacement of the membrane is given by $u(x, y)$ and the fixed surface is represented by the function $\chi(x, y)$, both defined for (x, y) belonging to an open set Ω with boundary Γ. If the contact domain is denoted by D, the displacement u will satisfy the boundary condition $u = g$ on Γ where g is the given boundary displacement ($g \geqq \chi$ on Γ) and

$$\Delta u = 0, \quad u > \chi \quad \text{in } \Omega - D, \qquad u = \chi \quad \text{in } D.$$

In the region D the force between the membrane and the rigid surface is $-\Delta u$ which must obviously be positive. These conditions lead to the complementarity equations

$$\begin{aligned} -\Delta u = 0, \quad u > \chi \quad &\text{in } \Omega - D, \\ -\Delta u \geqq 0, \quad u = \chi \quad &\text{in } D, \end{aligned}$$

or

$$-\Delta u \geqq 0, \quad u \geqq \chi, \quad (u - \chi)\Delta u = 0 \quad \text{in } \Omega,$$

with $u = g$ on Γ. This problem may be stated in a weak formulation as a variational inequality (Kinderlehrer and Stampacchia [9, pp. 40–45]) in the following manner:

Let $K = \{v \mid v \in H^1(\Omega),\ v = g \text{ on } \Gamma,\ v \geqq \chi \text{ a.e. in } \Omega\}$. Then $u \in K$ is the solution of

$$a(u, v - u) \geqq 0 \quad \forall v \in K$$

where

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, dS$$

($H^1(\Omega)$ is the Sobolev space of functions with generalized first partial derivatives).

In practice it is more convenient to find $w = u - \chi$ which then satisfies

$$w \in K = \{v \mid v \in H^1(\Omega), v = g - \chi \text{ on } \Gamma, v \geqq 0 \text{ a.e. in } \Omega\},$$

and w is the solution of

$$a(w, v - w) \geqq l(v - u)$$

where

$$l(v-u)=\int_\Omega \Delta\chi(v-u)\,dS.$$

(It has been assumed here that χ is a sufficiently differentiable function, e.g. $\chi \in H^2(\Omega)$.)

In the present work two such membrane contact problems are considered. In both Ω is the interior of the square $|x|<1$, $|y|<1$, and $\chi=\frac{1}{4}(\frac{1}{2}-x^2-y^2)\cdot(\Delta\chi=-1)$ so that $l(v)=-\int_\Omega v\,dS$. In the first problem $g=0$ ($w=\frac{1}{4}(x^2+y^2-\frac{1}{2})$ on Γ), and in the second $g=\frac{1}{16}[1-\ln\{4(x^2+y^2)\}]$ ($w=\frac{1}{16}[-1+4(x^2+y^2)-\ln 4(x^2+y^2)]$ on Γ).

The second example has the exact solution

$$w=\begin{cases}\frac{1}{16}[-1+4(x^2+y^2)-\ln 4(x^2+y^2)], & x^2+y^2>\frac{1}{4},\quad (x,y)\in\Omega,\\ 0, & x^2+y^2\leqq\frac{1}{4}.\end{cases}$$

The free boundary is the circle $x^2+y^2=\frac{1}{4}$.

2.2. The isotropic homogeneous dam seepage problem. The problem may be stated as follows (see e.g. Baiocchi et al. [2]):

Find u such that

$$\begin{array}{lll}\Delta u=0 & \text{in } \mathscr{D}, & \\ u=h_1 & \text{on } AB & \multirow{2}{*}{\Big\}\text{(interface with water at rest),}} \\ u=h_2 & \text{on } CD & \\ u=y & \text{on } DE & \Big\}\text{(interface with air),} \\ u=y & \text{on } \Gamma & \\ \dfrac{\partial u}{\partial n}=0 & \text{on } \Gamma & (\Gamma \text{ is a streamline}), \\ \dfrac{\partial u}{\partial y}=0 & \text{on } BC & \text{(impervious boundary).}\end{array}$$

u represents the hydraulic head and Γ is the free surface. DE is the seepage surface and E the seepage point. The letters refer to Fig. 1 which gives a representation of the problem.

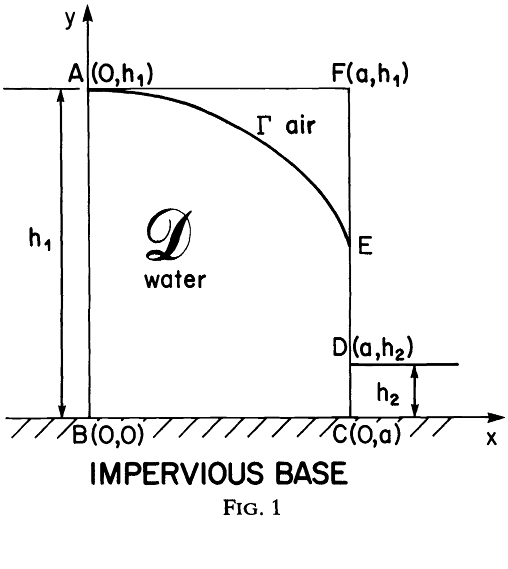


FIG. 1

This can be restated as a variational inequality by means of a method due to Baiocchi (see [2]) as follows. Let Ω be the interior of the rectangle $ABCF$ and define

$$\tilde{u} = \begin{cases} u & \text{in } \mathcal{D}, \\ y & \text{in } \Omega - \mathcal{D}. \end{cases}$$

Let

$$w(x, y) = \int_y^{h_1} (\tilde{u}(x, t) - t)\, dt$$

and

$$g = \begin{cases} \dfrac{h_1^2}{2} - \dfrac{(h_1^2 - h_2^2)}{2}\dfrac{x}{a} & \text{on } BC, \\ \frac{1}{2}(h_1 - y)^2 & \text{on } AB, \\ \frac{1}{2}(h_2 - y)^2 & \text{on } CD, \\ 0 & \text{on } DF \cap AF. \end{cases}$$

Then if $K = \{v \mid v \in H^1(\Omega), v = g \text{ on } \partial\Omega, v \geqq 0 \text{ a.e. in } \Omega\}$, $w \in K$ is the solution of

$$a(w, v - w) \geqq l(v - w) \quad \forall v \in K$$

where

$$a(v, w) = \int_\Omega \operatorname{grad} v \cdot \operatorname{grad} w \, dS,$$

$$l(v) = -\int_\Omega v \, dS.$$

This problem has been tackled numerically by several authors, for example Shaw and Southwell [11], Baiocchi et al. [2], Aitchison [1]. A much more complete survey is given in Cryer [4].

3. The numerical method. It is seen that the three problems of § 2 all reduce to essentially the same variational inequality on a rectangular region. In the membrane contact problems (2.1) symmetry is used to reduce the region to $0 \leqq x \leqq 1$, $-1 \leqq y \leqq 0$. The region is divided into triangles by three sets of parallel lines (see Fig. 2) and the nodes are denoted x_i, $i = 1, \cdots, N$. For convenience the region is oriented so that the contact region will be in the upper left corner.

The finite element method is used to discretize the variational inequality. The finite dimensional subspace V_N of $H^1(\Omega)$ is defined by

$$V_N = \left\{ v \,\middle|\, v = \sum_{i=1}^{N} v_i \phi_i,\ v_i = g(x_i),\ x_i \in \partial\Omega \right\}$$

where v_i are the nodal values $v(x_i)$ of v and ϕ_i are basis functions which are piecewise linear in each triangle and $\phi_i(x_j) = \delta_{ij}$. If V denotes the N-vector with components v_i, the closed convex subset K_N is defined by

$$K_N = \{V \mid v_i = g(x_i),\ x_i \in \partial\Omega,\ V \geqq 0 \text{ (i.e. } v_i \geqq 0, i = 1, \cdots, N)\}.$$

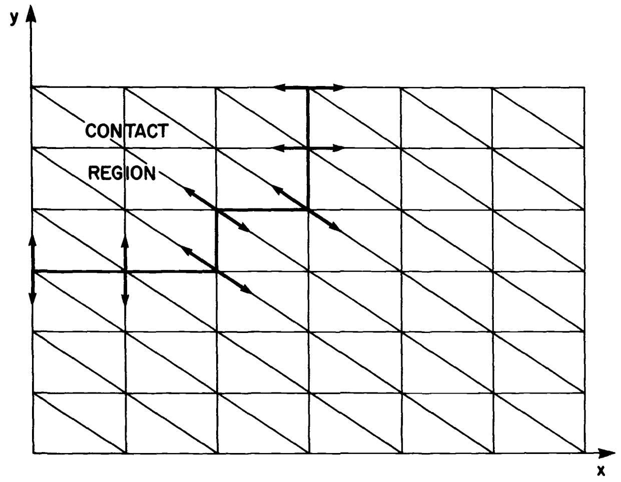


FIG. 2. *A typical finite element mesh with the approximate free boundary marked by the thick line (the contact region is in the upper left corner). The free boundary nodes are moved in the directions shown, i.e., parallel to the y-axis if its neighbors have the same x coordinates, parallel to the x-axis if its neighbors have the same y coordinates, along the diagonal otherwise.*

The approximate solution w with vector of nodal values W now satisfies the variational inequality:

Find $W \in K_N$ such that

$$(V-W)^T(AW-F) \geqq 0 \quad \forall V \in K_N$$

where A is the matrix with i, j entry a_{ij} given by $a_{ij} = a(\phi_i, \phi_j)$ and F is the vector with entries $f_i = l(\phi_i)$ (with adjustments due to boundary conditions). It may be seen that W satisfies

$$(3.1) \qquad AW-F \geqq 0, \quad W \geqq 0, \quad W^T(AW-F) = 0.$$

The equations (3.1), which are equivalent to a quadratic programming problem, are solved by a method of successive overrelaxation with projection, a method used by many authors, e.g. Glowinski, Lions and Tremolières [7, pp. 67–71], or Cryer [3] where proofs of convergence are given.

The iteration is started with an initial choice (in the current work $w_i^{(0)} = 0$) and the $(n+1)$st iteration is described as follows:

$$r_i^{(n)} = f_i - \sum_{j<i} a_{ij} w_j^{(n+1)} - \sum_{j \geqq i} a_{ij} w_j^{(n)},$$

$$w_i^{(n+1)} = \max\left\{0, w_i^{(n)} + \frac{\omega r_i^{(n)}}{a_{ii}}\right\}, \qquad i = 1, \cdots, N.$$

ω is the relaxation parameter. In the current work $\omega = 1.7$ was found to be satisfactory. Iterations were repeated until a stopping criterion of the following form was satisfied:

$$\left| \sum_{i=1}^{N} w_i^{(n+1)} - \sum_{i=1}^{N} w_i^{(n)} \right| < \varepsilon$$

where ε is a chosen tolerance.

The contact region is identified as the union of those triangles in which $w=0$ ($w_i=0$ at all three vertices) and the approximate free boundary is taken as the boundary of this region (note that Baiocchi et al. [2] use a slightly different choice for the identification of the free boundary).

At this stage the approximate free boundary suffers from the disadvantages noted by Kikuchi [8], but it does give a reasonably accurate approximation to the actual boundary, and from this point a trial free boundary iterative method is adopted to improve the location of the free boundary. Since $w_i=0$ at all free boundary nodes, one of the boundary conditions at the free boundary is satisfied. The second condition, which corresponds to the natural boundary condition at the boundary of the noncontact region, is used to move the boundary. Each boundary node is allowed to move in a specific "pseudonormal" direction which depends on the configuration of its neighbouring boundary nodes and which is recorded along with the node numbers of the boundary nodes as these nodes are identified. A diagram of the mesh and the three pseudonormal directions is given in Fig. 2. With the given mesh configuration one of three possible directions is determined for each node. The element numbers of all triangles, the boundary elements, with a free boundary node at a vertex and which lie in the noncontact region are also recorded. The algebraic equations which correspond to the natural boundary condition at the free boundary for the noncontact region depend on the boundary node coordinates, and these nodes are moved in the pseudo normal directions by amounts which are the solution of quasi-Newton equations with the node positions as variables and with fixed values for the w_i. The iterations are as follows beginning with the mesh points and w_i values obtained by the variational inequality method:

(i) Use the quasi-Newton equations derived from the natural boundary condition to obtain new coordinates for the free boundary nodes (w_i fixed in this step).

(ii) Recalculate the matrix A and load F of the variational inequality and resolve this problem using the current w_i as starting values for the S.O.R. routine.

(ii) Return to (i) until the maximum change in coordinates is below a given tolerance.

In step (i) the Jacobian is formed element by element using only the boundary elements. In this work the partial derivatives with respect to the nodal displacements were calculated by hand. In general a secant method might be preferable. The Jacobian for this system is tridiagonal so that the solution is obtained easily. These equations are solved once only with the current w_i before moving on to (ii).

In (ii) only the contributions to A and F from the boundary elements need be changed. Contributions from elements in the contact region do not affect the results.

At the same time that element contributions are adjusted, a check is made to see that the elements do not overlap. (In the examples considered this occurred only in the first boundary change for the dam seepage problem and was corrected by damping.) The same stopping criterion was used for the S.O.R. routine, but in this case a limit of five iterations was imposed if convergence had not been attained.

In variational inequality approaches to the dam seepage problem, the seeping point D is usually determined by extrapolation because $w=0$ on the seepage surface. In the current work two nodes belonging to the seepage surface are allowed to move. One is the node where the approximate free boundary first meets the seepage surface and the other is the node on the seepage surface immediately below the first. The second node gives a direct approximation of the seepage point after the boundary iteration.

In examining Fig. 2 it may be felt that the opposite diagonals would lead to a better approximation. This would be true of a direct use of variational inequalities, but when this is followed by boundary iteration the present orientation seems better suited to avoiding overlapping of elements or their distortion into thin triangles.

4. Numerical results.

4.1. First membrane contact problem. The results for the first membrane contact problem are illustrated in Fig. 3.

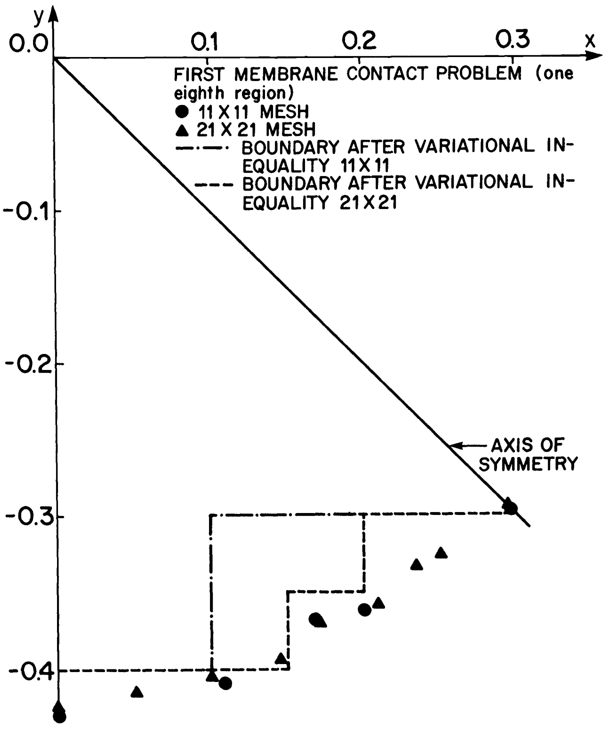


FIG. 3

The points marked ▲ are the location of the free boundary points after iteration in the case of a 21×21 mesh. The points marked ● are the locations of the final free boundary points with an 11×11 mesh. The free boundary on the fixed mesh obtained by the solution of the variational inequality is also shown.

4.2. Second membrane contact problem. The second membrane contact problem has the boundary of the contact region at $r=0.5$. Table 1 gives the coordinates of the final set of boundary points together with the value of $r=(x^2+y^2)^{1/2}$ for the case of 21×21 mesh. The final positions are also shown in Fig. 4 (marked ▲), together with the actual boundary.

The dam seepage problem. In these problems $h_1=24$, $a=16$ and $h_2=4$ or 0. The free boundary coordinates are compared with the results of Aitchison [1], $h_2=4$, and the coordinates of the seepage point with that given in Cryer [4], $h_2=4$ and 0. Since Aitchison uses a boundary iterative method in which boundary points are moved parallel to the vertical walls of the dam, the present results cannot be compared directly. To obtain boundary points with the same "x" coordinate, linear interpolation

TABLE 1

(x, y) coordinates of free boundary nodes before and after iteration of the boundary. The values of $\sqrt{x^2+y^2}$ are given for the final values. Only points in $\frac{1}{8}$ of the region are given because of the symmetry.

Before iteration		After iteration		
x	y	x	y	$\sqrt{x^2+y^2}$
0.5000	0.0000	0.5059	0.0000	.5059
0.5000	0.0500	0.4991	0.0500	.5016
0.5000	0.1000	0.4953	0.0953	.5044
0.4500	0.1000	0.4779	0.1279	.4948
0.4500	0.1500	0.4760	0.1500	.4991
0.4500	0.2000	0.4583	0.2000	.5003
0.4500	0.2500	0.4437	0.2437	.5062
0.4000	0.2500	0.4167	0.2667	.4947
0.4000	0.3000	0.4036	0.3036	.5050
0.3500	0.3000	0.3736	0.3236	.4943
0.3500	0.3500	0.3571	0.3571	0.5050

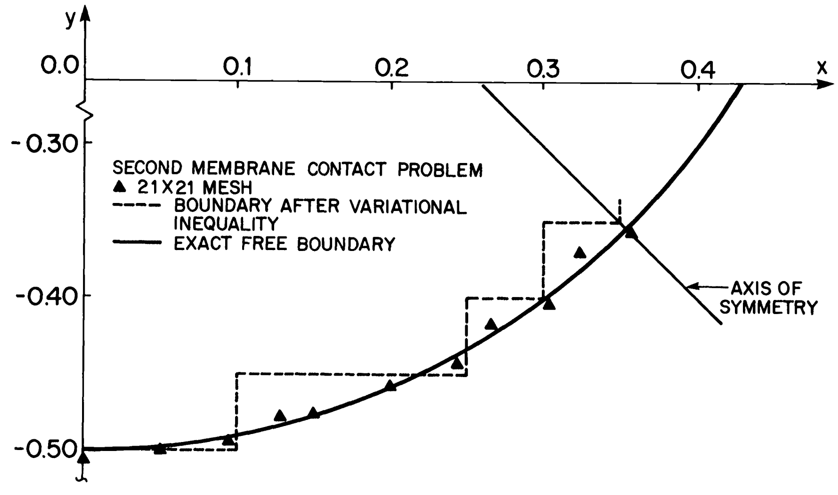


FIG. 4

of the nearest two of the present boundary points has been used. The results are given in Table 2. Columns 1 and 3 are Aitchison's results for 9 and 17 divisions in the x direction respectively and 2 and 4 are from the present work with 9×13 and 17×25 meshes respectively. The results marked with an asterisk are obtained by linear interpolation using the nearest two free boundary nodes obtained by the present method.

Cryer's results for the seepage point are obtained by an integral equation method, and comparisons are made with this result and many others given in Cryer [4] in Table 3. All results are quoted from Cryer [4, pp. 54–57]. The result marked with a dagger was obtained by Cryer graphically.

TABLE 2

Free boundary coordinates for the dam seepage problem.

	1	2	3	4
x	y	y	y	y
0	24.00	24.00	24.00	24.00
1			23.74	23.75
2	23.40	23.40	23.41	23.39*
3			23.02	22.96
4	.59	22.37*	22.59	22.60*
5			22.12	22.00*
6	21.60	21.56	21.60	21.60
7			21.04	20.97*
8	20.44	20.18*	20.43	20.42
9			19.78	19.74*
10	19.08	19.01	19.08	18.95*
11			18.31	18.20*
12	17.50	17.42*	17.48	17.46
13			16.57	16.55*
14	15.62	15.35*	15.54	15.53*
15			14.39	14.12*
16	12.85	12.34	12.79	12.48

TABLE 3

Seepage point for the dam problem. The y coordinate of the seepage point is given by various methods for $h_2 = 4$ and for $h_2 = 0$.

$h_2 = 4$	y-coord. seepage pt.	Method
Cryer	12.71	Integral equations
Aitchison	12.79	Trial free boundary 24 divisions of h_1
	12.75	Trial free boundary 48 divisions of h_1
Shaw and Southwell	12.75†	
Comincoli, Guerri, Volpi	12.89	Trial free boundary
Comincoli, Guerri, Volpi	14.4	Variational inequality 30 divisions
Present	12.34	9×13 mesh
	12.48	17×25 mesh
$h_2 = 0$		
Cryer	12.57	Integral equations
J. M. Taylor	12.64	Trial free boundary
McNawn, Hsu Yih	10.32	Trial free boundary
Present	12.25	9×13 mesh
	12.33	17×25 mesh

In Fig. 5 the free boundary points are plotted for the cases $h_2 = 4$ for 9×13 mesh ▲. Aitchison's points are also plotted ●.

In Fig. 6 the same are plotted for a 17×25 mesh.

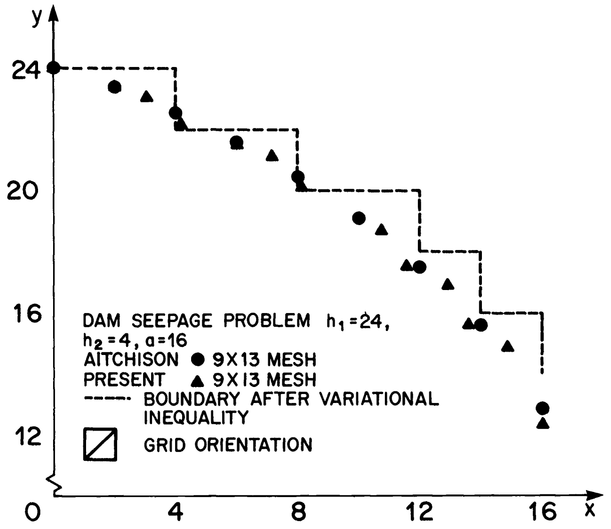


FIG. 5

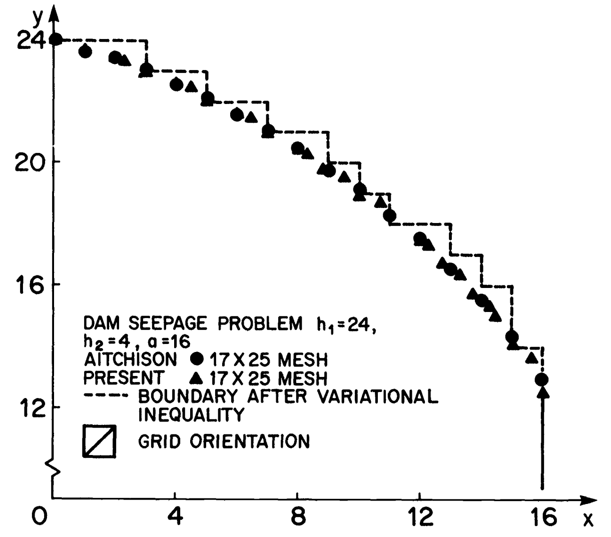


FIG. 6

Conclusion. The numerical results suggest that the method works successfully and overcomes Kikuchi's objections stated in the introduction, in that it provides a reasonably smooth boundary without excessive iterations. The seepage point is given directly in the dam problem.

There is of course still no proof that the boundary iteration process converges, nor that the approximate free boundary obtained by variational inequalities is sufficiently accurate that mesh distortion problems do not arise in the boundary iterations.

REFERENCES

[1] J. AITCHISON, *Numerical treatment of a singularity in a free boundary problem*, Proc. Roy. Soc. London, Ser. A, 330 (1972), pp. 573–580.

[2] C. Baiocchi, V. Comincioli, L. Guerri and G. Volpi, *Free boundary problems in the theory of fluid flow through porous media: A numerical approach*, Publ. 29 del L.A.N. del C.N.R., Pavia, 1973.

[3] C. W. Cryer, *The solution of a quadratic programming problem using systematic overrelaxation*, SIAM J. Control, 9 (1971), pp. 385–392.

[4] ———, *A survey of steady-state porous flow free boundary problems*, M.R.C. Tech. Sum. Rep. 1657, Mathematics Research Center, Univ. Wisconsin, Madison, July 1976.

[5] ———, *A survey of trial free boundary methods for the numerical solution of free boundary problems*, M.R.C. Tech. Sum. Rep. 1693, Mathematics Research Center, Univ. Wisconsin, Nov. 1976.

[6] G. Duvaut and J. L. Lions, *Inequalities in Mechanics and Physics*, Grundlehren der mathematischen Wissenschaften 219, Springer-Verlag, Berlin-Heidelberg-New York, 1976.

[7] R. Glowinski, J. L. Lions and R. Trémolières, *Numerical Analysis of Variational Inequalities*, North-Holland, Amsterdam, New York, Oxford, 1981. (Translation of Analyse numérique des inéquations variationelles, by the same authors, Dunod, Paris, 1976.)

[8] N. Kikuchi, *An analysis of the variational inequalities of seepage flow by finite element methods*, Quart. Appl. Math., 35 (1977), pp. 149–163.

[9] D. Kinderlehrer and G. Stampacchia, *An Introduction to Variational Inequalities and Their Applications*, Academic Press, New York, London, Toronto, Sydney, San Francisco, 1980.

[10] F. S. Shaw and R. V. Southwell, *Relaxation methods applied to engineering problems*, VII. *Problems relating to percolation of fluids through porous materials*, Proc. Roy. Soc. London, Ser. A, 178 (1941), pp. 1–17.

[11] R. L. Taylor and C. B. Brown, *Darcy flow solutions with a free surface*, in Proc. A.S.M.E., J. Hydraulics Div. 93 No. HY2 (1967), pp. 25–33.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

PRACTICAL USE OF SOME KRYLOV SUBSPACE METHODS FOR SOLVING INDEFINITE AND NONSYMMETRIC LINEAR SYSTEMS*

YOUCEF SAAD†

Abstract. The main purpose of this paper is to develop stable versions of some Krylov subspace methods for solving linear systems of equations $Ax = b$. As in the case of Paige and Saunders's SYMMLQ [SIAM J. Numer. Anal., 12 (1975), pp. 617–624], our algorithms are based on stable factorizations of the banded Hessenberg matrix representing the restriction of the linear application A to a Krylov subspace. We will show how an algorithm similar to the SYMMLQ can be derived for nonsymmetric problems and we will describe a more economical algorithm based upon the LU factorization with partial pivoting. In the particular case where A is symmetric indefinite the new algorithm is theoretically equivalent to SYMMLQ but slightly more economical. As a consequence, an advantage of the new approach is that nonsymmetric or symmetric indefinite or both nonsymmetric and indefinite systems of linear equations can be handled by a single algorithm.

Key words. numerical linear algebra, iterative methods, nonsymmetric systems, conjugate gradients

1. Introduction.

In the previous few years considerable attention has been devoted to solving large sparse sets of equations of the form

$$Ax = b \tag{1}$$

where A is an $N \times N$ real matrix. Linear systems of equations fall into four distinct classes:

1. A is symmetric positive definite;
2. A is symmetric indefinite;
3. A is nonsymmetric positive real, i.e., its symmetric part $(A + A^T)/2$ is positive definite;
4. A is nonsymmetric nonpositive real, i.e. its symmetric part is indefinite. We will then often say that A is indefinite.

Roughly speaking, the four classes above are ordered according to increasing difficulty of solution.

While the problems of the first class are well understood, the other classes have attracted much of the recent research in numerical linear algebra and are still under intensive investigation. Recent work by Vinsome [18], Axelsson [1], Elman, Eisenstat and Schultz [3], Elman [4], [5], Young and Jea [8] and Saad [14] concerns Krylov subspace methods for the case where A is nonsymmetric. A common feature of all of these methods is that the approximate solution x_m belongs to the affine space $x_0 + K_m$ where K_m is the Krylov subspace $K_m = \text{span}\,[r_0, Ar_0, \cdots, A^{m-1}r_0]$ and r_0 is the initial residual vector $r_0 = b - Ax_0$. Their principle is to attempt to make the residual vector r_m orthogonal to some subspace L_m, usually different from K_m [15]. These processes can also be regarded as different realizations of Galerkin projection methods on K_m, whereby the original problem is replaced by an m-dimensional problem with the linear operator $A_m = P_m A|_{K_m}$, where P_m is the projector onto K_m parallel to L_m. We will refer to A_m as a *section of A in K_m*.

Many of the Krylov subspace methods developed in the literature assume that the matrix A has a positive definite symmetric part, i.e. they deal with problems of the third class. Problems of the harder class 4 often occur when a preconditioning

* Received by the editors February 9, 1982, and in final revised form November 12, 1982. This work was supported in part by the U.S. Air Force grant AFOSR-81-0193 and in part by the National Science Foundation grant MCS-8104874.

† Computer Science Department, Yale University, New Haven, Connecticut 06520.

technique is used in conjunction with the iterative method. Then the resulting iteration matrix is nonpositive real even though the original matrix is positive real (see [5]).

The Incomplete Orthogonalization Method (IOM), closely related to the Galerkin process, was introduced in [14]. The IOM was devised to handle problems of classes 2, 3 and 4. Experience shows that it is effective on problems of the second and third class and on problems of class 4 that are mildly indefinite or mildly nonsymmetric. A peculiarity of the original version of IOM is that the solution is not available at each ep. Instead, one saves the basis vectors of the subspace K_m in secondary storage and forms the solution as a combination of these vectors only at the end of the process. A simple formula provides, at no extra cost, the residual norm at each step, thus determining when to stop. This implementation of the method is *indirect* in that the solution is not directly formed. It requires less core memory and computational work than its counterparts based on direct updating of the solution at every step, but has the disadvantage of using secondary storage. The purpose of this paper is to develop an equivalent version of the IOM that does not require secondary storage, while keeping the stability properties of the original version.

The principle of our approach is similar to that adopted by Paige and Saunders [10] which led to the successful SYMMLQ algorithm for solving symmetric indefinite problems. Let us recall that SYMMLQ was based upon the stable LQ factorization of a tridiagonal matrix representing the section A_m of A in K_m. For IOM, this representation becomes a banded Hessenberg matrix, which we will factor by resorting to the *LU factorization with partial pivoting.* Note that such a factorization can also be applied to tridiagonal matrices and therefore when the matrix A is symmetric, we obtain an alternative of the SYMMLQ algorithm which, incidentally, is slightly more economical than SYMMLQ. As a consequence, the new algorithm has the attractive feature that the same code can be used for any linear system regardless of symmetry or definiteness. However, it should be pointed out that when the skew-symmetric part of A is large and its symmetric part has the origin well inside the spectrum, the Krylov subspace methods are not recommended. The Krylov subspace methods are most effective in those cases where A is either nearly symmetric or nearly positive real or both.

We will compare our method with other Krylov subspace methods and will prove in particular that DIOM(k) is equivalent to Jea and Young's ORTHORES(k) algorithm.

In § 2 we will briefly recall the Incomplete Orthogonalization Method in its original form [14]. Section 3 describes an alternative version of the same algorithm, which will be called the Direct Incomplete Orthogonalization Method (DIOM). Then, in § 4, we will briefly indicate how similar techniques can be derived for Krylov subspace methods other than the IOM. A few practical considerations and heuristics will be presented in § 5 and some numerical experiments will be reported in the last section.

2. Krylov subspace methods.

2.1. The full orthogonalization method. Given a set of m linearly independent vectors $V_m = [v_1, v_2, \cdots, v_m]$, an orthogonal projection method on the subspace $K_m \equiv \text{span}[V_m]$, is by definition a method which obtains an approximate solution to (1) of the form

$$x_m = x_0 + V_m y_m \tag{2}$$

for which the residual $r_m = b - Ax_m$ is orthogonal to the subspace K_m. This Galerkin condition gives

$$V_m^T(b - Ax_m) = 0 \tag{3}$$

and therefore

$$y_m = [V_m^T A V_m]^{-1} V_m^T r_0. \tag{4}$$

In the basic full orthogonalization method (or Arnoldi's method) presented in [14], one first constructs an orthonormal sequence $V_m = [v_1, v_2, \cdots, v_m]$ from the recurrence:

$$h_{j+1,j} v_{j+1} = Av_j - \sum_{i=1}^{j} h_{ij} v_i \tag{5}$$

starting with the vector $v_1 = r_0/\|r_0\|$. In (5), the coefficients h_{ij}, $i = 1, \cdots, j+1$ are chosen such that the vector v_{j+1} is orthogonal to all the previous v_i's and $\|v_{j+1}\| = 1$.

The system V_m constitutes an orthonormal basis of the Krylov subspace $K_m = \operatorname{span}\{r_0, Ar_0, \cdots, A^{m-1} r_0\}$. An important property is that the matrix $V_m^T A V_m$ in (4) is precisely the $m \times m$ upper Hessenberg matrix whose nonzero entries are the h_{ij}'s defined in the algorithm. Furthermore, the vector $V_m^T r_0$ in (4) is equal to $\|r_0\| e_1$, with $e_1 = (1, 0, 0, \cdots, 0)^T$, by the definition of v_1. Therefore the solution y_m of (4) simplifies into

$$y_m = H_m^{-1} (\beta e_1) \tag{6}$$

where we have denoted by β the scalar $\|r_0\|$ for convenience.

An algorithm based upon the above considerations can be briefly described as follows:

ALGORITHM 1.
1. Compute $r_0 := b - Ax_0$, take $v_1 := r_0/(\beta := \|r_0\|)$ and choose an integer m.
2. For $j = 1, 2, \cdots, m$ compute the vectors v_j and the coefficient $h_{i,j}$ by (5).
3. Compute the approximate solution x_m from (2) and (6).

This will be referred to as the full orthogonalization method.

2.2. The incomplete orthogonalization method. A serious drawback of Algorithm 1 is that as j increases the process becomes intolerably expensive and requires the storage of the whole set of previous vectors v_i since these are used at every step. A remedy to this is to orthogonalize the current vector Av_j against k previous vectors instead of all the previous vectors. We will assume throughout that[1] $k \geqq 2$. The derived algorithm has been proposed in [14], and is called the Incomplete Orthogonalization Method (IOM). The only difference with the above algorithm of full orthogonalization lies in the definition of v_{j+1}, which now becomes:

$$v_{j+1} = \frac{1}{h_{j+1,j}} \left[Av_j - \sum_{i=j-k+1}^{j} h_{ij} v_i \right]. \tag{7}$$

In the above summation $j - k + 1$ is to be replaced by 1 whenever $j \leqq k - 1$. Here the coefficients $h_{i,j}$ are chosen such as to make v_{j+1} of norm one and orthogonal to the vectors v_i, $i = j - k + 1, \cdots, j$, that is:

$$h_{i,j} = (Av_j, v_i), \qquad i = j - k + 1, \cdots, j, \tag{8}$$

$$h_{j+1,j} = \left\| Av_j - \sum_{i=j-k+1}^{j} h_{ij} v_i \right\|. \tag{9}$$

[1] The method can also be defined for $k = 1$, and corresponds to the method of steepest descent in the symmetric positive definite case. We want to avoid this trivial case.

The new Hessenberg matrix which will still be denoted by H_m has the following banded structure when, for example, $m=9$, $k=4$:

$$H_m = \begin{bmatrix} x & x & x & x & & & & & \\ x & x & x & x & x & & & & \\ & x & x & x & x & x & & & \\ & & x & x & x & x & x & & \\ & & & x & x & x & x & x & \\ & & & & x & x & x & x & x \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \end{bmatrix}. \tag{10}$$

A simplistic version of the IOM algorithm can be described as follows.

ALGORITHM 2.

1. Compute $r_0 := b - Ax_0$, take $v_1 := r_0/(\beta := \|r_0\|)$ and choose an integer m.
2. Compute $V_m = [v_1, v_2, \cdots, v_m]$ and H_m by using (7) and (8) and (9), for $j = 1, \cdots, m$.
3. Compute

$$y_m := \beta H_m^{-1} e_1 \tag{11}$$

and form the approximate solution

$$x_m := x_0 + V_m y_m. \tag{12}$$

We will refer to the above algorithm as IOM (k) or simply IOM if there is no ambiguity. It is clear that when the number of steps m does not exceed k, the above algorithm is equivalent to the full orthogonalization method. For this reason we will denote by IOM (∞) the full orthogonalization method, since full orthogonalization corresponds to taking k larger than any step number m in the above algorithm.

One of the important details not made clear in the above implementation concerns the choice of the number of steps m. If the algorithm were to be applied with an arbitrarily chosen m, we would certainly have to restart the algorithm whenever m is so small that the accuracy of x_m is insufficient. In other words we would have to restart the above algorithm with the initial vector x_0 replaced by the latest computed approximate solution x_m, and this would be repeated until convergence. But it is also possible that m might be too large and that convergence would occur for some $m_0 < m$. This means that we must be able to test for convergence anywhere between $j=1$ and $j=m$. In fact all we need is a formula for computing the residual norm of the intermediate approximation x_j *without forming* x_j. Fortunately such a formula exists and is given by (see e.g. [14])

$$\|b - Ax_m\| = h_{m+1,m} |e_m^T y_m|. \tag{13}$$

As will be seen later, updating (13) requires only 2 multiplications per step provided that the factorization of H_j is updated at each step (this fact will be shown in the next section). Since the final factorization of H_m is needed for the solution of the $m \times m$ system involved in (11), it is not more costly to factor H_m by updating the factorization at each step, and hence the computation of the estimate (13) is virtually free. It should be added that (13) gives a quite accurate approximation of the residual norm in practice.

The above remarks suggest an efficient implementation of IOM which we briefly outline here (see [14] for more details). First choose an initial vector x_0, the parameter k and a maximum number of steps m max. Compute r_0, and $v_1 = r_0/\|r_0\|$. Then for $j = 1, 2, \cdots$, compute $h_{i,j}$ and v_{j+1} by (5). Write in secondary storage every vector v_{j+1} thus generated, one by one. At each step j, simultaneously update the LU factorization of H_j and the residual norm (13). If at step j the residual norm is small enough, recall the v_i's from secondary storage one by one and form the approximate solution x_j by (11) and (12). If j reaches m max and the residual norm is still unsatisfactory, form $x_{m\,\max}$ and restart the algorithm with this as initial vector.

We must emphasize that the central idea of the algorithm lies in the fact that it is possible to update at each step the factorization $H_j = L_j U_j$ with L_j unitary lower triangular, U_j upper triangular. Even more interesting is the fact that LU factorization *with partial pivoting* can also be updated at each step together with the estimate (13) of the residual norm. These observations have already been exploited in the previous paper [14]. The algorithm given above will be referred to as the *indirect version of* IOM as the approximate solution x_m is not updated at every step. We now show how to derive a few direct versions which are theoretically equivalent.

3. Incomplete orthogonalization method: direct versions. In all of the algorithms proposed by Axelsson [1], Eisenstat, Elman and Schulz [3], Elman [5], Young and Jea [8] and Vinsome [18], the approximate solution x_m is updated at every step in a conjugate-gradient-like algorithm. We show here that it is also possible to write similar versions for the IOM algorithm. The algorithms we are about to describe are based upon updating the LU factorization of H_m at each step. For the sake of clarity we first present a version that does not use partial pivoting. The more stable algorithm using partial pivoting will be the object of § 3.2.

3.1. Derivation of the algorithm. At each step m of IOM, the approximate solution x_m is given by the formula

$$x_m = x_0 + \beta V_m H_m^{-1} e_1 \tag{14}$$

where $\|r_0\|$ has been replaced by β for convenience.

Let H_m be factored as

$$H_m = L_m U_m \tag{15}$$

where L_m is an $m \times m$ lower bidiagonal matrix with diagonal elements equal to one, and U_m is a banded upper triangular matrix with k diagonals. That is:

$$H_m \qquad = \qquad L_m \qquad\qquad U_m$$

$$\begin{bmatrix} x & x & x & x & & & & & \\ x & x & x & x & x & & & & \\ & x & x & x & x & x & & & \\ & & x & x & x & x & x & & \\ & & & x & x & x & x & x & \\ & & & & x & x & x & x & x \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & & 1 & & \\ & & \ddots & \ddots & \\ & & & l_m & 1 \end{bmatrix} \begin{bmatrix} x & x & x & x & & & & & \\ & x & x & x & x & & & & \\ & & x & x & x & x & & & \\ & & & x & x & x & x & & \\ & & & & x & x & x & x & \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & x & x \\ & & & & & & & & x \end{bmatrix}$$

From (14) and (15) the solution x_m satisfies $x_m = x_0 + V_m U_m^{-1} L_m^{-1} (\beta e_1)$.

Following Paige and Saunders [10] let us set

$$W_m = V_m U_m^{-1} \tag{16}$$

and

$$z_m = \beta L_m^{-1} e_1. \tag{17}$$

We will denote by w_i the ith column of the $N \times m$ matrix W_m.

The key observation here is that we pass from the $(m-1)$-dimensional vector z_{m-1} to the m-dimensional vector z_m by just appending one component. In other words:

$$z_m = \begin{bmatrix} z_{m-1} \\ \\ \xi_m \end{bmatrix}. \tag{18}$$

The last element ξ_m of the vector z_m satisfies the recurrence

$$\xi_m = -l_m \xi_{m-1}, \qquad m = 2, 3, \cdots, \tag{19}$$

starting with $\xi_1 = \beta = \|r_0\|$. Recall that we denote by l_m the element in position $m, m-1$ of the matrix L_m. It is then clear that x_m can be updated at every step through the formula:

$$x_m = x_{m-1} + \xi_m w_m. \tag{20}$$

The vectors w_m can in turn be updated quite simply since we have from their definition (16)

$$w_m = \frac{v_m - \sum_{i=m-k+1}^{m-1} u_{im} w_i}{u_{mm}}. \tag{21}$$

Our first direct version of IOM (k) can therefore be summarized as follows:

ALGORITHM 3.

```
1. Start. Choose an initial vector $x_0$ and compute $r_0 = b - Ax_0$.
2. Iterate. For $m = 1, 2, \cdots$ until convergence do:
   • Compute $h_{im}$, $i = m-k+1, \cdots, m+1$, and $v_{m+1}$ by formulas (7), (8) and (9).
   • Update the $LU$ factorization of $H_m$, i.e. obtain the last column of $U_m$ and
     the last row of $L_m$. Then compute $\xi_m$ from (19).
   • Compute
         $w_m = \frac{v_m - \sum_{i=m-k+1}^{m-1} u_{im} w_i}{u_{mm}}$.
   • Compute
         $x_m = x_{m-1} + \xi_m w_m$.
```

We have intentionally skipped some of the details concerning in particular the way the LU factorization of H_m is updated. An important remark here is that (13) can easily be updated because the last element of y_m is just ξ_m / u_{mm}, hence

$$\|b - Ax_m\| = h_{m+1,m} \left| \frac{\xi_m}{u_{mm}} \right|, \tag{22}$$

which requires only two arithmetic operations per step.

Note that the division by u_{mm} in (21) can be avoided by simply using the vectors $w'_j = u_{jj}w_j$ in place of w_j and by keeping track of the scaling factors u_{jj}. With this, the cost per step of the above algorithm is approximately $(3k+2)N$ additions/multiplications plus one matrix by vector multiplication, and we need $(2k+1)$ vectors of storage (these are: x_m, $k-1$ vectors w_j, k vectors v_j, plus an extra vector for Av_m).

The above algorithm breaks down whenever u_{mm} vanishes. In fact even if u_{mm} does not vanish it is not recommended to use the above algorithm as it is based upon the potentially unstable LU factorization of H_m and can result in an unstable algorithm for solving $Ax = b$. This brings up the use of partial pivoting.

3.2. Using the *LU* factorization with partial pivoting. Instead of the straightforward factorization (15) we now introduce the following LU factorization with partial pivoting of the matrix H_m

$$H_m = P_2E_2P_3E_3 \cdots P_mE_mU_m \tag{23}$$

where each P_j is an elementary permutation matrix, and E_j is an elementary transformation [19] having the structure:

$$E_j = \begin{array}{c} \left[\begin{array}{ccccccccc} 1 &&&&&&&& \\ & 1 &&&&&&& \\ && 1 &&&&&& \\ &&& \ddots &&&&& \\ &&&& 1 &&&& \\ &&&& l_j & 1 & 0 & 0 & 0 \\ &&&& 0 && \ddots && \\ &&&& 0 &&& \ddots & \\ &&&& 0 &&&& 1 \end{array}\right] \\ \begin{array}{c} \uparrow \\ (j-1)\text{st column} \end{array} \end{array} \begin{array}{l} \\ \\ \\ \\ \\ \leftarrow j\text{th row.} \\ \\ \\ \\ \\ \end{array}$$

The elementary permutation matrix P_{j+1} is the one used to permute the rows j and $j+1$ if needed, i.e., if the element $h_{j+1,j}$ is larger in absolute value than the element u_{jj}. The matrix U_m is again a banded upper triangular matrix, this time with $k+1$ diagonals instead of k due to the permutations. The LU factorization with partial pivoting is a very stable process when the matrix H_m is tridiagonal since the elements obtained at any step r of the factorization, $r \leqq m$, are no longer than $2 \max \{|h_{ij}|, i = 1, m; j = 1, m\}$ (see Wilkinson [19, p. 219]). For banded Hessenberg matrices of bandwidth $k+1$, it is easy to generalize the above bound and show that the elements obtained at any step of the factorization do not exceed $k \max \{|h_{ij}|, i = 1, m; j = 1, m\}$.

As before the approximation x_m is defined by (14). Letting again

$$W_m = V_mU_m^{-1} \tag{24}$$

we get

$$x_m = x_0 + W_mz_m$$

where z_m is now defined as

$$z_m = E_m^{-1}P_mE_{m-1}^{-1}P_{m-1} \cdots E_2^{-1}P_2(\beta e_1).$$

Note that the vectors w_j can be updated in the same way as before by use of (21), except that the summation is now from $m-k$ to $m-1$. We claim that there are formulas similar to (19), (20) for updating the vectors z_m. This is because we have

$$z_m = E_m^{-1} P_m \bar{z}_{m-1} \tag{25}$$

with

$$\bar{z}_{m-1} = \begin{bmatrix} z_{m-1} \\ \\ 0 \end{bmatrix}. \tag{26}$$

Equations (25) and (26) show that there are two cases, depending on whether interchange has or has not been performed at the previous step:

1. either rows $m-1$ and m have *not* been interchanged, in which case the vector z_m is defined as before by (18) and (19);

2. or there has been an interchange of rows $(m-1)$ and m, in which case

$$z_m = \begin{bmatrix} z_{m-2} \\ 0 \\ \xi_{m-1} \end{bmatrix} \tag{27}$$

where ξ_{m-1} is the last element of z_{m-1}.

Practically, the use of the above observations is particularly simple. If interchange has not been performed at step $m-1$, then the approximate solution x_m is updated from x_{m-1} as before by (20) and (19). If, on the other hand, rows $m-1$ and m have been permuted, then the expression (27) of z_m shows that the only difference with the previous case is that the approximation x_m is now defined by $x_m = x_{m-2} + \xi_{m-1} w_m$. In other words x_{m-1} could be redefined as equal to x_{m-2} and the scalar ξ_m as equal to ξ_{m-1} in the formula (20). In other words if a permutation occurs all we have to do is *skip the application of the updating formulas* (20), (19) at the next step. In a practical implementation we must look ahead at the current step m and check whether permutation will or will not be necessary *in the next step* $m+1$. This is fortunately possible because the element $h_{m+1,m}$ is available at the mth step as well as the element u_{mm}, since the factorization is updated at each step.

Concerning the updating of the factorization of H_m at the mth step we can proceed as follows. First, using the $k+1$ previous pivots $l_{m-k}, l_{m-k+1}, \cdots, l_m$ transform the column $\{h_{im}\}_{i=1,m}$ into the column $\{u_{im}\}_{i=1,m}$. In order for this to be possible we must save these $k+1$ pivots. Note that the mth column of H_m (resp. U_m) has at most k (resp. $k+1$) nonzero elements. Second, compare the element u_{mm} thus obtained with $h_{m+1,m}$ to determine if interchange is needed. If $|u_{mm}| < h_{m+1,m}$, permute the elements $h_{m+1,m}$ and u_{mm} and compute the next pivot l_{m+1}. Clearly the matrices H_m, E_m, U_m need not be saved. All we need is the previous $k+1$ pivots l_j, the logical information perm $(j+1)$, $j = m-k, \cdots, m$, defined as perm $(j+1) =$ true if rows j and $j+1$ are permuted, and the $k+1$ nonzero elements of the mth column of H_m, which is transformed into the mth column of U_m. This constitutes little storage as k is small in general (typically $k < 10$). We describe below the Direct Incomplete Orthogonalization Method (DIOM (k)) algorithm derived from the above suggestions.

```
ALGORITHM 4. DIOM (k).
1. Start. Pick an iniital vector $x_0$, define $r_0 := b - Ax_0$, $\rho_0 := \|r_0\|$.
2. Iterate. For $m = 1, 2, \cdots$ do:
   • Compute $v_{m+1}$ and the $m$th column of $H_m$ by (7), (8), and (9).
   • Update the $LU$ factorization with partial pivoting of $H_m$, i.e. find the $m$th
     column of $U_m$ and compute $\rho_m := \beta_{m+1}|\xi_m/u_{mm}|$. Then interchange $u_{mm}$ and
     $h_{m+1,m}$ if necessary, and get the pivotal information to be used in the next step.
   • Compute
(28)     $w_m := \frac{v_m - \sum_{i=m-k}^{m-1} u_{im} w_i}{u_{mm}}$.
   • If perm $(m+1)$ = true and $\rho_m > \varepsilon$ then:
         $\xi_m := \xi_{m-1}$,   $x_m := x_{m-1}$.
   • Else compute
         $\xi_m := -l_m \xi_{m-1}$,   $x_m := x_{m-1} + \xi_m w_m$.
   • If $\rho_m \leqq \varepsilon$ stop.
```

Once again DIOM (∞) refers to the case of full orthogonalization, that is, to the case where the summations in (7) and (28) start from 1. In the above algorithm ρ_m represents the residual norm that would be obtained if we stop at step m. Thus the process is stopped when ρ_m is smaller than the tolerance ε. We should stress that u_{mm} in item 3 of iterate is the one obtained *after* interchange while the one used for computing ρ_m in item 2 is *before* interchange.

Notice that we force the application of formulas (19) and (20) when it is known from the residual norm ρ_m that the process will stop at the current step. Indeed, the algorithm *artificially* defines x_m to be equal to x_{m-1} whenever it is known that interchange will be necessary at the *next step*. When it is known from ρ_m that convergence is achieved, i.e. that the current step is the last one, we must imperatively define x_m by (20) and (19), regardless of the value of perm $(m+1)$. This is the reason for the more complicated test in item 4 of iterate. Another, perhaps simpler, way of correcting this is to always define perm $(m+1)$ as false if m is known to be the last step.

Because of the artifice used to define x_m when interchange occurs, the vectors x_m defined here are not the same as the vectors x_m of Algorithm 3 except when interchange is not performed. Thus, while the final solutions delivered by Algorithms 3 and 4 are identical, the intermediate vectors x_j are not necessarily the same. When interchange occurs, the approximation x_j defined by (14) is not computed but is defined as being equal to the previous approximation. For example, if interchange is needed at *every step*, the new sequence of approximations x_j obtained from Algorithm 4 is *stationary* until the final step is reached, i.e. until $\rho_m \leqq \varepsilon$. The residual estimate ρ_m corresponds to the *actual approximate solution* that would have been obtained if we had had to stop at the current step. Clearly, this may be infinite in case u_{ij} (before interchange) is zero, which reflects the fact that the approximate solution x_j defined by (14) does not exist when H_j is singular. But, while Algorithm 3 will break down in this situation, Algorithm 4 is able to construct the next approximations $x_{j+1}, x_{j+2}, \cdots$.

The amount of work is essentially the same as required for Algorithm 3. Indeed, when a permutation takes place we save one vector update of the form (20); i.e., we save N additions/multiplications. But then the permutation introduces an extra nonzero element in the triangular matrix U_{m+k}, which means N extra additions/multiplications when updating the vectors w_{m+k} at step $m+k$, and this compensates exactly the savings made at the current step. Concerning the storage, we need one more vector of storage, as the sum defining w_m is now from $m-k$ to $m-1$; i.e., we need a total of $2k+2$ vectors of storage instead of the $2k+1$ of Algorithm 3.

One might question the need for using Algorithm 4 instead of Algorithm 3 in some cases. In particular, is pivoting necessary at all if the original matrix A is positive real? Our numerical experiments show that interchange will indeed occur even in those cases. The fact that some pivots are quite small even when A is almost positive real suggests that it is in general better to use the more stable version of Algorithm 4, instead of Algorithm 3. Moreover, as shown above, it is not more costly to use pivoting except for one additional vector of storage required.

3.3. Using the *QR* factorization. The SYMMLQ algorithm described by Paige and Saunders in [10] uses the LQ factorization $H_m = L_m Q_m$. A similar algorithm using the stable QR factorization can also be developed for the incomplete orthogonalization method. Consider the orthogonal QR factorization

$$H_m = Q_m U_m \tag{29}$$

where U_m is, as in § 3.1, an $m \times m$ upper triangular matrix and Q_m is a unitary orthogonal matrix, i.e. $Q_m^T Q_m = I$. A remark which is essential is that since H_m is banded upper Hessenberg, U_m will be banded upper triangular.

The reason we prefer the QR factorization to the LQ factorization is that the implementation with QR is quite simple and resembles that of Algorithm 4. The only difference is that instead of the elementary matrices E_j we now use elementary rotation matrices F_j of the form

$$F_j = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & & \\ & & & c_j & -s_j & & & \\ & & & s_j & c_j & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \begin{array}{l} \\ \\ \\ \\ \leftarrow \text{row } j \\ \\ \\ \\ \end{array}$$

where $c_j = \cos(\theta_j)$, $s_j = \sin(\theta_j)$. It will be shown below that Q_m in (29) is the product of $m-1$ such rotation matrices; more precisely,

$$Q_m = F_2 F_3 \cdots F_m. \tag{30}$$

Letting as previously

$$W_m = V_m U_m^{-1}, \tag{31}$$

$$z_m = Q_m^{-1}(\beta e_1) = Q_m^T(\beta e_1), \tag{32}$$

we observe that the approximate solution x_m is again defined by

$$x_m = x_0 + W_m z_m \tag{33}$$

and that the updating of the vectors w_j is essentially the same as in the previous algorithms, since U_m is banded upper triangular.

Next we would like to show how to update the factorization (29) and the vector z_m at every step. Suppose that at a given step we have the factorization (29) and let us assume that one more step is performed in the sequence giving the v_i's, such that we now have the $(m+1)$st column of H_{m+1} available. We want to compute the next orthogonal matrix Q_{m+1}, or according to (30) the next rotation F_{m+1}, and the next last column of U_{m+1}. Let us denote by $\bar{Q}_j$ the $(m+1)\times(m+1)$ matrix obtained by completing the $j\times j$ matrix Q_j by adding zeros in all nondiagonal positions and ones in the diagonal positions. We will define in the same way the matrix $\bar{F}_j$.

Then we have

$$\tag{34} \bar{Q}_m^T H_{m+1} = \left[\begin{array}{cccccccc|c} x & x & x & x & & & & & 0 \\ & x & x & x & x & & 0 & & 0 \\ & & x & x & x & x & & & 0 \\ & & & x & U_m & & x & & 0 \\ & 0 & & & x & x & x & x & 0 \\ & & & & & x & x & x & x \\ & & & & & & x & x & x \\ & & & & & & & u_{mm} & x \\ \hline & & & & & & & \beta_{m+1} & \alpha_{m+1} \end{array}\right]$$

where we have denoted for convenience by β_{m+1} and α_{m+1} the elements $h_{m+1,m}$ and $h_{m+1,m+1}$ respectively. The elements of the last column of the above matrix are obtained by multiplying the last column of H_{m+1} by the successive rotations $F_j^T, j=2,3,\cdots,m$. In order to eliminate the element β_{m+1} in position $(m+1,m)$ it is clear that we need to premultiply the above matrix by the plane rotation F_{m+1}^T with c_{m+1} and s_{m+1} defined by

$$\tag{35} c_{m+1}=\frac{u_{mm}}{(u_{mm}^2+\beta_{m+1}^2)^{1/2}}, \qquad s_{m+1}=\frac{\beta_{m+1}}{(u_{mm}^2+\beta_{m+1}^2)^{1/2}},$$

which determines the next plane rotation. Note that the last column of U_{m+1} is now entirely available by premultiplying the last column of the matrix (34) by the rotation F_{m+1}^T. Since the elements $h_{i,m+1}$ are zeros for $i=1,2,\cdots,j-k$, only the previous k plane rotations are needed. The upper triangular matrix U_{m-1} will have $k+1$ diagonals as the premultiplication by the plane rotations will introduce an extra diagonal.

Consider now the vector z_{m+1} which we would like to update from z_m. This is possible because $z_{m+1}=F_{m+1}\bar{z}_m$ where

$$\bar{z}_m=\begin{bmatrix} z_m \\ \\ 0 \end{bmatrix}.$$

Hence

$$z_{m+1}=\begin{bmatrix} z_{m-1} \\ \bar{\xi}_m \\ \xi_{m+1} \end{bmatrix}$$

where $\bar{\xi}_m = c_{m+1}\xi_m$, $\xi_{m+1} = s_{m+1}\xi_m$. In the above, ξ_j denotes the last element of z_j. Hence the approximate solution may be updated from the equation

$$x_{m+1} = x_{m-1} + \bar{\xi}_m w_m + \xi_{m+1} w_{m+1}. \tag{36}$$

Clearly, such an updating formula is not the most economical, because in the computation of x_{m+1}, each w_i is accumulated twice with different coefficients as is seen from (36). However, a more efficient accumulation can be used. Consider instead of x_m the vector $\bar{x}_m$ defined by the recurrence $\bar{x}_m = \bar{x}_{m-1} + \bar{\xi}_m w_m$. Unlike x_{m+1}, this can be accumulated with N multiplications per step instead of $2N$. It is only at the last step that we need to compute x_{m+1}, which can be obtained from $x_{m+1} = \bar{x}_m + \xi_{m+1} w_{m+1}$. Just as pointed out in § 3.3, at the mth step we have all the information necessary to obtain the next plane rotation. Hence, we should look ahead at step m and obtain the rotation F_{m+1}. In this manner we have not only z_m but also the element $\bar{\xi}_m$ in position m of z_{m+1}. Since this will not be changed by the subsequent rotations, we see that the updating formula passing from $\bar{x}_{m-1}$ to $\bar{x}_m$ can be performed. Notice that similar arguments have been used in [10].

An algorithm based on the above developments can easily be implemented, but our experience contradicted the expectation, formulated in [14], that this approach is superior. Indeed, we found that it is needlessly more expensive and complicated than the version DIOM(k) of Algorithm 4. In effect, each step now requires N more operations than the equivalent method DIOM(k). This would not have been a high price to pay if it resulted in a substantial improvement. Such is not the case, however, as a number of numerical experiments show (see § 6.1). We think that the reason for this is that the main source of errors is not in the factorization of H_m and the formation of the solution as defined by (14), but rather mainly in the construction of the vectors v_j. Solving a banded Hessenberg system by Gaussian elimination with partial pivoting is a very stable process, as was seen in § 3.2. However, there might exist particularly difficult cases where the more stable QR factorization would be more effective, and although we prefer Algorithm 4, the technique outlined in this subsection should not be completely discarded.

3.4. Properties of the Incomplete Orthogonalization Method. In this subsection we wish to show a number of properties of Algorithm 4 assuming exact arithmetic. We would like first to establish a sufficient condition under which Algorithm 4 does not break down. Similar sufficient conditions for the symmetric Lanczos algorithm are expressed in terms of the minimal polynomial of the initial vector v_1 (or r_0). We will call minimal polynomial of a vector v with respect to the matrix A the polynomial p_s of smallest degree s such that $p_s(A)v = 0$ (cf. [19]).

PROPOSITION 1. *Assume that the minimal degree of r_0 is not less than m and that the mth approximate solution x_m as defined by* (14) *exists*, i.e. *that H_m is nonsingular. Then x_m can be computed by Algorithm* 4.

Proof. To prove this property consider the polynomial defined by the recurrence

$$h_{j+1,j}p_{j+1}(\lambda) = \lambda p_j(\lambda) - \sum_{i=j-k+1}^{j} h_{ij}p_i(\lambda), \qquad j = 1, 2, \cdots, m, \cdots \tag{37}$$

starting with $p_1(\lambda) = 1$. Denote by $\tilde{p}_{j+1}$ the polynomial on the right-hand side of (37). This is a polynomial of degree j such that $h_{i+1,i} = \|\tilde{p}_{i+1}(A)v_1\|$, $i = 1, 2, \cdots$. If at step j we had $h_{j+1,j} = 0$, this would mean that $\tilde{p}_{j+1}(A)v_1 = 0$. Since $\tilde{p}_{j+1}$ is of degree j, and because the minimal polynomial of v_1 is of degree larger than j, we conclude that $h_{j+1,j}$ cannot be equal to zero. This means that the algorithm does not break down at step j, because firstly the next vector v_{j+1} can be computed, and secondly since $\max\{h_{j+1,j}, |u_{jj}|\}$ is nonzero, the factorization of H_j does not breakdown and hence the vector w_{j+1} can be computed. The only difficulty may be encountered at the final step m where u_{mm} can be equal to zero. But this is excluded by the assumption that H_m is nonsingular. $\square$

If at the jth step, the Hessenberg matrix H_j is singular, then the jth approximation x_j as defined by (14) does not exist. Thus Algorithm 3 fails because it attempts to explicitly compute x_j for all j, while the key advantage of Algorithm 4 is that it does not need x_j to obtain the subsequent approximations. Note that when $h_{m+1,m}=0$ and $u_{mm}\neq 0$, the algorithm produces the exact solution at step m. This uncommon situation is a consequence of (13) and is often referred to as a "happy breakdown".

Next we would like to prove some orthogonality relations characterizing the residual vectors r_j and the directions w_j produced by the algorithm. In the symmetric case it is known that the residual of the approximate solution produced by the conjugate gradient algorithm is orthogonal to all the previous residuals and that the directions w_j, are conjugate with respect to A, i.e. $(Aw_j, w_i)=0$ if $i\neq j$. The situation is not as simple in the nonsymmetric case. For the residual vectors, it was shown in [14] that at each step j the residual r_j is orthogonal to the previous k residuals. This fact is a simple consequence of the following lemma:

LEMMA 2. *The residual vectors r_m produced by m steps of either Algorithm* 3 *or Algorithm* 4 *satisfy the relation*

$$r_m = -h_{m+1,m}e_m^T y_m v_{m+1} \tag{38}$$

where y_m is defined by (11).

Proof. The lemma is a consequence of the following important relation derived from the definition of the vectors v_j:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T \tag{39}$$

where $V_m=[v_1, v_2, \cdots, v_m]$. The residual r_m is such that

$$\begin{aligned} r_m &= b - Ax_m = b - A[x_0 + V_m y_m] = r_0 - AV_m y_m \\ &= \beta v_1 - [V_m H_m + h_{m+1,m} v_{m+1} e_m^T] y_m. \end{aligned}$$

The result (39) follows from the fact that $H_m y_m = \beta e_1$. $\square$

Among other things, the lemma asserts that the residual vectors are equal to the v_i except for length. Since the coefficients $h_{i,j}$ are chosen such that $(v_{m+1}, v_i)=0$, $i=m-k+1, \cdots, m$, it follows from (38) that ([14]):

PROPOSITION 3. *The residual vectors produced by Algorithm* 3 *or* 4 *satisfy the orthogonality relation*:

$$(r_m, r_i) = 0, \qquad i = m-k, m-k+1, \cdots, m-1.$$

In other words the current residual is orthogonal to the previous k residuals. Note that the above lemma also proves the result (13).

An important consequence of the above proposition is that when A is symmetric then the algorithm DIOM (2) is theoretically equivalent to the conjugate gradient method (A positive definite) or the SYMMLQ algorithm (A symmetric indefinite), for which it is known that the residuals satisfy the same property. In the symmetric case it is not necessary to take $k>2$, because all of the algorithms DIOM (k) with $k\geqq 2$ are equivalent to DIOM (2) (see [14]). The matrix H_m is then tridiagonal symmetric and the process of building the v_i's is nothing but the usual symmetric Lanczos process.

Concerning the directions w_i, it is natural to ask whether a conjugacy relation similar to that of the conjugate gradient method holds. Consider first the case of full orthogonalization. For the remainder of this subsection we will assume that no pivoting is performed throughout the process, i.e. we assume Algorithm 3 is used instead of Algorithm 4.

PROPOSITION 4. *For the direct version of the Full Orthogonalization Method* (DIOM (∞)) *with no pivoting, the following semiconjugacy relation is satisfied by the vectors* w_i:

$$(Aw_j, w_i) = 0 \quad \text{for } i < j, \quad j = 1, 2, \cdots.$$

Proof. Multiplying (39) on the right by U_m^{-1} gives

$$AW_m = V_m L_m + \left(\frac{h_{m+1,m}}{u_{mm}}\right) v_{m+1} e_m^T U_m^{-1}. \tag{40}$$

Multiplying the above equation by W_m^T on the left, and letting $\mu = h_{m+1,m}/u_{mm}$, we obtain

$$W_m^T A W_m = U_m^{-T}[V_m^T V_m] L_m + \mu U_m^{-T} V_m^T v_{m+1} e_m^T.$$

The last term in the above equation is a null vector because v_{m+1} is orthogonal to all previous vectors (full orthogonalization). Also because of the orthogonality of the v_i's, the $m \times m$ matrix $[V_m^T V_m]$ is the identity matrix. Finally we get

$$W_m^T A W_m = U_m^{-T} L_m,$$

which is a lower triangular matrix. This completes the proof. □

From the above proof it is easily seen that the proposition does not extend to the case of incomplete orthogonalization. A somehow weaker result is, however, proved below.

PROPOSITION 5. *The following orthogonality relations hold for* DIOM (k) *with no pivoting*:

$$(Aw_j, v_i) = 0 \quad \text{for } j - k + 1 < i < j, \quad j = 2, 3, \cdots.$$

Proof. Let us start with equation (40), which is still valid, and multiply both sides on the left by V_m^T to get

$$V_m^T A W_m = V_m^T V_m L_m + \mu V_m^T v_{m+1} e_m^T, \tag{41}$$

with μ defined in the proof of the previous proposition. Careful matrix interpretation of (41) shows that $V_m^T A W_m$ has the following structure:

$V_m^T A W_m =$

$$\begin{array}{ccccc}
V_m^T V_m & + & L_m & + & \mu V_m^T v_{m+1} \theta_m^T \\
\begin{bmatrix}
1 & & & x & \cdot & \cdot & \cdot \\
 & 1 & & & x & & \cdot \\
 & & \cdot & 0 & & x & \cdot \\
x & & & \cdot & & & x \\
\cdot & x & & 0 & \cdot & & \\
\cdot & & x & & & \cdot & \\
\cdot & \cdot & \cdot & x & & & 1
\end{bmatrix} & + &
\begin{bmatrix}
1 & & & & & & \\
x & 1 & & & & & \\
 & x & \cdot & & & 0 & \\
 & & x & \cdot & & & \\
 & & & x & \cdot & & \\
 & 0 & & & x & \cdot & \\
 & & & & & x & 1
\end{bmatrix} & + &
\begin{bmatrix}
 & & & & & 0 & x \\
 & & & & & 0 & x \\
 & & & & & \cdot & x \\
0 & & & & & \cdot & x \\
 & & & & & \cdot & 0 \\
 & & & & & & 0 \\
 & & & & & 0 & 0
\end{bmatrix}
\end{array}
\begin{array}{l}
\\ \\ \\ \\ \leftarrow \text{row } m-k+1. \\ \\ \\ \\
\end{array}$$

Hence

$$V_m^T A W_m = \begin{bmatrix} & 1 & & & x & & \cdot & \cdot & \cdot \\ & x & 1 & & & x & & & \cdot \\ & & x & \cdot & & 0 & x & & \cdot \\ x & & & x & \cdot & & & x & \\ \cdot & x & 0 & & x & \cdot & & & x \\ \cdot & & x & & & x & \cdot & & \\ \cdot & \cdot & \cdot & x & & & x & 1 & \end{bmatrix}.$$

The upper part of the above matrix has zero elements in position i, j whenever $j-k+1<i<j$ (post multiplication of $V_m^T V_m$ by L_m introduces an extra diagonal). The proof is complete. □

3.5. Comparison with other methods. In this section we will compare the Incomplete Orthogonalization Methods with other Krylov subspace methods and will in particular prove that DIOM (k) is theoretically equivalent to Young and Jea's ORTHORES (k) algorithm. The class of Krylov subspace methods includes the ORTHOMIN (k) algorithm [18], [3], [5], Young and Jea's ORTHODIR (k) and ORTHORES (k) algorithms and Axelsson's method [1]. A comparison of the work per step for each of the above methods must take into account the fact that the parameter k does not always have the same meaning. Thus ORTHOMIN (1) is equivalent to the conjugate residual method in the symmetric case while our DIOM (2) would be equivalent to the conjugate gradient method.[2] It is therefore more meaningful to compare ORTHOMIN (k) with DIOM $(k+1)$. With this in mind, the next table compares the work and storage requirements for a few representative Krylov subspace methods.

TABLE 1

Method	Mult./step	Storage
IOM $(k+1)$*	$(2k+5)N$	$(k+2)N$
DIOM $(k+1)$	$(3k+5)N$	$(2k+4)N$
ORTHOMIN (k)	$(3k+4)N$	$(2k+3)N$
ORTHODIR (k)	$(3k+4)N$	$(2k+3)N$
ORTHORES $(k+1)$	$(3k+6)N$	$(2k+3)N$
Axelsson (k)	$(3k+4)N$	$(2k+3)N$

* This method uses secondary storage. The table reports core memory requirement only.

Note that the number of multiplications in DIOM (k) can be reduced to roughly $(3k+4)N$ by normalizing the vectors v_i only when their norms become large (or small). Thus one might say that the number of multiplications for DIOM $(k+1)$, ORTHOMIN (k) and ORTHODIR (k) is nearly the same. The slight difference in storage is not significant. The indirect method IOM (k) is attractive in computing environments in which the I/O with disks is inexpensive, but this may vary significantly from one site to another.

[2] It is assumed in this discussion that the auxiliary matrix Z used in Young and Jea's notation [8] is the identity matrix.

Various experiments have been conducted to compare the above methods, in particular by Elman [4], [5]. The nontruncated versions ($k=\infty$) are impractical since one has to keep all the previous vectors in core memory. The truncated versions ($k<\infty$), on the other hand, are more suitable computationally but more difficult to study theoretically. The comparisons show that, in most cases, none of the above methods is superior by a large margin. The ORTHOMIN (k) methods, however, have a better theoretical foundation since proofs of convergence have been established for them [3], [5]. Some examples exhibited in [5] also show that ORTHODIR (k) may diverge. We know that this may happen for DIOM (k) as well when the symmetric part of A is not positive definite. It is not known whether this can occur in the positive real case, but all the numerical experiments conducted thus far have yielded convergence in such cases. It seems likely that for k sufficiently large the process will converge in the general case but this has not been proved so far.

Besides convergence, an important comparison criterion is the risk of breakdown presented by the different methods. In this respect, DIOM (k) is reliable, as shown by Proposition 1. ORTHOMIN (k) is known to be feasible when A is positive real and can breakdown otherwise. There is no proof that ORTHORES (k) *does not* break down for positive real problems, but examples showing that it does break down for nonpositive real problems are easy to construct [8].

It will be shown below that ORTHORES (k) is in fact equivalent to IOM (k)–DIOM (k). The result that ORTHORES (∞) is equivalent to IOM (∞) is straightforward and was mentioned in [5]. That the truncated versions are also equivalent does not seem to have been noticed.

We begin by briefly recalling the ORTHORES (k) methods and a few of its properties. For more details see [8].

ALGORITHM ORTHORES (k).

1. *Start.* Choose x_0, and compute $r_0=b-Ax_0$.
2. *Iterate.* For $i=0,1,\cdots$ until convergence do:

$$\begin{aligned} a_j^{(i)} &= (Ar_i, r_j)/(r_j, r_j), j=i, i-1, \cdots, i-k+1, \\ b_i &= \left[\sum_{j=i-k+1}^{i} a_j^{(i)}\right]^{-1}, \\ c_j^{(i)} &= b_i c_j^{(i)}, j=i-k+1, \cdots, i, \\ x_{i+1} &= b_i r_i + \sum_{j=i-k+1}^{i} c_j^{(i)} x_j, \\ r_{i+1} &= -b_i A r_i + \sum_{j=i-k+1}^{i} c_j^{(i)} r_j. \end{aligned} \tag{42}$$

The scalars $c_j^{(i)}$ are determined so that the residual r_{i+1} is orthogonal to the previous k residuals. The nontruncated version of the above algorithm, named ORTHORES (∞), consists in starting the above sums from 1 instead of $i-k+1$. The process may break down at any step because a division by zero can occur in computing b_i. It is not difficult to exhibit an example for which this happens at the first step [8].

The nontruncated version ORTHORES (∞) is equivalent to IOM (∞) (or DIOM (∞)) because for both algorithms and residuals satisfy the same orthogonality relation. Note that the truncated versions DIOM (k) and ORTHORES (k) also satisfy the same orthogonality relation, namely $(r_i, r_j)=0$, for $i-k+1<j<i$. This is a hint that the truncated versions are also equivalent, but a more careful proof is needed.

Let us recall that by Lemma 2 the residual vector r_n obtained at the nth step of DIOM (k) is equal to a scalar times the vector v_{n+1} of the truncated Arnoldi process. The residual vectors r_n obtained from any Krylov subspace method are known to satisfy $r_n = p_n(A)r_0$ where p_n is a polynomial of degree n, normalized so that $p_n(0) = 1$.

We will need the following lemma.

LEMMA 6. *Let the residual vectors r_i and r_i' produced by two Krylov subspace methods be such that $r_i = \alpha_i r_i'$, $i = 0, 1, \cdots, n$, where α_i is a scalar with $\alpha_0 = 1$. Assume that the degree of the minimal polynomial for the initial residual vector r_0 is not less than n. Then $r_j = r_j'$, $j = 0, 1, \cdots, n$.*

In other words, two Krylov subspace methods that start with the same initial vector and which produce proportional residual vectors will in fact produce identical iterates.

Proof. For the first method we have $r_i = p_i(A)r_0$ and for the second $r_i' = p_i'(A)r_0$ where the two polynomials p_n and p_n' are such that

$$p_n(0) = p_n'(0) = 1. \tag{43}$$

Since the degree of the minimal polynomial of r_0 is not less than n, the relation

$$r_i = \alpha_i r_i', \qquad i = 0, 1, \cdots, n, \tag{44}$$

yields

$$p_i(\lambda) \equiv \alpha_i p_i'(\lambda), \qquad i = 0, 1, \cdots, n. \tag{45}$$

From (45) and (43) we get $\alpha_i = 1$, $i = 0, 1, \cdots, n$, which completes the proof. □

PROPOSITION 7. *Assume that n steps of both* ORTHORES (k) *and* DIOM (k) *with no pivoting can be achieved starting with the same initial vector x_0 and that the degree of the minimal polynomial of r_0 is not less than n. Then the iterates x_j, $j = 0, 1, \cdots, n$, produced by both algorithms are identical.*

Proof. Let r_i be the residual vectors for ORTHORES (k). According to the lemma all we have to show is that this is proportional to the residual vector produced by DIOM (k), or, equivalently, to the vector v_{i+1} produced by the incomplete Arnoldi process. Note that (42) can be written:

$$r_{i+1} = -b_i\left[Ar_i - \sum_{j=i-k+1}^{i} a_j^{(i)} r_j\right], \tag{46}$$

which is of the same form as (7). Using the fact that the sequences $\{r_i\}_{i=0,1,\cdots}$ and $\{v_i\}_{i=1,2,\cdots}$ are generated by the similar formulas (7) and (46), and that for both sequences the current vector is orthogonal to the previous k vectors, it is easy to show by induction that r_i differs from v_{i+1} only by a multiplicative factor. This completes the proof. □

Clearly, this result can be extended to DIOM (k) with pivoting (Algorithm 4), by considering only the iterates corresponding to the steps where pivoting has not occurred. Though theoretically equivalent, the two methods are quite different in their implementation. ORTHORES (k) is slightly more expensive than DIOM (k) and IOM (k), as shown in Table 1, but requires less storage than DIOM (k). However, these differences are not significant because they represent a small fraction of the total work and storage in general. The computation of b_i in ORTHORES (k) may cause the algorithm to break down, and no result explaining when this may happen seems to exist. Implicitly, the ORTHORES (k) process attempts, like Algorithm 3, to compute the approximate solutions x_j defined by (14) for all j. When one of the intermediate Hessenberg matrices H_j is singular, this solution does not exist and the process breaks down. Algorithm 4, on the other hand, discards x_j in this situation by defining it as being equal to x_{j-1} and computes x_{j+1} instead (or some x_{j+i}, $i \geqq 1$).

4. Application to other Krylov subspace methods. As indicated in [15], many algorithms using Krylov subspaces can be described with equations similar to those of IOM. A sequence of vectors v_j, respesenting the residuals rescaled as in Lemma 2, is generated by requiring some orthogonality conditions. Then the solution is obtained by (11) and (12). The only difference between the various methods resides in the orthogonality conditions forced upon the residual vectors r_j, or equivalently the v_j's. An interesting question is whether the algorithms described earlier can also be adapted to other Krylov subspace methods. The main reason why such versions are sought is that when the matrix A is not positive real, then the regular versions may break down or become unstable, because they *implicitly* solve an upper Hessenberg system with the potentially unstable Gaussian elimination with no pivoting.

The use of pivoting will be very helpful in particular for the Lanczos algorithm, considered next, as the original direct version, called the biconjugate gradient algorithm, faces serious risks of instability and breakdown (see [15]).

4.1. The Lanczos biorthogonalization algorithm. For the following discussion we recall the essential of the Lanczos algorithm for solving a linear system of the form $Ax = b$. See [6], [9], [15].

ALGORITHM 5 (Lanczos).

1. Choose an initial vector x_0 and compute $r_0 = b - Ax_0$. Define $v_1 := u_1 := r_0/(\beta := \|r_0\|)$.
2. For $j = 1, 2, \cdots, m$ do:

$$\tilde{v}_{j+1} := Av_j - \alpha_j v_j - \beta_j v_{j-1}, \tag{47}$$

$$\tilde{u}_{j+1} := A^T u_j - \alpha_j u_j - \delta_j u_{j-1} \quad \text{with } \alpha_j := (Av_j, u_j), \tag{48}$$

$$\delta_{j+1} := |(\tilde{v}_{j+1}, \tilde{u}_{j+1})|^{1/2}, \qquad \beta_{j+1} := \text{sign}\,[(\tilde{v}_{j+1}, \tilde{u}_{j+1})]\delta_{j+1}, \tag{49}$$

$$v_{j+1} := \frac{\tilde{v}_{j+1}}{\delta_{j+1}}, \qquad u_{j+1} := \frac{\tilde{u}_{j+1}}{\beta_{j+1}}. \tag{50}$$

3. Form the approximate solution

$$x_m := x_0 + V_m y_m \tag{51}$$

in which $V_m = [v_1, v_2, \cdots, v_m]$ and

$$y_m = H_m^{-1}(\beta e_1) \tag{52}$$

where H_m is the tridiagonal matrix defined by

$$H_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \cdot & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \beta_m \\ & & & \delta_m & \alpha_m \end{bmatrix}. \tag{53}$$

A direct version of this algorithm exists and was due to Lanczos himself. The algorithm was neglected for a long time because it faces serious stability problems; see [6], [15].

The similarity of part 3 of this algorithm with part 3 of Algorithm 2 indicates that a direct version which uses the *LU* factorization with partial pivoting can easily be formulated. The development of the new algorithm is identical with that of DIOM (k) described in § 3, and we will omit the details. This does not, however, overcome all of the difficulties associated with this algorithm, because the process of building the sequence $\{v_i\}_{i=1,2,\cdots}$ can itself be troublesome. The problem of building the Lanczos vectors in the nonsymmetric case was addressed by Parlett and Taylor [13], who suggest an alternative algorithm which handles the breakdowns associated with the construction of the sequence v_i. This a combination of Parlett and Taylors' "look-ahead" Lanczos [13] process for constructing the Lanczos vectors v_j and our technique of obtaining the approximation x_m as implemented in Algorithm 4 constitute a more reliable version of Algorithm 5.

4.2. Krylov subspace methods based on conjugate residuals. In the context of symmetric indefinite problems, Paige and Saunders [10] proposed an algorithm, named MINRES, which at each step minimizes the residual norm of the residual vector over the Krylov subspace K_m. MINRES uses the same *LQ* factorization of the matrix H_m as SYMMLQ. The algorithm proposed in § 3.3 can also be extended in a similar way to yield a generalization of Paige and Saunders's MINRES to nonsymmetric problems. The details of the algorithm and its properties are beyond the scope of this paper. However, we would like to make an important remark. The nontruncated version of the resulting MINRES algorithm is equivalent to the ORTHOMIN (∞) algorithm [18], [5] which has the property that its residual vectors are semiconjugate, i.e. $(Ar_j, r_i) = 0$, $i < j$. One might then think that, if we truncate the process as is done for the SYMMLQ-like method of § 3.3, we will find an equivalent (and hopefully better) version of ORTHOMIN (k), the truncated version of ORTHOMIN (∞). A more careful analysis shows that this is not the case, i.e., that the nonsymmetric truncated MINRES-like extension of the method described in § 3.3 is not equivalent to ORTHOMIN (k).

An alternative is to generate a direct version by imposing on the v_i's the orthogonality condition known to be satisfied by the residual vectors of the original algorithm. In our case we would like the residual vectors to be semiconjugate [3], which means that the v_i's should satisfy

$$(Av_j, v_i) = 0, \qquad i = 1, 2, \cdots, j-1. \tag{54}$$

The above sequence of vectors would lead to the generalized conjugate residual method, or ORTHOMIN (∞) [3]. Again the computation of the system of vectors satisfying (54) becomes uneconomical as j increases, and a natural idea is to replace such a condition by an incomplete orthogonality condition. Note that the incomplete version of the algorithm thus obtained is equivalent neither to the truncated version ORTHOMIN (k) of ORTHOMIN nor to the truncated version of MINRES.

Computing the sequence of vectors v_i by the recurrence (7) in which the v_i's satisfy the new orthogonality conditions $(Av_j, v_i) = 0$, $i = j-k+1, j-k+2, \cdots, j-1$, and $\|Av_{j+1}\| = 1$, may break down or become unstable. The reason for this is that we are attempting to orthonormalize a sequence of vectors with respect to an indefinite inner product; i.e., we can have $(Av, v) = 0$ for $v \neq 0$. A process similar to the one suggested by Parlett and Taylor can be applied to the sequence of v_i's because in both cases we implicilty deal with the same problem of constructing a sequence of orthogonal polynomials with respect to some indefinite inner product; see [15], [7]. The combination of such a process and the technique using partial pivoting for forming the approximate solution of Algorithm 4 can again be combined to yield a more robust algorithm.

5. Practical considerations.

5.1. General comments. As mentioned in the introduction, one attractive feature of DIOM (k) is its ability to deal with symmetric indefinite systems and nonsymmetric systems. A comparison with SYMMLQ would indeed show that DIOM (2) requires one more vector of storage than SYMMLQ, while computationally each step of DIOM (2) requires $7N$ multiplications against $9N$ for SYMMLQ (see Table 2.) Note that according to the comments following Algorithm 4, each step of DIOM (k) would cost $(3k+2)N$, which for $k=2$ gives $8N$ operations instead of the $7N$ claimed. However, because of the symmetry of the problem, we save one inner product in the formation of v_{j+1}, which explains the result.

There exist other methods which are also less expensive than SYMMLQ. For example Chandra's SYMMBK version [2] based upon the use of 2×2 pivots in the LU factorization of H_m also requires $7N$ multiplications per step. However, the Bunch and Kaufman factorization is still potentially unstable while the LU factorization with partial pivoting is always stable for tridiagonal matrices, as is well known [19, p. 219]. Table 2 shows the work and storage required for several methods dealing with symmetric indefinite problems, including SYMMLQ [10], SYMMBK [2], MCR [2], Stoer and Freund's method [17]. The indirect Lanczos method, equivalent to IOM (2), was described by Parlett [11], who observed that it can be competitive in some environments where the I/O can be realized inexpensively. See also [16].

TABLE 2
Work and storage of some methods for solving symmetric indefinite systems.

Method	Mult./step	Storage
DIOM (2)	$7N$	$6N$
SYMMLQ	$9N$	$5N$
SYMMBK	$7N$	$6N$
MCR	$7N$–$9N$	$7N$
Stoer & Freund	$8N$	$7N$
Parlett (*)	$6N$	$3N$

* This method uses secondary storage. The table reports core memory requirement only.

These methods deal with symmetric problems, but most of them can easily be generalized to nonsymmetric problems, although this does not seem to have been considered so far in the literature.

Consider the four classes of problems enumerated in the introduction. For the class of symmetric positive definite linear systems, there are many effective methods. It seems that a common way of dealing with symmetric indefinite problems is the SYMMLQ algorithm. The third class of problems, dealing with nonsymmetric systems with indefinite symmetric parts, can be effectively handled by several methods including ORTHOMIN (k), IOM (k), DIOM (k), Chebyshev iteration, Concus, Golub and Widlund's generalized conjugate gradient method, etc.

The last class of problems, i.e. the class of nonpositive real problems, is more difficult. Our algorithm DIOM (k) can be applied especially in the cases where the nonsymmetric part $(A-A^T)/2$ is small compared with the symmetric part. When this is *not the case*, we observe that in order for the process to converge, the parameter k must be quite large, thus rendering the method uneconomical. The process may diverge or be very slow if k is too small.

These observations suggest that the Krylov subspace methods may not be suitable for nonpositive real systems with large nonsymmetric parts or such that the origin is well inside their spectra. In fact since k must be large (e.g. $k>9$) for convergence to hold, one might find it preferable to solve the normal equations using e.g. the conjugate gradient method. Moreover, note that in the extreme case where A is unitary, the eigenvalues are on the unit circle, and, as suggested by the theory [14], the convergence of Krylov subspace methods can become very slow. However, the normal equations are clearly trivial to solve since A^TA is the identity matrix. This suggests that the choice between solving the normal equations and using a Krylov subspace method is not always an easy one to make.

5.2. Heuristics. In order to enhance the efficiency of a code based upon IOM (k) or DIOM (k), a number of heuristics are needed. The most important of them are described below.

Dynamic choice of the parameter k. In an efficient implementation of IOM (k), or DIOM (k), we must include a process which chooses automatically the parameter k. Indeed, the user does not in general have any idea of a reasonable choice for k. The possibility of choosing k in a dynamical way is based on the fact that k can be *reduced* during the algorithm without changing the orthogonality relation of Proposition 5. Note, however, that k cannot be reincreased. What this means is that we can start the algorithm with some large k (in our code k starts with the value 9) and then reduce it progressively according to some criterion. The criterion that we use is related to the fact that when the matrix is almost symmetric (or skew-symmetric), then the elements $h_{i,j}$ of the jth column of H_m with $i<j-2$ are small, and can therefore be neglected. This suggests that at a given step j we should subtract from k the number of small elements h_{ij}, where i is between $j-2$ and $j-k+1$. Specifically we redefine k from

$$k := k - \max_{\mu}\left\{\mu \text{ s.t. } \sum_{i=j-k+1}^{j-\mu+1} |h_{ij}| < \text{tol}_1 \sum_{i=j-k+1}^{j} |h_{ij}|\right\}$$

where tol_1 is some tolerance parameter (In our code tol_1 was set to 1.e-03.). The formula above should be modified so as not to yield k less than 2.

With this empirical formula, symmetry or near-symmetry is easily detected, and as a consequence the computational work may be significantly reduced.

Restarting. In IOM (k), the version using secondary storage, it is a necessity to restart the algorithm because of storage. It is also often more effective to include a restarting strategy even for the direct version DIOM (k). Such a strategy would restart the algorithm whenever the convergence becomes unsatisfactory. More precisely the following heuristics have been found to be effective:

$$\text{If } \left(\frac{\rho_j}{\rho_{j-p}}\right) > \text{tol}_2 \quad \text{then restart}$$

where p is some fixed integer (e.g. 5 as in our code), and tol_2 is some positive tolerance parameter. The above criterion for restarting has the following interpretation: restart if the residual norms do not decrease by a sufficient amount in p steps. For IOM (k) we can restart with the vector x_{j-p} which has a smaller residual, but this cannot be done for DIOM (k) unless we save the vector x_{j-p}. The restarting strategy was found to be quite effective for some difficult cases (see § 6.3), but does not change much when convergence is fast enough.

Preconditioning. The efficiency of the incomplete orthogonalization methods can be improved by using preconditioning techniques. One difficulty, however, is that most of the known preconditions assume (directly or indirectly) that $(A+A^T)/2$ is positive definite. A very simple remedy is to add αI to the original matrix, where α is a scalar such that $B=A+\alpha I$ is positive real, and use as preconditioning the preconditioning associated with B. This can be effective in case α is not too large.

6. Numerical experiments. All the numerical experiments have been performed on a DEC-20 computer. Single precision (unit roundoff $\approx 3.7\times 10^{-09}$) is used in the first experiment while double precision (unit roundoff $\approx 10^{-19}$) is used elsewhere.

6.1. Symmetric indefinite problems. We begin with an experiment illustrating the behaviors of DIOM (2), SYMMLQ and the regular conjugate gradient (CG) method on a symmetric indefinite problem. Note that the regular conjugate gradient method may break down if A is indefinite, and is therefore not recommended in general for indefinite problems. It is applied here for the sole purpose of illustration. We choose to take an example from [10], in which the matrix A is of the form $A=B^2-\mu I$, where B is the tridiagonal matrix with typical nonzero row elements $(-1, 2, -1)$ and $\mu=\sqrt{3}$. Note that μ is not an eigenvalue of A, and that it is not near either extremity of the spectrum. In order to demonstrate the fact that IOM (2) and SYMMLQ have similar behavior on this example, we use single precision arithmetic. The problem solved is $Ax=b$, where $b=Ae$, $e=(1, 1, 1, \cdots, 1)^T$. The initial vector is a random vector, the same for the three methods DIOM (2), SYMMLQ, CG. Figure 6.1 compares the behaviors of the three methods for the steps $m=38, 39, \cdots, 65$. The first 37 steps yield almost identical residual norms for the three algorithms and have not been reproduced.

The figure shows that both SYMMLQ and DIOM (2) are superior to the regular conjugate gradient method, but SYMMLQ is not superior to DIOM (2). In fact DIOM (2) is slightly better on this example through the difference is not significant. More significant is the difference obtained when a less efficient way of generating the Lanczos vectors v_i is utilized. Indeed, in our first experiments with the above example we found DIOM (2) significantly more accurate than SYMMLQ. A careful analysis showed that the reason for this superiority was due to the fact that the original code of SYMMLQ was not using the best formula for generating the Lanczos vectors. As mentioned in [12], it is more accurate to generate the Lanczos vectors by the recurrence

$$q := Av_j - \beta_j v_j,$$

$$\alpha_j := (q, v_j),$$

$$q := q - \alpha_j v_j,$$

$$v_{j+1} := \frac{q}{(\beta_{j+1} := \|q\|)}$$

rather than by

$$q := Av_j,$$

$$\alpha_j := (q, v_j),$$

$$q := q - \alpha_j v_j - \beta_j v_j,$$

$$v_{j+1} := \frac{q}{(\beta_{j+1} := \|q\|)}.$$

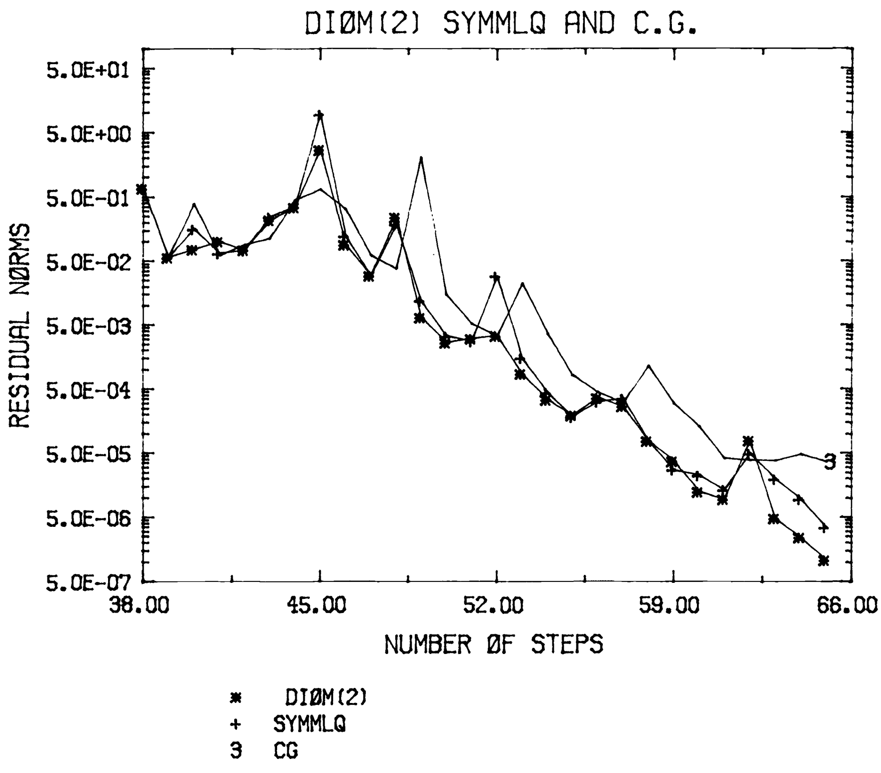


FIG. 6.1. *Comparison of* DIOM (2), SYMMLQ, CG *on a symmetric indefinite problem of dimension* 50.

Remarks. 1. The plot of Fig. 6.1 reports the numerical results corresponding to the first (best) of the above formulations for *both* SYMMLQ and DIOM (2).

2. Similar observations are made when double precision is used.

3. The residual norms in Fig. 6.1 are obtained for DIOM (2) by the formula (13) and for SYMMLQ by an equivalent formula. Both formulas have been checked to produce accurate result in the final step. Note that after step 65, these formulas deteriorate slowly as the maximum possible accuracy given the norm of A and the unit roundoff is being reached, so the estimates of the residual norms become meaningless.

All this demonstrates the important fact mentioned earlier that the main source of error lies in the computation of the v_i's rather than in the formation of the approximate solution by formulas (11) and (12).

6.2. Nonsymmetric problems with positive real matrices. In this test we compare the direct version DIOM (k) with the indirect version IOM (k) on the following model problem:

$$A=\begin{bmatrix} B & -I & & & \\ -I & B & & & \\ & & \ddots & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & B \end{bmatrix} \quad \text{with } B=\begin{bmatrix} 4 & t_1 & & & \\ t_2 & 4 & & & \\ & & 4 & \ddots & \\ & & \ddots & \ddots & t_1 \\ & & & t_2 & 4 \end{bmatrix}$$

and $\dim(A)=N=200$, $\dim(B)=n=10$, $t_1=-1+\delta$, $t_2=-1-\delta$, where δ is some parameter. The above example originates from the centered difference discretization of the operator $-\Delta+c\partial/\partial x$, where c is a constant.

The performances of IOM (4) and DIOM (4) have been compared on the above example with $\delta=0.5$, $b=Ae$, $e=(1,1,\cdots,1)^T$. This test was performed in double precision. For simplicity none of the heuristics has been used, i.e. the algorithm is not restarted and k is constantly equal to 4. The process is stopped as soon as the residual is less than 10^{-5}. This has required 57 steps. As expected, the residual norms and the final iterates produced by both algorithms are identical. The run times on the DEC-20 are approximately as follows:

IOM (4): 5.60 sec, DIOM (4): 3.60 sec.

Note that IOM (4) requires $5N$ vectors of core memory storage while DIOM (4) requires $9N$. It is interesting to decompose the run times for IOM (4) into I/O time and computing time. The time for writing the vectors v_i's into disk memory and reading them back when forming the solution is about 2.50 sec, i.e. 47% of the overall CPU time. The I/O time can be further decomposed into write time = 1.39 sec and read time = 0.91 sec. This distribution is obviously very much machine dependent, and the comparison may change completely for other architectures. It may even happen that IOM becomes faster than DIOM in cases where the I/O time can be masked by performing much of the computation and the I/O in parallel.

6.3. Indefinite and nonsymmetric problems. In this example we test DIOM (k) on the matrix $B=(A-\mu I)$, where A is defined as in the previous experiment, with δ set again to 0.5 and where μ is chosen equal to 0.25. This is a nonsymmetric and indefinite problem.

The right-hand side is defined as previously, and the initial vector is again a random vector with an initial residual norm of 19.08 .... The process is stopped as soon as the residual norm is below 10^{-5}. A straightforward application of DIOM (4), with a fixed k and no restarting, converged in 89 steps. Then we used a preconditioning matrix M the incomplete Choleski factorization associated with the Laplace operator, i.e. the incomplete Choleski factorization of $(A+A^T)/2$. The system $M^{-1}Ax=M^{-1}b$ was then solved by a call to DIOM (2). This preconditioned DIOM produced a generalized residual vector $M^{-1}r_n$ of norm less than 10^{-5} in 31 steps, thus significantly improving the previous performance. Note that, surprisingly, DIOM (k) with $k>2$ did not perform better than with $k=2$ since it took 41 steps for DIOM (3) to converge and 48 steps for DIOM (4).

As μ increases, the problem becomes more difficult to solve. When $\mu = 0.5$, for example, DIOM (k) either diverges (e.g. for $k = 2$) or showed signs of very slow convergence. Using the same preconditioning as above, IOM (7) converged in 125 steps. The restarting strategy used was the one described in § 5.2 with tol $2 = 1$. The criterion for restarting was tested every $p = 5$ steps and, at a minimum, 10 steps were taken at each iteration. With this strategy the process was restarted at steps 20, 30, 70, and 110. Note that we took tol $1 = 0$, which means that k was constantly equal to 7.

When μ becomes even larger, the above preconditioning does not improve the convergence, which can become very poor. It seems more appropriate to use the conjugate gradient method applied to the normal equations in those cases.

Acknowledgments. The mathematical equivalence between ORTHORES (k) and IOM (k) (§ 3.5) was established after a suggestive question was raised by one of the referees. The author is indebted to both referees for their instructive and helpful comments.

REFERENCES

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

[2] R. Chandra, *Conjugate gradient methods for partial differential equations*, PhD thesis, Tech. Rep. 129, Yale Univ., New Haven, CT, 1981.

[3] S. C. Eisenstat, H. C. Elman and M. H. Schulz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 345–357.

[4] H. C. Elman, *Preconditioned conjugate gradient methods for nonsymmetric systems of linear equations*, Advances in Computer Methods for Partial Differential Equations-IV, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, New Brunswick, NJ, 1981, pp. 409–417.

[5] ———, *Iterative methods for large sparse nonsymmetric systems of linear equations*, PhD thesis, Technical Report 229, Yale Univ., New Haven, CT, 1982.

[6] R. Fletcher, *Conjugate gradient methods for indefinite systems*, in Proceedings of the Dundee Biennial Conference on Numerical Analysis 1974, Univ. of Dundee, Scotland, New York, 1975, pp. 73–89.

[7] W. B. Gragg, *Matrix interpretation and applications of continued fraction algorithm*, Rocky Mountain J. Math., 4 (1974), pp. 213–225.

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

[9] C. Lanczos, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[10] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–624.

[11] B. N. Parlett, *A new look at the Lanczos algorithm for solving symmetric systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 323–246.

[12] ———, *The Symmetric Eignevalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[13] B. N. Parlett and D. Taylor, *A look ahead Lanczos algorithm for unsymmetric matrices*, Technical Report PAM-43, Center for Pure and Applied Mathematics, Berkeley, CA, 1981.

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

[15] ———, *The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems*, SIAM J. Numer. Anal., 19 (1982), pp. 470–484.

[16] H. D. Simon, *The Lanczos algorithm for solving symmetric linear systems*, PhD Thesis, Univ. of California at Berkeley, Berkeley, CA, 1982.

[17] J. Stoer and R. Freund, *On the solution of large indefinite systems of linear equations by conjugate gradient algorithms*, Technical Report, Institut für Angewandte Mathematik und Statistik, Universität Würzburg, Würzburg, W. Germany.

[18] P. K. W. VINSOME, ORTHOMIN, *an iterative method for solving sparse sets of simultaneous linear equations*, in Proceedings of the Fourth Symposium on Reservoir Simulation, Society of Petroleum Engineers of the American Inst. of Mechanical Engineers, 1976, pp. 149–159.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

AN EFFICIENT ALGORITHM FOR THE REGULARIZATION OF ILL-CONDITIONED LEAST SQUARES PROBLEMS WITH TRIANGULAR TOEPLITZ MATRIX*

LARS ELDÉN†

Abstract. Ill-conditioned least squares problems, where the matrix is triangular and has Toeplitz structure, arise when certain Volterra integral equations with stationary kernel are discretized. Tikhonov–Phillips regularization can be used to stabilize the solution. An efficient algorithm for computing regularized solutions of such problems is described. The number of arithmetic operations required is $O(n^2)$, where n is the dimension of the matrix. A numerical example is given, where an inverse heat conduction problem is solved.

Key words. least squares, ill-conditioned, triangular, Toeplitz, regularization, inverse problem, heat conduction

1. Introduction.

Consider the linear system of equations

$$Kf = g, \tag{1.1}$$

where K is an $n \times n$ upper triangular matrix with Toeplitz structure, i.e.

$$K = \begin{bmatrix} k_1 & k_2 & k_3 & \cdots & & k_{n-1} & k_n \\ & k_1 & k_2 & k_3 & & \cdots & k_{n-1} \\ & & \ddots & \ddots & \ddots & & \vdots \\ & & & \ddots & \ddots & \ddots & \vdots \\ & & & & k_1 & k_2 & k_3 \\ & 0 & & & & k_1 & k_2 \\ & & & & & & k_1 \end{bmatrix}. \tag{1.2}$$

Such systems arise when convolution type Volterra integral equations of the first kind,

$$\int_0^t k(t-s)f(s)\,ds = g(t), \qquad 0 \leqq t \leqq T_0, \tag{1.3}$$

are discretized; see [9], [13]. (Actually the discretization of (1.3) leads to a *lower* triangular system of equations [9], [13], but this can be put in upper triangular form simply by reordering the unknowns; it turns out that our algorithm is more conveniently described if we assume that the matrix is upper triangular.)

In many cases the linear system (1.1) arising from the discretization of (1.3) is relatively well conditioned; see [9], [2, p. 896ff.]. Then (1.1) can be solved simply by backward substitution.

However, there are important applications, where the discrete system (1.1) is so ill conditioned that a straightforward solution is impossible. This happens e.g. when the kernel function $k(t)$ in (1.3) is smooth and $k(0)$ is equal to zero, or is very small. Examples of such problems are inverse heat transfer problems [7] [16, p. 88], and deconvolution of time series [13]. In this case it is not meaningful to try to satisfy (1.1) exactly, because small perturbations of the right-hand side can cause very large perturbations of the solution. Note that (1.1) is equivalent to the least squares problem

$$\min_f \|Kf-g\|,$$

* Received by the editors March 10, 1982, and in revised form November 26, 1982. This work was supported by the Swedish Natural Science Research Council (NFR).

† Department of Mathematics, University of Linköping, S-581 83 Linköping, Sweden.

where the norm is the Euclidean vector norm. To alleviate ill-conditioning it is necessary to stabilize the problem. This can be done using the regularization method of Tikhonov [16], Phillips [14]. Then we solve

$$\min_f \{\|Kf-g\|^2+\mu^2\|Lf\|^2\}, \tag{1.4}$$

where L is a discretization of some differentiation operator, e.g.

$$L=1/h\begin{bmatrix} 1 & -1 & & & & \\ & 1 & -1 & & 0 & \\ & & 1. & -1. & & \\ & 0 & & \ddots & \ddots & \\ & & & & 1 & -1 \end{bmatrix}, \qquad L \text{ is } (n-1)\times n, \tag{1.5}$$

and h is a steplength parameter. The value of the *regularization parameter* μ controls the degree of smoothness of the solution.

For a discussion of regularization and related methods we refer to [16], [11], [12]. Regularization in connection with Volterra equations of the first kind is considered in [15]. Surveys on numerical algorithms are given in [3], [17].

In § 2 of this paper we develop an efficient algorithm for solving (1.4), in the case when both K and L are upper triangular Toeplitz matrices. Note that discretizations of differentiation operators often lead to Toeplitz matrices; see (1.5). The algorithm is based on the computation of a QR-decomposition of the matrix

$$\begin{bmatrix} K \\ \mu L \end{bmatrix},$$

using plane rotations. Therefore the algorithm has good stability properties.

The Toeplitz structure of the matrices is exploited so that the algorithm requires $O(n^2)$ operations. The implementation of the algorithm is briefly discussed in § 3. There we also compare it to alternative algorithms for problems with Toeplitz structure [10], [5], [13], [4]. The results of timing experiments are given, verifying the $O(n^2)$ behavior.

In § 4 we apply the new algorithm to an inverse heat conduction problem.

2. Description of the algorithm. The regularized least squares problem (1.4) is equivalent to

$$\min_f \left\|\begin{pmatrix} K \\ \mu L \end{pmatrix} f-\begin{pmatrix} g \\ 0 \end{pmatrix}\right\|, \tag{2.1}$$

which can be solved as follows.

Determine an orthogonal matrix Q, such that

$$Q^T\begin{pmatrix} K \\ \mu L \end{pmatrix}=\begin{pmatrix} K_\mu \\ 0 \end{pmatrix}, \qquad \bar{g}=Q^T\begin{pmatrix} g \\ 0 \end{pmatrix}=\begin{pmatrix} \bar{g}_1 \\ \bar{g}_2 \end{pmatrix}, \tag{2.2}$$

where K_μ is upper triangular. The solution of (2.1) can now be obtained by solving

$$K_\mu f = \tilde{g}_1. \tag{2.3}$$

This is the QR-decomposition method [8].

In our algorithm the matrix Q is a product of plane rotations. It is not necessary to form Q explicitly.

We now describe how the decomposition (2.2) can be computed. The algorithm is conveniently developed in terms of a small example ($n = 5$). Consider the augmented matrix

$$\left[\begin{array}{c|c} K & g \\ \mu L & 0 \end{array}\right] = \left[\begin{array}{ccccc|c} k_1 & k_2 & k_3 & k_4 & k_5 & g_1 \\ & k_1 & k_2 & k_3 & k_4 & g_2 \\ & & k_1 & k_2 & k_3 & g_3 \\ & & & k_1 & k_2 & g_4 \\ & & & & k_1 & g_5 \\ l_1 & l_2 & l_3 & l_4 & l_5 & 0 \\ & l_1 & l_2 & l_3 & l_4 & 0 \\ & & l_1 & l_2 & l_3 & 0 \\ & & & l_1 & l_2 & 0 \\ & & & & l_1 & 0 \end{array}\right]$$

where l_i is equal to μ times a component of L.

Usually the matrix L is a band matrix, but it turns out not to be possible to exploit the band structure (cf. § 3). For the description of the algorithm we therefore assume that L is a full upper triangular matrix. It is not necessary to assume that L is a square matrix (cf. (1.5.)). For definiteness we here make that assumption.

In the first step of the algorithm we zero all diagonal elements in L. This is done by n rotations in the planes $(1, n+1), (2, n+2), \cdots, (n, 2n)$. Because of the Toeplitz structure all rotations involve the same angle of rotation. The result of these transformations is

$$\left[\begin{array}{ccccc|c} k_1' & k_2' & k_3' & k_4' & k_5' & g_1' \\ & k_1' & k_2' & k_3' & k_4' & g_2' \\ & & k_1' & k_2' & k_3' & g_3' \\ & & & k_1' & k_2' & g_4' \\ & & & & k_1' & g_5' \\ 0 & l_2' & l_3' & l_4' & l_5' & g_6' \\ & 0 & l_2' & l_3' & l_4' & g_7' \\ & & 0 & l_2' & l_3' & g_8' \\ & & & 0 & l_2' & g_9' \\ & & & & 0 & g_{10}' \end{array}\right].$$

Note that the Toeplitz structure is not destroyed. However, the zeros in the lower part of the right-hand side are filled in. Obviously only $4n$ elements need be computed, namely the elements in rows 1 and $n+1$, and the right-hand side.

The first superdiagonal of the (transformed) L matrix is zeroed in the second step. We apply $n-1$ rotations in the planes $(2, n+1), (3, n+2), \cdots, (n, 2n-1)$. Again the same angle of rotation can be used in all transformations. The result is

$$\left[\begin{array}{ccccc|c}
k'_1 & k'_2 & k'_3 & k'_4 & k'_5 & g'_1 \\
 & k''_1 & k''_2 & k''_3 & k''_4 & g''_2 \\
 & & k''_1 & k''_2 & k''_3 & g''_3 \\
 & & & k''_1 & k''_2 & g''_4 \\
 & & & & k''_1 & g''_5 \\
0 & 0 & l''_3 & l''_4 & l''_5 & g''_6 \\
 & 0 & 0 & l''_3 & l''_4 & g''_7 \\
 & & 0 & 0 & l''_3 & g''_8 \\
 & & & 0 & 0 & g''_9 \\
 & & & & 0 & g'_{10}
\end{array}\right].$$

In this step $4(n-1)$ elements in the argumented matrix must be computed. Note that rows 2 to n of the upper block, and the whole lower block, still have Toeplitz structure. After n steps of this algorithm we have computed the matrix K_μ and the right-hand side $\bar{g}_1$ in the decomposition (2.2).

It is now seen that in step i of the algorithm, $4(n-i+1)$ matrix elements must be computed. If we use standard plane (Givens) rotations each transformed element requires two multiplications to be performed. Therefore the total count is

$$2 \cdot \sum_{i=1}^{n} 4(n-i+1) \approx 4n^2$$

multiplications. In addition to that approximately $n^2/2$ operations are needed for solving (2.3).

The algorithm can be modified to handle the case where K and L are almost triangular. Assume, e.g., that K and L are upper Hessenberg matrices. This structure may arise when we discretize (1.3) using piecewise linear polynomials and collocation on an equidistant grid. Then we can partition

$$\text{(2.4)} \qquad \begin{bmatrix} K \\ \mu L \end{bmatrix} = \begin{bmatrix} \kappa_1^T & \sigma \\ K_1 & \kappa_2 \\ \lambda_1^T & \tau \\ \mu L_1 & \lambda_2 \end{bmatrix},$$

where K_1 and L_1 are upper triangular Toeplitz matrices of dimension $n-1$, $\kappa_i, \lambda_i, i = 1, 2$, are $(n-1)$-vectors, and σ and τ are scalars.

Using the above algorithm we can transform (2.4) into

$$\begin{bmatrix} \kappa_1^T & \sigma \\ K'_1 & \kappa'_2 \\ \lambda_1^T & \tau \\ 0 & \lambda'_2 \end{bmatrix}.$$

Then by $3n-2$ plane rotations (or $2n-2$ plane rotations and a Householder transformation) we can put all elements below the main diagonal equal to zero. Thus the total operation count is $O(n^2)$.

3. Implementation and efficiency considerations. The algorithm developed in § 2 needs $4n$ memory locations for storing the right-hand side and one row in K and L respectively. Since K_μ does not have Toeplitz structure its $n(n+1)/2$ elements must also be stored, preferably in a one-dimensional array. From the derivation of the algorithm we see that once a row of K_μ, say the ith row, has been computed, it is not needed again until the back substitution, when f_i is to be computed. This means that the algorithm will perform well on a computer with virtual storage, since no unnecessary page faults will occur (note that this is usually not true if K_μ is stored in a two-dimensional Fortran array).

Further, the algorithm can be efficiently implemented on a computer with limited fast memory, if it has a direct access auxiliary store. Once a row of K_μ has been computed it can be written out on auxiliary store. Only $4n$ memory locations are needed in fast memory for storing matrix elements and vectors. This may be important in applications, where the problems are very large; see [13], where a time series problem with $n=512$ is solved.

Other algorithms have been suggested for ill-conditioned problems with Toeplitz structure. In [5] a method based on eigenvector expansion is described. The operation count is $O(n^3)$. An algorithm for solving the normal equations

$$(K^TK+\mu^2L^TL)f=K^Tg,$$

where K and L have Toeplitz structure, is given in [10]. The number of multiplications is Cn^2, where C is fairly large: approximately 20 (S. Ljung, private communication). It may be possible to reduce the value of C somewhat, by utilizing the triangular structure. Thus our algorithm is faster. Further, it has the advantage that it is based on orthogonal transformations.

The algorithms of O'Leary [13] and Björck [4] are both based on Lanczos bidiagonalization. In [13] it is shown how they can be adapted for the efficient solution of problems with Toeplitz structure. If n is a power of two, then the number of multiplications is approximately

$$11nk+4nk\log_2 n,$$

where k denotes the number of Lanczos steps (k is assumed to be relatively small compared to n).

To compare the operation count for these algorithms to that of ours we assume that $k=\alpha n$ and solve the equation

$$\alpha(11n^2+4n^2\log_2 n)=4.5n^2 \tag{3.1}$$

for a couple of values of α. The results are given in Table 3.1. The table shows that if k is taken as large as $0.15n$ then our algorithm is faster for all n larger than 27. On the other hand, if k is equal to $0.1n$, then the Lanczos based algorithms are faster for most problems that can be solved without using auxiliary storage.

TABLE 3.1
Approximate solution of (3.1) *for different values of* α.

α	0.05	0.1	0.15
Sol. of (3.1)	$8.8\cdot 10^5$	362	27

It is seen that the comparison depends very critically on k (the number of Lanczos steps). The problem of choosing a good value of k is nontrivial; see [13], [4].

Note that the above arguments on efficiency are based on operation counts only. A fair comparison should also take into account storage, accuracy, ease of use, and actual timing experiments.

To verify the $O(n^2)$ behavior of our algorithm we ran a number of tests with different values of n. The program was written in Fortran and run on a DEC-10 computer. Only the timing of the in-core version was recorded. The results are given in Table 3.2.

In some cases both K and L have band structure. It is not possible to modify the new algorithm to take advantage of band structure to obtain an $O(n)$ operation count. The reason is that even if only one of the identical rotations need be applied to the matrix elements, all rotations must be applied to the right-hand side components.

TABLE 3.2
CPU-*time in seconds for our algorithm. * The average over* 10 *runs*; ** *The average over* 5 *runs.*

n	25	50	100
CPU-time	0.013*	0.062*	0.249**

Algorithms for problems with band structure are given in [6] (this issue, pp. 237–254). The number of operations is $O(n)$.

4. A numerical example. To illustrate the use of the new algorithm we apply it to an inverse heat conduction problem. This application will be described in more detail in [7].

Consider the parabolic equation

$$\begin{aligned}
&u_t = \kappa u_{xx}, && 0 \leqq x \leqq 1, \quad 0 \leqq t \leqq 1,\\
&u(0,t) = g(t), && 0 \leqq t \leqq 1,\\
&u_x(0,t) = 0, && 0 \leqq t \leqq 1,\\
&u(x,0) = 0, && 0 \leqq x \leqq 1,\\
&u(1,t) = f(t), && 0 \leqq t \leqq 1, \quad \text{unknown.}
\end{aligned}$$

This Cauchy problem for a parabolic equation is ill posed. It can be formulated as a Volterra integral equation of the first kind,

$$\int_0^t k(t-s)f(s)\,ds = g(t), \qquad 0 \leqq t \leqq 1, \tag{4.1a}$$

where

$$k(t) = \pi\kappa \sum_{n=0}^{\infty} (-1)^n (2n+1) \exp\left(-(2n+1)^2 \frac{\pi^2 \kappa t}{4}\right). \tag{4.1b}$$

(4.1) was discretized using the midpoint method [9]. With a prescribed solution $f(t)$ we computed the data function $g(t)$ by solving the well-posed problem

$$\begin{aligned} &u_t = \kappa u_{xx}, && 0 \leqq x \leqq 1, \quad 0 \leqq t \leqq 1, \\ &u_x(0,t) = 0, && 0 \leqq t \leqq 1, \\ &u(1,t) = f(t), && 0 \leqq t \leqq 1. \\ &u(x,0) = 0, && 0 \leqq x \leqq 1, \end{aligned}$$

using the Crank–Nicolson method. The steplengths in time and in space were both taken equal to 0.02. The dimension of K was equal to 50. We used $\kappa = 0.5$.

In Fig. 4.1 the function $k(t)$ is plotted.

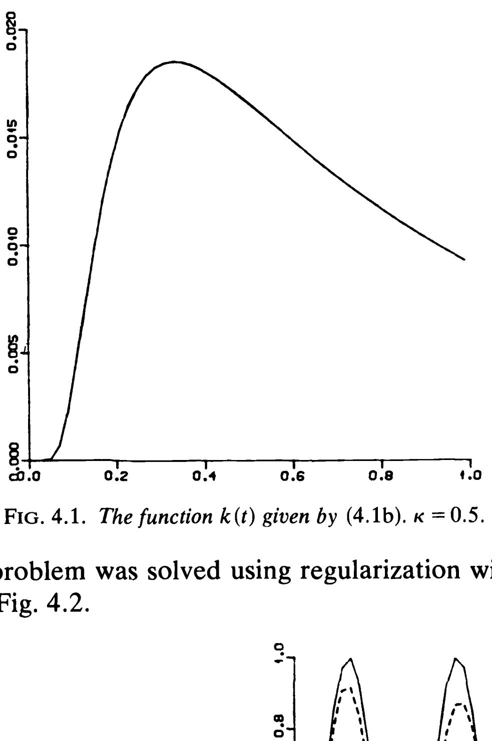


FIG. 4.1. *The function* $k(t)$ *given by* (4.1b). $\kappa = 0.5$.

The discretized problem was solved using regularization with L as in (1.5). The results are plotted in Fig. 4.2.

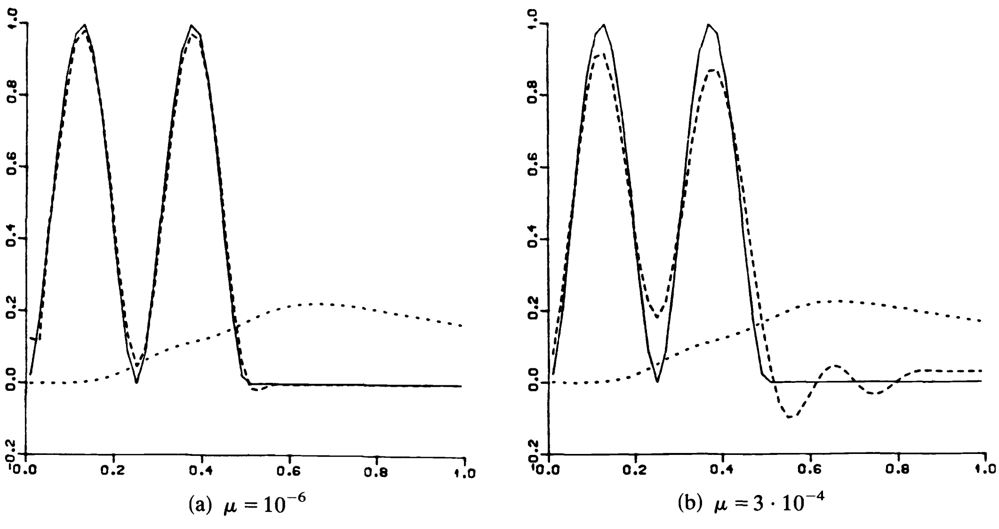


FIG. 4.2. *The numerical solution of* (4.1) *is illustrated. Correct solution* $f(t)$ (*solid*), *approximate solution* (*dashed*), *and data function* $g(t)$ (*dotted*). *In* (b) *the data were perturbed,* $\|\delta g\|/\|g\| = 6.3 \cdot 10^{-3}$.

In Fig. 4.2(b) the data were perturbed:

$$\tilde{g}_i = g_i + \eta_i,$$

where the η_i were pseudorandom numbers, taken from a normal distribution with mean zero and standard deviation equal to 10^{-3}.

REFERENCES

[1] R. S. ANDERSSEN, F. R. DE HOOG AND M. A. LUKAS, eds., *The Application and Numerical Solution of Integral Equations*, Sijthoff & Noordhoff, Alphen aan den Rijn, Netherlands, 1980.

[2] C. T. H. BAKER, *The Numerical Treatment of Integral Equations*, Clarendon Press, Oxford, 1977.

[3] Å. BJÖRCK AND L. ELDÉN, *Methods in numerical algebra for ill-posed problems*, LiTH-MAT-R-33-1979, Dept. Mathematics, Linköping Univ., Linköping, Sweden, 1979.

[4] Å. BJÖRCK, *A bidiagonalization algorithm for solving ill-posed systems of equations*, LiTH-MAT-R-80-33, Dept. Mathematics, Linköping Univ., Linköping, Sweden, 1980.

[5] M. P. EKSTROM AND R. L. RHOADS, *On the application of eigenvector expansions to numerical deconvolution*, J. Comp. Phys., 14 (1974), pp. 319–340.

[6] L. ELDÉN, *An algorithm for the regularization of ill-conditioned, banded least squares problems*, this Journal, this issue, pp. 237–254.

[7] ———, *The numerical solution of a noncharacteristic Cauchy problem for a parabolic equation*, in Numerical Treatment of Inverse Problems in Differential and Integral Equations, P. Deuflhard and E. Hairer, eds., Birkhäuser, Boston, 1983, pp. 244–267.

[8] G. H. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206–216.

[9] P. LINZ, *A survey of methods for the solution of Volterra integral equations of the first kind*, in [1], pp. 183–194.

[10] S. LJUNG AND L. LJUNG, *Fast numerical solution of Fredholm integral equations with stationary kernels*, BIT, 22 (1982), pp. 54–72.

[11] M. A. LUKAS, *Regularization*, in [1], pp. 151–182.

[12] G. F. MILLER, *Fredholm equations of the first kind*, in Numerical Solution of Integral Equations, L. M. Delves and J. Walsh, eds., Clarendon Press, Oxford, 1974.

[13] D. P. O'LEARY AND J. A. SIMMONS, *A bidiagonalization-regularization procedure for large scale discretizations of ill-posed problems*, this Journal, 2 (1981), pp. 474–489.

[14] D. L. PHILLIPS, *A technique for the numerical solution of certain integral equations of the first kind*, J. Assoc. Comput. Mach., 9 (1962), pp. 84–96.

[15] W. W. SCHMAEDEKE, *Approximate solution of Volterra integral equations of the first kind*, J. Math. Anal. Appl., 23 (1968), pp. 604–613.

[16] A. N. TIKHONOV AND V. Y. ARSENIN, *Solutions of Ill-Posed Problems*, Winston & Sons, Washington, DC, 1977.

[17] J. M. VARAH, *A practical examination of some numerical methods for linear discrete ill-posed problems*, SIAM Rev., 21 (1979), pp. 100–111.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 1, March 1984

AN ALGORITHM FOR THE REGULARIZATION OF ILL-CONDITIONED, BANDED LEAST SQUARES PROBLEMS*

LARS ELDÉN†

Abstract. An efficient and stable algorithm for solving least squares problems

$$\min_f \{\|Kf-g\|_2^2+\mu^2\|Lf\|_2^2\},$$

where the matrix K is ill conditioned, and K and L have band structure, is studied. The algorithm, which is a variant of an algorithm given by George and Heath, is based on QR-decomposition by plane rotations. The amount of work depends linearly on the dimension. It is shown that the algorithm does not introduce any unnecessary fill-in. The efficiency and stability of the new algorithm are compared to those of the algorithm based on normal equations.

The algorithm can also be used for computing smoothing splines.

Key words. least squares, regularization, band structure, QR-decomposition, plane rotations, ill-conditioned, numerical stability, fill-in, smoothing spline

1. Introduction.

Extremely ill-conditioned least squares problems,

$$\min_f \|Kf-g\|_2, \tag{1.1}$$

where K is an $m\times n$ matrix, $m\geqq n$, arise when Fredholm integral equations of the first kind,

$$\int_a^b k(x,y)f(y)\,dy=g(x), \qquad c\leqq x\leqq d, \tag{1.2}$$

with smooth kernel k, are discretized (see e.g. [17, pp. 175–188]). If

$$k(x,y)=0 \quad \text{for } |x-y|>M,$$

then the discretization can be performed so that the matrix K has band structure. For the preliminary discussion it is sufficient to define a matrix to have band structure if all elements in the upper right and the lower left corners are zero. Note that we allow $m>n$. Such problems arise e.g. in image restoration; see [2], [10].

Usually K is so ill conditioned that the straightforward solution of (1.1) using any standard method will give a completely meaningless result. Typically such a solution oscillates very rapidly. The ill-conditioning can be alleviated, and the solution can be forced to be smooth, if (1.1) is replaced by

$$\min_f \{\|Kf-g\|_2^2+\mu^2\|Lf\|_2^2\}, \tag{1.3}$$

where L is a $p\times n$ matrix, which is usually chosen equal to the identity matrix or some discretization of a differentiation operator. Typically L is a band matrix with small band width; see [15], [9].

This is the *regularization method* of Tikhonov and Phillips [20], [18]. The degree of smoothness of the solution can be controlled by choosing a suitable value of the *regularization parameter* μ. In many cases a good value in some sense (see [17, p. 182]) is not known a priori, and it is desirable to solve (1.3) for several different μ. Usually μ is chosen rather small.

* Received by the editors October 28, 1981, and in final revised form January 3, 1983.

† Department of Mathematics, University of Linköping, S-581 83 Linköping, Sweden.

A related way of dealing with the ill-conditioning is to impose a bound on the solution of (1.1) and consider the least squares problem with a quadratic constraint

$$\min_{f\in B} \|Kf-g\|_2^2, \qquad B=\{f: \|Lf\|_2 \leqq \omega\}. \tag{1.4}$$

Using the method of Lagrange multipliers, (1.4) can be reduced to solving iteratively a nonlinear equation, where in each iteration a least squares problem (1.3) must be solved; see [9], [11].

Banded least squares problems of the type (1.3) also occur in connection with smoothing splines [6].

Algorithms for the solution of (1.3) in the case when K is a dense matrix have been given by several authors; see e.g. [9], [11], [14], [15], [4], [21], and the papers cited there. George and Heath [12] have recently developed a general algorithm for the solution of sparse least squares problems using Givens (plane) rotations. In this paper we study how their algorithm can be used for the efficient solution of (1.3), where we have band structure.

The algorithm, which we shall call the G2B algorithm (Givens rotations on a double band matrix), is described in § 2. We show that the number of operations needed for the solution of (1.3) is approximately $2n(w_1^2+w_2^2)$, where w_1 and w_2 are the band widths of K and L respectively. In [12] a "suboptimal heuristic" row ordering is proposed. We show that when applied to our problem this ordering is indeed optimal in the sense that no unnecessary fill-in is introduced.

An alternative to the G2B algorithm for solving (1.3) is to form the normal equations and solve these by Cholesky decomposition. The band structure can easily be utilized, and, in fact, the normal equations algorithm is faster. However, the G2B algorithm has better stability properties. In § 3 we discuss the drawbacks of the normal equation algorithm and compare it to the G2B algorithm.

In § 4 we present some computer tests where we compare the efficiency of the G2B algorithm and the normal equation method. We also give a numerical example, which shows that the G2B algorithm has much better stability properties.

2. Description of the G2B algorithm. Since our algorithm is based on a QR-decomposition, it is natural to write (1.3) in the form

$$\min_f \left\| \begin{pmatrix} K \\ \mu L \end{pmatrix} f - \begin{pmatrix} g \\ 0 \end{pmatrix} \right\|_2. \tag{2.1}$$

This can be reduced as follows:

Determine an orthogonal matrix Q_1 such that

$$Q_1^T K = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \begin{matrix} \}\, n \\ \}\, m-n \end{matrix}, \qquad Q_1^T g = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \begin{matrix} \}\, n \\ \}\, m-n \end{matrix} \tag{2.2}$$

where R_1 is an upper triangular band matrix. We now assume that $r_{ij}=0$, if $j-i>b_1$, i.e. the band width of R_1 is equal to $w_1=b_1+1$.

The decomposition (2.2) can be computed by a sequence of plane rotations or Householder transformations. A detailed description of an algorithm for computing the QR-decomposition of a band matrix is given in [16, p. 212]. More generally (2.2) can be computed efficiently using the algorithm in [12]. It is not necessary to form Q_1 explicitly or store the transformations that make up Q_1.

In many cases the matrix L is originally an upper triangular band matrix. If not so, it can be reduced to upper triangular form by an orthogonal transformation

$$Q_2^T L = R_2, \tag{2.3}$$

where R_2 is an upper triangular band matrix with band width $w_2 = b_2 + 1$; for simplicity we assume that L (and R_2) has full row rank, i.e. rank $(L) = p$. In the following we also assume that $w_2 \leqq w_1$. This is no restriction, as will be seen below.

Since the Euclidean norm is invariant under orthogonal transformations, we may introduce the orthogonal matrix

$$\begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}$$

of dimension $m+p$ in (2.1). Using (2.2) and (2.3) we now see that (2.1) is equivalent to

$$\min_f \left\| \begin{pmatrix} R_1 \\ \mu R_2 \end{pmatrix} f - \begin{pmatrix} g_1 \\ 0 \end{pmatrix} \right\|_2. \tag{2.4}$$

Starting out from (2.4) we can solve (2.1) efficiently for different values of μ. By a sequence of plane rotations, the matrix in (2.4) is reduced to upper triangular form

$$Q_3^T \left(\begin{array}{c|c} R_1 & g_1 \\ \mu R_2 & 0 \end{array} \right) = \left(\begin{array}{c|c} R_\mu & \tilde{g}_1 \\ 0 & \tilde{g}_2 \end{array} \right). \tag{2.5}$$

The solution of (2.1) is now computed from

$$R_\mu f = \tilde{g}_1. \tag{2.6}$$

In the computation of (2.5) we use a variant of the algorithm of George and Heath [12].

We first note that the algorithm given in [12] is designed for the solution of very general sparse least squares problems. Thus the determination of a data structure for storing the upper triangular matrix corresponding to R_μ is an important part of the algorithm. In our problem this part is trivial: it is sufficient to note that the matrix R_μ in (2.5) has the same band width as R_1.

We then have only one remaining step of the algorithm as formulated in [12, p. 72]:

Compute R_μ by processing the rows of $\begin{pmatrix} R_1 \\ \mu R_2 \end{pmatrix}$ one by one, using Givens rotations.

The order in which the rows are processed is very critical for the efficiency of the algorithm as reported in [12]. George and Heath suggest the following "suboptimal heuristic" row ordering [12, p. 79]:

Sort the rows into increasing order with respect to the maximum column subscript.

These two rules form the basis of the G2B algorithm, which we shall now describe in some more detail. First we illustrate the algorithm using a small example; then we discuss the general case and make an operation count. Finally we show that the above row ordering is optimal for our problem.

In our description of the algorithm we implicitly assume that all transformations are also applied to the right-hand side; cf. [12].

2.1. A small example. Below we describe the G2B algorithm for computing (2.5) in terms of a small example with $n=9$, $b_1=4$, $p=7$, and $b_2=2$:

$$\begin{bmatrix} R_1 \\ \mu R_2 \end{bmatrix} = \begin{bmatrix}
\times & \times & \times & \times & \times & & & & \\
& \times & \times & \times & \times & \times & & & \\
& & \times & \times & \times & \times & \times & & \\
& & & \times & \times & \times & \times & \times & \\
& & & & \times & \times & \times & \times & \times \\
& & & & & \times & \times & \times & \times \\
& & & & & & \times & \times & \times \\
& & & & & & & \times & \times \\
& & & & & & & & \times \\
\times & \times & \times & & & & & & \\
& \times & \times & \times & & & & & \\
& & \times & \times & \times & & & & \\
& & & \times & \times & \times & & & \\
& & & & \times & \times & \times & & \\
& & & & & \times & \times & \times & \\
& & & & & & \times & \times & \times
\end{bmatrix}. \tag{2.7}$$

We denote the nonzero elements by $\times$. An element annihilated in the present transformation is represented by $\bigcirc$, and new nonzero elements by $+$.

Since we shall process the rows in the order determined by their maximum column subscript (for nonzero elements), we see that we can immediately move the first three rows from R_2 to R_μ without doing any computations:

$$R_\mu := \begin{bmatrix}
\times & \times & \times & & & & \\
& \times & \times & \times & & & \\
& & \times & \times & \times & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & &
\end{bmatrix}.$$

We then have to process the first row in R_1:

$$R_\mu := \begin{bmatrix}
\times & \times & \times & & & & \\
& \times & \times & \times & & & \\
& & \times & \times & \times & & \\
\times & \times & \times & \times & \times & & \\
& & & & & & \\
& & & & & & \\
& & & & & &
\end{bmatrix}.$$

By a sequence of rotations in the $(1, 4)$, $(2, 4)$, and $(3, 4)$ planes we can annihilate the first three elements in the fourth row of R_μ:

$$R_\mu := \left[\begin{array}{ccccccc}
\times & \times & \times & + & + & & \\
 & \times & \times & \times & + & & \\
 & & \times & \times & \times & & \\
\bigcirc & \bigcirc & \bigcirc & \times & \times & & \\
 & & & & & & \\
 & & & & & & \\
 & & & & & &
\end{array}\right].$$

Now we shall process the fourth row of R_2 and the second row of R_1:

$$R_\mu := \left[\begin{array}{ccccccc}
\times & \times & \times & \times & \times & & \\
 & \times & \times & \times & \times & & \\
 & & \times & \times & \times & & \\
 & & & \times & \times & & \\
 & & & \times & \times & \times & \\
 & \times & \times & \times & \times & \times & \\
 & & & & & &
\end{array}\right].$$

By a sequence of rotations in the $(4, 5)$, $(2, 6)$, $(3, 6)$, $(4, 6)$, and $(5, 6)$ planes we can now zero five elements in rows 5 and 6 of R_μ. The result is

$$R_\mu := \left[\begin{array}{ccccccc}
\times & \times & \times & \times & \times & & \\
 & \times & \times & \times & \times & + & \\
 & & \times & \times & \times & + & \\
 & & & \times & \times & + & \\
 & & & \bigcirc & \times & \times & \\
 & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \times & \\
 & & & & & &
\end{array}\right].$$

Note that no more fill-in is created in the first row.

In the next step we shall process the fifth row of R_2 and the third row of R_1.

$$R_\mu := \left[\begin{array}{ccccccc}
\times & \times & \times & \times & \times & & \\
 & \times & \times & \times & \times & \times & \\
 & & \times & \times & \times & \times & \\
 & & & \times & \times & \times & \\
 & & & & \times & \times & \\
 & & & & & \times & \\
 & & & & \times & \times & \times \\
 & & \times & \times & \times & \times & \times
\end{array}\right].$$

After we have zeroed two elements of the seventh row we have the picture

$$\begin{bmatrix} \times & \times & \times & \times & \times & & \\ & \times & \times & \times & \times & \times & \\ & & \times & \times & \times & \times & \\ & & & \times & \times & \times & \\ & & & & \times & \times & + \\ & & & & & \times & + \\ & & & & \bigcirc & \bigcirc & \times \\ & & \times & \times & \times & \times & \times \end{bmatrix}.$$

We now see that all the elements in the eighth row can be annihilated. The result is

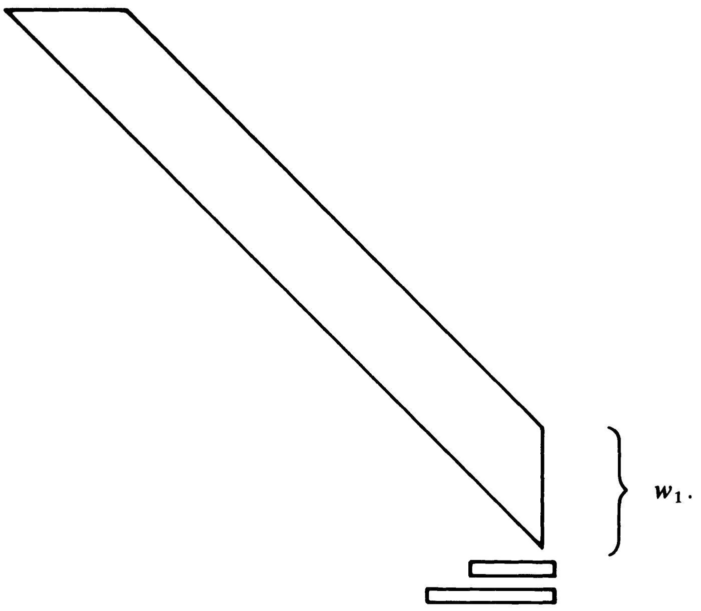
$$\begin{bmatrix} \times & \times & \times & \times & \times & & \\ & \times & \times & \times & \times & \times & \\ & & \times & \times & \times & \times & + \\ & & & \times & \times & \times & + \\ & & & & \times & \times & \times \\ & & & & & \times & \times \\ & & & & & & \times \\ & & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc \end{bmatrix}.$$

It is now obvious how the algorithm proceeds. In every step (apart from the last few) two rows are processed: one makes up a new row in R_μ, and one is completely annihilated.

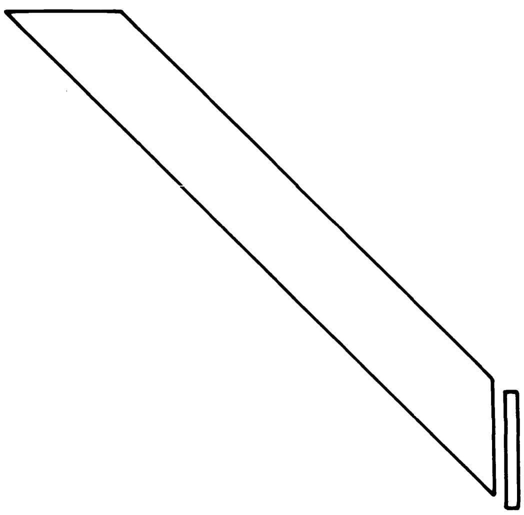
The last few steps are somewhat different, mainly because there, no more fill-in is created, and all rows processed are completely annihilated.

2.2. The general case. Here we shall briefly consider the general case and count the number of rotations and operations in the G2B algorithm. From the preceding discussion it is now apparent that before we have performed the rotations in a step of the algorithm, the problem has the structure

(2.8)



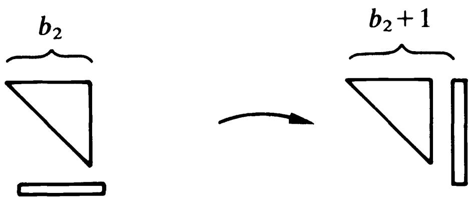
The shorter row at the bottom comes from R_2 and has w_2 nonzero elements. The longer row comes from R_1 and has w_1 nonzero elements. Using plane rotations we annihilate all elements but one in the shorter row, and all elements in the longer row in (2.8). Thus a typical step involves w_1+w_2-1 rotations. At the end of a step we have the picture



The first and last few steps of the algorithm are somewhat different. For the operation count we can ignore these special cases, if we assume that w_1 and w_2 are small compared to n and p.

We shall not make a detailed operation count, but shall show that the number of multiplications is $O(n(w_1^2+w_2^2))$ and ignore lower order terms. This may be inaccurate if w_1 and w_2 are very small. In that case the work for setting up a rotation will be significant, and the operation count itself may be dubious. Also, we shall assume that standard Givens rotations are used, involving two multiplications per element changed; cf. [12, p. 78]. We emphasize that operation counts should not be taken as the only measure of efficiency, but should be supplemented by timing experiments.

When we zero w_2-1 $(=b_2)$ elements in the shorter row in (2.8) we use a $b_2\times b_2$ upper triangular submatrix:



It is easily seen that the number of multiplications is approximately $2w_2^2$. By an analogous argument we find that the number of multiplications needed for annihilating the longer bottom row in (2.8) is $2w_1^2$ approximately.

The whole algorithm consists of n steps, and thus we obtain the following approximate total counts:

$$n(w_1+w_2-1) \quad \text{rotations,}$$

and

$$2n(w_1^2+w_2^2) \quad \text{multiplications.}$$

The G2B algorithm can easily be generalized for solving problems involving three band matrices (cf. [20])

$$\min \left\| \begin{pmatrix} & R_1 \\ \mu_2 & R_2 \\ \mu_3 & R_3 \end{pmatrix} f - \begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix} \right\|_2 ,$$

where R_1, R_2 and R_3 are upper triangular band matrices with band widths w_1, w_2 and w_3 respectively. The number of rotations is then approximately

$$n(w_1+w_2+w_3-1).$$

Lawson and Hanson [16, pp. 212–219] (see also [19]) give an algorithm for the solution of banded least squares problems using Householder transformations. They also discuss the solution of (2.1), but they do not describe the row ordering in detail.

An algorithm given in [9] can be considered as a special case of the G2B algorithm.

2.3. The G2B algorithm creates no unnecessary fill-in. We here show that the G2B algorithm is optimal in the sense that it does not introduce any additional nonzero elements, which must be annihilated in subsequent steps. Further, all other row orderings (except some, which differ only trivially from the G2B ordering) lead to the creation of fill-in. The column ordering is assumed to be fixed (to keep the band width of R_μ equal to w_1).

We only consider the case when the rows are processed one by one as described earlier. This restriction is made for definiteness, since it is possible to formulate algorithms which look different (e.g. using implicit row orderings), but which are equivalent to a row-oriented algorithm as far as fill-in is concerned. It is easily seen that such a row-oriented algorithm is equivalent to a column-oriented algorithm, which can be formulated as follows:

> Write out the matrix with the rows in the order that they are to be processed. Annihilate the elements below the main diagonal, column by column, in the order of increasing column subscripts. In column j, annihilate the elements in the order of increasing row subscripts, using the (j, j) element as pivot element.

The column-oriented version of the G2B algorithms applied to the example in § 2.1 starts out from the matrix

$$\begin{bmatrix}
\times & \times & \times & & & & & & \\
 & \times & \times & \times & & & & & \\
 & & \times & \times & \times & & & & \\
\times & \times & \times & \times & \times & & & & \\
 & & & \times & \times & \times & & & \\
 & \times & \times & \times & \times & \times & & & \\
 & & & & \times & \times & \times & & \\
 & & \times & \times & \times & \times & \times & & \\
 & & & & & \times & \times & \times & \\
 & & & \times & \times & \times & \times & \times & \\
 & & & & & & \times & \times & \times \\
 & & & & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times \\
 & & & & & & \times & \times & \times \\
 & & & & & & & \times & \times \\
 & & & & & & & & \times
\end{bmatrix}.$$

After six columns have been processed we have the picture

$$\begin{bmatrix}
\times & \times & \times & \times & \times & & & & \\
 & \times & \times & \times & \times & \times & & & \\
 & & \times & \times & \times & \times & \times & & \\
 & & & \times & \times & \times & \times & \times & \\
 & & & & \times & \times & \times & \times & \times \\
 & & & & & \times & \times & \times & \times \\
 & & & & & & \times & & \\
 & & & & & & \times & & \\
 & & & & & & \times & \times & \\
 & & & & & & \times & \times & \\
 & & & & & & \times & \times & \times \\
 & & & & & & \times & \times & \times \\
 & & & & & & \times & \times & \times \\
 & & & & & & \times & \times & \times \\
 & & & & & & & \times & \times \\
 & & & & & & & & \times
\end{bmatrix}.$$

We shall now annihilate the elements in the seventh column using the seventh row as pivot row. Due to our row ordering, fill-in will be created in the seventh row only, above the main diagonal. Also, any nontrivial reordering of the rows will lead to the introduction of fill-in below the main diagonal. (An interchange of rows 9 and 10 is an example of a trivial reordering.) If we examine the first steps of the G2B algorithm we see that some rows can be reordered there without producing fill-in.

3. The normal equations and G2B algorithms—efficiency and numerical stability. One obvious alternative to the G2B algorithm described in § 2 is to form the normal equations of (2.1)

$$(K^TK+\mu^2L^TL)f=K^Tg. \tag{3.1}$$

The matrix of coefficients is symmetric and positive definite, and therefore (3.1) can be solved by Cholesky decomposition. It is easily seen that the upper triangular matrix R_μ in (2.5) is the Cholesky factor

$$R_\mu^TR_\mu=K^TK+\mu^2L^TL.$$

This means that the storage requirements are the same for both methods.

The band Cholesky algorithm is described in [8]. The number of operations for solving (3.1) is approximately $nw_1^2/2$, so that this algorithm is always faster than the G2B algorithm. Note that if (3.1) is to be solved for several different values of the regularization parameter, K^TK need only be formed once.

However, the algorithm based on the normal equations has several important drawbacks. Firstly, unless the (3.1) is formed and solved in double precision, numerical information is lost due to rounding errors; see e.g. [13]. Secondly, the condition number of (3.1) is essentially the square of that of (2.1) [3], and therefore the normal equation algorithm is much more sensitive to data and rounding errors. This is clearly seen in the numerical example given in § 4.

Thirdly, if μ is taken very small then it may happen that the addition $K^TK+\mu^2L^TL$ cannot be represented in (single precision) computer arithmetic. In that case, instead of solving (3.1) we effectively solve

$$K^TKf=K^Tg,$$

which means that no regularization is applied. The G2B algorithm solves the problem essentially in the form (2.1) using plane rotations, and provided that the rotations are computed as in [16] or LINPACK [8], similar problems will not occur until much smaller values of μ are taken. Note that when the data vector g has errors of very small magnitude, it is essential to choose μ small; see § 4.

For most problems Givens transformations are stable independently of the row ordering. However, if we are solving a weighted least squares problem with widely disparate weights, instability may occur in certain cases [5]. Consider the following example, where $w \gg 1$:

$$\begin{pmatrix} 1 & w & w \\ 1 & 1 & 2 \\ w & 0 & 0 \end{pmatrix}.$$

If we annihilate the (2, 1) and (3, 1) elements in that order the transformations are unstable. If we interchange rows 1 and 3 and then annihilate the (2, 1) and (3, 1) the transformations are stable.

In the applications we are interested in, the diagonal elements of R_1 are much larger than those in μR_2; cf. § 4. If we examine the G2B algorithm, we can see that in this case the unstable situation will not occur.

4. Numerical tests. Operation counts show that the G2B algorithm is about four times slower than the normal equation algorithm. Since the G2B algorithm uses plane rotations, which are applied to relatively few matrix elements, the time for constructing the rotations may be significant compared to the time for applying the rotations. In view of this one might suspect that actual timings would be even more in favor of the normal equation algorithm. It turned out not to be so.

The G2B algorithm was programmed in FORTRAN and run on a DEC-10 computer. It was compared to the subroutine FO4ACF from the NAG library, which solves positive definite, banded systems. We measured the time to compute the decomposition (2.5) and solve (2.6). The NAG subroutine solved the normal equations (3.1).

The system subroutine, which measures CPU time on the DEC-10 computer, also includes some system overhead in the times reported. Thus timings of different runs of the same program may differ, especially when the problem is small. To reduce the arbitrariness we report mean values of 5–25 runs.

In Table 4.1 we give the timings of the G2B algorithm and the normal equation algorithm.

The tables show clearly that in both algorithms the amount of work depends linearly on the dimension n. For the values of w_1 that we have used, the timings do not show a typical $O(w_1^2)$ behavior, probably for the reason explained at the beginning of this section. A least squares estimate based on the timings of Table 4.1 indicates that the normal equation algorithm is 3.3–3.4 times faster than the G2B algorithm.

TABLE 4.1
CPU-*time in seconds for the two algorithms. L is bidiagonal. The figures are mean values.*

(a) *The* G2B *algorithm.*

w_1 \ n	25	50	100	200
3	0.013	0.029	0.051	0.10
5	0.024	0.048	0.097	0.20
9	0.046	0.11	0.22	0.46
17	0.099	0.27	0.61	1.3

(b) *The normal equation algorithm.*

w_1 \ n	25	50	100	200
3	0.007	0.007	0.022	0.041
5	0.009	0.019	0.036	0.072
9	0.017	0.037	0.075	0.15
17	0.032	0.085	0.19	0.39

We then solved numerically a problem with the 50×50 matrix

$$(K)_{ij}=\begin{cases}0 & \text{if } |i-j|>8,\\ \frac{4}{51}k(0.15, x_i-x_j) & \text{otherwise,}\end{cases}$$

where

$$x_i=-2+\frac{4i}{51},\qquad i=1,2,\cdots,50,$$

(4.1) $$k(\sigma,t)=\frac{1}{2\sqrt{\pi}\,\sigma}\exp\left(-\frac{t^2}{4\sigma^2}\right).$$

The integral equation (1.2) with kernel function (4.1) is a prototype equation in many image restoration contexts [1, p. 63], [2], [7]. The condition number of the matrix K is approximately $8.38\cdot 10^5$. K was transformed to upper triangular form by a series of rotations (which were also applied to the right-hand side). The value of w_1 was equal to 17.

The solution vector was taken to be $f=(f_1,f_2,\cdots,f_n)^T$, where

$$f_i=f(x_i),$$
$$f(x)=0.5k(0.1,x+0.9)+k(0.05,x-0.8).$$

The right-hand side vector g was computed by multiplying f by K. These computations were performed in double precision so that g was very accurate.

In Fig. 4.1 we have plotted the solution f and the right-hand side g. Note that g is a smoothed version of f. To simulate measurement errors we perturbed the right-hand side

$$\tilde{g}=g+\varepsilon,$$

where the components of ε were taken from a normal distribution with mean zero and standard deviation s. We then solved the perturbed problems

(4.2) $$\min_f\{\|Kf-\tilde{g}\|_2^2+\mu^2\|f\|_2^2\},$$

for several values of the regularization parameter μ. The solution of (4.2) is denoted $\tilde{f}_\mu$. The error in the computed solution was measured as

$$\left(\frac{1}{50}\sum_{i=1}^{50}((\tilde{f}_\mu)_i-f_i)^2\right)^{1/2},$$

which can be considered as an approximation of an L^2-norm of the error.

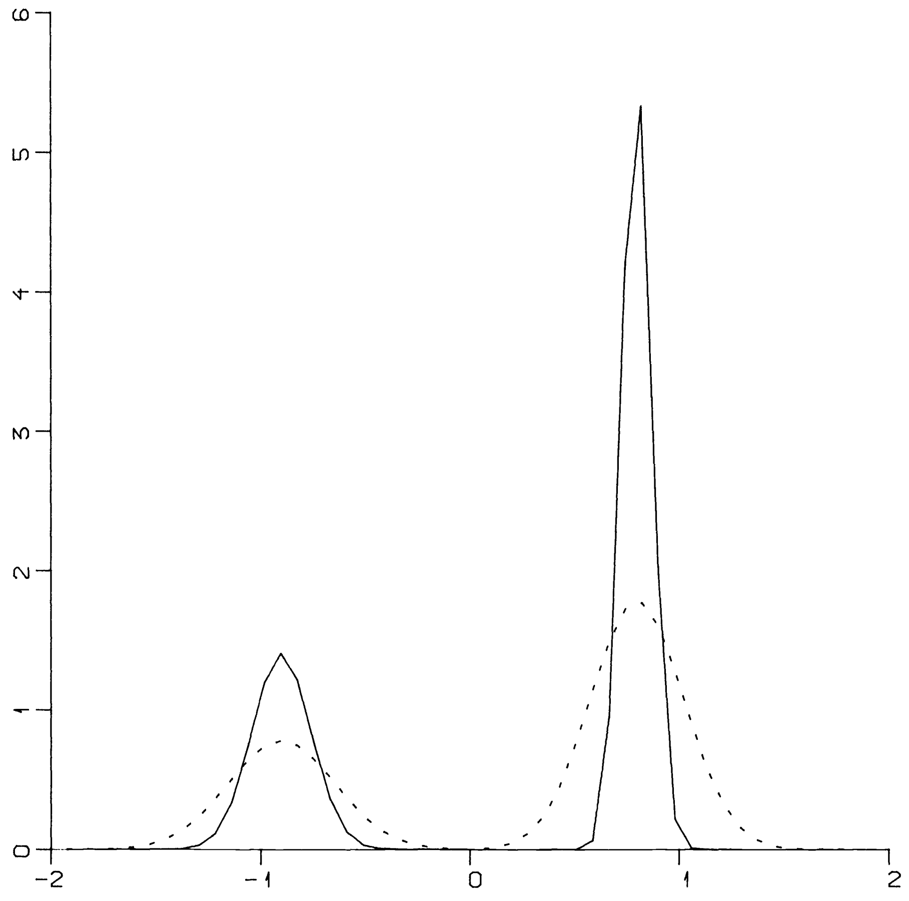
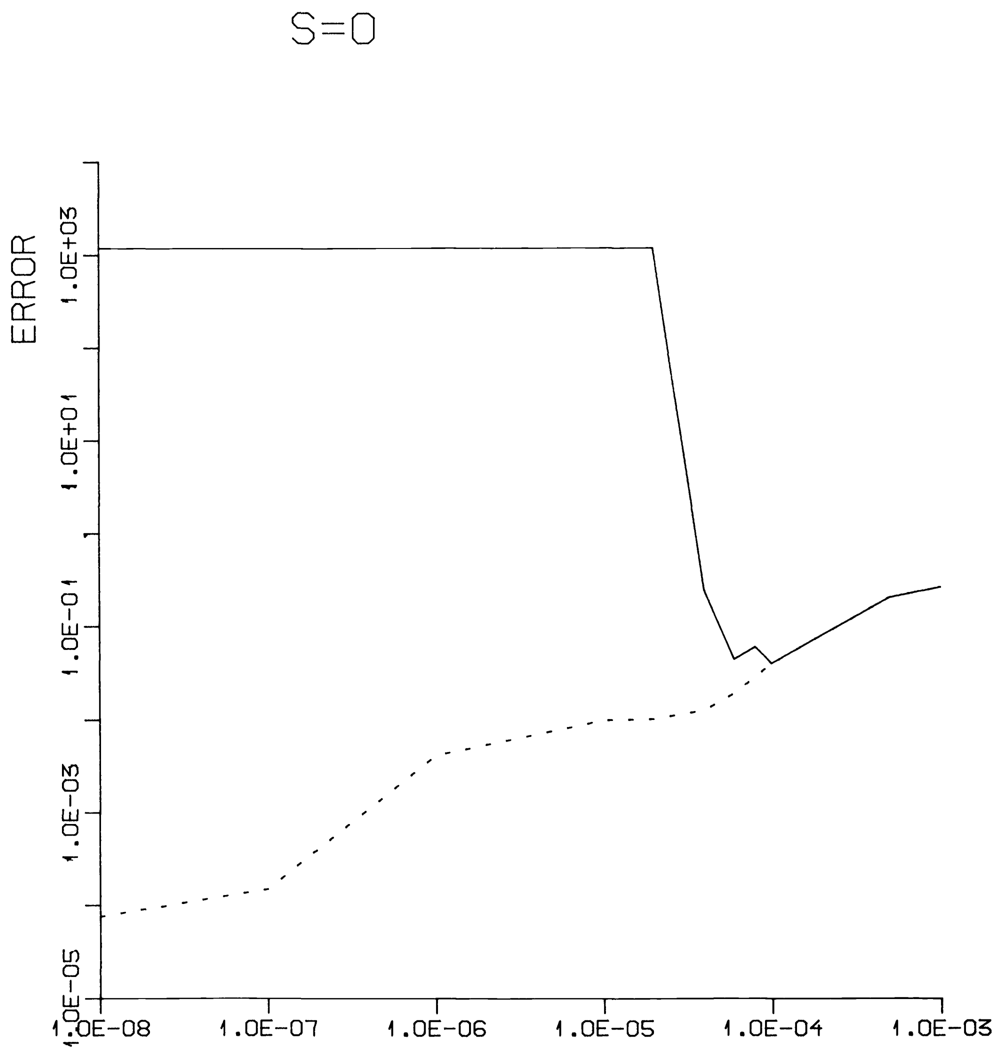


FIG. 4.1. *The solution f (solid line) and the right-hand side (dotted).*

The problem (4.2) was solved using the G2B and the normal equations algorithms. The errors obtained with different values of the regularization parameter μ are plotted in Fig. 4.2. The G2B algorithm produced better solutions than the normal equations algorithm in all test cases, and the difference between the two algorithms was larger for small perturbations of the data.

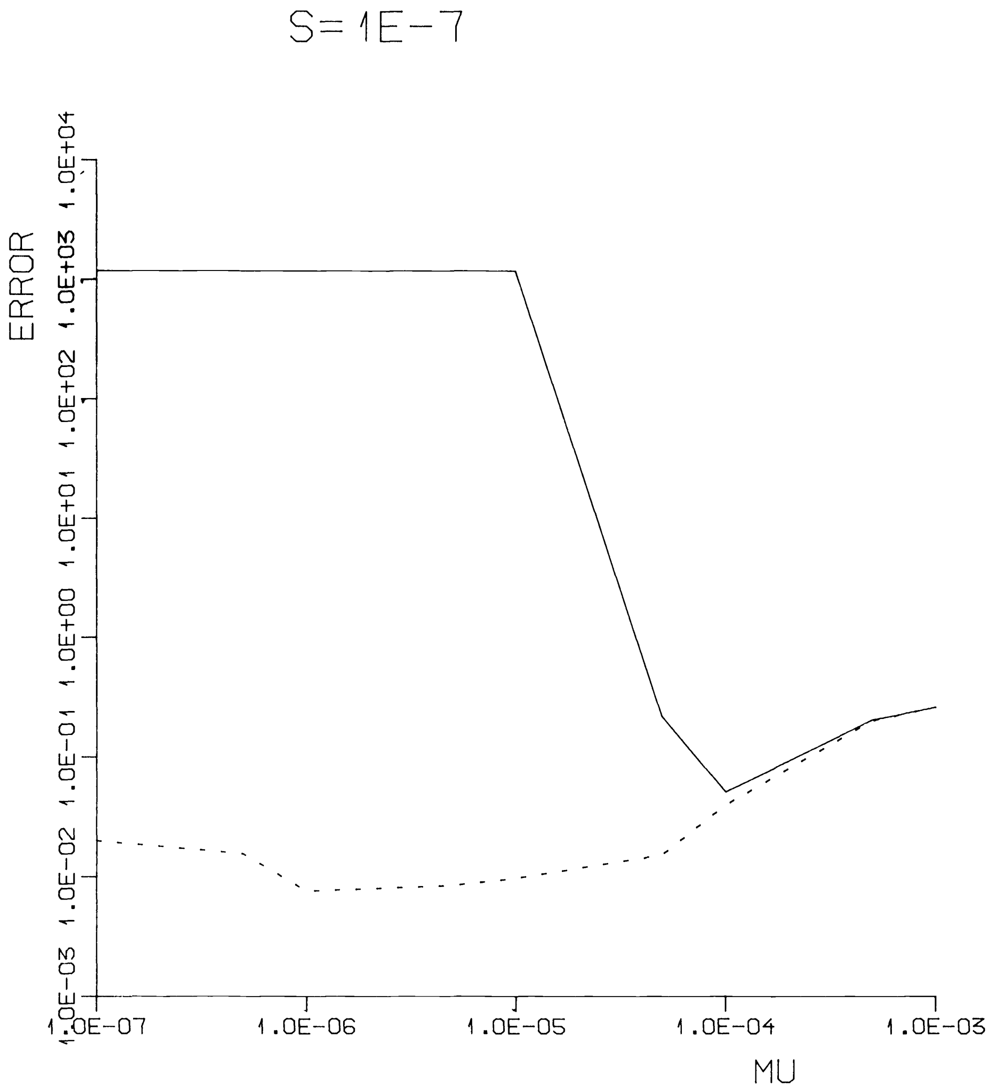
The flat parts of the error curves for the normal equations algorithm show where $K^TK+\mu^2L^TL$ is represented as K^TK in the computer arithmetic (on the DEC-10 computer the smallest value of ε for which $1+\varepsilon>1$ is approximately $1.5 \cdot 10^{-8}$). We also tested the algorithms on an ill-conditioned problem with large residual. Not unexpectedly, it was here necessary to take a large value of the regularization parameter in order to stabilize the solution, and the two algorithms produced results of approximately the same accuracy.

Our numerical tests indicate that the G2B algorithm has better stability properties than the normal equations algorithm; when applied to ill-conditioned problems with relatively accurate data, the G2B algorithm is superior.



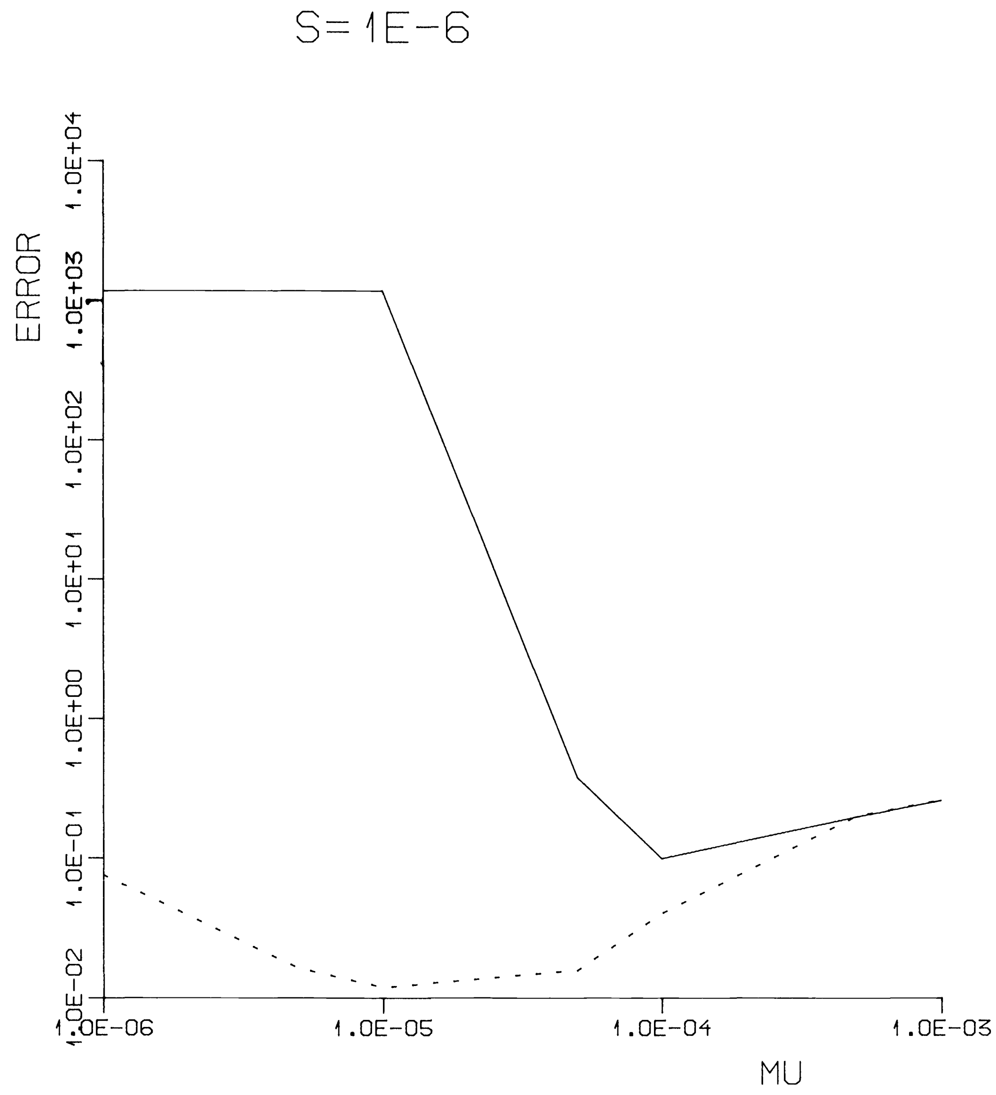
(a)

FIG. 4.2. *Error obtained for different values of the regularization parameter μ using the normal equations algorithm (solid line) and the* G2B *algorithm (dotted). The value of the standard deviation s is given in the plot.*


S=1E-7
ERROR
1.0E-03
1.0E-02
1.0E-01
1.0E+00
1.0E+01
1.0E+02
1.0E+03
1.0E+04
1.0E-07
1.0E-06
1.0E-05
1.0E-04
1.0E-03
MU

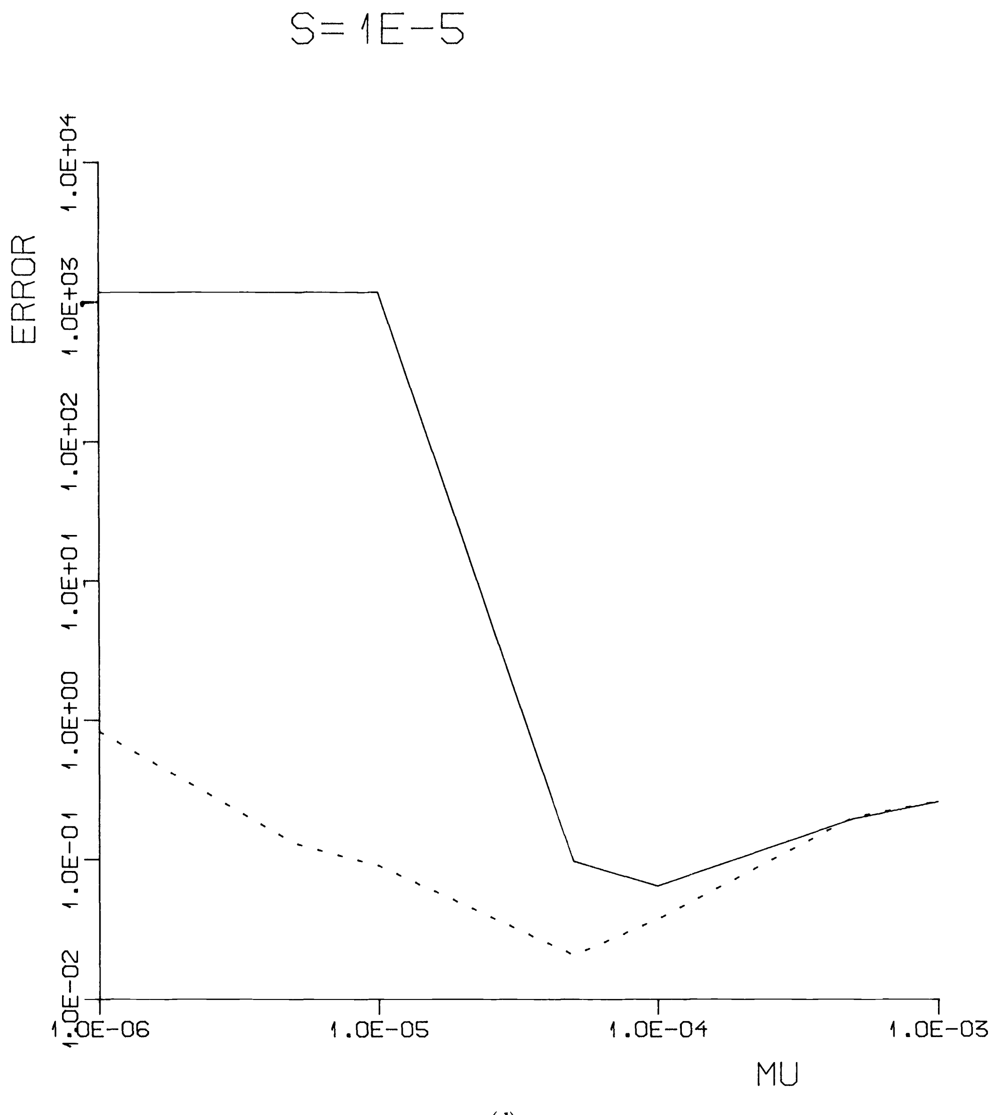
(b)

FIG. 4.2 (*cont.*)


S=1E-6
ERROR
1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E+00
1.0E-01
1.0E-02
1.0E-06
1.0E-05
1.0E-04
1.0E-03
MU

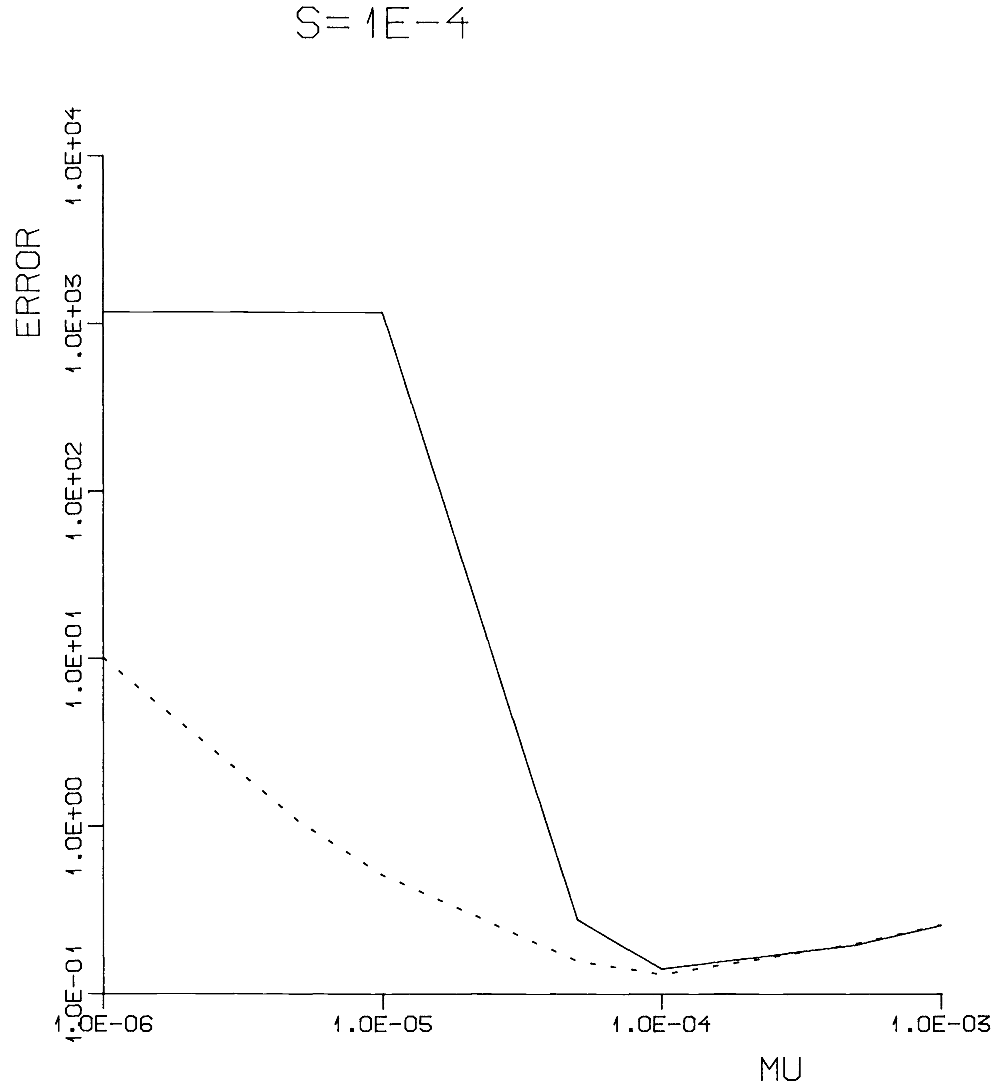
(c)

FIG. 4.2 (*cont.*)


S= 1E-5
ERROR
1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E+00
1.0E-01
1.0E-02
1.0E-06
1.0E-05
1.0E-04
1.0E-03
MU

(d)

FIG. 4.2 (*cont.*)


S= 1E-4
ERROR
1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E+00
1.0E-01
1.0E-06
1.0E-05
1.0E-04
1.0E-03
MU

(e)

FIG. 4.2 (*cont.*)

5. Acknowledgments. I wish to thank Professors Å. Björck and G. H. Golub for valuable advice. The algorithm of this paper is related to an algorithm due to them (private communication).

Note added in proof. The algorithm of this paper can be implemented in a systolic array. This will be discussed in a forthcoming report.

REFERENCES

[1] H. C. ANDREWS AND B. R. HUNT, *Digital Image Restoration*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[2] E. S. ANGEL AND A. K. JAIN, *Restoration of images degraded by spatially varying pointspread functions by a conjugate gradient method*, Appl. Optics, 17 (1978), pp. 2186–2190.

[3] Å. BJÖRCK, *Solving linear least squares problems by Gram–Schmidt orthogonalization*, BIT, 7 (1967), pp. 1–21.

[4] Å. BJÖRCK AND L. ELDÉN, *Methods in numerical algebra for ill-posed problems*, LiTH-MAT-R-33-1979, Dept. Math., Linköping Univ., Linköping, Sweden, 1979.

[5] Å. BJÖRCK, *Stability of Givens transformations for weighted least squares problems*, in preparation.

[6] 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.

[7] F. R. DE HOOG, *Review of Fredholm integral 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, Alphen aan den Rijn, Netherlands, 1980.

[8] J. J. DONGARRA, C. B. MOLER, J. R. BUNCH AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[9] L. ELDÉN, *Algorithms for the regularization of ill-conditioned least squares problems*, BIT, 17 (1977), pp. 134–145.

[10] L. ELDÉN AND I. SKOGLUND, *Algorithms for the regularization of ill-conditioned least squares problems with tensor product structure, and application to space-variant image restoration*, Report LiTH-MAT-R-1982-48, Department of Mathematics, Linköping Univ., Linköping, Sweden, 1982.

[11] W. G. GANDER, *On the linear least squares problem with a quadratic constraint*, Report STAN-CS-78-697, Computer Science Department, Stanford Univ., Stanford, CA, 1978.

[12] A. GEORGE AND M. T. HEATH, *Solution of sparse linear least squares problems using Givens rotations*, in Large Scale Matrix Problems, Å. Björck, R. J. Plemmons and H. Schneider, eds., North-Holland, New York, 1981, pp. 69–83.

[13] G. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206–216.

[14] G. H. GOLUB, *Some modified eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318–334.

[15] R. J. HANSON AND J. L. PHILLIPS, *An adaptive numerical method for solving linear Fredholm integral equations of the first kind*, Numer. Math., 24 (1975), pp. 291–307.

[16] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[17] L. M. DELVES AND J. WALSH, eds., *Numerical Solution of Integral Equations*, Clarendon Press, Oxford, 1974.

[18] D. L. PHILLIPS, *A technique for the numerical solution of certain integral equations of the first kind*, J. Assoc. Comput. Mach., 9 (1962), pp. 84–96.

[19] J. K. REID, *A note on the least squares solution of a band system of linear equations by Householder reductions*, Comput. J., 10 (1967–1968), pp. 188–189.

[20] A. N. TIKHONOV, *Solution of incorrectly formulated problems and the regularization method*, Dokl. Akad. Nauk SSSR, 151 (1963), pp. 501–504, = Soviet Math. Dokl., 4 (1963), pp. 1035–1038.

[21] J. M. VARAH, *A practical examination of some numerical methods for linear discrete ill-posed problems*, SIAM Rev., 21 (1979), pp. 100–111.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

AN ITERATIVE ALGORITHM FOR SOLVING INVERSE PROBLEMS OF TWO-DIMENSIONAL DIFFUSION EQUATIONS*

J. Q. LIU† AND Y. M. CHEN‡

Abstract. The applicability of the iterative numerical algorithm of the pulse-spectrum technique (PST) to solve inverse problems of two-dimensional linear diffusion equations is demonstrated. Numerical simulations are carried out to test the feasibility and to study the general characteristics of this technique without the real measurement data. It is found that PST also gives excellent results and is as robust as for solving the inverse problem of the one-dimensional linear diffusion equation.

Key words. iterative algorithm, inverse problem, two-dimensional diffusion equation

Introduction. The problem of determining the diffusion coefficient of the diffusion equation from known data or the solution and its first order derivatives either on the boundary or in the interior of a bounded space domain can be formulated as an ill-posed inverse problem for the diffusion equation. Usually the solution of an inverse problem is not unique and does not depend continuously on the given data. Applications can be found in remote sensing of the thermal conductivity of a nonhomogeneous solid, inferring the transmissivity or permeability from known data of the pressure and pressure gradient in oil reservoir and aquifer simulation, etc.

Many numerical and analytic methods for constructing approximate solutions of this type of inverse problem have been developed by researchers in the past and present. In particular, the questions of existence and uniqueness of the solution, and techniques for constructing approximate solutions of the inverse problem of the one-dimensional diffusion equation, have been treated by Jones [1], Douglas and Jones [2], Cannon [3], [4], and Cannon and DuChateau [5]–[8]. Recently, a much superior numerical algorithm the "pulse-spectrum technique" (PST), has been introduced and developed by Chen and Liu [9] to determine the approximate diffusion coefficient of the one-dimensional diffusion equation. For two-dimensional cases, history matching techniques based on the minimization of functionals of the differences between the observations and the computed solutions of the diffusion equation have been developed by Chen, Gavalas, Seinfeld and Wasserman [10], Chavent, DuPay and Lemonnier [11], Chang and Yeh [12]. On the other hand, various other techniques have been used to solve this type of inverse problems under steady state conditions by Frind and Pinder [13], Nutbrown [14], and Richter [15].

The basic idea of the pulse-spectrum technique (PST) is that data are measured in the time domain as any arbitrary functions which are Laplace transformable, and the synthesis of the unknown diffusion coefficient is carried out numerically in the complex frequency domain by a special iterative algorithm. The PST was first introduced by Tsien and Chen [16] for solving an idealized one-dimensional velocity inversion problem in fluid dynamics; then it was further developed by Chen and Tsien [17] to have the capability of handling noisy, poorly distributed and inadequately measured data. Later it was used to solve a one-dimensional inverse problem in electromagnetic wave propagation by Tsien and Chen [18]. Next, it was extended successfully by Hatcher and Chen [19] to solve inverse problems of a one-dimensional

* Received by the editors August 16, 1982, and in revised form January 24, 1983.

† Department of Applied Mathematics, Harbin Institute of Technology, Harbin, People's Republic of China.

‡ Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, New York 11794.

nonlinear acoustic wave equation. In a different direction, the PST has also been modified with great success [9] to solve an inverse problem of a one-dimensional diffusion equation. Moreover, the discretized version of this iterative algorithm under idealized conditions has been proved to converge quadratically [20], which is quite efficient from the numerical computation point of view.

In this paper, PST is presented and extended for solving this type of inverse problem for the two-dimensional diffusion equation. It is found that PST fares very well in regard to the following four practical criteria for the evaluation of any numerical method:

(a) *Universality criterion.* Can a numerical method which is effective in problems in one space dimension be extended with similar success into higher space-dimensional applications? Can a solution method which is effective for solving inverse problems for one type of equation, e.g., hyperbolic or parabolic type, be extended to solve inverse problems for the other type of partial differential equation with similar success and minimum effort?

(b) *Economy of data acquisition criterion.* The numerical method should be able to keep to a minimum the difficulties and the cost expenditure of acquiring or measuring the necessary data for a successful calculation.

(c) *Economy of programming effort criterion.* The numerical method should be as close to a nondedicated program as possible, for existing methods of programming new dedicated numerical methods for every special type of problem can be unacceptably costly in many practical circumstances. Furthermore, the computer code should as far as possible use modules based on canned subroutines.

(d) *Economy of computing cost criterion.* The numerical method should keep the cost of I/O and CPU time and memory storage to a minimum.

For simplicity in the next section the formulation of the inverse problem of a linear two-dimensional diffusion equation is presented and the basic numerical algorithm of PST is given. Then numerical simulations are carried out to test the feasibility and to study the intrinsic characteristics of this numerical algorithm with artificially generated data. Finally, a comprehensive discussion is given of the numerical results, their implication in actual implementing this computational algorithm, and the merits of PST.

Numerical algorithm (PST). Consider the following initial-boundary value problem for a two-dimensional linear diffusion equation:

$$(1)\qquad \begin{gathered} \frac{\partial(k(x,y)\partial u/\partial x)}{\partial x}+\frac{\partial(k(x,y)\partial u/\partial y)}{\partial y}-\rho(x,y)\frac{\partial u}{\partial t}=0, \qquad (x,y)\in\Omega, \quad 0<t<\infty, \\ u(x,y,0)|_{\Omega}=0, \qquad u(x,y,t)=f(x,y,t) \quad \text{for } (x,y)\in\Gamma, \end{gathered}$$

where Ω is a bounded region in x–y space and Γ is the boundary of Ω. For solving the inverse problem, in addition to the initial and boundary conditions in (1) one needs at least one more condition either on $\partial u/\partial n$ at a subinterval of Γ or on u in a subregion of Ω. Here for convenience we adopt

$$(2)\qquad \frac{\partial u}{\partial n}=h(x,y,t) \quad \text{for } (x,y)\in\tilde{\Gamma}$$

where $\tilde{\Gamma}$ is a section of Γ. Here the inverse problem is to determine the unknown diffusion coefficient $k(x,y)$ from known $\rho(x,y)$, $f(x,y,t)$ and $h(x,y,t)$ where $f(x,y,t)$ and $h(x,y,t)$ are Laplace transformable.

PST calls for the Laplace transformation of (1) and (2) so that the entire system is transformed from the time domain to the complex frequency domain, the corresponding system is

$$\text{(3)}\quad \begin{aligned} &\frac{\partial(k(x,y)\partial v/\partial x)}{\partial x}+\frac{\partial(k(x,y)\partial v/\partial y)}{\partial y}-\rho(x,y)s\,v=0, \qquad (x,y)\in\Omega,\\ &v(x,y,s)=F(x,y,s) \quad \text{for } (x,y)\in\Gamma, \end{aligned}$$

$$\text{(4)}\quad \frac{\partial v}{\partial n}=H(x,y,s) \quad \text{for } (x,y)\in\tilde{\Gamma},$$

where $v(x,y,s)$, $F(x,y,s)$ and $H(x,y,s)$ are the Laplace transforms of $u(x,y,t)$, $f(x,y,t)$ and $h(x,y,t)$ respectively. Now the inverse problem is to determine $k(x,y)$ from $\rho(x,y)$, $F(x,y,s)$ and $H(x,y,s)$.

The iterative numerical algorithm begins by setting

$$\text{(5)}\quad v_{n+1}=v_n+\delta v_n, \qquad k_{n+1}=k_n+\delta k_n, \qquad n=0,1,2,3,\cdots,$$

where $k_0(x,y)$ is the initial guess for the unknown coefficient $k(x,y)$, $\|k_n\|>\|\delta k_n\|$ and $\|v_n\|>\|\delta v_n\|$, and $k_0|_\Gamma=k|_\Gamma$. Upon substituting (5) into (3), neglecting terms of order δ^2 and higher, one obtains a system for v_n,

$$\text{(6)}\quad \begin{aligned} &\frac{\partial(k_n\partial v_n/\partial x)}{\partial x}+\frac{\partial(k_n\partial v_n/\partial y)}{\partial y}-s\rho v_n=0, \qquad (x,y)\in\Omega,\\ &v_n|_\Gamma=F(x,y,s), \end{aligned}$$

and a system for δv_n,

$$\text{(7)}\quad \begin{aligned} &\frac{\partial(k_n\partial\delta v_n/\partial x)}{\partial x}+\frac{\partial(k_n\partial\delta v_n/\partial y)}{\partial y}-s\rho\delta v_n=-\frac{\partial(\delta k_n\partial v_n/\partial x)}{\partial x}-\frac{\partial(\delta k_n\partial v_n/\partial y)}{\partial y},\\ &\delta v_n|_\Gamma=0. \end{aligned}$$

By using a Green's function, the partial differential equation (7) can be changed to a Fredholm integral equation of the first kind which relates $\delta k_n(x,y)$ to $\delta v_n(x,y,s)$:

$$\text{(8)}\quad \iint_\Omega G_n(x,y,x',y',s)\left\{\frac{\partial(\delta k_n\partial v_n/\partial x')}{\partial x'}+\frac{\partial(\delta k_n\partial v_n/\partial y')}{\partial y'}\right\}dx'\,dy'=-\delta v_n(x,y,s),$$

where $G_n(x,y,x',y',s)$ is the Green's function of the differential operator in (7). For the particular auxiliary condition in (4), a more useful expression of (8) can be derived by taking the normal derivative of (8) and setting (x,y) at $\tilde{\Gamma}$ to obtain

$$\text{(9)}\quad \iint_\Omega \left.\frac{\partial G_n}{\partial n}\right|_{\tilde{\Gamma}}\left\{\frac{\partial(\delta k_n\partial v_n/\partial x')}{\partial x'}+\frac{\partial(\delta k_n\partial v_n/\partial y')}{\partial y'}\right\}dx'\,dy'=-\left.\frac{\partial\delta v_n}{\partial n}\right|_{\tilde{\Gamma}}.$$

For the purpose of accelerating the rate of convergence, $\partial v_{n+1}/\partial n|_{\tilde{\Gamma}}$ on the right side of (9) can be replaced by $\partial v/\partial n|_{\tilde{\Gamma}}$. With the help of (4), (9) can be written as

$$\text{(10)}\quad \iint_\Omega \left.\frac{\partial G_n}{\partial n}\right|_{\tilde{\Gamma}}\left\{\frac{\partial(\delta k_n\partial v_n/\partial x')}{\partial x'}+\frac{\partial(\delta k_n\partial v_n/\partial y')}{\partial y'}\right\}dx'\,dy'=-H|_{\tilde{\Gamma}}+\left.\frac{\partial v_n}{\partial n}\right|_{\tilde{\Gamma}}$$

which is a Fredholm integral equation of the first kind for $\delta k_n(x,y)$.

Equations (5), (6) and (10) form the basic structure for each iteration in the iterative numerical algorithm of PST. First, a numerical integration subroutine is used to evaluate the Laplace transforms $F(x, y, s)$ and $H(x, y, s)$ at $s = s_\sigma, \sigma = 1, 2, 3, \cdots, \Sigma$. Then these discrete values will be used to solve (6) and (10) numerically.

The boundary value problem (6) and the Green's function of (7) can be solved numerically by simply using the following first order finite difference method. Assume that Ω can be approximated by a collection of small quadrilaterals or triangles and each computational grid point is denoted by a pair of numbers (x, y). The finite difference approximation at an interior point (x_i, y_j) is derived by considering the area integration of an element area centered at (i, j)(Fig. 1) as

$$\iint_{\Omega_{ij}} \left\{ \frac{\partial(k_n \partial v_n/\partial x)}{\partial x} + \frac{\partial(k_n \partial v_n/\partial y)}{\partial y} - s_\sigma v_n \right\} dx\, dy = 0. \tag{11}$$

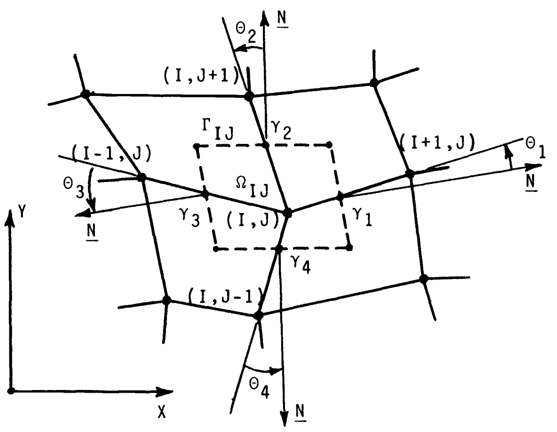


FIG 1. *Computational grid of the finite difference method at an interior point* (i, j).

By Green's formula, (11) becomes

$$\sum_{l=1}^{4} \int_{\Gamma_{ijl}} k_n \frac{\partial v_n}{\partial n} d\gamma - \iint_{\Omega_{ij}} s_\sigma v_n \, dx\, dy = 0. \tag{12}$$

The line integral of (12) can be approximated by

$$\begin{aligned}
&\int_{\Gamma_{ij1}} k_n \frac{\partial v_n}{\partial n} d\gamma \sim \frac{k_{ni,j} + k_{ni+1,j}}{2\cos\theta_1} \left\{ \frac{v_{ni+1,j} - v_{ni,j}}{\gamma_{(i,j)(i+1,j)}} - \frac{v_{ni,j+1} - v_{ni,j-1}}{2\gamma_1} \sin\theta_1 \right\} \cdot \gamma_1, \\
&\int_{\Gamma_{ij2}} k_n \frac{\partial v_n}{\partial n} d\gamma \sim \frac{k_{ni,j} + k_{ni,j+1}}{2\cos\theta_2} \left\{ \frac{v_{ni,j+1} - v_{ni,j}}{\gamma_{(i,j+1)(i,j)}} - \frac{v_{ni-1,j} - v_{ni+1,j}}{2\gamma_2} \sin\theta_2 \right\} \cdot \gamma_2, \\
&\int_{\Gamma_{ij3}} k_n \frac{\partial v_n}{\partial n} d\gamma \sim \frac{k_{ni,j} + k_{ni-1,j}}{2\cos\theta_3} \left\{ \frac{v_{ni-1,j} - v_{ni,j}}{\gamma_{(i-1,j)(i,j)}} - \frac{v_{ni,j-1} - v_{ni,j+1}}{2\gamma_3} \sin\theta_3 \right\} \cdot \gamma_3, \\
&\int_{\Gamma_{ij4}} k_n \frac{\partial v_n}{\partial n} d\gamma \sim \frac{k_{ni,j} + k_{ni,j-1}}{2\cos\theta_4} \left\{ \frac{v_{ni,j-1} - v_{ni,j}}{\gamma_{(i,j-1)(i,j)}} - \frac{v_{ni+1,j} - v_{ni-1,j}}{2\gamma_4} \sin\theta_4 \right\} \cdot \gamma_4,
\end{aligned} \tag{13}$$

and the area integral of (12) can be approximated by

$$\iint_{\Omega_{ij}} s_\sigma v_n \, dx\, dy \sim s_\sigma v_{nij} \Delta_{ij}, \tag{14}$$

where the θ's are the angles between the normal vectors and the local axes of the computational grid measured counterclockwise (Fig. 1), the coordinates of the corner point of Ω_{ij} are the arithmetic mean of those of its surrounding grid points, the γ_l's are the lengths of Γ_{ijl}'s, $\gamma_{(i,j)(p,q)}$ is the distance between (i,j) and (p,q) and Δ_{ij} is the area of the small quadrilateral Ω_{ij}. It is assumed that the θ_j's are not too large, say, $|\theta_j|<\pi/4$. With the help of (12), (13) and (14), the boundary value problem (6) is discretized and becomes a linear algebraic system,

$$\underset{=}{A}_n(k_n, s_\sigma)\cdot \underset{-}{V}_n(s_\sigma)=\underset{-}{B}(s_\sigma), \tag{15}$$

where $\underset{=}{A}_n(k_n, s_\sigma)$ is a known pentadiagonal matrix, $\underset{-}{V}_n(s_\sigma)$ is the vector with all $v_{ni,j}(s_\sigma)$'s as its components and the known vector $\underset{-}{B}(s_\sigma)$ corresponds to the boundary condition.

The Green's function of (7) satisfies (11) also except that the right-hand side contains an additional term

$$\iint_{\Omega_{ij}} \delta\{\gamma_{(i,j)(p,q)}\}\,dx\,dy=\begin{cases}1 & \text{for } (i,j)=(p,q),\\ 0 & \text{for } (i,j)\neq(p,q).\end{cases}$$

Hence the discretized Green's function satisfies the similar linear algebraic system,

$$\underset{=}{A}_n(k_n, s_\sigma)\cdot \underset{-}{G}_n(x_p, y_q, s_\sigma)=\underset{-}{C}(x_p, y_q), \tag{16}$$

where the components of the unknown vector $\underset{-}{G}_n(x_p, y_q, s_\sigma)$ are all $G_n(x_p, y_q, x_i', y_j', s_\sigma)$ for fixed x_p, y_q and s_σ, and the known vector $\underset{-}{C}(x_p, y_q)$ corresponds to the boundary condition and the nonhomogeneous term.

At the boundary, the normal derivatives in the integral equation (10) are approximated by a more accurate finite difference scheme. Assuming that $p=1$ corresponds to the boundary (Fig. 2), the approximations are

$$\left.\frac{\partial v_n}{\partial n}\right|_{\Gamma}\sim\frac{4v_{n2,q}-3F(x_1, y_q, s_\sigma)-v_{n3,q}}{\gamma_{(1,q)(2,q)}+\gamma_{(2,q)(3,q)}}\cos\theta+\frac{F(x_1, y_{q+1}, s_\sigma)-F(x_1, y_{q-1}, s_\sigma)}{\gamma_{(1,q)(1,q+1)}+\gamma_{(1,q-1)(1,q)}}\sin\theta \tag{17}$$

and

$$\left.\frac{\partial G_n}{\partial n}\right|_{\Gamma}\sim\frac{4G_n(x_2, y_q, x_i', y_j', s_\sigma)-G_n(x_3, y_q, x_i', y_j', s_\sigma)}{\gamma_{(1,q)(2,q)}+\gamma_{(2,q)(3,q)}}\cos\theta. \tag{18}$$

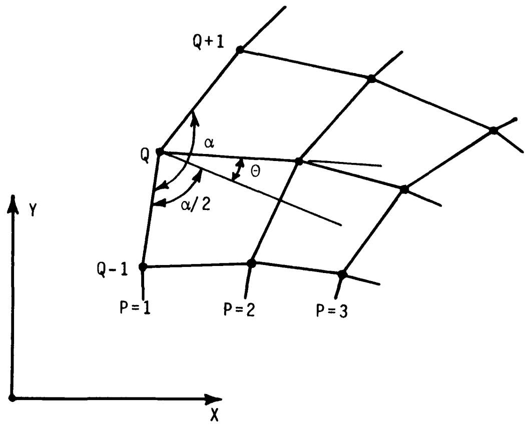


FIG. 2. *Computational grid for the finite difference approximation of the normal derivative at a boundary point* $(1,q)$.

In view of (18), one only needs to compute $G_n(x_2, y_q, x'_i, y'_j, s_\sigma)$ and $G_n(x_3, y_q, x'_i, y'_j, s_\sigma)$ for solving the inverse problem. Hence for computational efficiency, one solves the combination of (15) and (16),

(19) $\underline{\underline{A}}_n(k_n, s_\sigma) \cdot \{\underline{V}_n(s_\sigma), \underline{G}_n(x_2, y_q, s_\sigma), G_n(x_3, y_q, s_\sigma)\} = \{\underline{B}(s_\sigma), \underline{C}(x_2, y_q), \underline{C}(x_3, y_q)\}$.

The Fredholm integral equation of the first kind (10) can be discretized by simply using the rectangle rule and the derivatives in the integrand are approximated by the following finite difference approximation,

(20)

$$\frac{\partial \delta k_n}{\partial x'} \sim \frac{1}{\sin(\phi_{ij} - \psi_{ij})}\left\{\frac{\delta k_{ni+1,j} - \delta k_{ni-1,j}}{\gamma_{(i+1,j)(i,j)} + \gamma_{(i,j)(i-1,j)}} \sin \phi_{ij} - \frac{\delta k_{ni,j+1} - \delta k_{ni,j-1}}{\gamma_{(i,j+1)(i,j)} + \gamma_{(i,j)(i,j-1)}} \sin \psi_{ij}\right\},$$

$$\frac{\partial \delta k_n}{\partial y'} \sim \frac{1}{\sin(\phi_{ij} - \psi_{ij})}\left\{\frac{\delta k_{ni+1,j} - \delta k_{ni-1,j}}{\gamma_{(i+1,j)(i,j)} + \gamma_{(i,j)(i-1,j)}} (-\cos \phi_{ij}) + \frac{\delta k_{ni,j+1} - \delta k_{ni,j-1}}{\gamma_{(i,j+1)(i,j)} + \gamma_{(i,j)(i,j-1)}} \cos \psi_{ij}\right\},$$

where ϕ_{ij} and ψ_{ij} are the average angles of the local computational grid coordinates with respect to the x-axis (Fig. 3).

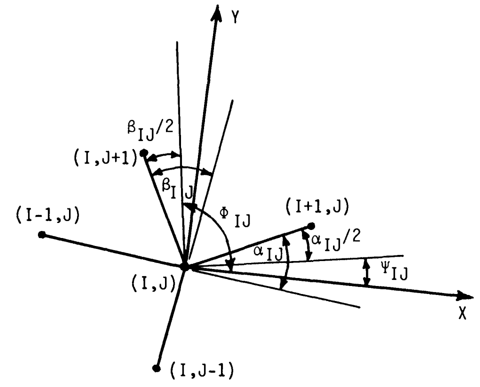


FIG. 3. *Definitions of ϕ_{ij} and ψ_{ij} in the finite difference approximation of derivatives in the integrand of the integral equation at (i, j).*

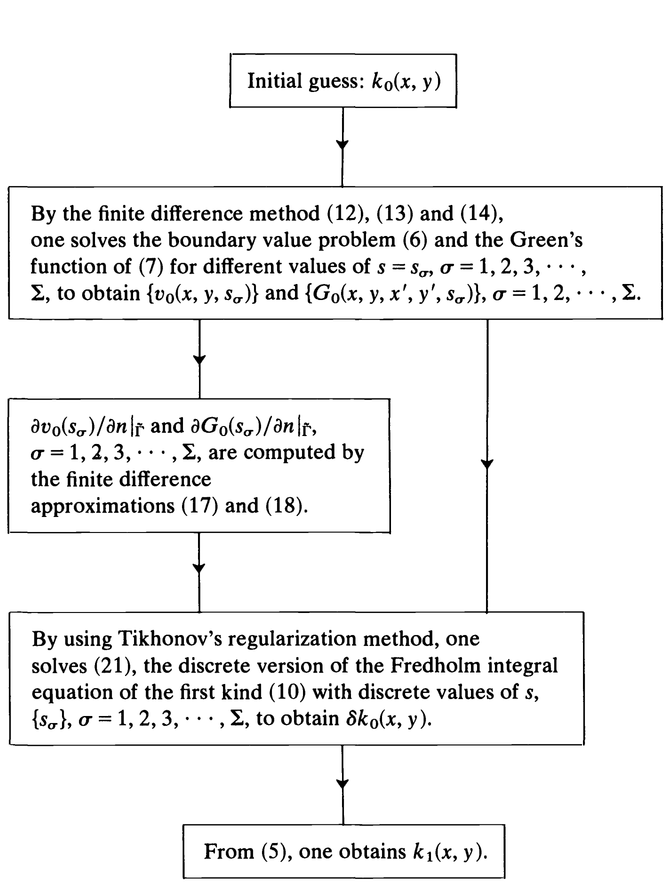
In a similar way the partial derivatives $\partial v_n/\partial x'$, $\partial v_n/\partial y'$, $\partial^2 v_n/\partial x'^2$ and $\partial^2 v_n/\partial y'^2$ are approximated by the same center difference scheme. Hence the integral equation (10) is reduced into a linear algebraic system,

(21) $$\underline{\underline{M}}_n(v_n, G_n, s_\sigma) \cdot \delta \underline{K}_n(x, y) = \underline{S}_n(s_\sigma),$$

where the known matrix $\underline{\underline{M}}_n(v_n, G_n, s_\sigma)$ comes from the discretization of the integral. It is an ill-conditioned full matrix with each row containing a different s_σ. The unknown vector $\delta \underline{K}_n(x, y)$ consists of all $\delta k_{ni,j}$ as its components, and the known vector $\underline{S}_n(s_\sigma)$ comes from the discretization of the right-hand side of (10) with components containing the corresponding complex frequency parameter s_σ, $\sigma = 1, 2, 3, \cdots, \Sigma$.

Since $\underline{\underline{A}}_n$ is a symmetric, positive-definite, narrow banded, and well-conditioned matrix, (19) can be solved by any modern efficient sparse matrix technique. However, $\underline{\underline{M}}_n$ is either a rectangular matrix or an ill-conditioned square matrix; therefore Tikhonov's regularization method with second-order stabilizers [21] is used to solve

(21). The essence of the first cycle of iteration is given in the accompanying diagram and the procedure for other cycles is exactly the same.



Initial guess: $k_0(x, y)$

By the finite difference method (12), (13) and (14), one solves the boundary value problem (6) and the Green's function of (7) for different values of $s = s_\sigma$, $\sigma = 1, 2, 3, \cdots, \Sigma$, to obtain $\{v_0(x, y, s_\sigma)\}$ and $\{G_0(x, y, x', y', s_\sigma)\}$, $\sigma = 1, 2, \cdots, \Sigma$.

$\partial v_0(s_\sigma)/\partial n|_\Gamma$ and $\partial G_0(s_\sigma)/\partial n|_\Gamma$, $\sigma = 1, 2, 3, \cdots, \Sigma$, are computed by the finite difference approximations (17) and (18).

By using Tikhonov's regularization method, one solves (21), the discrete version of the Fredholm integral equation of the first kind (10) with discrete values of s, $\{s_\sigma\}$, $\sigma = 1, 2, 3, \cdots, \Sigma$, to obtain $\delta k_0(x, y)$.

From (5), one obtains $k_1(x, y)$.

It is important to notice that each cycle of iteration consists basically of first solving the direct boundary value problem (6) and the Green's function of (7) Σ times and then solving the Fredholm integral equation of the first kind (10) once.

Numerical simulation. In order to test the feasibility and to study the general characteristics of the PST computational algorithm for solving two-dimensional inverse problems for the linear diffusion equation without real measurement data, the following numerical simulation procedure is carried out. A direct problem is solved for a given coefficient $k^*(x, y)$, and the PST algorithm is used to see whether k^* is recovered or not. First, one chooses a $k^*(x, y)$ which represents the correct diffusion coefficient, and also one chooses the boundary condition $f(x, y, t)$ which represents a part of the measured data. Its Laplace transform $F(x, y, s)$ is numerically computed for a chosen discrete set of $s = s_\sigma$, $\sigma = 1, 2, 3, \cdots, \Sigma$. Then the boundary value problem (3) is solved by the finite difference method (12), (13) and (14); thus one generates the rest of the data $H(x, y, s_\sigma)$, $= 1, 2, 3, \cdots, \Sigma$, for the PST algorithm by the finite difference approximation (17). Next, $k_0(x, y)$ is chosen. Hence upon solving (5), (6) and (10) numerically, $k_1(x, y)$ is obtained. $k_2(x, y)$ can be obtained in a similar manner. One continues this procedure until finally a numerical limit $k_N(x, y)$ is reached. Other than the truncation, round-off, numerical integration and finite difference approximation errors in both generating the numerical data and computing $k_N(x, y)$, any norm $\|k^*(x, y)-k_N(x, y)\|$ can be used as a criterion for evaluating the performance of the computational algorithm of PST.

The numerical simulation here is carried out for a general class of $k^*(x, y)$ and $k_0(x, y)$, e.g., constants, piecewise-linear continuous functions and oscillatory functions. Previous experiences have shown that the interchange of the functions for $k^*(x, y)$ and $k_0(x, y)$ results in no significant differences in $k_N(x, y)$'s except in some of the fine details. Hence for avoiding the expense in generating numerical data, a single constant $k^*(x, y)$ and various different $k_0(x, y)$'s are used for most of the numerical examples here. Also, for simplicity, only rectangles, quadrilaterals and triangles are used for the domain Ω. Furthermore, to avoid the expense of performing the numerical Laplace transformation $f(x, y, t)=1-e^{-t}$ on Γ is used. Hence $F(x, y, s)=1/s(s+1)$ on Γ. Here $s_\sigma=\sigma$, $\sigma=1, 2, 3, \cdots, 11$, are chosen in our computation.

The numerical results are plotted in Figs. 4–13. The maximum norms of $k^*(x, y)-k_N(x, y)$ and $k^*(x, y)-k_0(x, y)$ for all cases can be estimated from the graphs in these figures. The L_2 norms, $I_n=\|k^*(x, y)-k_n(x, y)\|_2$, $n=0, N$, and $\|k^*\|_2$ for all cases are tabulated in Table 1.

TABLE 1

Fig. #	4	5	6	7	8	9	10	11	12	13
N	10	5	13	10	2	2	9	12	10	19
I_0	1.60	3.19	0.97	2.52	6.71	5.00	2.50	0.74	1.60	2.50
I_N	0.19	0.37	0.19	0.50	2.75	3.43	0.25	0.07	0.21	0.26
$I_N/\|k^*\|_2$	0.019	0.025	0.019	0.037	0.239	0.295	0.028	0.007	0.021	0.026

Discussion. Although only a small number of computational zones are used in the numerical simulation here, the numerical results in Figs. 4–13 have demonstrated that the PST iterative numerical algorithm does give excellent results in solving two-dimensional inverse problems for the linear diffusion equation and it is as robust as for the one-dimensional case [9]. The accuracy of the numerical algorithm can be improved greatly if a larger number of computational zones are used; more effort is made in computing each individual step and in discretization of the partial differential and integral equations in the numerical algorithm; and more of s_σ's are used and their values are properly chosen according to either the minimum error criterion [22] or the well-conditioned matrix criterion [23] in solving the Fredholm integral equation of the first kind.

Mathematically, the inverse problem is a nonlinear problem regardless whether the original partial differential equation is linear. Hence in general the solution of an inverse problem with a minimum number of constraints is not unique. To be sure, the PST iterative numerical algorithm is not a method for settling the question of the uniqueness of the solution of an inverse problem. Rather, it is a constructive method for constructing one of the approximate solutions of the inverse problem. This approximate solution is unique in the sense of being the closest one to the initial guess in the L_2 norm. Moreover, it is clear from our numerical simulation that for different initial guess $k_0(x, y)$ the iterations converge to slightly different $k_N(x, y)$'s. However, if this numerical algorithm is reasonably robust, then any one of the approximate solutions will be an acceptable approximation. This numerical computation

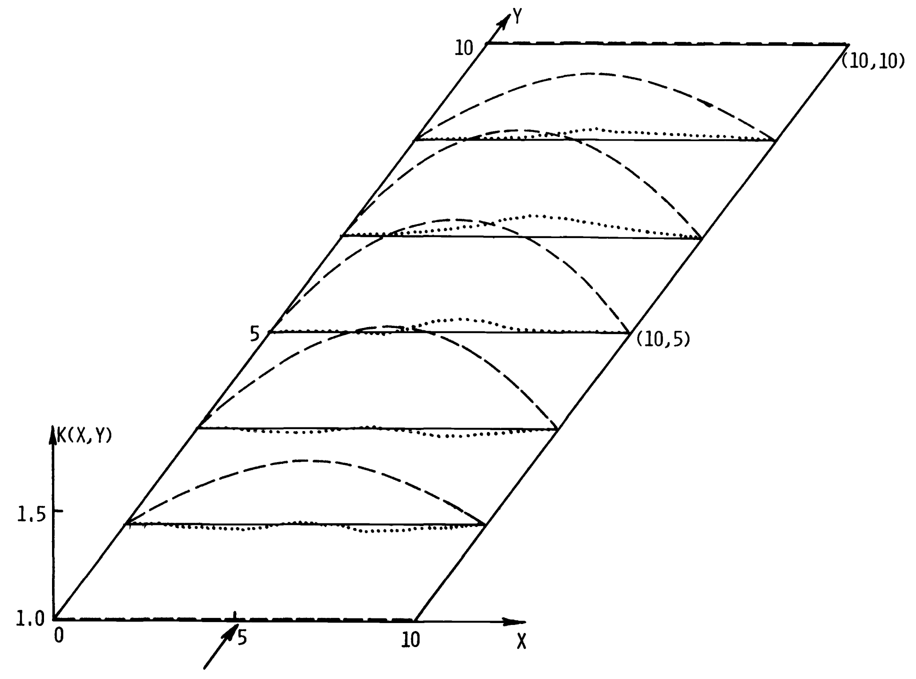


FIG. 4. *Comparison of the calculated* $k_{10}(x, y)$ $\cdots\cdots$ *and the exact* $k^*(x, y)$—— *with the initial guess* $k_0(x, y)$ - - - - -, $\Delta x = \Delta y = 1.67$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $y = 0$.

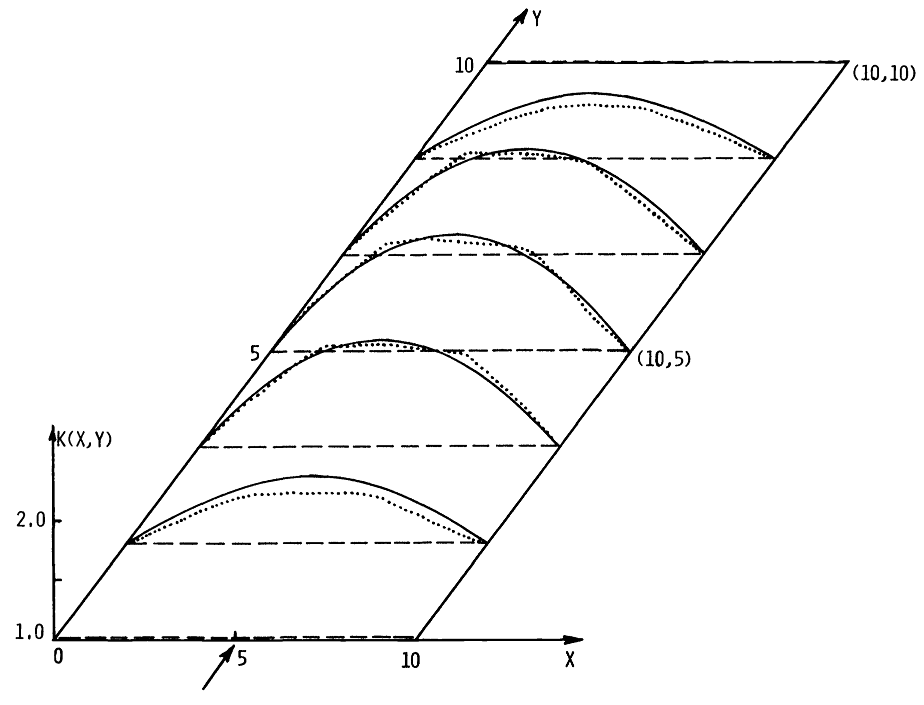


FIG. 5. *Comparison of the calculated* $k_5(x, y)$ $\cdots\cdots$ *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , $\Delta x = \Delta y = 1.67$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $y = 0$.

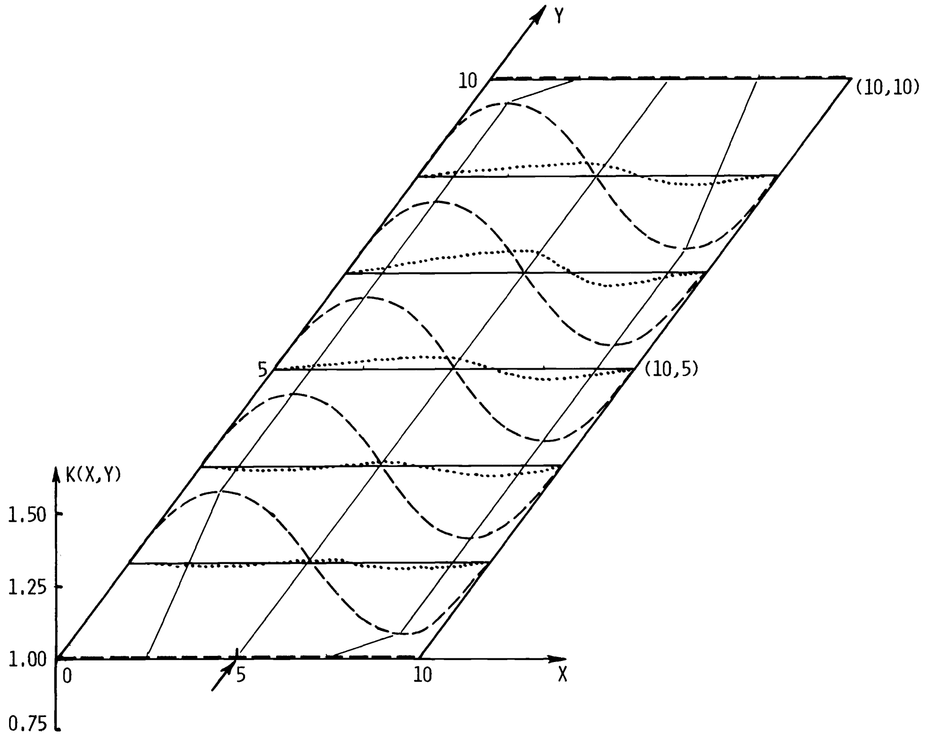


FIG. 6. *Comparison of the calculated* $k_{13}(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , $\Delta x = \Delta y = 1.67$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $y = 0$.

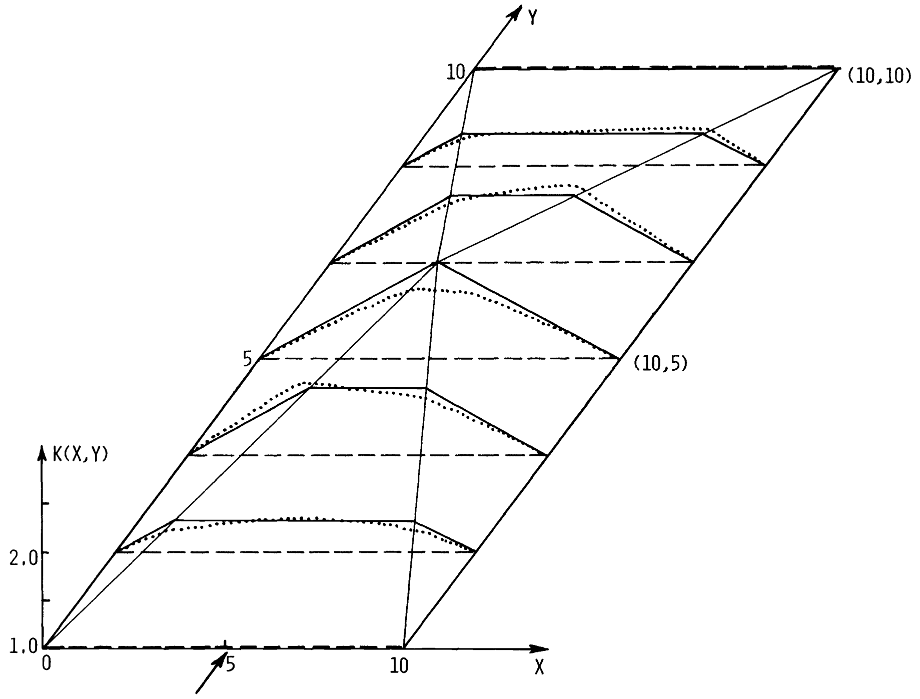


FIG. 7. *Comparison of the calculated* $k_{10}(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , $\Delta x = \Delta y = 1.67$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $y = 0$.

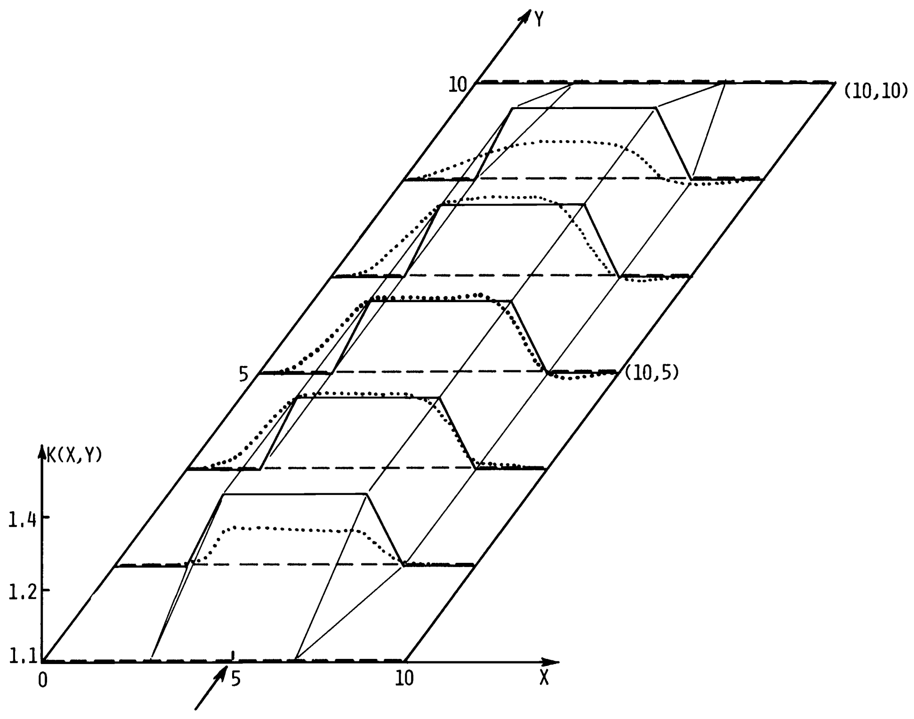


FIG. 8. *Comparison of the calculated* $k_2(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , $\Delta x = \Delta y = 1.67$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $y = 0$.

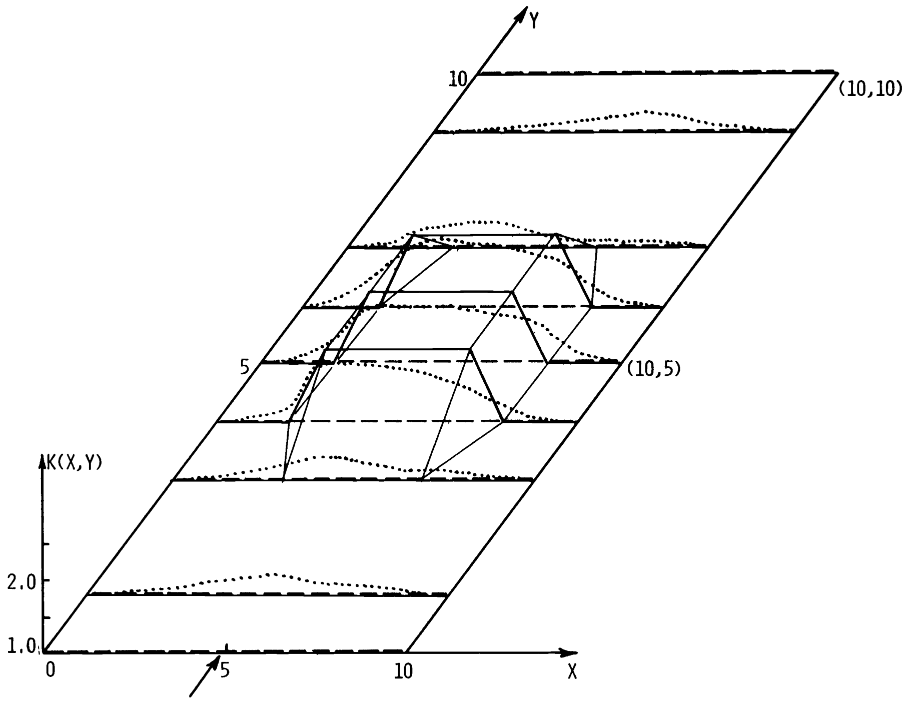


FIG. 9. *Comparison of the calculated* $k_2(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , $\Delta x = \Delta y = 1$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $y = 0$.

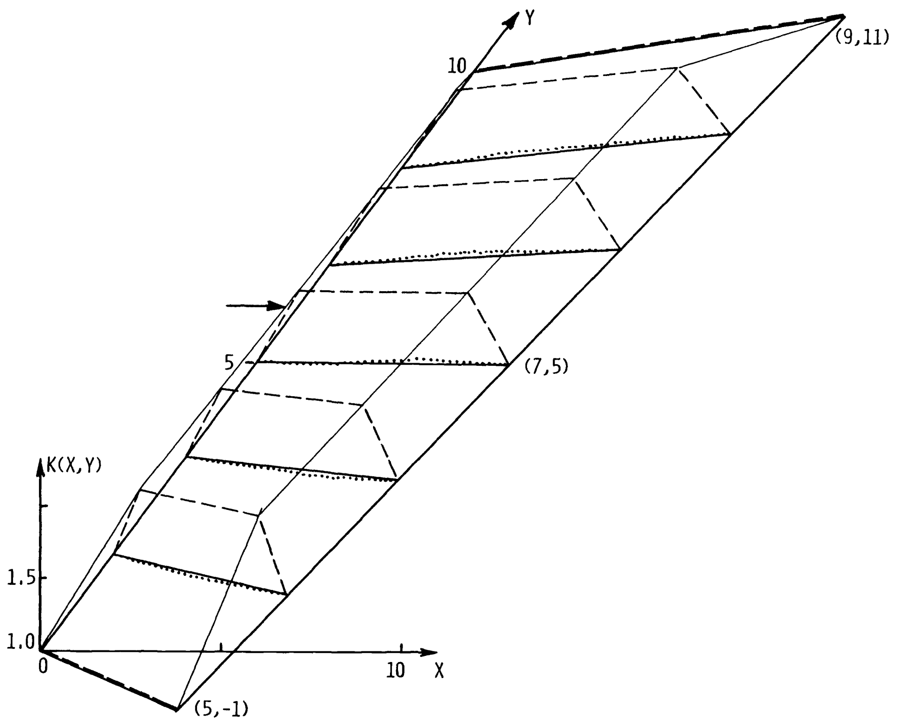


FIG. 10. *Comparison of the calculated* $k_9(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , *each side of the quadrilateral being divided into* 6 *equal-distance subintervals and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $x = 0$.

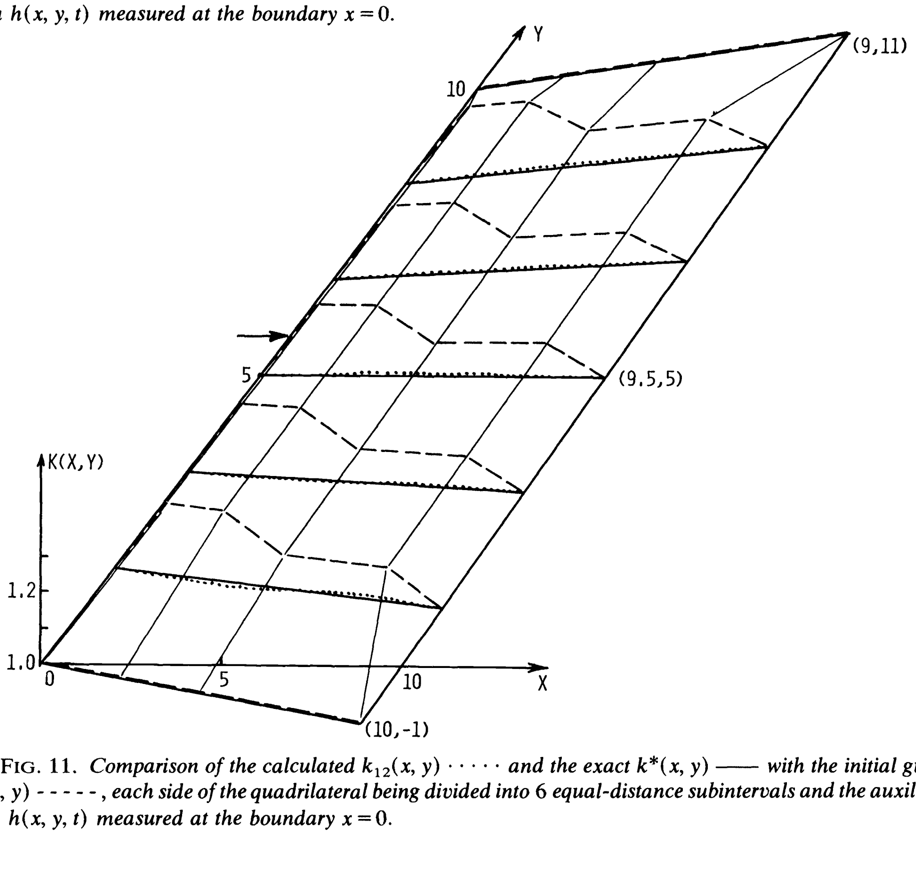


FIG. 11. *Comparison of the calculated* $k_{12}(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , *each side of the quadrilateral being divided into* 6 *equal-distance subintervals and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $x = 0$.

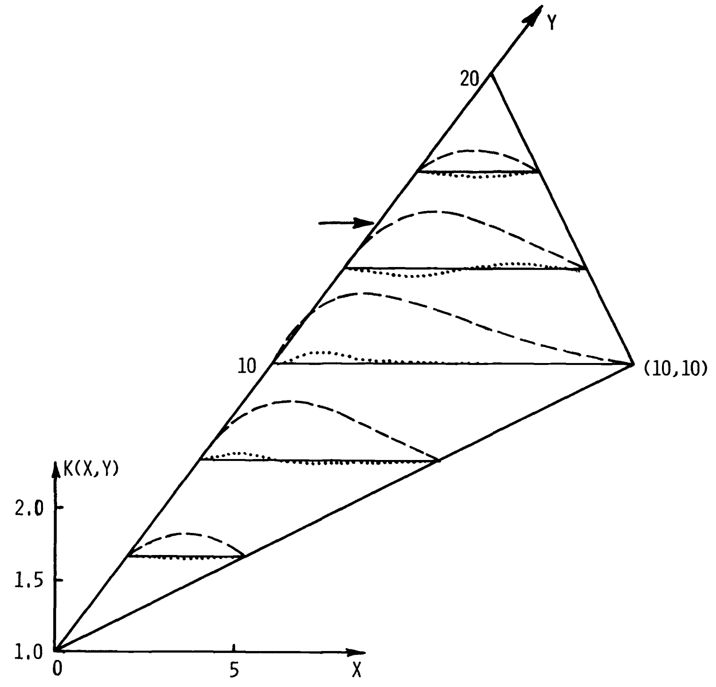


FIG. 12. *Comparison of the calculated* $k_{10}(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , $\Delta x = \Delta y = 1.67$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $x = 0, 10 \leq y \leq 20$.

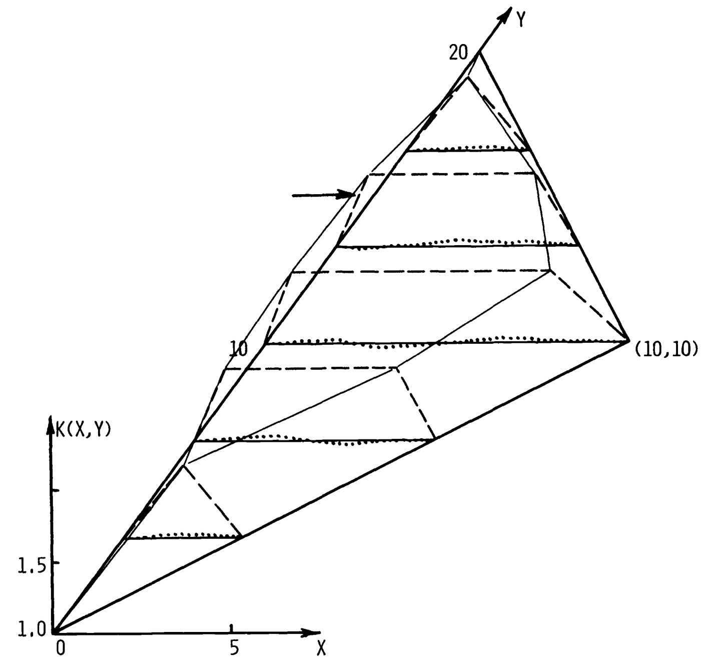


FIG. 13. *Comparison of the calculated* $k_{19}(x, y)$ · · · · · *and the exact* $k^*(x, y)$ —— *with the initial guess* $k_0(x, y)$ - - - - - , $\Delta x = \Delta y = 1.67$ *and the auxiliary data* $h(x, y, t)$ *measured at the boundary* $x = 0, 10 \leq y \leq 20$.

phenomenon can be attributed to the accumulation of the nonnegligible errors in computing each iterate. Here, most of these computational errors come from the regularization procedure in solving the ill-posed Fredholm integral equation of the first kind.

The PST iterative numerical algorithm can be extended to solve three-dimensional inverse problems in a straightforward manner, because the finite difference method or the finite element method is just as adaptable to solve any three-dimensional boundary value problem with arbitrary finite domain as to solve the two-dimensional boundary value problem, and the Tikhonov's regularization method is also as adaptable to solve the three-dimensional Fredholm integral equation of the first kind as to solve the two-dimensional case. Moreover, since the synthesis procedure is carried out in the frequency domain where the governing equation is an elliptic partial differential equation coming either from the Laplace transformed hyperbolic equation or the Laplace transformed parabolic equation as is the case here, the PST iterative numerical algorithm can be used to solve inverse problems of both hyperbolic and parabolic partial differential equations with trivial changes in the computer code of PST. Hence PST fares very well in regard to the universality criterion.

As has been demonstrated in the previous sections, measurement data are needed only at a portion of the boundary to solve two-dimensional inverse problems of the linear diffusion equation successfully. Hence the PST fares very well in regard to the economy of data acquisition criterion in comparison with the history matching techniques [10]–[12] where additional data must be measured in the interior.

The programming for the PST is basically nondedicated, because the changeover from solving the inverse problem for a class of hyperbolic partial differential equations to solving the inverse problems for a class of parabolic partial differential equations is simply a matter of changing a few coefficients (changing a card) in the elliptic equation solver. Moreover, one does not have to program a subroutine for the elliptic equation solver, for there are many finite difference and finite element computer codes available in the public domain for solving two-dimensional and three-dimensional elliptic partial differential equations. Hence the PST again fares very well in regard to the economy of programming effort criterion.

Finally, it also fares rather well in regard to the economy of computing cost criterion, mainly because there are relatively few data measured at sparse locations on the boundary to be processed, in comparison with the history matching techniques where many additional interior data are needed. However, the actual computing costs depend very much on the particular computer hardware and software and one cannot be sure of this until a benchmark comparison test is performed.

Efforts to carry out the generalization of the PST iterative numerical algorithm to solve two-dimensional inverse problems for the wave equation and three-dimensional inverse problems for the diffusion equation are underway, and their results will be reported in the near future. It is also important to point out that the PST iterative numerical algorithm also can be generalized to determine several unknown coefficients of a system of partial differential equations simultaneously and similar efforts have been started.

REFERENCES

[1] B. F. JONES, JR., *The determination of a coefficient in a parabolic differential equation Part* I. *Existence and uniqueness*, J. Math. Mech., 11 (1962), pp. 907–918.

[2] J. DOUGLAS, JR. AND B. F. JONES, JR., *The determination of a coefficient in a parabolic differential equation Part* II. *Numerical approximation*, J. Math. Mech., 11 (1962), pp. 919–926.

[3] J. R. CANNON, *Determination of an unknown coefficient in a parabolic differential equation*, Duke Math. J., 30 (1963), pp. 313–323.

[4] ———, *Determination of certain parameters in heat conduction problems*, J. Math. Anal. Appl., 8 (1964), pp. 188–201.

[5] J. R. CANNON AND P. DUCHATEAU, *Determination of unknown physical properties in heat conduction problems*, Int. J. Engrg. Sci., 11 (1973), pp. 783–794.

[6] ———, *Determination of unknown coefficients in a nonlinear heat conduction problem*, SIAM J. Appl. Math., 24 (1973), pp. 298–314.

[7] ———, *Determination of unknown coefficients* in *parabolic operators from overspecified initial-boundary data*, J. Heat Transfer, 100 (1978), pp. 503–507.

[8] ———, *An inverse problem for a nonlinear diffusion equation*, SIAM J. Appl. Math., 39 (1980), pp. 272–289.

[9] Y. M. CHEN AND J. Q. LIU, *A numerical algorithm for remote sensing of thermal conductivity*, J. Comput. Phys., 43 (1981), pp. 315–326.

[10] W. H. CHEN, G. R. GAVALAS, J. H. SEINFELD AND M. L. WASSERMAN, *A new algorithm for automatic history matching*, Soc. Pet. Eng. J., 257 (1974), pp. 593–608.

[11] M. G. CHAVENT, M. DUPAY AND L. LEMONNIER, *History matching by use of optimal theory*, Soc. Petrol. Eng. J., 259 (1975), pp. 74–86.

[12] S. CHANG AND W. YEH, *A proposed algorithm for the solution of the large-scale inverse problem in groundwater*, Water Resource Res., 12 (1976), pp. 365–374.

[13] E. O. FRIND AND G. F. PINDER, *Galerkin solution of the inverse problem for aquifer transmissivity*, Water Resource Res., 9 (1973), pp. 1397–1410.

[14] D. A. NUTBROWN, *Identification of parameters in a linear equation of groundwater flow*, Water Resource Res., 11 (1975), pp. 581–588.

[15] G. R. RICHTER, *An inverse problem for the steady state diffusion equation*, SIAM J. Appl. Math., 41 (1981), pp. 210–221.

[16] D. S. TSIEN AND Y. M. CHEN, *A numerical method for nonlinear inverse problems in fluid dynamics*, Computational Methods in Nonlinear Mechanics, Proc. Int. Conf. Computational Methods in Nonlinear Mechanics, Univ. Texas, Austin, 1974, pp. 935–943.

[17] Y. M. CHEN AND D. S. TSIEN, *A numerical algorithm for remote sensing of density profiles of a simple ocean model by acoustic pulses* J. Comput. Phys., 25 (1977), pp. 366–385.

[18] D. S. TSIEN AND Y. M. CHEN, *A pulse-spectrum technique for remote sensing of stratified media*, Radio Science, 13 (1978), pp. 775–783.

[19] R. P. HATCHER AND Y. M. CHEN, *An iterative method for solving inverse problems of a nonlinear wave equation*, this Journal, 4 (1983), pp. 149–163.

[20] Y. M. CHEN, *Numerical methods for solving a class of inverse matrix problems in active remote sensing*, Proc. International Symposium Ill-posed Problems: Theory and Practice, Univ. Delaware, Newark, Oct. 1979.

[21] A. N. TIKHONOV AND V. Y. ARSENIN, *Solutions of Ill-Posed Problems*, John Wiley, New York, 1979.

[22] E. TSIMIS, *On the inverse problem by means of the integral equation of the first kind*, Ph.D. thesis, Dept. Applied Mathematics and Statistics, State Univ. New York at Stony Brook, 1977.

[23] F. HAGIN, *On the construction of well-conditioned systems for Fredholm* I *problems by mesh adapting*, J. Comput. Phys., 36 (1980), pp. 154–169.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

DESIGN FEATURES OF A FRONTAL CODE FOR SOLVING SPARSE UNSYMMETRIC LINEAR SYSTEMS OUT-OF-CORE*

I. S. DUFF†

Abstract. We discuss design features of a code which solves sparse unsymmetric systems of linear equations using a frontal method. We consider in particular the user interface, the internal data structures, the pivoting strategy, and the isolation of machine dependencies. We illustrate the performance of our code on a variety of test problems on both the IBM 3033 and the CRAY-1.

Key words. sparse unsymmetric linear equations, unsymmetric frontal methods, out-of-core solver, vectorization, Gaussian elimination, finite-element equations, finite-difference equations.

1. Introduction. In this paper we discuss the design of a code for the solution of sets of n unsymmetric linear equations by the frontal method. A code implementing this method was first produced by Hood (1976) and our software is based on work of Cliffe, Jackson et al. (1978) who developed a version of Hood's code for the solution of finite-element problems in flow modelling. It is our belief that ours is the first unsymmetric frontal code which meets the standards required for incorporation in a general mathematical software library.

We summarize the frontal algorithm in § 2 and show how it can be extended to handle any unsymmetric system. Pivoting strategies are examined in § 3. In § 4, we consider the design of the user interface and the isolation of machine dependencies. We examine some of the data structures employed and their efficient manipulation in § 5. Finally, in § 6, we look briefly at the performance of our code on the IBM 3033 and the CRAY-1 and discuss related work.

The reader more interested in the finer details of the code should consult Duff (1981) which is essentially a user's guide and incorporates the specification sheets as an appendix.

The software described in this report is in the Harwell Subroutine Library under the generic name MA32. A card deck or magnetic tape of the source code for this subroutine can be obtained by writing to S. Marlow, Harwell Subroutine Library, Building 8.9, A.E.R.E. Harwell, Oxon OX11 0RA. A version for the CRAY-1 computer is also available on request.

2. The frontal method and its extension to nonelement problems. The basis for all frontal schemes is Gaussian elimination. That is, we perform the LU decomposition of a permutation of A which we can write as

$$A = PL \cdot UQ, \tag{2.1}$$

where P, Q are permutation matrices and L and U are lower and upper triangular matrices respectively. We permit problems that are so large that PL and UQ need to be held on an auxiliary storage device. An important observation is that only the factors PL and UQ are used during the solution process and, in frontal schemes, we make use of this by never storing (or indeed generating) the whole of A at one time.

Although frontal schemes were originally developed (Irons (1970)) for the solution of finite element discretizations in structural analysis where the resulting assembled stiffness matrix is positive definite, they are applicable to a far wider class of problems and can be modified to work when the resulting matrix is indefinite or even unsym-

* Received by the editors December 7, 1981, and in revised form December 14, 1982. This paper was written by an employee of the United Kingdom Atomic Energy Authority. © UKAEA.

† Computer Science and Systems Division, A.E.R.E. Harwell, Oxfordshire, England OX11 0RA.

metric. Indeed the software we will describe in this paper can be used to solve any general unsymmetric set of linear equations although it will not always be the most efficient method.

It is easiest to describe the frontal method by reference to its application in the solution of a finite-element problem. Here the matrix A is a sum

$$A = \sum_{l} B^{(l)}, \tag{2.2}$$

where each $B^{(l)}$ has nonzeros in only a few rows and columns and corresponds to contributions to the matrix from finite element l. It is normal to hold $B^{(l)}$ in packed form as a small full matrix together with an indexing vector to identify where the nonzeros belong in A. The basic "assembly" operation when forming A is thus of the form

$$a_{ij} \leftarrow a_{ij} + b_{ij}^{(l)}. \tag{2.3}$$

It is evident that the basic operation in Gaussian elimination

$$a_{ij} \leftarrow a_{ij} - a_{ik}[a_{kk}]^{-1} a_{kj} \tag{2.4}$$

may be performed before all assemblies (2.3) are completed, provided only that the terms in the triple product in (2.4) are all fully summed (that is, have no more sums of the form (2.3) to come) before execution of the elimination operation (2.4).

Since variables can only be eliminated after they are fully summed, the assembly order (which we discuss further in § 6) will determine, to a large extent, the order of elimination. At any stage during the assembly and elimination, the fully- or partially-summed variables are held in an in-core frontal matrix. If we permute all the fully-summed variables to the first rows and columns of this matrix, it will have the form shown in Fig. 2.1.

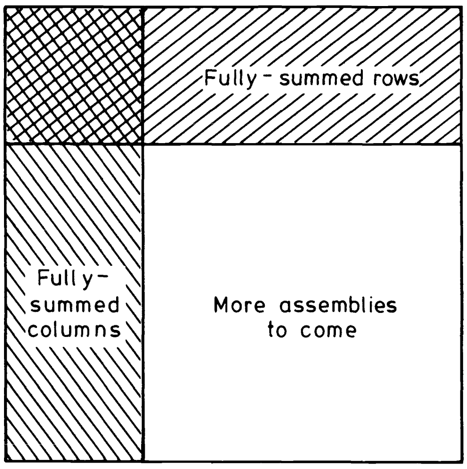


FIG. 2.1. *Frontal matrix.*

Pivots can be chosen from within the doubly-shaded region. For positive-definite systems, they can be taken from the diagonal in order but for indefinite systems numerical pivoting is required for stability. A suitable scheme, discussed by Duff and Reid (1982), would be to use block pivots from the diagonal of order 1 or 2 (Bunch and Parlett (1971)). In the unsymmetric case, pivots can be chosen from anywhere within the block subject to satisfying some numerical tolerance. We discuss this further in the following section.

A judicious choice of assembly order, dependent on the geometry and connectivity of the underlying problem, will keep the size of this frontal matrix small thus reducing arithmetic operations and storage requirements. For example, if we assemble the elements shown in Fig. 2.2 in the order indicated by the numbering in that figure, then the size of the partially-assembled matrix (which we call the frontal matrix) need never exceed the number of variables in two elements. The problem of obtaining a good assembly order for an arbitrary system is a difficult one which we discuss further in § 6.

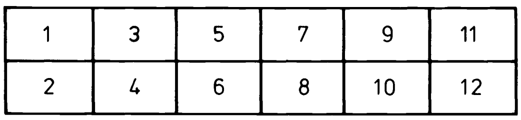
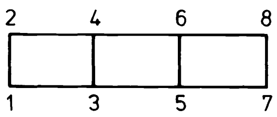


FIG. 2.2. *Assembly order in a finite-element problem.*

Many finite-element formulations include variables which are internal to the element and can thus be eliminated without reference to any other elements. It is much more efficient to perform these "static condensations" within the element itself rather than after assembling the element into the frontal matrix since the work involved will be a function of element rather than front size. Thus, in most problems, such eliminations are obtained almost for free and we have found cases (Cliffe, private communication) where savings of over 30% in execution time have been obtained.

The summation (2.2) is not restricted to a finite-element application. Each $B^{(l)}$ could represent a substructure, a low rank change to the matrix, or indeed a single row of the assembled matrix. We have made our code efficient in the latter case so that it can be used to solve systems arising from finite-difference discretizations. To our knowledge, this is the only unsymmetric frontal code to offer this facility. In principle, we could use our code to solve any unsymmetric system. We discuss this further in § 6.

For nonelement problems, the rows (equations) are added to the frontal matrix in turn and a variable is regarded as fully summed whenever the equation in which it last appears is added. This situation is illustrated in Fig. 2.3(b) where we show the frontal matrix after the input of equation 3 from the five-point discretization of the Laplacian operator on a 2×4 grid, as shown in Figure 2.3(a). At this stage no further equations will cause any nonzero entries to appear in the first column of Fig. 2.3(b), so this column is effectively fully summed and any entry in it (subject perhaps to a numerical criterion) can be chosen as pivot.



(a)

$$\begin{array}{ccccc} -4 & 1 & 1 & & \\ 1 & -4 & 0 & 1 & \\ 1 & 0 & -4 & 1 & 1 \end{array}$$

(b)

FIG. 2.3. *Illustration of equation input in frontal schemes.* (a) 2×4 *grid.* (b) *First three rows of matrix (entries to the right are zero).*

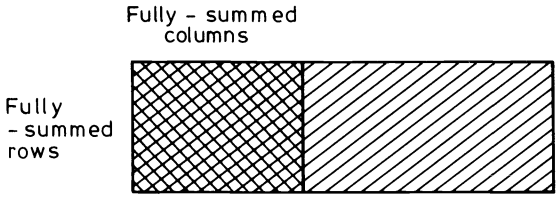


FIG. 2.4. *Frontal matrix (equation entry).*

The frontal matrix will, in this case, be rectangular and the counterpart to Fig. 2.1 is shown in Fig. 2.4 where, as in Fig. 2.1, pivots can be chosen from anywhere within the doubly-shaded region.

The user interface (see § 4) has been designed so that input by elements or by equations is equally easy and the equation entry has been used very successfully to solve equations arising from the dynamics of rapidly rotating gases. We include, in the illustration of the performance of our code in § 6, systems which arise from finite-difference discretizations.

The nonelement scheme just described is a generalization of variable band methods (Jennings (1966)) and will be superior to band schemes on most problems (for example, Hood (1976)).

3. Pivoting strategies. Hood (1976) uses two parameters, MAXFRT and MINFRT, to control the size of the frontal matrix. He performs assemblies until the front size reaches MAXFRT and then performs eliminations using the largest entry in the fully-summed block (double-shaded region in Figs. 2.1 and 2.4) as pivot until the front size reaches MINFRT.

We do not like this approach for two reasons. The first is that arithmetic may be performed on a very much larger frontal matrix than is necessary or desirable (the parameters are static and are set on entry). The second is that it is not so much pivot size as the size of the pivot relative to other entries in its row and column which is important in maintaining stability. The largest entry in the fully-summed block need not be large compared to the largest entry in its row or column.

From our past experience with general sparse codes, we have found a threshold criterion of the form

$$|a_{lk}| \geqq u \cdot \max_i |a_{ik}|, \tag{3.1}$$

where u is a user set parameter in the range (0, 1], to be a satisfactory test for a pivot a_{lk}. Note that partial pivoting corresponds to a u value of 1.0. It is possible to incorporate threshold pivoting in an unsymmetric frontal scheme. We refer to Fig. 2.1. Pivots can only be chosen from within the doubly-shaded region although the maximum in the stability test (3.1) must be over all rows of the frontal matrix. This is a legitimate test because all columns from which pivots can be chosen are fully-summed. We search the fully-summed columns for a pivot and the first entry in the doubly-shaded region to satisfy the threshold criterion is used as the next pivot. If we search several columns before finding a suitable pivot, we look for the next pivot starting from the first unsearched fully-summed column since it is quite likely that the previously searched columns will still not yield a pivot. Although this strategy shows little overall effect compared to that of searching from the first column each time, it is marginally beneficial and is very easy to implement.

It is possible that we cannot choose pivots from all the fully-summed columns because of large entries in the singly-shaded region of the fully-summed columns. In this case we leave the corresponding columns (and rows) in the frontal matrix and continue with further assemblies. It is always possible to complete the factorization using this pivoting strategy since eventually all rows and columns become fully summed and hence the largest element in the entire column can be used as pivot. Naturally, if many eliminations were delayed because of the test (3.1), the worry is that the order of the frontal matrix might be noticeably larger than it would have been without numerical pivoting. In practice, we have found that such an increase is slight and indeed, since pivots are chosen as soon as the test (3.1) is satisfied, the front size is not much greater than dictated by the underlying geometry. In particular, although our code requires the input of a maximum order for the frontal matrix, we do not require Hood's parameters and may often do much less work than he would.

We note that the pivoting strategy simplifies when we are inputting the matrix by equations. Here there are no non-fully-summed rows in the fully-summed columns (see Fig. 2.4) and so partial pivoting can be used without any possibility of delaying eliminations.

On Jackson's advice, we allow the user to limit the pivot search to a specified number of fully-summed rows and columns and to force eliminations to keep the number of fully-summed rows and columns under a preset level. Even in this case, candidates which come closest to satisfying (3.1) are chosen in preference to those which are numerically largest. In addition to enabling the solution of problems whose size forbids more exhaustive searching, these controls permit pivoting down the diagonal to be forced when such pivoting is known, a priori, to be stable.

4. Some software considerations. In this section we discuss some aspects of our user interface and briefly touch on the isolation of machine dependencies.

Input to the frontal solver will be by elements (the $B^{(l)}$ of § 2) in a finite-element calculation but, as we have discussed previously, it is desirable that the interface also allows input by rows (equations). Communication between the user and the package in the earlier codes was through calls by the factorization routine to a user-written subroutine whose parameter list was necessarily fixed. To avoid this limitation and in order that the equation and element entries can be handled with equal ease, we have chosen to use "reverse communication" in our user interface. A similar interface is used by SPARSPAK (George et al., (1980)). This means that control is returned to the user after each assembly. Thus, the call structure is of the form shown in Fig. 4.1. An added advantage of this way of working is that additional information can be easily passed to the user's program which can monitor

```
for each element or equation do
  begin
    input or generate element or equation;
    call subroutine:
  end
```

FIG. 4.1. *Use of reverse communication with the frontal solver.*

the progress of the elimination. For instance, it is possible that the user may terminate the sequence of calls early and we discuss this further below.

In addition to the numerical values, the user must provide integer information on the position of these nonzeros in the assembled matrix. As we saw in § 2, the frontal solver also requires information on when variables become fully summed. One way of providing this (adopted by Hood (1976) and Cliffe et al. (1978)) is to negate the integer information for variables appearing for the last time. We reject this approach since it imposes an onus on the user to store or generate the flags. Instead we perform a prepass, on the integer information only, which generates a single integer array of length n which records, for each variable, the call at which it appears for the last time. This array, which can be set directly by the user if the geometry is known, provides the necessary information on fully-summed variables to the main solve routine and can additionally be used within the main routine for temporary work space. We allow the user to solve for a specified number of right-hand sides since, if the application allows the solution of more than one system at a time, the I/O costs are less of an overhead.

For the calls to the frontal solver, the user must provide values for the maximum size of the frontal matrix and must allocate auxiliary storage for the matrix factors. If the problem is complicated or unfamiliar it may be hard to choose appropriate values, so we feel it is very important to return useful information should the run fail due to insufficient space. For computing systems which require the auxiliary storage to be allocated a priori (for example, IBM), if such an allocation is exceeded, we simply continue the decomposition throwing away the factors and calculating the space required for subsequent runs. If insufficient space is allocated to the frontal matrix, the situation is slightly more complicated. In this case, we continue with a symbolic factorization only, whose space requirements are linear rather than quadratic in the front size. We can thus continue as long as the front size does not exceed the square of the order of the frontal matrix and can return the size of frontal matrix required. Although the storage so determined will be sufficient for subsequent runs without pivoting, the user may wish to increase it slightly to allow for numerical pivoting. In both cases, a flag is set so that the run can be optionally terminated immediately if the allocated space is exceeded. This flexibility is greatly facilitated by our use of reverse communication.

We provide a separate entry for the solution of subsequent systems with the same coefficient matrix. This is just a single call with the assembled right-hand side(s) supplied as an argument. Again we allow more than one right-hand side to be input.

We have isolated all communication with auxiliary storage to a single short subroutine. The factors of PL and UQ (2.1) are blocked in in-core buffers before being passed to this I/O subroutine. It is normally better to reduce actual I/O requests by buffering and we allow the user to choose the size of these buffers to optimize for system characteristics.

Since the rows of UQ must be read in the opposite order to which they are written, it is advantageous on some machines (for example, the IBM 3033) to use direct access because the cost of the backspace operation is very high. We have chosen to do this in our current IBM version and also use nonstandard Fortran in the CRAY-1 version. With the exception of this well isolated I/O, we have written our code in the subset of ANSI-66 Fortran supported by the PFORT verifier (Ryder (1974)).

Since it is not possible to allocate data sets dynamically using Fortran on an IBM 3033, we have written an initialization routine for the IBM version of our code which calls assembly-coded routines to perform this allocation. We feel this option is very desirable in a library environment particularly when the user does not wish to preserve the factors and so does not need even to know of the existence of data sets on backing store.

The threshold parameter (u in (3.1)) and other pivoting controls mentioned in § 3 together with the stream number for output messages are all in a single named common block. On the basis of fairly extensive testing we have chosen default values for these parameters and have set the defaults in a block data subprogram (for the CRAY-1 version the defaults are set within the main subroutine in a DATA statement). In spite of this minor portability difficulty we use this approach to setting default values because we believe that these parameters will not be of concern to the average user and so should not need to be referenced in his code.

In summary, our interface with the user is particularly simple and flexible. The four routines which the user calls directly also check the input data and subdivide workspace thus screening the user from the complexities of 16 subroutines (in a 4 level hierarchical structure) and 1785 statements in the complete package. A much fuller discussion of the structure of the package is given by Duff (1981).

5. Data structures employed. A discussion of the data structures used in frontal codes divides naturally into three, corresponding to input, structures internal to the package, and output. The user is screened from all of the data structures except those relating to input. We present only a very brief discussion of data structures in this paper and refer the interested reader to Duff (1981) for full details.

The user need only input the reals corresponding to each $B^{(l)}$ of equation (2.2) together with an integer indexing array denoting the columns of A in which the entries lie. The user is not required to flag when variables are fully summed. A sequence of calls to a prepass routine (see § 4) where only the integer indexing arrays need be supplied generates an array which indicates, for each variable, the assembly at which it becomes fully summed. If the user can generate the fully-summed information in other ways, then the calls to the prepass routine can be avoided.

After any static condensations (see § 2), the input element or equation is loaded into the frontal matrix which is held as a full rectangular array (square in the case of element entries). All the subsequent pivoting and elimination operations take place in this frontal matrix. In addition to the frontal matrix, we need indexing information to relate positions within this matrix to the overall coefficient matrix. We follow Hood by holding two integer arrays which, for each row and column of the frontal matrix, indicate the corresponding row and column of the coefficient matrix. Two are necessary even with the element entry because pivoting from off the diagonal will destroy the symmetry in the frontal matrix.

Our other data structures internal to the factorization routines are designed to facilitate the assembly of incoming elements or equations, the identification of the submatrix in A (2.2) corresponding to the current frontal matrix, and the access to and identification of fully-summed rows and columns. Our data structures, described in detail by Duff (1981), are different from earlier codes. In particular, it is possible to tell immediately whether an incoming entry is in the frontal matrix without needing to scan any array. The storage required is only four times the maximum number of columns in the front plus twice the maximum number of rows. This storage could be reduced slightly, since the sets of fully and non-fully-summed variables partition the frontal variables, but would result in more opaque coding. In any case, the same storage was required by the earlier codes.

As we mentioned in § 4, the rows of UQ and columns of PL are buffered before output to backing store. We do not allow a single row or column to span a buffer but we have found empirically that this fragmentation incurs a storage overhead of only about 2 to 3%. Since each variable enters and leaves the front once only, we need only record the indices of each when it enters the front during forward elimination or back substitution. By doing this an index list of variables in the current front can be maintained during the solution process and there is no need to follow Hood in storing one index for every nonzero in the factors. A consequence of this sometimes substantial saving in storage is that the storage for single rows and columns becomes more complicated. The resulting storage scheme is discussed in detail by Duff (1981).

6. Comments on performance and related work. Although the principal advantage of frontal codes is that main storage requirements are low, another benefit arises from the use of direct addressing in the inner loop of the factorization routine. This benefit will be particularly evident on a machine like the CRAY-1. Naturally, a frontal code cannot take so much advantage of sparsity as a general sparse code so the greater efficiency of the inner loop in frontal codes will be offset by the fact that more operations must be performed.

Our frontal code will be most efficient when large finite elements are input in which case most of the factorization time will be spent in the inner loop itself. However, as we mentioned earlier, the code has been designed to accept equation input and we would like it to be efficient in this case also. We have compared our code, MA32, with two variable band codes on the model problem arising from 5-point discretizations of the Laplacian operator on a regular grid. The two variable band codes assume the matrix is symmetric and do no numerical pivoting. We would thus expect both the storage and factorization times to be half that of an unsymmetric code. The Harwell code MA15 (Reid (1972)) stores the factors on auxiliary storage while the profile solver from SPARSPAK (George et al. (1980)) uses a reverse Cuthill–McKee ordering (RCM) and works entirely in main memory. Because SPARSPAK does not use auxiliary storage, its solution times are noticeably faster. The results in Table 6.1 indicate that MA32 is competitive with established codes when used in its equation entry mode. For the runs in this table, an RCM ordering was used to order the equations prior to entry to both MA32 and MA15.

The frontal code, MA32, has recently been used on the CRAY-1 by Andrew Cliffe of the Theoretical Physics Division at Harwell in the solution of problems in buoyancy driven flow in a square cavity using finite elements. We show his results in

TABLE 6.1
Performance on a model problem on an $m \times n$ grid. Times in seconds on an IBM 3033.

Grid	m / n	10 / 10	10 / 100	16 / 16	32 / 32	50 / 50
Factorization time						
MA32		.053	.632	.194	1.332	6.038
MA15		.030	.244	.094	.744	3.481
SPARSPAK (RCM ordering)		.019	.265	.072	.629	3.014
Solution time						
MA32		.008	.100	.027	.178	.635
MA15		.012	.060	.030	.156	.519
SPARSPAK (RCM ordering)		.004	.048	.013	.079	.293
Main storage in kbytes						
MA32		40	45	45	60	90
MA15		5	20	5	20	40
SPARSPAK (RCM ordering)		10	110	30	200	740

Table 6.2 where the elements used for the runs in the first two columns were five node rectangular elements with three variables at each corner and one at the centroid. The element for the run in column three was a rectangle having nodes at the corners, mid-points and centroid with three variables defined at each node. We have augmented these results by runs on two artificial problems. The first (column 4) is a grid of ten node elements similar to those used in the runs in column 3 but with five variables at each node. The second (column 5) is a finite difference problem arising from the five-point discretization of the Laplacian operator on a square grid. In every case the elements or equations were assembled in a pagewise ordering along the side of shorter dimension.

TABLE 6.2
Performance of MA32 *on the* CRAY-1

Dimensions of grid of elements or equations	28×28	36×36	20×32	16×16	64×64
Order (degrees of freedom)	3,023	5,039	9,480	5,445	4,096
Maximum front size	117	149	141	195	65
Total time in seconds for solution (including back substitution)	3.83	8.63	13.43	12.58	3.55
Time in innermost loop (in seconds)	1.70	4.29	6.96	7.63	1.49
Number of static condensations	0	0	3346	1620	0
Number of flops in inner loop (in millions)	44.2	120.9	198.6	238.4	32.6
Inner-loop Megaflops	26	28.2	28.5	31.2	21.9
Total Megaflops	11.5	14.0	14.9	19.4	9.2

The results in Table 6.2 illustrate the performance of our frontal code on realistic finite-element problems and furthermore indicate that the inner loop vectorizes on the CRAY-1. All the MA32 code is written in Fortran and we might expect (Duff (1982)) even faster times by CAL (CRAY assembly language) coding of the inner loop. We see that, in the larger problems, the overall Megaflop rate (which includes assembly, pivoting, output of factors and back substitution) exceeds half that of the innermost loop. The two artificial cases have been run on the IBM 3033 at Harwell and the times for the element and the equation input were 133.3 and 26.5 seconds respectively. Thus the overall increase in speed due to the vectorizing capability of the CRAY-1 was 10.6 and 7.5 with the inner loop running over 15 and 11 times faster on the element and equation problem respectively. Since the scalar speed differential of the CRAY-1 over the IBM 3033 is only about 2, the effect of vectorization is evident.

As we explained earlier, it is possible to use our frontal code to solve any general set of sparse equations. We compare it with a general purpose code for unsymmetric sparse equations, MA28 (Duff (1977)), on a range of test problems in Table 6.3, where the equations were input to MA32 in the natural order. Many numerical experiments have been performed. A selection of these has been chosen to represent a range of problem classes. The 1,176 case arose in electrical networking, the 113 case in statistical modelling, the 199 case in stress analysis, the 130 case in a laser investigation, and the 577 case from a finite-element triangulation of an L-shaped region. On very general systems (columns 2–4 of the table), MA32 is generally slower

TABLE 6.3
A comparison with MA28. *Times in seconds on an* IBM 3033.

Order	1,176	113	199	130	577
Nonzeros	18,552	655	701	1,282	3,889
Factorization time					
MA32	2.801	.081	.379	.134	1.311
MA28 (pivot order unknown)	4.143	.059	.105	.160	6.339
MA28 (pivot order given)	1.046	.013	.019	.035	.785
Solution time					
MA32	.461	.016	.061	.025	.158
MA28	.049	.003	.005	.003	.044
Main storage in kbytes					
MA32	160	45	140	60	90
MA28	310	15	25	20	260

than even the main MA28 entry (which finds a suitable pivotal sequence in addition to factorizing the matrix) and the large front size means that MA32 requires even more main storage than the in-core code. On systems with identifiable substructures (column 1) or arising from finite-element discretizations (column 5), MA32 is noticeably more competitive and requires much less main storage than MA28. Since a general sparse code like MA28 uses indirect addressing in the inner loop of the factorization, it does not vectorize. Thus, we might expect MA32 to be even more competitive on a machine like the CRAY-1. Indeed while the MA28 factorization times on the CRAY-1 are typically half those for the IBM 3033, the MA32 times are cut by a factor of around 4.

Naturally, because the assembly order to a large extent determines the pivot order, the order in which the equations (or elements) are input to MA32 will have a significant effect on its efficiency. The investigation of suitable orderings is a research area in its own right and is outwith the scope of this paper. Because of the similarity of our frontal scheme to variable band schemes, we might, however, expect that orderings good in the variable band case would benefit our approach also. We therefore used an RCM ordering generated by SPARSPAK to determine the order in which equations (rows) are input to MA32. We show some results in Table 6.4, where the 113 and 130 cases are as before, the case of order 292 is a normal-equations matrix

TABLE 6.4
The effect of ordering on MA32. *Times in seconds on an* IBM 3033.

Order	113	292	130	443	1,561	503
Nonzeros	655	2,208	1,282	1,623	10,681	6,027
Ordering time	.016	.022	.019	.024	.128	.068
Factorization time						
With ordering	.089	.271	.163	.543	3.738	1.046
Without ordering	.081	.282	.134	1.046	7.645	4.181
Solution time						
With ordering	.014	.037	.032	.116	.357	.120
Without ordering	.016	.040	.025	.214	.693	.214

from a least-squares problem in surveying, the 443 case is from an electrical power network problem, the 1,561 case from the triangulation of an L-shaped region, and the 503 case from a structural analysis problem.

It is apparent that considerable savings are possible on element or structures problems (columns 5 and 6 in the table) and on very sparse network problems (column 4) where the initial ordering produced a large frontal matrix. Unfortunately, the RCM ordering is not always beneficial as we can see from the other columns in the table. Of course, in many cases, for example in the fluid calculations at Harwell, a good ordering is evident from the underlying geometry of the problem.

Duff and Reid (1982) are developing ordering and factorization routines based on multifrontal techniques where the matrix is symmetric (but not necessarily positive definite) and Reid is continuing his work (Reid (1978)) on a multifrontal solver (where the frontal matrix may also be out-of-core) for large symmetric positive–definite element problems.

Acknowledgments. I am grateful to Peter Jackson, Andrew Cliffe, and John Reid for many fruitful discussions concerning the design of the MA32 code and to John Reid for his helpful comments on draft versions of this paper. I am also grateful to the referees for their constructive suggestions.

REFERENCES

J. R. BUNCH AND B. N. PARLETT (1971), *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal., 8, pp. 639–655.

K. A. CLIFFE, C. P. JACKSON, J. RAE AND K. H. WINTERS (1978), *Finite element flow modelling using velocity and pressure variables*. Harwell Report, AERE R.9202, HMSO, London.

I. S. DUFF (1977), MA28—*a set of Fortran subroutines for sparse unsymmetric linear equations*, Harwell Report AERE R.8730, HMSO, London.

——— (1981), MA32—*a package for solving sparse unsymmetric systems using the frontal method*, Harwell Report AERE R.10079, HMSO, London.

——— (1982), *The solution of sparse linear equations on the* CRAY-1. Harwell Report CSS 125. Presented at the Cray Research, Inc. Symposium "Science, Engineering, and the CRAY-1", Minneapolis, 5–7 April 1982. Shortened version in CRAY CHANNELS 4 (3) Dec. 1982.

I. S. DUFF AND J. K. REID (1982), *The multifrontal solution of indefinite sparse symmetric linear systems*, Harwell Report CSS 122, HMSO, London. To appear in ACM Trans. Math. Softw.

A. GEORGE, J. LIU AND E. NG (1980), *User Guide for* SPARSPAK: *Waterloo Sparse Linear Equations Package*, Report CS-78-30 (Revised Jan. 1980). Dept. of Computer Science, Univ. Waterloo, Waterloo, Ontario.

P. HOOD (1976), *Frontal solution program for unsymmetric matrices*, Int. J. Numer. Meth. Engng, 10, pp. 379–399.

B. M. IRONS (1970), *A frontal solution program for finite element analysis*, Int. J. Numer. Meth. Engng, 2, pp. 5–32.

A. JENNINGS (1966), *A compact storage scheme for the solution of symmetric linear simultaneous equations*, Comput. J., 9, pp. 281–285.

J. K. REID (1972), *Two Fortran subroutines for direct solution of linear equations whose matrix is sparse, symmetric and positive-definite*, Harwell Report R.7119, HMSO, London.

——— (1978), *A package of subroutines for solution of very large sets of linear finite element equations*, Harwell Report AERE M.2947, HMSO, London.

B. G. RYDER (1974), *The* PFORT *verifier*, Softw. Pract. Exper., 4, pp. 359–377.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

ADI AS A PRECONDITIONING FOR SOLVING THE CONVECTION-DIFFUSION EQUATION*

RAYMOND C. Y. CHIN†, THOMAS A. MANTEUFFEL‡ AND JOHN DE PILLIS§

Abstract. We examine a splitting of the operator obtained from a steady convection-diffusion equation with variable coefficients in which the convection term dominates. The operator is split into a dominant and a subdominant parts consistent with the inherent directional property of the partial differential equation. The equations involving the dominant parts should be easily solved. We accelerate the convergence of this splitting or the outer iteration by a Chebyshev semi-iterative method. When the dominant part has constant coefficients, it can be easily solved using alternating direction implicit (ADI) methods. This is called the inner iteration. The optimal parameters for a stationary two-parameter ADI method are obtained when the eigenvalues become complex. This corresponds to either the horizontal or the vertical half-grid Reynolds number larger than unity. The Chebyshev semi-iterative method is used to accelerate the convergence of the inner ADI iteration. A two-fold increase in speed is obtained when the ADI iteration matrix has real eigenvalues, and the increase is less significant when the eigenvalues are complex. If either the horizontal or the vertical half-grid Reynolds number is equal to one, the spectral radius of the optimal ADI iterative matrix is zero. However, a high degree of nilpotency impairs rapid convergence. This problem is removed by introducing a more implicit iterative method called ADI/Gauss–Seidel (ADI/GS). ADI/GS resolves the nilpotency and, thus, converges more rapidly for half-grid Reynolds number near 1. Finally, our methods are compared with several well-known schemes on test problems.

Key words. boundary value problems, singular perturbation, convection-diffusion equation, iterative methods, matrix splittings, acceleration techniques, optimal acceleration parameters

1. Introduction.

Consider the steady convection-diffusion equation

$$-\varepsilon \Delta u + a(x, y)u_x + b(x, y)u_y + c(x, y)u = f(x, y) \tag{1.1}$$

on a bounded convex domain D with appropriate boundary conditions. The functions a, b, c, f are assumed to be in C^2, ε and $c > 0$. Equation (1.1) is a model for the Navier–Stokes equation and for the equations of mass and heat transfer.

If $\varepsilon \ll 1$, then the boundary value problem is singularly perturbed. The solution can have boundary layers. Singular perturbation problems for linear differential equations of elliptic type such as (1.1) have been studied by Wasow [22], Levinson [13], Vishik and Liusternik [21], Eckhaus and de Jager [4], O'Malley [18], Eckhaus [3] and Howes [10]–[12].

Levinson has proved for a first boundary value problem with Dirichlet conditions prescribed along the boundary that the solution $u(x, y; \varepsilon)$ is asymptotically a sum of two contributions:

$$u(x, y; \varepsilon) = V(x, y) + z(x, y; \varepsilon) + O(\varepsilon^{1/2})$$

where $V(x, y)$ is the solution of the reduced equation

$$a(x, y)\frac{\partial V}{\partial x} + b(x, y)\frac{\partial V}{\partial y} + c(x, y)V = f(x, y) \tag{1.2}$$

* Received by the editors October 21, 1981, and in revised form February 4, 1983. This work was performed in part under the auspices of the U.S. Department of Energy, Office of Basic Energy Sciences, Applied Mathematics and Statistics Division, by the Lawrence Livermore National Laboratory under contract W-7405-ENG-48 and by the Los Alamos National Laboratory under contract W-7405-ENG-36.

† Lawrence Livermore National Laboratory, Livermore, California 94550.
‡ Los Alamos National Laboratory, Los Alamos, New Mexico 87545.
§ University of California, Riverside, California 92521.

which takes on the given boundary value u on a part S_1 of the boundary ∂D, and $z(x, y, \varepsilon)$ is a boundary layer correction called an exponential boundary layer having the generic form

$$z = e^{-g(x,y)/\varepsilon} h(x, y) \tag{1.3}$$

and is important only near $S_2 = \partial D - S_1$ (see Fig. 1.1). Here, it is assumed that $a(x, y)$ and $b(x, y)$ do not vanish simultaneously in D. In other words, there are no turning points of (1.1) in D.

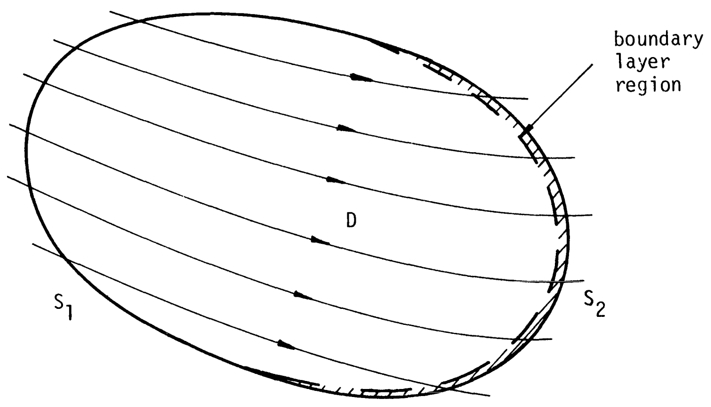


FIG. 1.1

Equation (1.2) is hyperbolic and, therefore, the solution $V(x, y)$ is oriented along the characteristic curves given by the solution of

$$\frac{dx}{a(x, y)} = \frac{dy}{b(x, y)}. \tag{1.4}$$

For well-posedness of the Cauchy problem for (1.2), S_1 must be that segment of the boundary along which the characteristic curves are incoming.

Furthermore, Eckhaus and de Jager [4] have shown that the boundary layer correction has different behaviors depending on whether the boundary curve is a characteristic curve. If the boundary curve is a characteristic curve, then the boundary layer term has a parabolic boundary layer behavior whose solution satisfies a parabolic equation

$$\alpha(\xi, \eta)\frac{\partial z}{\partial \xi} = \varepsilon \frac{\partial^2 z}{\partial \eta^2}$$

where ξ, η are tangential and normal coordinates along the boundary curve. If the boundary curve is not a characteristic curve, then the boundary layer correction has the form (1.3).

The above discussion suggests that the solution of the convection-diffusion equation for ε small has a directional property over most of the domain. The lower order partial derivative terms are mostly responsible for this behavior.

Motivated by the above discussion on the property of the solution, we generate an iterative scheme for the numerical solution of the convection-diffusion equation as follows. Set

$$M(u) = -\varepsilon \Delta u + \bar{a}u_x + \bar{b}u_y + \bar{b}u_y + \bar{c}u \tag{1.5}$$

and

$$N(u) = (a - \bar{a})u_x + (b - \bar{b})u_y + (c - \bar{c})u$$

where $\bar{a}$, $\bar{b}$ and $\bar{c}$ are functions that make $M(u) = f$, with appropriate boundary conditions, an easily solved boundary value problem. Here, the inclusion of the lower order terms in $M(u)$ is essential in developing an accurate and efficient numerical algorithm. The functions $\bar{a}$, $\bar{b}$ and $\bar{c}$ should be chosen to approximate a, b, c as closely as possible. In general $M(u)$ will not be selfadjoint.

Thus (1.1) becomes

$$M(u) = N(u) + f, \tag{1.6}$$

where $N(u)$ represents a measure of the deviation from the coefficients a, b and c over the domain D.

Let M_h and N_h be the discrete analogues of M and N in (1.6), and let v be the discrete approximation to u and let g be a discrete analogue of f plus boundary value information. The splitting

$$A_h = M_h - N_h$$

gives rises to the basic iterative scheme

$$M_h v^{(i+1)} = N_h v^{(i)} + g \tag{1.7}$$

with a dominant part M_h and a subdominant part N_h. The iterative matrix becomes

$$G = M_h^{-1} N_h.$$

This splitting is motivated by the behavior of the solution of the convection-diffusion equation when convection dominates. In contrast, the splitting of the matrix into a symmetric and a skew-symmetric parts only makes sense for problems when diffusion dominates.

The basic scheme (1.7) may be accelerated by a number of methods, for example, Chebyshev [14], two-part splitting [19], generalized conjugate gradient [2], orthomin [20], etc. We use either Chebyshev or two-part splitting as follows. To initialize, set

$$M_h \bar{v}^{(1)} = N_h v^{(0)} + g,$$
$$r^{(0)} = \bar{v}^{(1)} - v^{(0)},$$
$$\Delta^{(0)} = \alpha_0 r^{(0)},$$
$$v^{(1)} = v^{(0)} + \Delta^{(0)}.$$

Then, for $n > 1$, we set

$$M_h \bar{v}^{(n+1)} = N_h v^{(n)} + g, \tag{1.8a}$$

$$r^{(n+1)} = \bar{v}^{(n+1)} - v^{(n)}, \tag{1.8b}$$

$$\Delta^{(n+1)} = \alpha_n r^{(n)} + \beta_n \Delta^{(n-1)}, \tag{1.8c}$$

$$v^{(n+1)} = v^{(n)} + \Delta^{(n)}, \tag{1.8d}$$

where the parameters α_n and β_n are determined by the particular acceleration scheme (Chebyshev or two-part). In fact, the parameters may be estimated adaptively using Manteuffel's procedure [15].

Note that M_n will not, in general, be positive-definite symmetric due to the first order terms au_x and bu_y. However, M_h inherits the directional properties of the reduced (1.2). In solving (1.8a) for $v^{(n+1)}$ one should choose a method that exploits this directional dependence.

One such method is an alternating direction implicit (ADI) iterative scheme. Another method is a variant of ADI called the ADI/Gauss–Seidel (ADI/GS). This variation performs better than ADI for a certain range of the parameters. Both schemes may be accelerated by a Chebyshev semi-iterative method. Gourlay [5]–[6] has proposed and analyzed a Chebyshev semi-iterative method accelerating an ADI scheme when the matrix is symmetric and positive definite.

In § 2, we discuss the matrix splitting and the computation of the optimal parameters for the ADI method when $M(u)$ in (1.5) has constant coefficients and is defined on a rectangle. In § 3, we consider the acceleration of the ADI splitting by the Chebyshev semi-iterative method. In § 4, the ADI/Gauss–Seidel splitting is presented. Finally, we present in § 5 numerical results for (1.1) with variable coefficients. The results on optimal ADI parameters for a nonsymmetric matrix are apparently new, as is the four-parameter Chebyshev acceleration method discussed in § 3. The splitting of the matrix A_h into an easily solved nonsymmetric dominant part and subdominant part consistent with the directional properties of the solution of (1.1) has not appeared previously.

2. Matrix splitting. To facilitate the solution of (1.8a), the operator $M(u)$ in (1.5) can be split into two parts that correspond to the derivatives in the x direction and the y direction,

$$M(u)=H(u)+V(u). \tag{2.1}$$

A discrete analogue may be written

$$M_h=H+V.$$

This splitting gives rise to the iterative scheme known as Alternating-Direction-Implicit (ADI) (cf. Young [23]). We have

$$\begin{aligned}(\rho I+H)v^{i+1/2}&=(\rho I-V)v^{i}+g,\\ (\rho' I+V)v^{i+1}&=(\rho' I-H)v^{i+1/2}+g.\end{aligned} \tag{2.2}$$

Because of the structure of H and v, both equations in (2.2) are easily solved. The iterative matrix is

$$G=(\rho' I+V)^{-1}(\rho' I-H)(\rho I+H)^{-1}(\rho I-V). \tag{2.3}$$

If $M(u)$ has constant coefficients and D is rectangular, then H and V commute and the eigenvalues of G, say λ, may be found in terms of the eigenvalues of H and V. We have

$$\lambda_{ij}=\frac{\rho'-\mu_i}{\rho+\mu_i}\,\frac{\rho-\nu_j}{\rho'+\nu_j} \tag{2.4}$$

where μ_i is an eigenvalue of H and ν_j is an eigenvalue of V.

Suppose either Dirichlet or Neumann boundary conditions are prescribed. If (1.5) is discretized using 5-point central differencing, then the eigenvalues μ_i and ν_i will be the eigenvalues of tridiagonal matrices of the form

$$\begin{bmatrix} \hat{a} & \hat{b} & & \\ \hat{c} & \ddots & \ddots & \\ & \ddots & \ddots & \ddots \\ & & \ddots & \ddots \end{bmatrix}$$

where for H we have

$$\hat{a}=2+\frac{\bar{c}h^2}{2\varepsilon},\quad \hat{b}=-\left(1-\frac{\bar{a}h}{2\varepsilon}\right),\quad \hat{c}=-\left(1+\frac{\bar{a}h}{2\varepsilon}\right), \tag{2.5}$$

and for V we have

$$\hat{a}=2+\frac{\bar{c}h^2}{2\varepsilon},\quad \hat{b}=-\left(1-\frac{\bar{b}h}{2\varepsilon}\right),\quad \hat{c}=-\left(1+\frac{\bar{b}h}{2\varepsilon}\right).$$

If Neumann boundary conditions are used, then the first or last row may be altered, but the eigenvalues are not significantly changed. In fact, it can be shown that

$$\mu_i=2(1+\beta+\gamma\sqrt{1-R_x^2}),\qquad \nu_i=2(1+\beta+\alpha\sqrt{1-R_y^2})$$

for some $-1<\gamma<1$, $-1<\alpha<1$. Here $\beta=\bar{c}h^2/4\varepsilon$, and $R_x=\bar{a}h/2\varepsilon$ and $R_y=\bar{b}h/2\varepsilon$ are the half-grid Reynolds numbers. The exact intervals of γ and α depend upon boundary conditions and h.

Four cases arise depending on R_x and R_y. If $R_x>1$, $R_y<1$, then μ_i is complex with real part $2(1+\beta)$ and ν_i is real. If $R_x\leqq 1, R_y>1$ the roles of μ_i and ν_i are reversed. Finally, if $R_x>1$, $R_y>1$, then both μ_i and ν_i are complex with real part $2(1+\beta)$.

The optimal ρ, ρ' for the first case are known and will be given below for completeness. The optimal ρ and ρ' for the second case is derived here. The third case is found by reversing the roles of x and y. The final case is much more complicated and is not analyzed here.

Suppose $R_x<1$, $R_y<1$. Then the eigenvalues of G are all real. Suppose the eigenvalues of H and V are such that $q-r<\mu<q+r$, $s-t<\nu<s+t$. Let $S(G)$ be the spectral radius of G. The values of ρ and ρ' that minimize the spectral radius of G can be found in terms of q, r, s, and t (cf. Young [23, pp. 511–514]). The spectrum of G is contained in the interval

$$-\left(\frac{\kappa^{1/2}-1}{\kappa^{1/2}+1}\right)^2<\lambda<\left(\frac{\kappa^{1/2}-1}{\kappa^{1/2}+1}\right)^2, \tag{2.6}$$

where

$$\kappa=(1+\theta+\sqrt{(1+\theta)^2-1})\quad\text{and}\quad \theta=\frac{8rt}{(q+s)^2-(r+t)^2}.$$

The quantity κ is closely related to the condition of the matrices H and v.

Suppose $R_y<1<R_x$. The eigenvalues of V are real while the eigenvalues of H appear in complex conjugate pairs. Suppose the eigenvalues of H and V are contained in the regions

$$\mu=q+i\xi,\qquad -r<\xi<r,$$
$$\nu=s+\eta,\qquad -t<\eta<t.$$

The relationship between the spectrum of G and the spectrum of H and V is given by the map

$$\lambda(\rho, \rho'; \xi, \eta) = \frac{\mu - \rho'}{\mu + \rho} \frac{\nu - \rho}{\nu + \rho'}. \tag{2.7}$$

We would like to find ρ, ρ' to satisfy the minimax problem

$$\min_{\rho, \rho'} \max_{\substack{|\xi| \leq r \\ |\eta| \leq t}} |\lambda|^2. \tag{2.8}$$

The Möbius transformation $\zeta = (\mu - \rho')/(\mu + \rho)$ maps the vertical line $\mu = q + i\xi$, $-\infty < \xi < \infty$, in the complex μ-plane onto a circle centered at

$$c_0 = \frac{q + (\rho - \rho')/2}{q + \rho} \tag{2.9}$$

and with radius

$$r_0 = \frac{1}{2} \frac{\rho + \rho'}{q + \rho} \tag{2.10}$$

in the ζ-plane (cf. Kober [9]). Moreover as $|\mu| \to \infty$, $\zeta \to 1$ (see Fig. 2.1). Note that

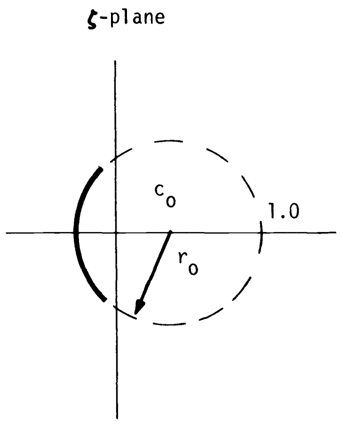


FIG. 2.1

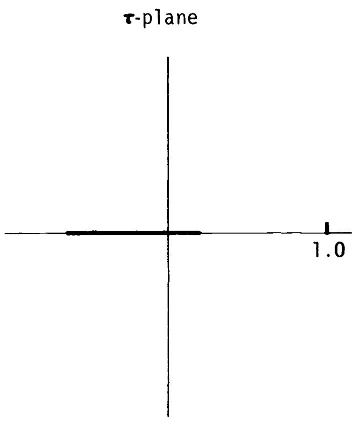


FIG. 2.2

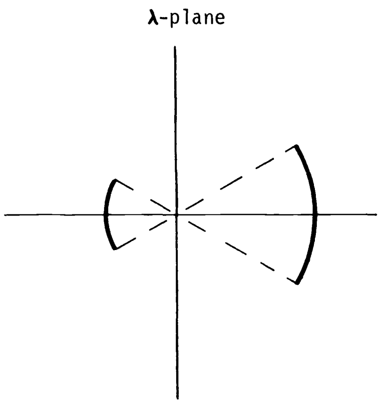


FIG. 2.3

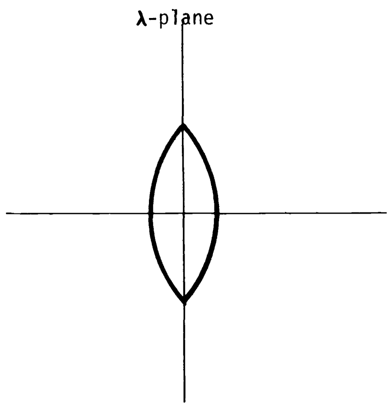


FIG. 2.4

we must have $\rho \neq -q$ or the radius is infinite. The portion of the circle with $-r<\xi<r$ is denoted by the heavier line. This arc is greater than a semicircle only if $\rho<r-q$.

The expression $\tau=(\nu-\rho)/(\nu+\rho')$ maps the line segment $\nu+s+i\eta$, $-t<\eta<t$, in the ν-plane onto the real interval (see Fig. 2.2)

$$\left[\frac{s-t-\rho}{s-t+\rho'}, \frac{s+t-\rho}{s+t+\rho'}\right] \tag{2.11}$$

in the τ-plane. Note that we must have $\rho'>-(s-t)$ or this interval will be infinite.

The region containing the spectrum of G is the cross product of the arc in Fig. 2.1 and the interval in Fig. 2.2. By choosing various values of ρ and ρ' many interesting shapes can be formed (see Fig. 2.3).

Let us rewrite $|\lambda|^2$ as

$$|\lambda|^2=\left(\frac{(q-\rho')^2+\xi^2r^2}{(q+\rho)^2+\xi^2r^2}\right)\left(\frac{s+\eta-\rho}{s+\eta+\rho'}\right)^2. \tag{2.12}$$

It can be shown by taking partial derivatives with respect to ξ and η that the point in the region with the largest modulus lies either on the midpoint or at the end of one of the bounding arcs (see Fig. 2.3). It is the end of the arc if the circle corresponding to Fig. 2.1 has radius less than 1, i.e. $c_0>0$. Notice that $c_0=0$ corresponds to

$$\rho'=\rho+2a. \tag{2.13}$$

The bounding arc with largest modulus is determined by which end of the segment (2.11) has largest absolute value. The interval is perfectly centered when

$$\rho'=\frac{s^2-t^2-s\rho}{\rho-s}. \tag{2.14}$$

The equations (2.13) and (2.14) divide the ρ, ρ' plane into four regions. The expression

$$\max_{\substack{|\xi|\leq r\\ |\eta|\leq t}} |\lambda|^2$$

has a unique expression in each region. Application of elementary calculus in each region and along the boundaries of the regions yields the solution to the minimax problem (2.8). If $r^2+t^2<(q+s)^2$, then the solution lies along the curve (2.14) at the intersection with curve

$$(q+\rho)(q-\rho')+r^2=0. \tag{2.15}$$

This yields

$$\begin{aligned} \rho&=-q+\sqrt{\frac{((s+q)^2-(r^2+t^2))^2}{4(s+q)^2}+r^2}+\frac{(s+q)^2-(r^2+t^2)}{2(s+q)},\\ \rho'&=q+\sqrt{\frac{((s+q)^2-(r^2+t^2))^2}{4(s+q)^2}+r^2}-\frac{(s+q)^2-(r^2+t^2)}{2(s+q)}. \end{aligned} \tag{2.16}$$

The curve (2.15) corresponds to the end of the arc in Fig. 2.1 having real part equal to zero. The curve (2.14) implies a symmetry around the imaginary axis. The spectrum of G is contained in a lens shaped region (see Fig. 2.4). The spectral radius is the height along the imaginary axis and is given by

$$S(G)=\left(\frac{t}{\rho'+s}\,\frac{r}{\rho+q}\right)^{1/2}. \tag{2.17}$$

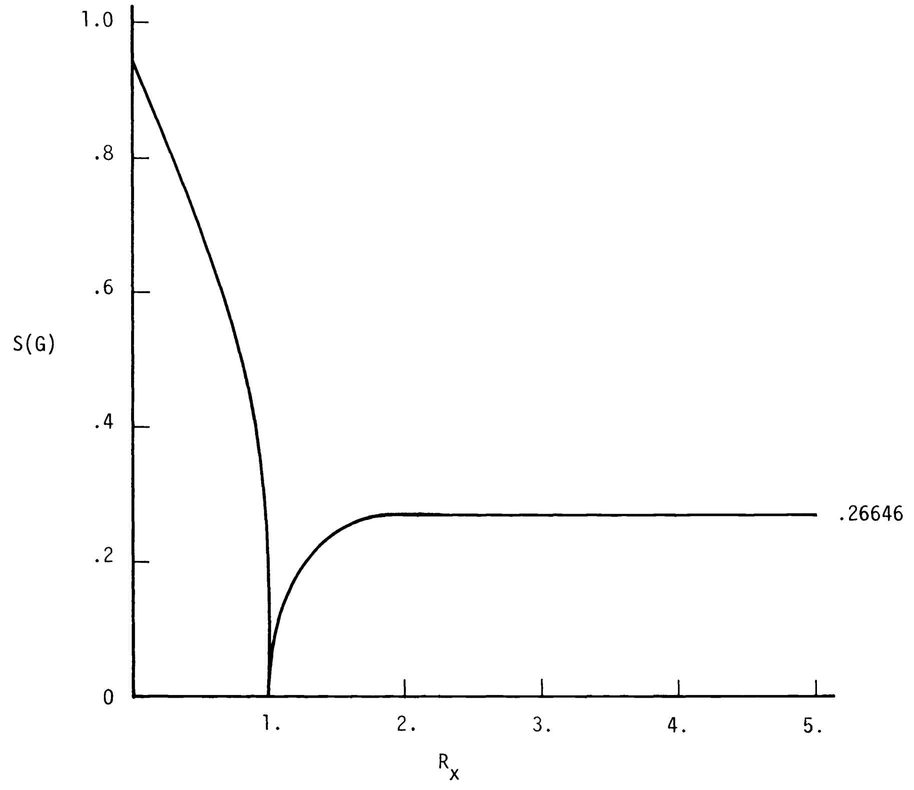


FIG. 2.5

If $r^2+t^2>(s+q)^2$, then the solution is the intersection between the curves (2.13) and (2.14). We have

$$\rho=-q+\sqrt{(s+q)^2-t^2}, \qquad \rho'=q+\sqrt{(s+q)^2-t^2} \tag{2.18}$$

and the spectral radius is

$$S(G)=\left[\frac{s+q}{t}\left(1-\sqrt{1-\left(\frac{t}{s+q}\right)^2}\right)\right]^{1/2}=\left(\frac{t}{\rho'+s}\right)^{1/2}. \tag{2.19}$$

In this case the region is a circle centered at the origin.

Suppose we fix $R_y=0$ and let R_x vary. Figure 2.5 shows the spectral radius of G, $S(G)$, with the optimal ρ, ρ' as a function of R_x. Notice that for $R_x=1$, that is, $r=0$ in (2.6) and (2.19), we have $S(G)\equiv 0$. However, $\lambda=0$ is an eigenvalue with nonlinear elementary divisors of high degree. This will be shown to be important later. Notice that for $r^2-t^2>(s+q)^2$, that is for R_x large, $S(G)$ is independent of R_x (see Fig. 2.5). Figure 2.6 shows the optimal values of ρ and ρ' as functions of R_x.

3. Acceleration of the ADI splitting. Consider the operator $I-G$. Since H and V commute, a bit of algebra yields

$$\begin{aligned}(I-G)&=(\rho+\rho')(\rho I+H)^{-1}(\rho' I+V)^{-1}(H+V)\\&=(\rho+\rho')(\rho I+H)^{-1}(\rho' I+V)^{-1}M_h.\end{aligned} \tag{3.1}$$

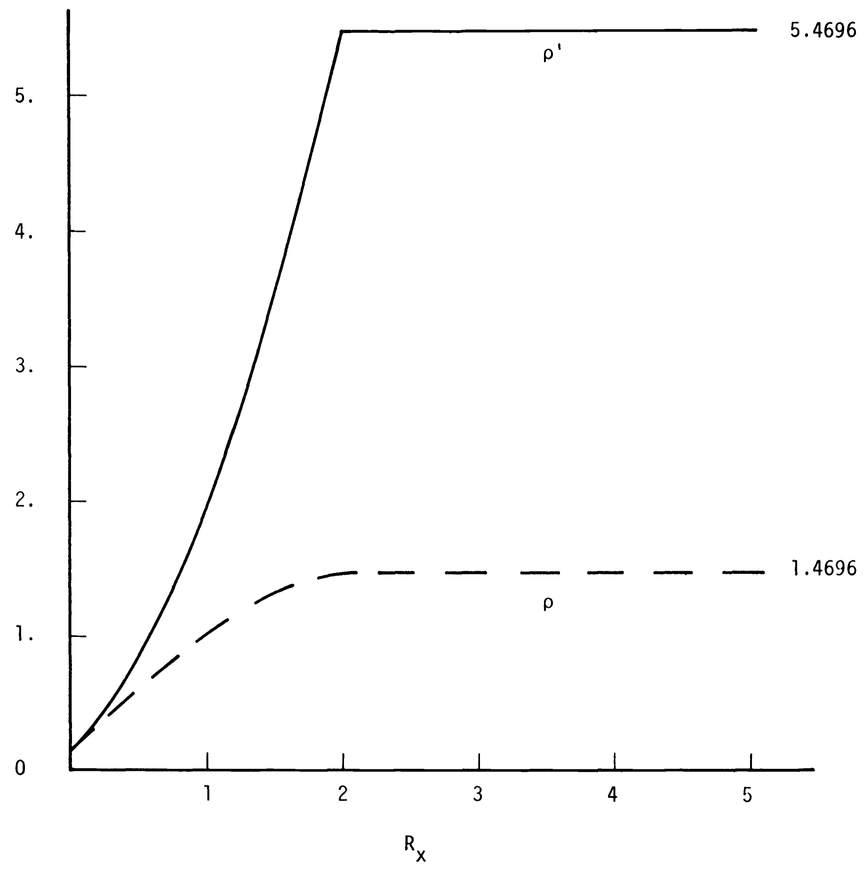


FIG. 2.6

We see that ADI as a preconditioning can be organized much as can other preconditionings such as SSOR [1], SIC [14], and MIC [7].

The ADI splitting can be accelerated by a Chebyshev iteration as described in (1.8). The parameters α_n, β_n are functions of two parameters d, e which are determined by the eigenvalues of the operator $I-G$ (cf. Manteuffel [14]). If the optimal ρ and ρ' for ADI iteration alone have been chosen, the spectrum of G, and thus $I-G$, can be calculated. Optimum d and e for this spectrum can be calculated as described in Manteuffel [15]. However, since it is the convergence of the total iteration (ADI accelerated by a Chebyshev iteration) that is of interest, we must consider choosing the four parameters ρ, ρ', d and r simultaneously. After some analysis to reduce the possible range of ρ, ρ', an optimization routine can be used to calculate the four-parameter optimum.

If $R_y < 1$, $R_x < 1$, that is, if H and V have real spectra, then the four-parameter optimal yields the same ρ, ρ' as in the ADI analysis. The optimal d, e are

$$(3.2) \qquad d = 1.0, \qquad e = \left(\frac{\kappa^{1/2}-1}{\kappa^{1/2}+1}\right)^2$$

and the convergence factor, the asymptotic factor of error reduction per step, is given by

$$\mathrm{cf} = \frac{(\kappa^{1/2}+\kappa^{-1/2})^{1/2}-2^{1/2}}{(\kappa^{1/2}+\kappa^{-1/2})^{1/2}+2^{1/2}}.$$

Notice that for $\kappa \sim 1$, $\mathrm{cf} \sim \frac{1}{2}S(G)$. For large κ, the number of iterations required to reduce the error by a factor of 10^{-N} is approximately $(N/4)\kappa^{1/2}$ for ADI alone, but is approximately $(N/2)\ \kappa^{1/4}$ for the combined iteration.

If $R_y < 1 < R_x$, then the four-parameter optimum yields ρ, ρ' that are not the same as those found for ADI alone. The spectrum of G for optimal ADI is shown in Fig. 2.4 to be a lens shape which becomes a circle as R_x increases. The spectrum of G produced by using the ρ, ρ' from the four-parameter optimum is shown in Fig. 3.1. Here as in Fig. 2.4 we have $R_x = 1.4$, $R_y = 0$ and $\beta = 0$. Notice that the ends are slightly crossed.

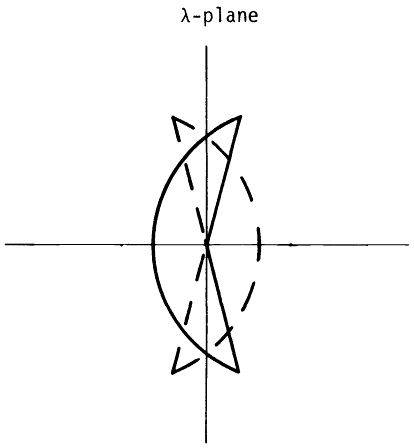


FIG. 3.1

The four-parameter optimal values for ρ, ρ' differ very little from the ADI optimal ρ, ρ' for the class of problems of interest here. Using the ADI optimal ρ, ρ' and then choosing the optimal d, e for these values gives a convergence factor that is slightly larger for R_x in the region of interest, that is, $0 < R_x < 2$. Figure 3.2 shows the convergence factor of the combined iteration and the spectral radius of ADI as a function of R_x. Here $R_y = 0$. Near $R_x = 1$, $cf \sim \frac{1}{2}\ S(G)$. For large R_x, say $R_x = 3$, $cf \sim S(G)$.

4. ADI/Gauss–Seidel splitting. The ADI splitting can be made more implicit by further splitting H and V. Let us divide H and V into lower triangular, upper triangular and diagonal parts,

$$H = L + D + U, \qquad V = L' + D' + U'.$$

As in (2.2) we may write

$$(4.1)\qquad \begin{aligned} (\rho I + H + L')v^{(i+1/2)} &= [\rho I - (D' + U')]v^{(i)} + g, \\ (\rho' I + V + L)v^{(i+1)} &= [\rho' I - (D + U)]v^{(i+1/2)} + g. \end{aligned}$$

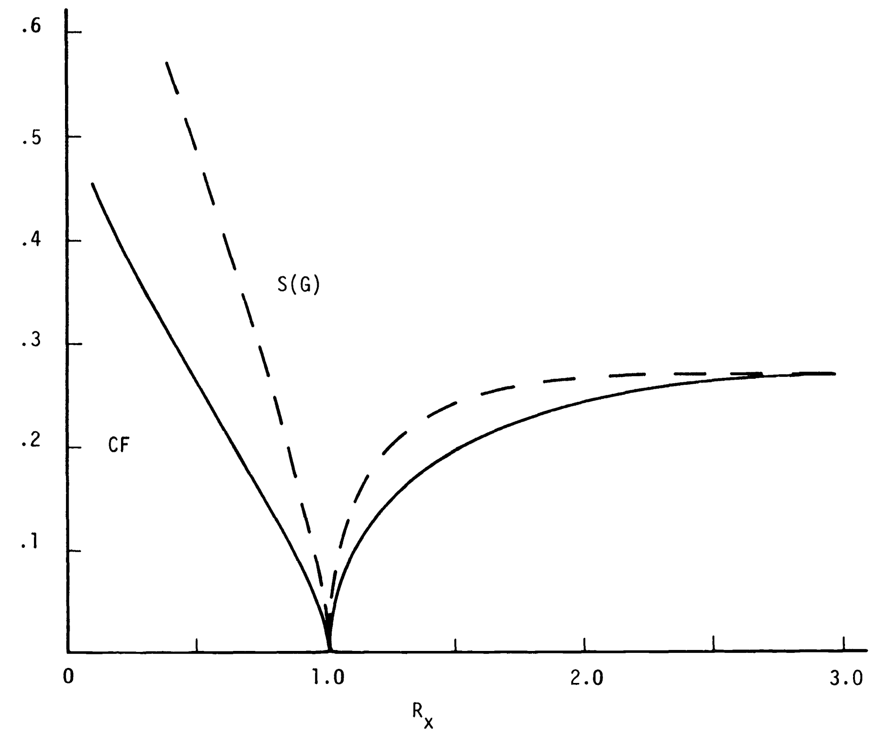


FIG. 3.2

Now we have

$$G=(\rho'I+V+L)^{-1}[\rho'I-(D+U)](\rho I+H+L')^{-1}[\rho I-(D'+U')]. \tag{4.2}$$

This splitting is very much like an alternating block SOR. Unfortunately, the terms do not commute, even for constant coefficients. Thus, the spectral properties of G defy analysis. Notice, however, that for $R_x=1.0$ we have $U\equiv 0$ (see (2.5)). Since $D=2I$, we may set $\rho'=2$ and then we have $G\equiv 0$. For R_x in the neighborhood of 1, G is nearly zero. Now for ADI and $R_x=1$ the spectral radius, $S(G)$, is zero, but $G\neq 0$. There is a high degree of nilpotency. This is reflected in tests that show that ADI requires one less iteration than the dimension of H to converge, whereas ADI/GS yields the exact solution in one iteration. The implicit L resolves the nilpotency of G. Figure 4.1 shows the number of iterations required to reduce the initial error by a factor of 10^{-10} for various values of R_x near 1.0 with $R_y=0$.

Since it is the second half of (4.1) that has nice properties ($U\equiv 0$) for R_x near 1 and $R_y=0$, one might consider making only the second step implicit. In Fig. 4.1, the middle curve shows the results of applying ADI on the first half of (4.1) and ADI/GS on the second half. As expected, the iteration is only slightly slower.

This splitting may also be accelerated by a Chebyshev iteration. It was found experimentally that using the parameters computed for the ADI/Cheb iteration worked well for the ADI/GS/Cheb iteration.

5. Numerical results with variable coefficients. In this section, we apply our algorithm to the solution of the convection-diffusion equation with a variable coefficient

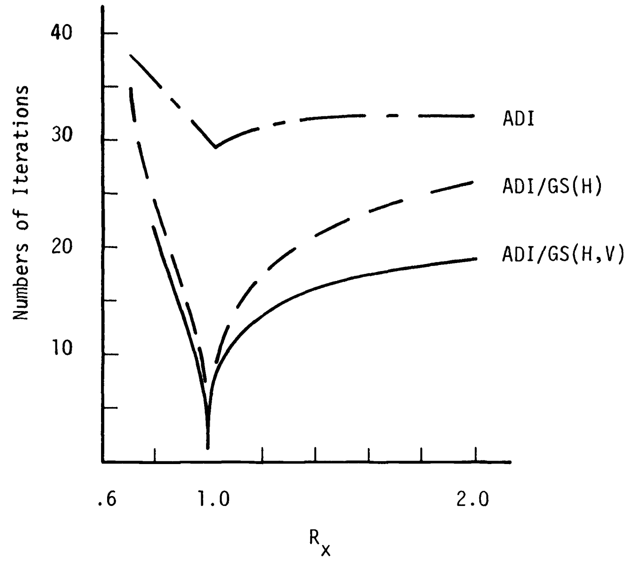


FIG. 4.1

$a(x, y)$ under the condition that

$$b(x, y)=c(x, y)=f(x, y)=0.$$

We consider two examples:

Case I. $\qquad a(x, y)=\delta y^n, \qquad 0<n<1.$

Case II. $\qquad a(x, y)=\delta y(1-y),$

on a unit square.

The boundary conditions along $y=0$, $x=0$, and $x=1$ are given by

$$u(x, 0)=0, \quad u(0, y)=1 \quad \text{and} \quad u_x(1, y)=0.$$

Along the top boundary $y=1$, we have for Case I

(5.1) $$u(x, 1)=1$$

and for Case II

(5.2) $$u(x, 1)=0.$$

The solution for Case I with boundary condition (5.1) has a developing parabolic boundary layer along $y=0$ and is, otherwise, nearly equal to unity. Lines parallel to the x-axis are characteristic curves.

The solution for Case II with boundary condition (5.2) has developing parabolic boundary layers along boundaries $y=0$ and $y=1$. The boundary layers can merge if the boundary layer development distance is sufficiently short. The characteristic curves of the reduced equations are once again lines parallel to the x-axis. Note that in both Cases I and II, an exponential boundary will form at $x=1$ if a Dirichlet boundary condition is imposed there.

The numerical calculations use second order difference approximations to the partial derivatives to develop the discrete analogues M_h and N_h on a 32×32 grid with equal mesh widths. The dominant part of the splitting, M_h, uses an average coefficient $\bar{a}$:

$$\bar{a} = \frac{\int_D a \, d\Omega}{\int_D d\Omega}.$$

We have for Case I

$$\bar{a} = \frac{\delta}{1+n}$$

and for Case II

$$\bar{a} = \frac{\delta}{6}.$$

It is clearly seen for $a(x, y) = \delta y^n$ that as n decreases toward 0 the region on which $\bar{a}$ is close to $a(x, y)$ increases. Therefore, we expect our matrix splitting to be more effective as n decreases. The computations are terminated when

$$\frac{\|r^{(n)}\|_2}{\|r^{(0)}\|_2} < 10^{-10}$$

where $r^{(n)} = g - A_h v^{(n)}$ is the residual vector.

We perform our computations with

$$.80 < R_x = \frac{\bar{a}h}{2\varepsilon} < 1.20.$$

The parameter $\bar{a}h/2\varepsilon$ is the half-grid Reynolds number and measures the relative importance of the convective to the diffusive effects. For this range of R_x the numerical solution would be reasonably accurate and yet efficient with ε small if a nonoutflow boundary condition were imposed at the exit boundary (see Hedstrom and Osterheld [8]). For a developing parabolic boundary layer solution for which the outflow boundary condition is prescribed, Hedstrom and Osterheld find that the numerical solution is well approximated by the following modified equation:

$$\bar{a}u_x = \varepsilon u_{yy} + \varepsilon\left(1 + \frac{R_x^2}{3}\right)u_{xx}.$$

In the parabolic boundary layer region, we have

$$\varepsilon u_{yy} = O(1), \quad u_x = O(1) \quad \text{and} \quad u_{xx} = O(1).$$

It follows that the numerical solution is accurate even for sufficiently large half-grid Reynolds number. Hedstrom and Osterheld find that small discernible differences between the exact and the numerical solution occur for $R_x = 20$.

In all of the calculations, one ADI/GS/Cheb inner iteration per outer iteration is used. This is suggested by numerical experiments using more than one ADI/GS/Cheb iteration per outer iteration. For example, it is found that a 10–15% increase in convergence rate is obtained with three inner iterations per outer iteration. However, the relative cost of an inner iteration negates the increase in speed of the outer iteration.

To assess the effectiveness of splitting $A(u)$ into a dominant part involving the average of the coefficient and a subdominant part involving the deviation from the average of the coefficient, we plot in Figs. 5.1–5.3 the convex hull of the approximate

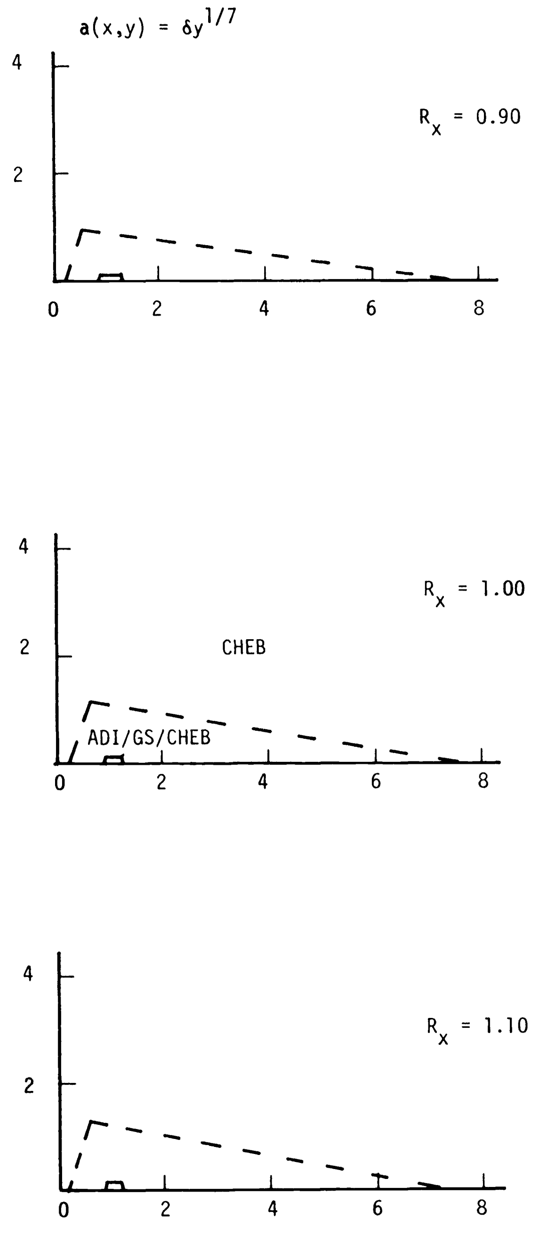


FIG. 5.1

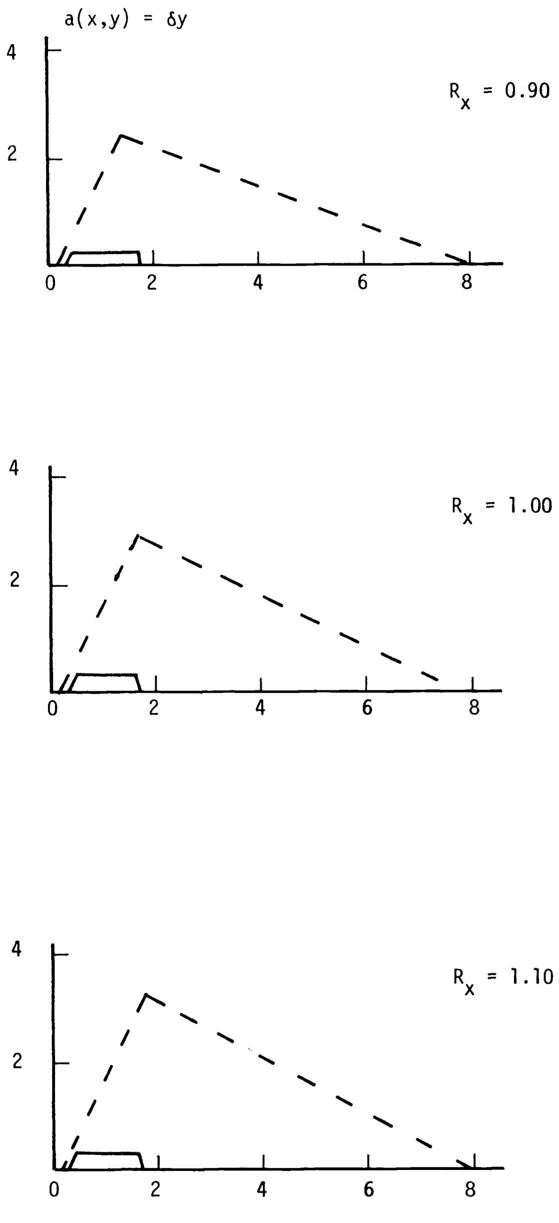


FIG. 5.2

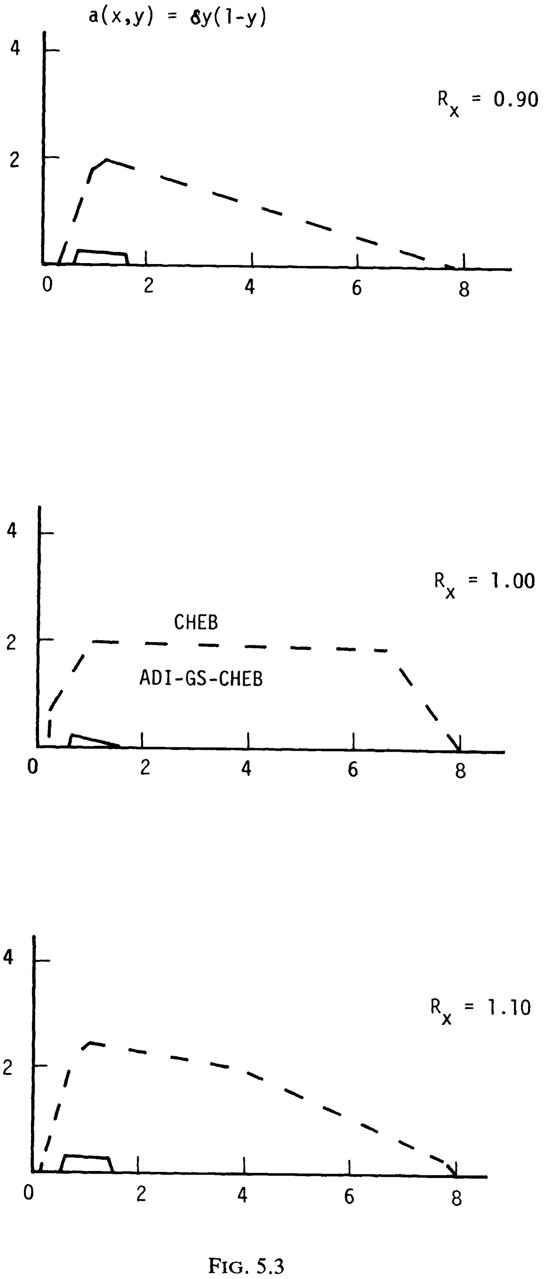


FIG. 5.3

spectrum of A_n and $M^{-1}A_n$ as computed by the TCHEB program (cf. Manteuffel [15]) for $a(x, y) = \delta y^{1/7}$, $a(x, y) = \delta y$, and $a(x, y) = \delta y(1-y)$ and for values of R_x near 1. In each case, the convex hull of the approximate spectrum using M_h as a preconditioning is close to 1.0. This implies that the operator $M_h^{-1}A_h$ provides a "good" approximation to the identity operator.

This preconditioning is most effective when $a(x, y)$ can be closely approximated by a constant. In Fig. 5.4 the number of iterations required for convergence is plotted for values of R_x for the three choices of $a(x, y)$ listed above. The results confirm our expectations that the better $\bar{a}$ approximates a, the more rapid the convergence rate is.

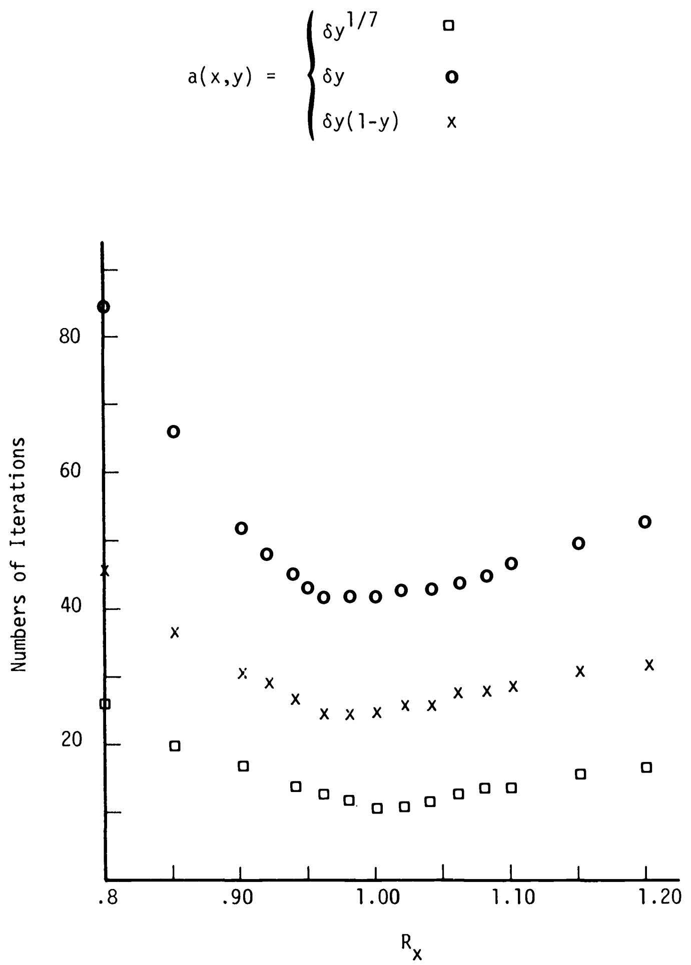


FIG. 5.4

To compare with other presently used methods of preconditioning, we plot in Figs. 5.5 and 5.6 the convergence rate in numbers of Chebyshev iterations versus R_x for the SIC, SSOR and MIC methods. Each of these preconditionings has comparable work requirements. It is seen that ADI preconditioning is most effective for $a(x, y) = \delta y^{1/7}$ and becomes less effective as the exponent increases. Common to all of these

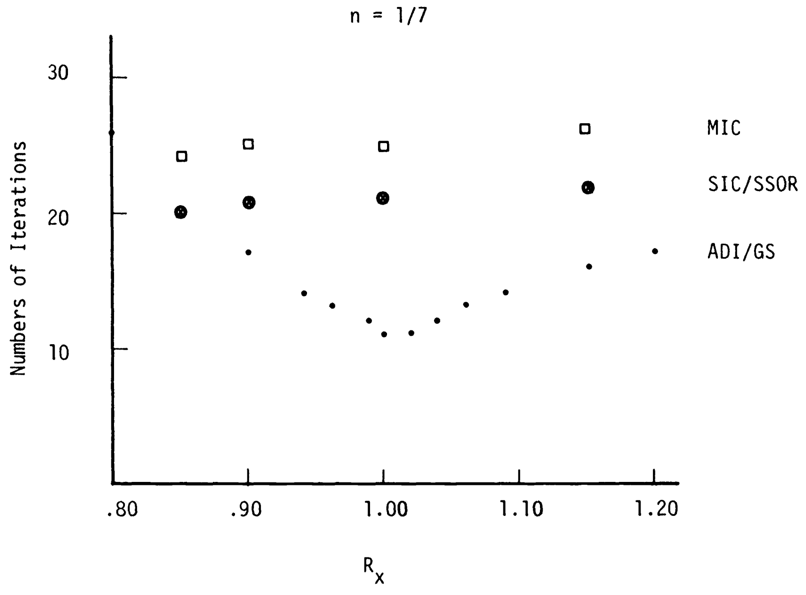

n = 1/7
Numbers of Iterations
30
20
10
.80
.90
1.00
1.10
1.20
R_x
MIC
SIC/SSOR
ADI/GS

FIG. 5.5

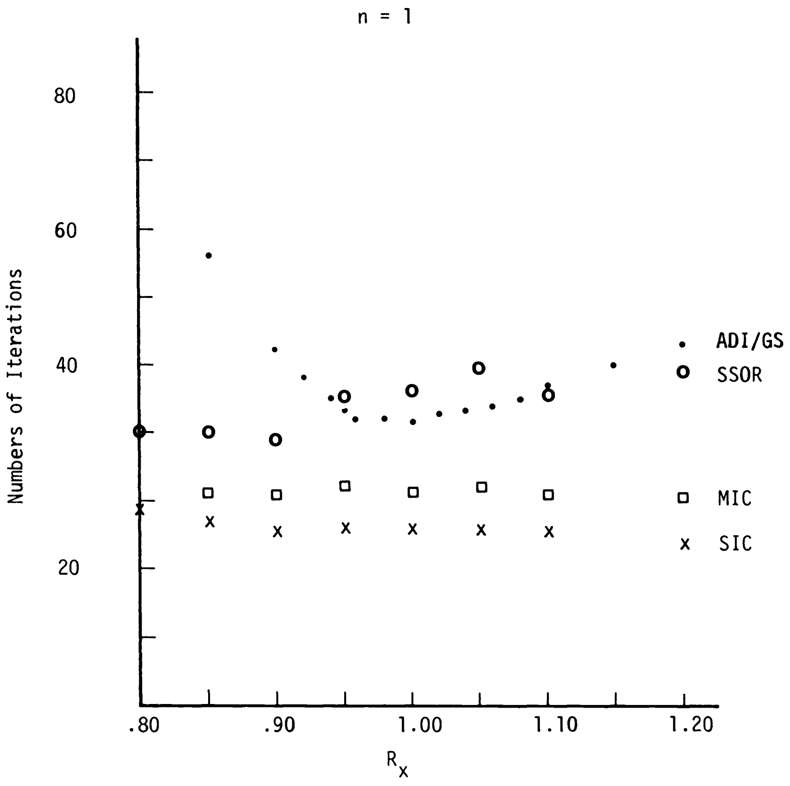

n = 1
Numbers of Iterations
80
60
40
20
.80
.90
1.00
1.10
1.20
R_x
ADI/GS
SSOR
MIC
SIC

FIG. 5.6

preconditioning methods is the retention of the directional property of the solution by incorporating the skew-symmetric part into the preconditioning matrix.

Acknowledgments. We thank Gerald Hedstrom for helpful discussions and Vance Faber for his proclivity for voluminous algebraic calculation.

REFERENCES

[1] O. AXELSSON, *On preconditioning and convergence acceleration in sparse matrix problems*, Report CERN 74-10, Data Handling Division, Laboratory I, CERN European Organization for Nuclear Research, 8 May 1974.

[2] P. CONCUS, G. H. GOLUB AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, Publ. LBL-4604, Lawrence Berkeley Laboratory, Berkeley, CA, 1975.

[3] W. ECKHAUS, *Boundary layers in linear elliptic singular perturbation problems*, SIAM Rev., 14 (1972), pp. 225–270.

[4] W. ECKHAUS AND E. M. DEJAGER, *Asymptotic solutions of singular perturbation problems for linear differential equations of elliptic type*, Arch. Rational Mech. Anal., 23 (1966), pp. 26–86.

[5] A. R. GOURLAY, *The acceleration of the Peaceman–Rachford method by Chebyshev polynomials*, Comput. J., 10 (1968), pp. 378–382.

[6] ———, *On Chebyshev acceleration procedure for alternating direction iterative methods*, J. Inst. Math. Appl., 6 (1970), pp. 1–11.

[7] I. GUSTAFSSON, *A class of first order factorizations*, BIT, 18 (1978), pp. 142–156.

[8] G. W. HEDSTROM AND A. OSTERHELD, *The effect of cell Reynolds number on the computation of a boundary layer*, J. Comput. Phys., 37 (1980), pp. 399–421.

[9] H. KOBER, *Dictionary of Conformal Representations*, Dover, New York, 1957.

[10] F. A. HOWES, *Singularly perturbed semilinear elliptic boundary value problems*, Comm. Partial Differential Equations, 4 (1979), pp. 1–39.

[11] ———, *Some singularly perturbed nonlinear boundary value problems of elliptic type*, in Proc. Conference on Nonlinear Partial Differential Equations in Applied Science and Engineering, Steinberg et al., eds., Marcel Dekker, New York, 1980.

[12] ———, *Perturbed boundary value problems whose reduced solutions are nonsmooth*, Indiana Univ. Math. J., 30 (1981), pp. 267–280.

[13] N. LEVINSON, *The first boundary value problem for $\varepsilon\Delta u + A(x, y)u_x + B(x, y)u_y + C + (x, y)u = D(x, y)$ for small ε*, Ann. of Math., 51 (1950), pp. 428–445.

[14] T. A. MANTEUFFEL, *The Tchebyschev iterations for nonsymmetric linear systems*, Numer. Math., 28 (1977), pp. 307–327.

[15] ———, *Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration*, Numer. Math., 31 (1978), pp. 183–208.

[16] ———, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473–497.

[17] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[18] R. E. O'MALLEY, JR., *Topics in singular perturbations*, Adv. in Math., 2 (1968), pp. 365–470.

[19] J. DE PILLIS, *How to embrace your spectrum for faster iterative results*, Linear Algebra Appl., 34 (1980), pp. 125–143.

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

[21] M. I. VISHIK AND L. A. LIUSTERNIK, *Regular degeneration and boundary layers for linear differential equations with small parameters*, Amer. Math. Soc. Transl. Ser. 2, 20 (1961), pp. 239–364.

[22] W. WASOW, *Asymptotic solution of boundary value problems for the differential equation $\Delta U + \lambda(\partial U/\partial x) = \lambda f(x, y)$*, Duke Math. J., 11 (1944), pp. 405–415.

[23] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

A METHOD FOR CONSTRUCTING LOCAL MONOTONE PIECEWISE CUBIC INTERPOLANTS*

F. N. FRITSCH† AND J. BUTLAND‡

Abstract. A method is described for producing monotone piecewise cubic interpolants to monotone data which is completely local and which is extremely simple to implement.

1980 AMS mathematics subject classifications. Primary 65D05, secondary 65D07

Key words. monotone interpolation, cubic Hermite interpolation

In [4] Fritsch and Carlson gave necessary and sufficient conditions for a piecewise cubic interpolant to monotone data to be monotonic. The algorithm which they proposed for computing such an interpolant suffers from three defects: (1) it requires two passes over the data; (2) the result is dependent upon the order in which the data are processed; and (3) it is potentially nonlocal (i.e., a correction introduced in the first interval might ripple through the entire interpolant).

The purpose of this note is to acquaint the mathematical community with a technique proposed by Butland [2] for obtaining monotone piecewise cubic interpolants which avoids all of these problems. Before we describe Butland's method, we recall some notation from [4]. Let $x_1 < x_2 < \cdots < x_n$ $(n > 2)$ and let $f_i = f(x_i)$ $i = 1, \cdots, n$ be the values of a monotone function at these points. Let $p(x)$ be a piecewise cubic function such that $p(x_i) = f_i$ and $p'(x_i) = d_i$, $i = 1, \cdots, n$. Let $\Delta_i = (f_{i+1} - f_i)/(x_{i+1} - x_i)$, $i = 1, \cdots, n-1$.

A piecewise cubic interpolation scheme is a method for selecting the values of the derivatives d_i. Butland's idea is to construct a function G such that

$$d_i = G(\Delta_{i-1}, \Delta_i), \qquad i = 2, \cdots, n-1, \tag{1}$$

and $p(x)$ is monotonic. Since d_i depends only on neighboring slopes, a method based on formula (1) is necessarily one-pass and local. If G is a symmetric function of its arguments, the result will also be independent of the direction of processing, as desired.

Sufficient conditions for an acceptable G-function are given in [2], where Butland observes that the harmonic mean

$$G_H(S_1, S_2) = \begin{cases} \dfrac{2S_1S_2}{S_1 + S_2} & \text{if } S_1S_2 > 0, \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

satisfies all of these conditions. Unfortunately, formula (2) tends to produce interpolants that are "too flat", because the values $(\alpha_i, \beta_i) = (d_i/\Delta_i, d_{i+1}/\Delta_i)$ are restricted to the small square $[0, 2] \times [0, 2]$, contained in the monotonicity region. We have experimented with various G-functions that fill out the square[1] $[0, 3] \times [0, 3]$ more completely, thus producing more "reasonable" curves. One such family of functions

* Received by the editors February 25, 1983.

† Mathematics and Statistics Division, Lawrence Livermore National Laboratory, Livermore, California 94550. The work of this author was supported by the U.S. Department of Energy under contract W-7405-Eng-48 and by its Office of Basic Energy Sciences, Applied Mathematical Sciences Program.

‡ Computer Officer, Postgraduate School of Electrical and Electronic Engineering, University of Bradford, England BD7 1DP.

[1] This is the largest square contained in the monotonicity region.

is described in [3], where the recommended formula is

$$(3)\qquad G_3(S_1, S_2) = \begin{cases} 0 & \text{if } S_1S_2 \leqq 0, \\ \operatorname{sign}(S_1)\dfrac{3|S_1||S_2|}{|S_1|+2|S_2|} & \text{if } |S_2| \leqq |S_1|, \\ G(S_2, S_1) & \text{otherwise.} \end{cases}$$

One negative aspect of (1) is that it gives the same value for d_i regardless of the relative spacing of the surrounding x-values. One way to remedy this is to replace (1) with

$$(4)\qquad d_i = G(\Delta_{i-1}, \Delta_i, h_{i-1}, h_i), \qquad i = 1, \cdots, n-1,$$

where $h_j = x_{j+1} - x_j$, $j = 1, \cdots, n-1$. The conditions for an acceptable G-function now are that, for all S_1, S_2, and all positive h_1 and h_2:

A. $$G(S_2, S_1, h_2, h_1) = G(S_1, S_2, h_1, h_2).$$

[This makes the formula independent of the order of the data points.]

B. $$\min(S_1, S_2) \leqq G(S_1, S_2, h_1, h_2) \leqq \max(S_1, S_2).$$

[Thus the slope of the curve lies between the slopes of the two adjacent data segments. While not necessary for monotonicity, this condition seems intuitively reasonable.]

C. $$G(S_1, S_2, h_1, h_2) = 0 \text{ if } S_1S_2 \leqq 0.$$

[For $S_1S_2 = 0$, this is a necessary condition for monotonicity. For $S_1S_2 < 0$, this implies that the curve stays within the limits of the data.]

D. $$|G(S_1, S_2, h_1, h_2)| \leqq \min(3|S_1|, 3|S_2|).$$

[This insures that $(\alpha_i, \beta_i) = (d_i/\Delta_i, d_{i+1}/\Delta_i)$ lies inside the square $[0, 3]\times[0, 3]$.]

When coupled with appropriate boundary conditions, (4) with any acceptable G-function gives a monotone piecewise cubic interpolant to the given data which has all the properties we desire.

In the discussion of [1], Brodlie proposed a formula which can be written in the form of (4) with

$$(5)\qquad G(S_1, S_2, h_1, h_2) = \begin{cases} \dfrac{S_1S_2}{\alpha S_2 + (1-\alpha)S_1} & \text{if } S_1S_2 > 0, \\ 0 & \text{otherwise,} \end{cases}$$

where

$$\alpha = \frac{1}{3}\left(1 + \frac{h_2}{h_1+h_2}\right) = \frac{h_1+2h_2}{3(h_1+h_2)}.$$

We observe that this G-function satisfies conditions A–D and it reduces to (2) when $h_1 = h_2$.

Much experimentation indicates that (4) and (5), when coupled with the boundary conditions in either [2] or [4], generally produce interpolants that are at least as "visually pleasing" as (1) and (3). Furthermore, it can be shown that for uniformly spaced data (5) gives an $O(h^2)$ approximation to $f'(x_i)$, whereas (3) is only $O(h)$.

In Figs. 1–4, we exhibit the curves produced by the four methods discussed here when applied to the data set Akima 3 of [4]. We conclude that the technique described

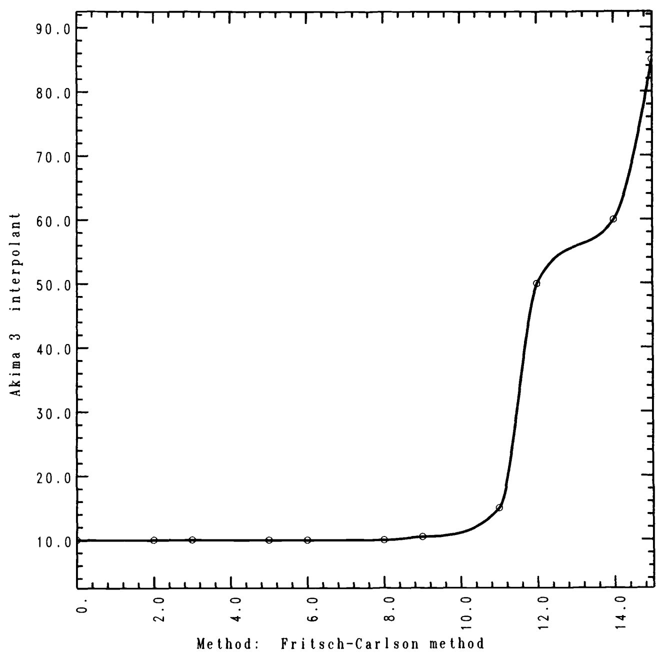


FIG. 1. *Result from the algorithm proposed in* [4] *when applied to the* Akima 3 *data.* (*This and all following examples used boundary derivatives computed by the standard noncentral three-point formula.*)

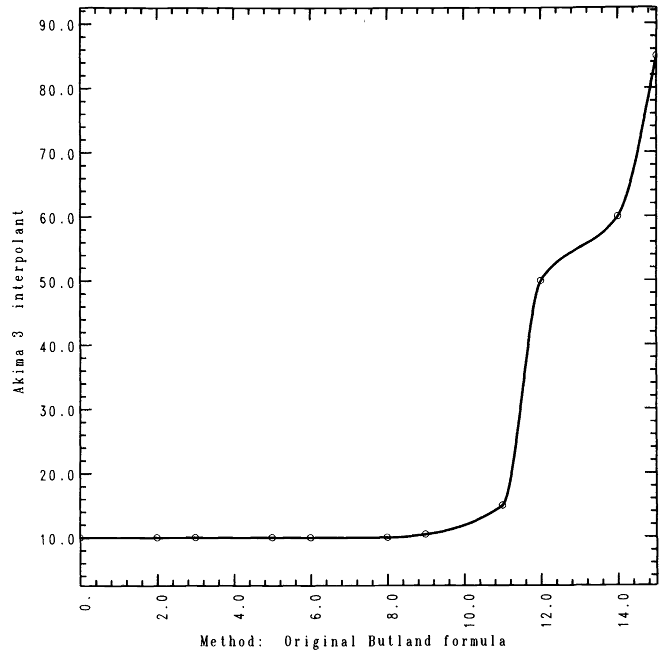


FIG. 2. *Result of applying formulas* (1) *and* (2) *to the* Akima 3 *data.*

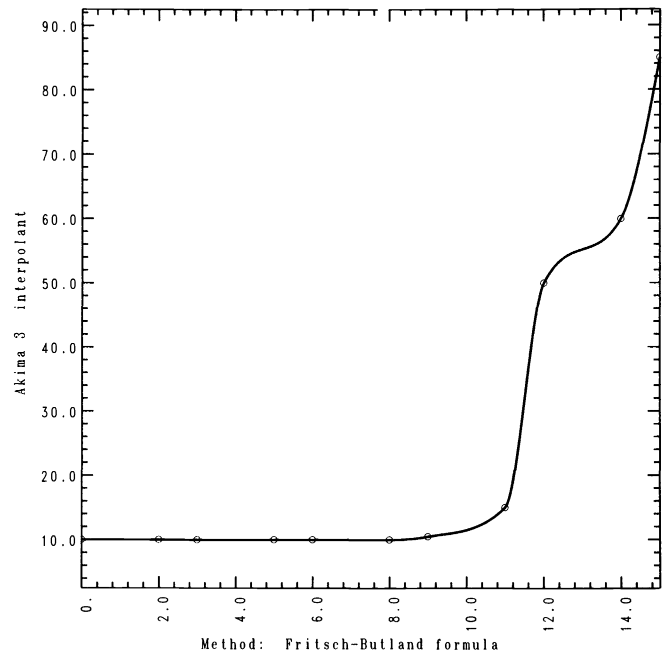


FIG. 3. *Result of applying formulas* (1) *and* (3) *to the* Akima 3 *data.*

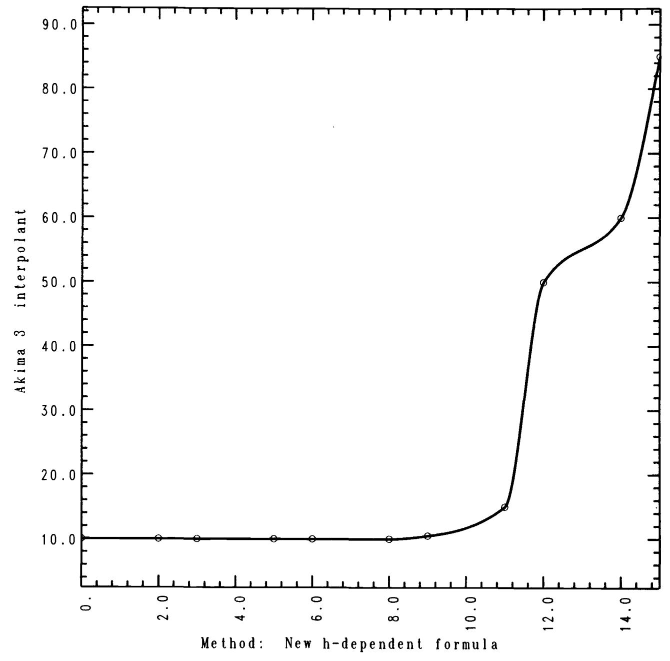


FIG. 4. *Result of applying formulas* (4) *and* (5) *to the* Akima 3 *data.*

here leads to a method for computing monotone piecewise cubic interpolants which is simple, symmetrical, and completely local. We remark that the method also produces reasonable results when applied to piecewise monotone data. Software implementing this algorithm may be obtained by writing to the first author.

Acknowledgment. The first author wishes to thank R. E. Carlson and J. M. Hyman for valuable discussions during the course of this work.

REFERENCES

[1] K. W. BRODLIE, *A review of methods for curve and function drawing*, in Mathematical Methods in Computer Graphics and Design, K. W. Brodlie, ed., Academic Press, London, 1980, pp. 1–37.

[2] J. BUTLAND, *A method of interpolating reasonable-shaped curves through any data*, Proc. of Computer Graphics 80, Online Publications Ltd., Northwood Hills, Middlesex, UK, 1980, pp. 409–422.

[3] F. N. FRITSCH AND J. BUTLAND, *An improved monotone piecewise cubic interpolation algorithm*, Rep. UCRL-85104, Lawrence Livermore Laboratory, Livermore, CA, October 1980.

[4] F. N. FRITSCH AND R. E. CARLSON, *Monotone piecewise cubic interpolation*, SIAM J. Numer. Anal., 17 (1980), pp. 238–246.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

QUASICONFORMAL MAPPINGS AND GRID GENERATION*

C. W. MASTIN† AND J. F. THOMPSON‡

Abstract. A finite difference scheme is developed for constructing quasiconformal mappings for arbitrary simply and doubly-connected regions of the plane. Computational grids are then generated to reduce elliptic equations to canonical form. Examples of conformal mappings on two-dimensional surfaces are also included.

Key words. quasiconformal mapping, grid generation, elliptic equation

1. Introduction. The motivation for constructing quasiconformal mappings lies in their application to the generation of curvilinear coordinate systems for two-dimensional regions. Quasiconformal mappings may be used to reduce any second order linear elliptic partial differential equation to canonical form (i.e., the principal part of the differential operator reduces to the Laplacian). Consequently, if one is solving an elliptic boundary value problem, an appropriate quasiconformal mapping could be used simultaneously to fit the boundary contours with coordinate lines and simplify the original partial differential equation. The equation, in canonical form, could possibly be solved more efficiently or by methods which would not be applicable to the original equation in cartesian coordinates. A related application is in the construction of conformal mappings on two-dimensional surfaces.

Quasiconformal mappings have been studied extensively by complex analysts and even generalized to higher dimensions. However, very little work has been done on the numerical construction of quasiconformal mappings. Of the methods which have been proposed, that of Belinskii et al. [2] uses a fixed boundary correspondence which determines the mapping parameters, and the method of Mastin and Thompson [4] would be difficult to implement on arbitrary regions. A finite element version of the latter method developed by Weisel [9] appears promising. However, the class of mappings and the type of regions presented in the examples are very limited. In recent years a finite-difference method for constructing conformal mappings developed by Allen [1] has been used by Mobley and Stewart [5], Pope [6], and Yen and Lee [8] in the construction of orthogonal coordinate systems. Although all of these authors use essentially the same numerical method, there are differences in the way the boundary values and the conformal module of the region are computed. This method is not as accurate or efficient as other conformal mapping methods using integral equations or series expansions, but it does have the advantage of simplicity since the module of the region, the boundary correspondence, and the interior grid points are determined in a single iterative procedure. Modeling this conformal mapping procedure, it will be shown how quasiconformal mappings can be constructed and applied to the reduction of elliptic equations to canonical form and the construction of conformal mappings on surfaces. Except for the method of Godunov and Prokopov [3], this appears to be the only conformal mapping method which can be easily adapted to handle the problem of constructing quasiconformal mappings.

* Received by the editors June 11, 1982, and in revised form January 11, 1983. This research was sponsored by NASA Langley Research Center under grant NSG 1577.

† Department of Mathematics and Statistics, Mississippi State University, Mississippi State, Mississippi 39762.

‡ Department of Aerospace Engineering, Mississippi State University, Mississippi State, Mississippi 39762.

2. Boundary value problem for quasiconformal mappings. Let D be a bounded simply-connected region in the xy-plane whose boundary C is a simple closed contour. Let z_1, z_2, z_3 and z_4 be distinct boundary points ordered by the orientation on C. There exists a unique quasiconformal mapping of D onto the interior of a rectangle such that the points z_i map to the vertices which are also ordered by the orientation of the rectangle. The ratio of the length to the width of the rectangle is a quasiconformal invariant called the module of D and will be denoted by m. The quasiconformal mapping of the region D onto a rectangular region can be obtained by constructing the mapping of D onto a square region S which satisfies the linear system

$$\xi_x = m(c\eta_y + b\eta_x),\qquad \xi_y = -m(a\eta_x + b\eta_y), \tag{1}$$

where $ac - b^2 = 1$. On setting $(\mu, \nu) = (\xi, m\eta)$, it is obvious that μ and ν, as functions of x and y, satisfy the Beltrami system, and hence we arrive at the desired quasiconformal mapping (see [4] for further details).

It is easily shown that x and y, as functions of ξ and η, satisfy

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = J^2(a_x + b_y),\qquad \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = J^2(b_x + c_y), \tag{2}$$

where

$$\alpha = ay_\eta^2 - 2bx_\eta y_\eta + cx_\eta^2,$$
$$\beta = ay_\xi y_\eta - b(x_\xi y_\eta + x_\eta y_\xi) + cx_\xi x_\eta,$$
$$\gamma = ay_\xi^2 - 2bx_\xi y_\xi + cx_\xi^2,$$
$$J = x_\xi y_\eta - x_\eta y_\xi.$$

At each boundary point of D, one of the functions, ξ or η, is constant while the other satisfies an oblique derivative condition. This implies that the condition $\beta = 0$ must be satisfied by x and y at each boundary point of S (except for vertices). Note that in the solution of (1) we would have $\beta = 0$ throughout S. Thus the computed value of β can serve as a test of the accuracy of our solution. It also follows from (1) that $m = \sqrt{\alpha/\gamma}$. Therefore, the equations (2) can be written as

$$m^2 x_{\xi\xi} + x_{\eta\eta} = mJ(a_x + b_y),\qquad m^2 y_{\xi\xi} + y_{\eta\eta} = mJ(b_x + c_y). \tag{3}$$

The same procedure is used for doubly-connected regions with a periodicity condition applied on two opposite sides of S.

The system (2) or (3) may be solved using an iterative procedure with (i) the right-hand sides of the equations, (ii) the boundary values of x and y, and (iii) either α, β, γ, or an approximation of m, re-evaluated at each iteration. As in the previously discussed conformal mapping methods, both (2) and (3) have performed equally well in numerical examples.

3. Reduction of elliptic equations to canonical form. The application of quasiconformal mappings in the solution of elliptic equations is well known. An elliptic equation of the form

$$au_{xx} + 2bu_{xy} + cu_{yy} = f(u, u_x, u_y), \qquad ac - b^2 = 1$$

transforms to an equation of the form

$$m^2u_{\xi\xi}+u_{\eta\eta}=g(u, u_\xi, u_\eta)$$

under the transformation defined by (1). This transformation will make the solution of many problems much easier. For example, if a, b, and c are constants and $f=0$, then it can be shown that $g=0$ and we need only solve $m^2u_{\xi\xi}+u_{\eta\eta}=0$ on a square region. This is efficiently done by separation of variables or a direct numerical method. Computational grids for solving this problem on a simply and doubly-connected region are given in Fig. 1.

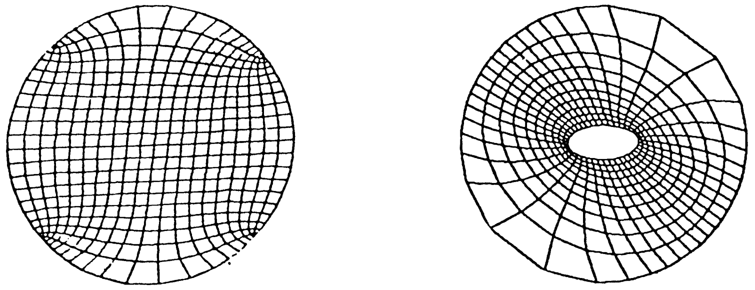


FIG. 1. *Quasiconformal grids for simply and doubly-connected regions.* $a=1, b=\frac{1}{2}, c=\frac{5}{4}$.

The practicality of using quasiconformal mappings in solving elliptic equations will be further examined in the following example. The function $u=\cos(x-y)$ is a solution of the partial differential equation

$$u_{xx}+u_{xy}+u_{yy}+\cos(x-y)=0. \tag{4}$$

We will solve this equation numerically for $0\leqq x, y\leqq \pi$ with Dirichlet boundary conditions prescribed by the known solution. In terms of curvilinear coordinates, this equation can also be written as

$$m^2u_{\xi\xi}+u_{\eta\eta}+\frac{2Jm}{\sqrt{3}}\cos(x-y)=0. \tag{5}$$

The quasiconformal grid for solving (5) is illustrated in Fig. 2. For the purpose of assessing the influence of the error in the iterative solution of (2) on the error in the solution of (5), the iteration was stopped occasionally and the solution of (5) was computed. A comparison of a normalized value of β with the error in the solution of (5) is plotted in Fig. 3. Note that β has been normalized so that for the construction

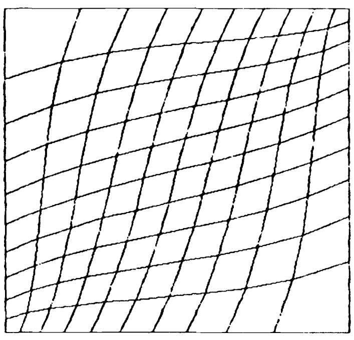


FIG. 2. *Quasiconformal grid for the solution of elliptic equation.* $a=c=2/\sqrt{3}$, $b=1/\sqrt{3}$.

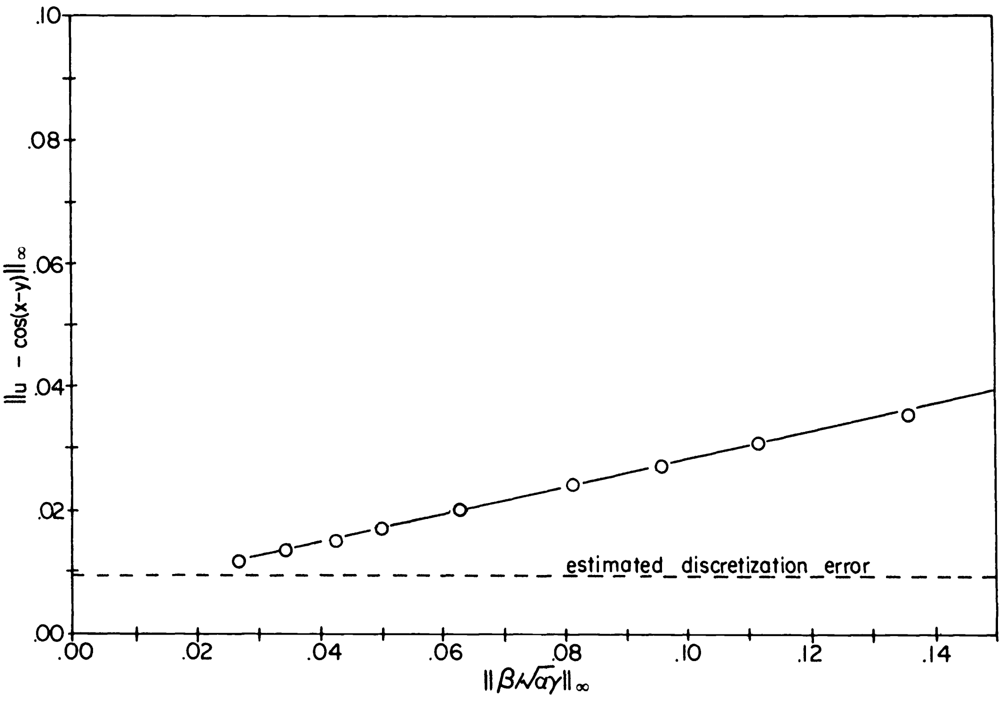


FIG. 3. *Effect of error in the quasiconformal mapping on error in the solution of the partial differential equation.*

of conformal mappings ($a = c = 1, b = 0$), the values along the abscissa would represent the degree of nonorthogonality. Equation (4) was also solved on a uniform rectangular mesh with the same number of grid points. The error in this solution serves as an approximation of the discretization error which would result in solving (5) with the exact quasiconformal mapping. It also serves as a test of our method against the traditional method for solving (4). Fig. 3 indicates a nearly linear relation between the plotted variables. This is to be expected since the major part of the truncation error for larger β, is due to the omission of the mixed derivative term, which is a linear function of β.

A few remarks concerning the numerical solution of (2) and (5) will be made. The system (2) was solved using point SOR in the same way one would construct a conformal mapping (see [5]). For the condition $\beta = 0$ on the boundary, a form of one-side "upwind" differencing was necessary to maintain convergence for the value of $b = 1/\sqrt{3}$ in this example. The elliptic equation (5) was solved using a direct elliptic solver (see [7]). After 35 iterations of (2) the maximum error in the solution of (5) was within 125% of what we estimated to be the maximum discretization error. At this point the value of $|\beta|$ was still decreasing, but at a very slow rate. The exact value for m in this example is 1 due to symmetry. The computed estimate, which was the root-mean-square value of $\sqrt{\alpha/\gamma}$, was 1.00006 after 35 iterations.

This example does not illustrate an efficient use of quasiconformal mappings. The method would be very efficient when one had to solve an elliptic equation with many different boundary conditions or inhomogeneous terms. In that case the quasiconformal mapping would only have to be constructed once.

4. Conformal mapping on surfaces. A second area of application of quasiconformal mappings is in the construction of conformal (isothermal) coordinates on a surface.

Let M be a smooth bounded surface in xyz-space which is defined by the parametric equations

$$x = x(\phi, \theta), \quad y = y(\phi, \theta), \quad z = z(\phi, \theta).$$

The parameter region in the $\phi\theta$-plane may be an arbitrarily shaped simply or doubly-connected region but it is assumed that the boundary is composed of simple closed contours and the mapping from the parameter region to the surface has a nonvanishing Jacobian.

A conformal mapping of M onto a rectangular region can be constructed by constructing a mapping from a square region of the $\xi\eta$-plane onto M which satisfies

$$P_\xi \cdot P_\eta = 0 \quad \text{and} \quad m|P_\xi| = |P_\eta|,$$

where $P = (x, y, z)$ and m is the module of the surface M. If these equations are written in terms of the parametric variables ϕ and θ, we conclude that ϕ and θ satisfy

$$m\theta_\xi = b\theta_\eta - c\phi_\eta, \qquad m\phi_\xi = a\theta_\eta - b\phi_\eta,$$

where

$$a = \frac{|P_\theta|^2}{d}, \qquad b = \frac{P_\phi \cdot P_\theta}{d},$$

$$c = \frac{|P_\phi|^2}{d}, \qquad d = |P_\phi \times P_\theta|.$$

However, this is equivalent to (1) with (x, y) replaced by (ϕ, θ). In this case the quantity which corresponds to $\beta/\sqrt{\alpha\gamma}$ would be the cosine of the angle between a $\xi =$ constant and a $\eta =$ constant coordinate line on the surface M.

Conformal grids have been constructed for several simply-connected surfaces. Three surfaces are listed below. In the first two cases, the parametric region was the projection of the surface onto the xy-plane.

(i) paraboloid: $z = 1 - x^2 - y^2, x^2 + y^2 \leqq 1$;
(ii) bicubic: $z = x^2y^3, -1 \leqq x, y \leqq 1$;
(iii) torus: $x = (2 + \sin\phi)\cos\theta$,
$y = 1 - (2\phi/\pi)^2$,
$z = (2 + \sin\phi)\sin\theta, -\pi/2 \leqq \phi \leqq \pi/2, 0 \leqq \theta \leqq \pi$.

The plots of these surfaces appear in Fig. 4. It is difficult to visualize the orthogonality from the plots, but the departure from orthogonality was less than one degree except near vertices on the boundary where the orthogonality condition was not imposed.

The advantages of conformal coordinates are well known. Problems involving heat conduction, ideal fluid flow, and electric fields can be solved as easily on the surface as they can on a rectangular region in the cartesian plane.

5. Conclusions and discussion. A finite difference method, which has been widely used for the construction of conformal mappings, has been generalized to construct quasiconformal mappings. This development will increase the class of problems which can be solved using the currently available fast elliptic solvers developed by Swarztrauber and Sweet [7]. Even when iterative methods are required, the absence of a mixed derivative and a rectangular region both would tend to give faster convergence especially when optimal iteration parameters are known.

We will conclude this report with an open problem. It is known that if $a = c = 1$ and $b = 0$, then the solution of (2) which satisfies $\beta = 0$ on the boundary of the square

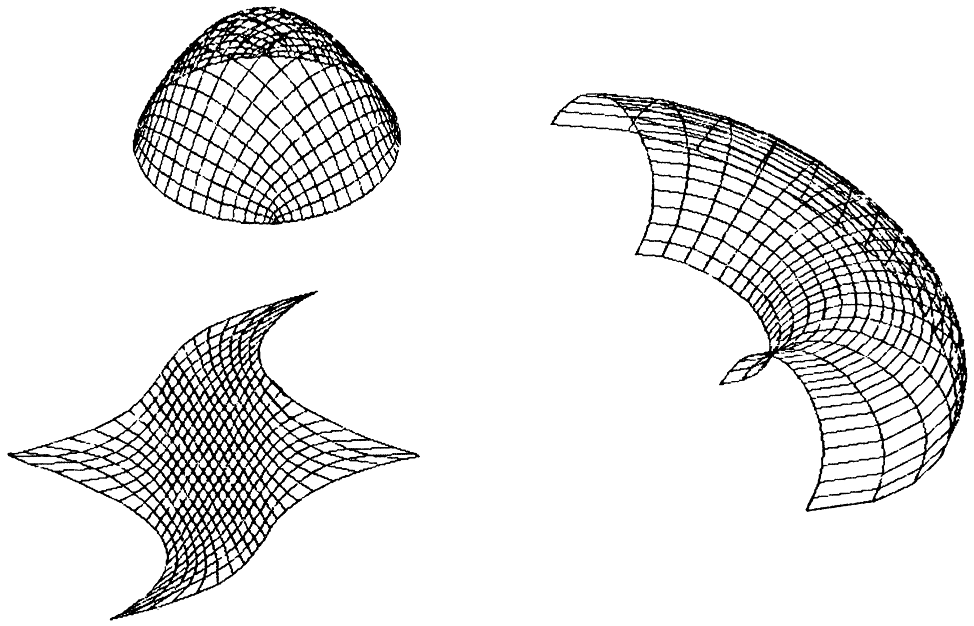


FIG. 4. *Conformal grids on subsets of a paraboloid, torus, and bicubic surface.*

S will also satisfy the system (1), and hence determines a conformal mapping. This follows directly once it is noted that the quotient

$$\frac{\eta_y + i\eta_x}{\xi_y + i\xi_x}$$

is an analytic function. Here it has been assumed, and numerical results tend to verify, that the same result holds for arbitrary quasiconformal mappings. However, no proof has been found.

REFERENCES

[1] C. N. DE G. ALLEN, *Relaxation methods applied to conformal transformation*, Quart. J. Mech. Appl. Math., 15 (1962), pp. 35–42.

[2] P. P. BELINSKII, S. K. GODUNOV AND I. K. YANENKO, *The use of a class of quasiconformal mappings to construct difference nets in domains with curvilinear boundaries*, USSR Comp. Math. Math. Phys., 15 (1975), pp. 133–144.

[3] S. K. GODUNOV AND G. P. PROKOPOV, *On the computation of conformal transformations and the construction of difference meshes*, USSR Comp. Math. Math. Phys., 7 (1967), pp. 89–124.

[4] C. W. MASTIN AND J. F. THOMPSON, *Discrete quasiconformal mappings*, Z. Angew. Math. Phys., 29 (1978), pp. 1–11.

[5] C. D. MOBLEY AND R. J. STEWART, *On the numerical generation of boundary-fitted orthogonal curvilinear coordinate systems*, J. Comp. Phys., 34 (1980), pp. 124–135.

[6] S. B. POPE, *The calculation of turbulent recirculating flows in general orthogonal coordinates*, J. Comp. Phys., 26 (1978), pp. 197–217.

[7] P. N. SWARZTRAUBER AND R. A. SWEET, *Efficient Fortran subprograms for the solution of separable elliptic partial differential equations*, ACM Trans. Math. Software, 5 (1979), pp. 352–364.

[8] S. M. YEN AND K. D. LEE, *Design criteria and generation of optimum finite element meshes*, in Lecture Notes in Physics, 90, Springer-Verlag, Berlin, 1979, pp. 579–586.

[9] J. WEISEL, *Numerische Ermittlung Quasikonformer Abbildungen mit Finiten Elementen*, Numer. Math., 35 (1980), pp. 201–222.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

CONDITION ESTIMATES*

WILLIAM W. HAGER†

Abstract. A new technique for estimating the l_1 condition number of a matrix is developed and compared to an earlier scheme.

Key word. condition number

1. Introduction. Given an $n\times n$ matrix A and a vector $\mathbf{b}\in R^n$, the condition number measures the sensitivity of $\mathbf{x}=A^{-1}\mathbf{b}$ to changes in A or $\mathbf{b}$. If $\mathbf{x}+\delta\mathbf{x}$ satisfies

$$A(\mathbf{x}+\delta\mathbf{x})=\mathbf{b}+\delta\mathbf{b},$$

then it is well known [6, p. 285] that

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|}\leqq\|A\|\,\|A^{-1}\|\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

where $\|\cdot\|$ denotes both a vector norm and the corresponding matrix norm defined by

$$\|A\|=\max\{\|A\mathbf{z}\|: \|\mathbf{z}\|=1\}. \tag{1}$$

The parameter $\kappa=\|A\|\|A^{-1}\|$ is called the *condition number*. Similarly, if $\mathbf{x}+\delta\mathbf{x}$ satisfies

$$(A+\delta A)(\mathbf{x}+\delta\mathbf{x})=\mathbf{b},$$

we have [6, p. 285]:

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}+\delta\mathbf{x}\|}\leqq\kappa\frac{\|\delta A\|}{\|A\|}.$$

In practice, the most common norms are the l_1, l_2, and l_∞ norms given by

$$\|\mathbf{x}\|_1=\sum_{i=1}^n|x_i|,\quad \|\mathbf{x}\|_2=\left(\sum_{i=1}^n x_i^2\right)^{1/2},\quad \|\mathbf{x}\|_\infty=\max\{|x_1|,|x_2|,\cdots,|x_n|\}.$$

It is well known [5, p. 21–22] that the corresponding matrix norms (1) can be expressed as follows:

$$\|A\|_1=\max_j\sum_{i=1}^n|a_{ij}|,\quad \|A\|_2=\rho(A^TA),\quad \|A\|_\infty=\max_i\sum_{j=1}^n|a_{ij}|,$$

where a_{ij} is the element in row i and column j or A, ^{T} denotes transpose, and ρ is the spectral radius. Both $\|A\|_2$ and $\|A^{-1}\|_2$ can be estimated by the power method [7, Chapter 9] while $\|A\|_1$ and $\|A\|_\infty$ can be evaluated explicitly. We focus on the problem of determining $\|A^{-1}\|_1$ and $\|A^{-1}\|_\infty$. Of course, this problem is trivial when A^{-1} is known. But since A^{-1} is rarely needed in scientific computations and the cost of inverting a matrix is often 3 or more times the cost of factoring a matrix, it is important to estimate $\|A^{-1}\|_1$ from A's factors, not from the inverse. Also note that any scheme for computing the l_1 norm of A^{-1} can be used to evaluate the l_∞ norm since $\|A^{-1}\|_\infty=\|A^{-T}\|_1$.

* Received by the editors March 16, 1982, and in revised form January 12, 1983. This research was partly supported by the National Science Foundation under grant MCS 8101892.

† Department of Mathematics, Pennsylvania State University, University Park, Pennsylvania 16802.

Cline, Moler, Stewart and Wilkinson [1] give a strategy for estimating $\|A^{-1}\|$ that involves solving two systems:

$$A^T\mathbf{x}=\mathbf{b}, \qquad A\mathbf{y}=\mathbf{x}$$

where $\mathbf{b}$ is chosen during the substitution process to "enhance" the growth of $\mathbf{x}$. Their estimate is

$$\|A^{-1}\|_1 \sim \|\mathbf{y}\|_1/\|\mathbf{x}\|_1.$$

This scheme is incorporated in LINPACK [2], a collection of programs for solving linear systems. To study reliability, O'Leary [4] computed the average ratio

$$r=\frac{\text{estimated } \|A^{-1}\|_1}{\text{actual } \|A^{-1}\|_1}$$

for 100 matrices of dimensions ranging from 5 to 50 where the a_{ij} were taken from a uniform distribution on $[-1, 1]$. Obviously, $r \leqq 1$ and $r=1$ if and only if the estimate is perfect. Column 2 of Table 1 is extracted from [4, Table 1]. O'Leary points out that for negligible cost, the strategy [1] can be improved slightly.

TABLE 1

n	Average r	Average s
5	.69	.61
10	.60	.55
20	.52	.42
40	.43	.40

On the surface, the reliability seems good. If the condition number is "big", then its estimate is big, on the average. However, these results are disappointing in the following respect: Setting

$$\mathbf{x}=\begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix},$$

let us solve $A\mathbf{y}=\mathbf{x}$ and consider the estimate $\|A^{-1}\|_1 \sim \|\mathbf{y}\|_1$. That is, $\|A^{-1}\|_1$ is approximated by the absolute sum of elements from column 1 of A^{-1}. Column 3 of Table 1 lists the average ratio

$$s=\frac{\sum_{i=1}^n |a_{i1}^{-1}|}{\|A^{-1}\|_1}$$

where a_{ij}^{-1} is the (i, j) entry of A^{-1}. Observe that this simple strategy is almost as good as the sophisticated approach! The next section presents a new scheme for estimating $\|A^{-1}\|_1$.

2. A new idea. Before developing our algorithm, let us note that for certain matrices with special structure, $\|A^{-1}\|_1$ can be computed very quickly. For example, if every element of A^{-1} is nonnegative, we can evaluate $\|A^{-1}\|_1$ by solving $A^T\mathbf{x}=\mathbf{1}$ where $\mathbf{1}$ is the vector whose components are all 1. Since the elements of A^{-1} are nonnegative, the components of $\mathbf{x}$ are the column sums of A^{-1}, and $\|A^{-1}\|_1$ is the biggest component of $\mathbf{x}$. Our goal, however, is to develop an algorithm that is suitable for matrices whose elements are generated randomly.

Given an $n \times n$ matrix B, define $f: R^n \to R$ by

$$f(\mathbf{x}) = \|B\mathbf{x}\|_1 = \sum_{i=1}^{n} \left| \sum_{j=1}^{n} b_{ij}x_j \right|.$$

Thus we have

$$\|B\|_1 = \max \{f(\mathbf{x}): \|\mathbf{x}\|_1 \leqq 1\}.$$

Abstractly, $\|B\|_1$ is the maximum of the convex function f over the convex set

$$S = \{\mathbf{x} \in R^n: \|\mathbf{x}\|_1 \leqq 1\}.$$

It is well known that a convex function defined on a convex, compact set attains its maximum at an extreme point. The $2n$ extreme points of S are simply

$$\{\pm \mathbf{e}^j : j = 1, \cdots, n\}$$

where $\mathbf{e}^j$ is the unit vector whose components are all 0 except for the jth component which is 1. Since f is convex, it satisfies the inequality

$$f(\mathbf{y}) \geqq f(\mathbf{x}) + \partial f(\mathbf{x})(\mathbf{y} - \mathbf{x}) \tag{2}$$

for all $\mathbf{x}, \mathbf{y} \in R^n$ where $\partial f(\mathbf{x})$ denotes a subgradient of f at $\mathbf{x}$. If

$$\sum_{j=1}^{n} b_{ij}x_j \neq 0$$

for each i, then $\partial f(\mathbf{x})$ is the usual gradient vector. Defining for $i = 1$ to n,

$$\xi_i = \begin{cases} 1 & \text{if } \sum_{j=1}^{n} b_{ij}x_j \geqq 0, \\ -1 & \text{otherwise,} \end{cases} \tag{3}$$

the chain rule gives us

$$\partial f(\mathbf{x}) = \boldsymbol{\xi}^T B. \tag{4}$$

Note that if one or more components of $B\mathbf{x}$ are zero at some point $\mathbf{x}$, then the function $f(\cdot)$ has a corner at $\mathbf{x}$, and the set of subgradients has many elements at this point. That is, if $(B\mathbf{x})_i = 0$, then equation (4) gives us a different element of this set for each value of ξ_i between -1 and 1. Thus equations (3) and (4) specify a particular element of the subgradient set at the corners of $f(\cdot)$. In the special case $B = A^{-1}$, computing $\partial f(\mathbf{x})$ by equations (3) and (4) is equivalent to solving two systems:

$$A\mathbf{y} = \mathbf{x}, \qquad A^T\mathbf{z} = \boldsymbol{\xi} \tag{5}$$

where

$$\xi_i = \begin{cases} 1 & \text{if } y_i \geqq 0, \\ -1 & \text{otherwise,} \end{cases}$$

and $\partial f(\mathbf{x}) = \mathbf{z}^T$.

Our algorithm for estimating $\|B\|_1$ starts at a point $\mathbf{x}$ on the boundary of S. We then find a j for which

$$|\partial f(\mathbf{x})_j| = \max_i |\partial f(\mathbf{x})_i|. \tag{6}$$

If $|\partial f(\mathbf{x})_j| \leqq \partial f(\mathbf{x})\mathbf{x}$, then stop. (Below we show that this $\mathbf{x}$ is a "local maximum" of f over the polytope S). Conversely, suppose that $|\partial f(\mathbf{x})_j| > \partial f(\mathbf{x})\mathbf{x}$. By the convexity inequality (2) and the fact that $f(\mathbf{e}^j) = f(-\mathbf{e}^j)$, we conclude that $f(\mathbf{e}^j) > f(\mathbf{x})$. Replacing $\mathbf{x}$ by $\mathbf{e}^j$, this process repeats. Since f is strictly increasing, vertices of S are visited only once, and the iterations terminate in a finite number of steps. A Fortran code for our algorithm is included in [3].

To prove that the final point $\mathbf{x}$ generated by this algorithm is a local maximum, we assume that every component of $B\mathbf{x}$ is nonzero. In the case that some component of $B\mathbf{x}$ is zero, we should modify (6) by letting the index j correspond to the maximum absolute component over the entire set of subgradient vectors. The algorithm still makes sense without this modification, but $\mathbf{x}$ may not be a local maximum of f. When the components of $B\mathbf{x}$ are nonzero, $f(\cdot)$ is linear near $\mathbf{x}$. Hence $\mathbf{x}$ is a local maximum of f over S if and only if

$$\partial f(\mathbf{x})(\mathbf{y}-\mathbf{x}) \leqq 0$$

for every $\mathbf{y} \in S$. If $\mathbf{y}$ is a vertex of S, then $\partial f(\mathbf{x})\mathbf{y} = \pm \partial f(\mathbf{x})_i$ for some i since all but one component of $\mathbf{y}$ is zero. If $|\partial f(\mathbf{x})_i| \leqq \partial f(\mathbf{x})\mathbf{x}$ for each i, it follows that $\partial f(\mathbf{x})(\mathbf{y}-\mathbf{x}) \leqq 0$ whenever $\mathbf{y}$ is a vertex of S. Since S is the convex hull of its vertices, $\partial f(\mathbf{x})(\mathbf{y}-\mathbf{x}) \leqq 0$ for every $\mathbf{y} \in S$, and $\mathbf{x}$ is a local maximum of f over S.

To test this scheme, we computed the ratio

$$t_1 = \frac{\text{estimated } \|A^{-1}\|_1}{\text{actual } \|A^{-1}\|_1}$$

for 200 matrices of the same dimension where the a_{ij} are taken from a uniform distribution on $[-1, 1]$. Our initial guess is $\mathbf{x} = n^{-1}\mathbf{1}$. Column 3 of Table 2 gives the

TABLE 2

n	Average t_1	Average steps	Probability $t_1 \geqq .99$
5	.96	2.1	.82
10	.97	2.1	.83
20	.98	2.1	.88
40	.97	2.1	.85
80	.98	2.1	.86

average termination step, counting the initial guess $\mathbf{x} = n^{-1}\mathbf{1}$ as step 1. Column 4 is the proportion of the cases where $t_1 \geqq .99$. With few exceptions, $t_1 \geqq .99$ if and only if the algorithm actually found the vertex $\mathbf{e}^j$ for which $\|A^{-1}\mathbf{e}^j\|_1 = \|A^{-1}\|_1$. It appears that the reliability is independent of n. Since the average termination step is 2.1, the scheme starts from $\mathbf{x} = n^{-1}\mathbf{1}$ and almost always moves straight to a locally maximizing vertex of S. Of course, each step involves solving the two systems (5). In column 4 of Table 2, we see that the local maximum computed by the algorithm is a global maximum with high probability.

To estimate $\|A^{-1}\|_1$ more precisely, our scheme is applied repeatedly to suitable subspaces. During the first cycle described above, we visit vertices $\{\mathbf{v}^1, \cdots, \mathbf{v}^m\}$ and stop at a local maximum. Let $\{\mathbf{v}^{m+1}, \cdots, \mathbf{v}^n\}$ be the remaining vertices; that is,

$$\{\mathbf{v}^{m+1}, \cdots, \mathbf{v}^n\} = \{\mathbf{e}^1, \cdots, \mathbf{e}^n\} - \{\mathbf{v}^1, \cdots, \mathbf{v}^m\}.$$

Then starting at the point

$$\mathbf{x} = \frac{1}{n-m} \sum_{i=m+1}^{n} \mathbf{v}^i,$$

we apply the same scheme to the polytope S_2 with vertices

$$\{\pm\mathbf{v}^i : i = m+1, \cdots, n\}.$$

This leads us to a local maximum on S_2. Our estimate for $\|A^{-1}\|_1$ is the bigger local maximum. Letting t_2 be the ratio between the estimated $\|A^{-1}\|_1$ and the actual $\|A^{-1}\|_1$, our results for the two cycle process are summarized in Table 3.

TABLE 3

n	Average t_2	Average steps	Probability $t_2 \geqq .99$
5	.993	4.2	.94
10	.991	4.2	.94
20	.993	4.2	.95
40	.987	4.2	.90
80	.995	4.3	.95

Finally, the three cycle process yields Table 4.

TABLE 4

n	Average t_3	Average steps	Probability $t_3 \geqq .99$
5	.997	6.2	.98
10	.995	6.4	.97
20	.997	6.5	.96
40	.996	6.4	.97
80	.997	6.6	.97

The worst condition estimate that we detected for the 200 random matrices is shown in Table 5. If the hyperplanes $\{\mathbf{x} \in R^n : \sum_{j=1}^{n} b_{ij}x_j = 0\}$ do not intersect some face of S and $\mathbf{v}$ is any vertex of S, then one step of our algorithm starting from $\mathbf{v}$ takes us to a global maximum of f over S. This situation corresponds to f being linear on a face of S. On the other hand, when the hyperplanes intersect all the faces of S, then f has corners on each face, and it is possible to hide the global maximum behind a corner.

TABLE 5

n	t_1	t_2	t_3
5	.32	.67	.70
10	.39	.67	.76
20	.46	.62	.74
40	.43	.44	.78
80	.46	.71	.71

REFERENCES

[1] 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.

[2] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[3] W. W. HAGER, *Computing*, book in preparation.

[4] D. P. O'LEARY, *Estimating matrix condition numbers*, this Journal, 1 (1980), 205–209.

[5] J. M. ORTEGA, *Numerical Analysis, A Second Course*, Academic Press, New York, 1973.

[6] G. STRANG, *Linear Algebra and Its Applications*, Academic Press, New York, 1980.

[7] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, London, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

FAST APPROXIMATION OF DOMINANT HARMONICS[*]

GEORGE CYBENKO[†]

Abstract. This paper presents a fast method for estimating dominant harmonics in a sequence of data. In a stochastic sense, the proposed method finds the autoregressive scheme with a pure point spectrum that best describes the data, while from a deterministic point of view, the method is a special case of the Lanczos algorithm for finding eigenvalues of a symmetric matrix. Eigenvalue approximations come into play because every circulant matrix is diagonalized by the discrete Fourier transform matrix, and so using the Lanczos algorithm with the given data as the initial vector on a simple circulant matrix, the eigenvalues that are first approximated are the eigenvalues corresponding to eigenvectors which are dominant in the initial vector. It is shown that this method is related to "lattice methods" for linear prediction and to Prony's method for exponential approximation.

Key words. time series analysis, eigenvalue approximation, Lanczos algorithm, spectral analysis

1. Introduction.

In this paper, we address the problem of approximating a given sequence of data by a sum of sinusoids. This problem arises in many fields of application such as astronomy, meteorology, geology and tidal analysis [27], [37]. Lanczos called the problem "the search for hidden periodicities," while others have called a version of it "the retrieval of harmonics" problem. The problem of finding dominant harmonic components in a time series is a fundamental question [2] and motivated much of the early work in the field [45], [47]. This problem is also basic to the estimation of point (discrete) spectra, as will soon be shown.

Specifically, given a sequence of data $s_1, s_2, \cdots, s_n$ we wish to find a collection of amplitudes $a_1, a_2, \cdots, a_p$, a collection of frequencies $w_1, w_2, \cdots, w_p$ and a collection of phase angles $\theta_1, \theta_2, \cdots, \theta_p$ with $0 \leqq w_k \leqq \pi$ so that

$$s_j \approx \sum_{k=1}^{p} a_k \cos(jw_k + \theta_k) \tag{1.1}$$

where $\approx$ denotes some sort of approximation. Typically, p is taken to be significantly smaller than n. It is significant that the frequencies w_k are not assumed to be rational multiples of π such as would occur in a discrete Fourier transform analysis of the data sequence.

The exact sense in which the approximation is made is of course central to the formulation and ultimate solution of this problem, and we shall now present some of the more important direct approaches to formulating the problem.

1. The nonlinear least-squares problem of determining the best fit according to

$$\text{minimize} \sum_j \left| s_j - \sum_k a_k \cos(jw_k + \theta_k) \right|^2$$

over all a_k, θ_k and w_k. Notice that in this solution, the residual vector $e = (e_j)$,

$$s_j = \sum_k a_k \cos(jw_k + \theta_k) + e_j$$

* Received by the editors July 15, 1982, and in revised form January 3, 1983. This work was supported in part by the National Science Foundation under grant MCS-8003364. Parts of this paper were presented at the SIAM Conference on Applied Linear Algebra, Raleigh, North Carolina, April 26–29, 1982, in a talk entitled "Numerically Stable Alternatives to Prony's Method."

† Department of Mathematics, Tufts University, Medford, Massachusetts 02155, and Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

is orthogonal to the vectors $c(k)=(\cos(jw_k+\theta_k))$ by the properties of linear least-squares solutions. Such nonlinear least-squares problems have been discussed by a number of authors [15], [16], [21], [35], [40] in a very general setting.

2. Modeling the sequence s_j as

$$s_j=\sum_k a_k \cos(jw_k)+e_j$$

where the residuals e_j form a vector orthogonal to each of the $c(k)=(\cos(jw_k))$ and have zero autocorrelations, that is, also satisfy

$$\sum_j e_j e_{j+k}=0 \quad \text{for } k=1,\cdots,p-1.$$

This formulation and a technique for the estimation of this model were proposed by Pisarenko [6], [25], [37]. The statistical model for this formulation assumes that a deterministic sinusoidal signal is contaminated with additive white noise.

3. Fitting the sequence by an autoregressive model with a pure point spectrum in the following sense: minimize the expression

$$\sum_j |s_j+b_{p-1}s_{j-1}+b_{p-2}s_{j-2}+\cdots+b_0 s_{j-p}|^2$$

subject to the condition that the roots of the polynomial

$$b(z)=\sum_{k=0}^{p} b_k z^k, \qquad b_p=1,$$

all lie on the unit circle of the complex plane. To see the relationship between this problem and the sinusoidal fitting problem, let Z be the unit delay operator defined by $(Zs)_j=s_{j-1}$. Then solutions to the homogeneous difference equation

$$b(Z)s=0$$

are precisely the weighted sums of sinusoids arising in the approximation, so that the b determined by the above problem is in a sense "the best difference equation" with sinusoidal solutions satisfied by the data s. Once the difference equation is found, the homogeneous solution to it then gives frequencies for approximating the data. Notice that phases are not present in this solution but can be reconstructed as will be shown later in the paper. The statistical model for this formulation is that the given data satisfies an equation of the form

$$s_j+b_{p-1}s_{j-1}+\cdots+b_0 s_{j-p}=w_j$$

where w_j is white noise.

It should be noted here that the discrete Fourier transform (DFT) is of limited use since the transform of n data points results in n frequency amplitudes which amounts to no data compression or modeling whatsoever. Even after the computation of the DFT using a fast Fourier transform, the problem would remain of finding the "most dominant" collection of frequencies, thus requiring at least $n(\log_2 n)$ operations and the use of complex arithmetic. As we shall see, the method proposed in this paper is faster when p is less than $\log_2 n$ and uses only real arithmetic. In fact, Lanczos [17] did propose a method based on the interpolation of the discrete Fourier transform, but his approach is completely heuristic and does not correspond to any model or error assumptions.

A significant observation is that the first formulation of above, namely the nonlinear least-squares version, is in a sense ill-posed, as the following considerations show. Any data vector $s=(s_j)$, $j=1,\cdots,2n+1$, is expressible as a combination of the $2n+1$ vectors

$$c(k)=\left(\cos\left(\frac{jk\pi}{2(n+1)}\right)\right) \quad \text{for } k=1,\cdots,2n+1,$$

say

$$s=\sum_k x_k c(k).$$

Let us assume that only a few of the x_k are nonzero and that the nonzero x_k have k even. Furthermore, assume that we have $p=1$ for simplicity. Thus we are to find a and w to minimize

$$\sum_j |s_j - a\cos(jw)|^2.$$

Writing $d(w)=(\cos(jw))$, the value of a minimizing the expression for a given w is $a=s^T d(w)/d(w)^T d(w)$, to that for a given w the attainable minimum is

$$\|s-ad(w)\|^2 = s^T s - \frac{(s^T d(w))^2}{d(w)^T d(w)}. \tag{1.2}$$

Now $c(k)^T d(w)=0$ for w an odd multiple of $\pi/2(n+1)$. Thus every such w is a local optimum and between every adjacent pair of odd multiples of $\pi/2(n+1)$ is a local minimum. Hence, in terms of w there are about $2n$ local optima. The standard approach to solving such separable least-squares problems is to find critical points of the expression in (1.2), and as we have just seen, there are about $2n$ such critical points. In a problem with $n=1000$, the density of such extreme points is so great that it is extremely unlikely that any random starting point for the nonlinear least-squares formulation of the problem will converge to a global optimum.

Although this type of example is admittedly specially structured, it raises some very serious questions about the well-posedness of the problem in Formulation 1, above. The potential density of local optima at which an iterative algorithm for solving the approximation in Formulation 1 can get trapped necessitates a re-evaluation of the straightforward nonlinear least-squares approach. The method presented in this paper can be used as a technique for generating starting values for the nonlinear iterations required for solving Formulation 1, and numerical experiments support this claim.

This paper presents a solution to Formulation 3 above, and it has been empirically observed that these solutions yield excellent starting values for the nonlinear iterations arising in Formulation 1. An analysis of the method actually makes it possible to predict situations where the technique is not expected to behave well.

Our method formulates the problem in terms of a symmetric averaging operator rather than in terms of the shift operator, thereby forcing functions of the symmetrized variable to have zeros occurring in reciprocal pairs. The pth degree polynomial in this averaging operator minimizing the 2-norm is seen to be the pth degree monic orthogonal polynomial with respect to a classical real inner product in the real interval $[-2,2]$, and is easily computed from the data sequence using the symmetric Lanczos algorithm [18], [26], [36]. The zeros of this polynomial can be obtained from the eigenvalues of the tridiagonal matrix determined by the Lanczos iteration. The zeros are guaranteed to be in the interval $[-2,2]$ by a classical theorem of Fejer and the frequencies are obtained as the arccosines of one half of the zeros thus obtained. The frequencies can then be used to estimate optimal amplitudes and phases.

Goodman and Miller [19] have shown how the monic polynomial occurring in the solution to Formulation 3 can be computed efficiently using the Levinson–Durbin algorithm [11], [29] or the "lattice" orthogonalization method [3], [4], [22], [31]. The actual estimated dominant frequencies are then obtained by finding the zeros of that polynomial. The method presented here derives a tridiagonal matrix whose eigenvalues are simply related to the estimated dominant frequencies. In theory, the zeros of the polynomial obtained by Goodman and Miller and the arccosines of one-half the eigenvalues in this paper are identical. However, simple methods for symmetric tridiagonal eigenvalue problems are known to be more efficient and accurate than polynomial root-finding methods [46]. Thus, in addition to describing an alternate approach for computing the desired polynomial, the method of this paper offers a complete, efficient and accurate solution to the problem of computing the actual dominant frequencies.

It should be noted that many other methods exist for spectral resolution problems of the type discussed here. In particular, the papers [19], [44] address different formulations (closer to our Formulation 1) that use exhaustive search techniques for estimating dominant harmonics. The work by Marple [32] discusses Prony's method and Pisarenko's method for solving Formulation 2.

Complete details of our approach are given in § 2, while connections between this method and currently popular methods for spectral estimation and resolution are presented in § 3. The results of some numerical experiments are in § 4.

2. The algorithm. Recall that we are interested in solving the formulation to the sinusoidal approximation problem as given in the third approach above. For ease of presentation, we shall consider the case where the data sequence is extended with zeros to be of arbitrary length—that is, our problem is to find a monic polynomial $q(z)$ so that the expression

$$\|q(Z)s\|^2 = \sum_{j=-\infty}^{\infty} |(q(Z)s)_j|^2$$

is minimized, subject to the constraint that $q(z)$ have roots on the unit circle. Here as in the remainder of the paper, an upper case Z denotes the shift operator

$$(Zs)_j = s_{j-1}$$

while lower case z denotes a complex scalar variable.

Although the summation is written as an infinite one, there are no more than $n+p$ nonzero terms. A feature of this approach is that we are padding the data artificially with zeros to get a convenient form, but as we shall see, this is precisely one of the ingredients of the formulation that make the problem tractable. Another approach is obtained by summing only over interior indices j where the explicit padding by zeros is not required, and although this formulation is solvable by ideas similar to those presented in this paper, the introduction of a nonclassical orthogonal polynomial theory becomes essential (this nonclassical theory is intimately related with "close-to-Hankel" matrices as in [23]), and the root locations of the minimizing polynomial are no longer easily kept on the unit circle. That extension will be presented in a forthcoming paper.

In the present case, we restrict our attention to the above situation where data is augmented with zeros. For problems with n large and p small, the difference between the two formulations is expected to be quite small. Furthermore, results and algorithms are cleaner and more complete in this case.

A real polynomial with zeros on the unit circle is factorable into quadratic terms of the form

$$z^2+\gamma z+1,$$

where γ lies in the interval $[-2, 2]$, and possibly linear terms of the form

$$z-1 \quad \text{and} \quad z+1.$$

For the moment, we shall only consider the situation where $q(z)$ has quadratic factors with $|\gamma|<2$, and it will subsequently be clear how to accommodate the situation where linear factors are allowed. That is, the possible real factors of $q(z)$ that we consider are

$$z^2+\gamma z+1$$

with γ in $(-2, 2)$ only. This means that $q(z)$ has even degree, say $2p=p'$. The minimization of above is therefore equivalent to the problem

$$\text{minimize } \|Z^{-p}q(Z)s\|^2$$

where $q(Z)$ is a polynomial of degree $2p$ in Z with zeros on the unit circle. Such a polynomial is clearly a function of Z^*+Z because of the symmetry of the real quadratic factors of q. Letting $\Omega=Z^*+Z$, we are to solve

$$\text{minimize } \|r(\Omega)s\|^2$$

where the constraint that $q(z)$ is monic with zeros on the unit circle has become the constraint that $r(Z)$ has zeros in the interval $(-2, 2)$ and is monic.

Our first result is that the constraint on root locations for $r(Z)$ is actually built into the formulation of the problem and is therefore automatically satisfied by our reformulation in terms of the symmetric operator Ω.

THEOREM 1. *The monic polynomial* $r(z)$ *determined by*

$$\text{minimum } \|r(\Omega)s\|^2 \tag{2.1}$$

has all roots in the interval $(-2, 2)$.

Proof. Consider the inner product defined by

$$\langle u, v\rangle = s^Tuv(\Omega)s$$

where u and v are polynomials in z. Although s and Ω are infinite, the inner product is determined by finite sections so we can replace s and Ω by finite versions, sufficiently large, whose size depends on the order of $uv(z)$. In fact, we can replace Ω by the finite circulant

$$\Omega'=\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 1 & \ddots & & 0 \\ 0 & 1 & 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & \ddots & 1 \\ 1 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix},$$

so that for u and v of degree less than p, say, we have

$$\langle u, v\rangle = s^Tuv(\Omega')s=(Fs)^*uv(D)(Fs)$$

where F is the discrete Fourier transform matrix and D is the diagonal matrix of eigenvalues of Ω' [8]. The eigenvalues of Ω' are in fact

$$d_k = 2\cos\left(\frac{2k\pi}{N}\right), \qquad k=0,\cdots,N-1,$$

all of which lie in the interval $[-2, 2]$. Thus

$$\langle u, v\rangle = \sum_j |(Fs)_j|^2 uv(d_j),$$

and so the inner product thus induced is seen to be an inner product with respect to a discrete measure on the interval $[-2, 2]$.

Furthermore, it is well known that the solution to

$$\text{minimum } \|q(\Omega)s\|^2 = \text{minimum } \langle q, q\rangle$$

over monic $q(z)$ of degree p is precisely the pth monic orthogonal polynomial with respect to the inner product $\langle\cdot,\cdot\rangle$ [7].

Fejer's theorem [7], [12] states that the zeros of this polynomial all lie in the interval $[-2, 2]$. The fact that the zeros actually lie in the interior of this interval is seen from the fact that the orthogonal polynomials form a Sturm sequence. This aspect is reviewed after the properties of the polynomials are developed in the following. □

Thus solutions to (2.1) over monic $r(z)$ must have zeros in the desired interval. Hence, the factors of $r(\Omega)$ are of the form

$$Z^* + Z - \gamma I, \quad \text{where } -2<\gamma<2.$$

Each of these linear factors in $Z+Z^*$ gives rise to a conjugate pair of zeros of $q(Z) = r(Z+Z^*)Z^p$,

$$z = \frac{\gamma}{2} + i\sqrt{1-(\gamma/2)^2}, \qquad z^* = \frac{\gamma}{2} - i\sqrt{1-(\gamma/2)^2},$$

which lie on the unit circle.

Hence, we have reduced the problem to computing the orthogonal polynomials with respect to the above inner product and then finding the zeros of the pth polynomial obtained. The sequence of orthogonal polynomials can be easily computed using the triple recursion relationship and the Lanczos algorithm, after which a tridiagonal eigenvalue problem determines the zeros of the pth orthogonal polynomial. The details of this are now presented.

First, we notice that the operator Ω of above is symmetric, so that the inner product in question satisfies

$$\langle xu, v\rangle = \langle u, xv\rangle,$$

which is precisely the requirement for a triple recursion relationship to hold between triples of successive orthogonal polynomials. In fact, it has already been shown that this inner product is induced by a positive discrete measure over the interval $(-2, 2)$. Letting $q_j(x)$ be the jth orthogonal polynomial, we have

$$q_{j+1}(x) = (x-\alpha_j)q_j(x) - \beta_j q_{j-1}(x)$$

where

$$\alpha_j = \frac{\langle xq_j, q_j\rangle}{\langle q_j, q_j\rangle}, \qquad \beta_j = \frac{\langle xq_j, q_{j-1}\rangle}{\langle q_{j-1}, q_{j-1}\rangle}.$$

A more balanced version of this recursion with better numerical properties is [36]

$$\beta_0 \leftarrow \sqrt{\langle 1,1\rangle}, \qquad r_0 \leftarrow 1.$$

```
For $j=1$ to $p+1$ do
    $q_{j-1}(x) \leftarrow r_{j-1}(x)/\beta_{j-1}$
    $r_j(x) \leftarrow xq_{j-1}(x) - \beta_{j-1}q_{j-2}(x) \quad (q_{-1}=0)$
    $\alpha_j \leftarrow \langle q_{j-1}, r_j\rangle$
    $r_j(x) \leftarrow r_j(x) - \alpha_j q_{j-1}(x)$
    $\beta_j \leftarrow \sqrt{\langle r_j, r_j\rangle}$.
```

A fundamental feature of this iteration is that the characteristic polynomial of the matrix

$$\begin{bmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \\ & \beta_2 & \ddots & \beta_{p-1} \\ 0 & & \beta_{p-1} & \alpha_p \end{bmatrix} = T(\alpha_1, \cdots, \alpha_p; \beta_1, \cdots, \beta_{p-1})$$

is the desired monic polynomial minimizing (2.1). This is easily shown by induction on the order of the matrix. Hence, the eigenvalues of the matrix are the zeros of the monic minimizer of (2.1).

A significant simplification of the above is the observation that the only required quantities are the vectors

$$q_j(\Omega)s, \qquad r_j(\Omega)s$$

since they are sufficient for computing the inner products giving the coefficients α_j and β_j.

Hence, we now state this vector version, which we shall call "the symmetric lattice algorithm", a name to be explained in the next section. Here s, q, r, and v are vectors of length n.

```
THE SYMMETRIC LATTICE ALGORITHM.
$s=(s_i)$ given with $s_i = 0 = s_{n-i+1}$ for $i=1$ to $p$,

        $b_0 \leftarrow \sqrt{s^Ts}, \quad r \leftarrow \frac{s}{b_0}, \quad v \leftarrow 0.$

For $j=1$ to $p+1$ do
        $q \leftarrow r$
        For $i=2$ to $n-1$ do
            $r_i \leftarrow (q_{i+1}+q_{i-1})/b_{j-1} - b_{j-1}v_i$
(2.2)   $a_j \leftarrow q^Tr$
        $r \leftarrow r - a_jq$
        $v \leftarrow q$
        $b_j \leftarrow \sqrt{r^Tr}$.
```

This part of the algorithm requires about $4np$ multiplications. The second phase of the algorithm is then to find the eigenvalues of the tridiagonal matrix $T(a_1, \cdots, a_p; b_1, \cdots, b_{p-1})$. This requires about p^3 multiplications using the symmetric Francis QR algorithm [13], [18], [36].

We now state a simple fact about such a tridiagonal T as computed by our algorithm above.

LEMMA 1. *The matrix T of above is unreduced; that is, none of the b_i are zero.*

Proof. Recall that

$$b_i^2 = \|r\|^2 = \|r(\Omega)s\|^2$$

for some polynomial $r(\Omega)$ of degree i, which will only be 0 if s satisfies exactly a $2i$th order difference equation. Since $s \neq 0$ but is padded with zeros, this is clearly impossible and the result is proved. $\square$

The fact that T is unreduced allows for the direct use of fast and accurate methods for solving symmetric tridiagonal eigenvalue problems. A description of methods and their actual implementations may be found in [13], [18], [36], for instance.

The method described above solves the problem where the minimization is over all polynomials with roots on the unit circle, except at the distinguished points $+1$ and -1, which would lead to linear real factors in the polynomial and could therefore not be accommodated by the above technique. To include these linear factors, it seems to be necessary to solve the auxiliary systems

$$\min \|q(\Omega)(Z-1)s\|^2, \qquad \min \|q(\Omega)(Z+1)s\|^2,$$
$$\min \|q(\Omega)(Z^2-1)s\|^2$$

where the minimizations are done over monic $q(\Omega)$ of degree $p-1$, $p-1$ and $p-2$ respectively. The four minimizations can then be compared and the smallest picked. (The fourth minimization is without any linear factors.)

A fundamental question concerns the behavior, as p increases, of the residual norm

$$E_p = \min \|q(\Omega)s\|, \qquad \text{degree } (q) = p,$$

over monic q with roots on the unit circle. In a problem where p is not known a priori, it is essential that the algorithm provide some information about which order p is optimal. This is in fact possible, as the following results show.

LEMMA 2. $E_p = (\prod_{i=1}^{p} b_i)$.

Proof. The above results have shown that E_p is the norm of the monic orthogonal polynomial of degree p with respect to the norm $\langle \cdot, \cdot \rangle$. The symmetric lattice algorithm generates orthonormal polynomials in such a way that the relationship between successive leading coefficients is

$$q_i(x) = \frac{xq_{i-1}(x)}{b_{i-1}} + \text{lower order terms},$$

establishing the result. $\square$

LEMMA 3.

$$0 < b_i \leqq \sqrt{2}.$$

In particular, E_i need not be a monotonic decreasing function of i.

Proof. Since $b_i^2 = (q_i(\Omega)s)^T(\Omega q_{i-1}(\Omega)s)$ and $\|q_{i-1}(\Omega)s\|^2 = 1$, an application of the Cauchy–Schwarz inequality establishes the result once we note that the operator 2-norm of Ω is 2. The fact that some of the b_i may actually be larger than 1, that is, that E_i need not be decreasing as a function of i, is seen from the examples of § 4. $\square$

A few words are now appropriate to discuss the phase angles which have not played a role in the procedure thus far. We note firstly that a sinusoid determined by a sine function such as $\sin(jw+\theta)$ is trivially expressible as a cosine function with a phase shift. Now the operator

$$Z+Z^*-2\cos(w)I$$

annihilates any cosine function of the form $\cos(wj+\theta)$ regardless of the phase, θ, so that we must do some extra work to recover the phase angle from a term such as

$$\cos(jw+\theta).$$

Now suppose that a value, $2\cos(w)$, has been computed as one of the roots of the minimizing polynomials so that a component of the form $a\cos(wj+\theta)$ is expected to be significant in the data. We wish to recover the phase angle θ. Using the elementary trigonometric identity

$$\cos(A+B)=\cos(A)\cos(B)-\sin(A)\sin(B), \tag{2.3}$$

we can find the least-squares components of the vectors

$$c=(c_j),\qquad c_j=\begin{cases}0 & \text{for } 1\leqq j\leqq p \text{ and } n\leqq j\leqq n+p,\\ \cos(jw) & \text{otherwise,}\end{cases}$$

$$s=(s_j),\qquad s_j=\begin{cases}0 & \text{for } 1\leqq j\leqq p \text{ and } n\leqq j\leqq n+p,\\ \sin(jw) & \text{otherwise}\end{cases}$$

in the data vector s. Using the coefficients computed by the solution, it is then possible to reconstruct the phase by using the identity (2.3).

The computation of the phase angles would of course significantly increase the computational complexity—requiring about $O(np^2)$ additional algebraic operations and $2np$ evaluations of the elementary trigonometric functions, sine and cosine. If only the primary angles w are desired, however, this extra effort is not necessary. An implementation of these ideas is presented in § 4, where numerical examples and comments on them are given.

3. Connections with Prony's method and lattice algorithms. The basic connection between the method described in this work and the classical Prony method for fitting data with exponentials [24], [38], [39] is through the use of shift operators and their action on exponential functions. The essential idea in both cases is quite simply described as follows.

Prony's method. To approximate a sequence s_j by

$$f_j=\sum_k a_k e^{-jw_k}$$

note that

$$P(Z)=\prod_k (Z-e^{-w_k})$$

annihilates $f=(f_j)$. Hence we can estimate w_k by finding the zeros of the polynomial $P(Z)$ that minimizes the expression

$$\|P(Z)s\|_2^2.$$

Symmetric Prony's method. To approximate the sequence s_j by

$$f_j=\sum_k a_k e^{iw_k j}$$

where the w_k occur in conjugate pairs and coefficients a_k for these conjugate pairs are equal, note that such an $f=(f_j)$ is annihilated by a polynomial of the form

$$P(Z)=\prod_k (Z+2\cos(w_k)+Z^{-1})$$

where $P(Z)$ may be written as a polynomial in $Z+Z^{-1}$, say $Q(\Omega)$, where $\Omega = Z+Z^{-1}$. Hence, we can estimate w_k by finding the zeros of the monic polynomial $Q(\Omega)$ that minimizes $\|Q(\Omega)s\|^2$ over all such Q.

Autoregressive modeling of time series [2] and linear prediction [30] use the same basic construction as does Prony's method, although with quite different goals. An autoregressive time series of order p, s_j satisfies an equation of the form

$$s_j + b_{p-1}s_{j-1} + \cdots + b_0 s_{j-p} + w_j = 0$$

where w_j is white noise and the polynomial

$$b(z) = z^p + b_{p-1}z^{p-1} + \cdots + b_0$$

has roots inside the unit circle. Hence, given a time series, the problem of finding the best model of a given order that describes the time series in a least-squares sense involves computing coefficients of $b(z)$ that minimize the expression

$$\|b(Z)s\|^2,$$

with the constraint that $b(z)$ have roots inside the unit circle. The polynomial $b(z)$ can also be used very successfully to estimate the spectrum of the time series using a method known as "maximum entropy spectral estimation" [3], [10].

There are basically two direct ways of solving for the coefficients of $b(z)$. Noting that the vector of coefficients is the solution to a linear least-squares problem, it is possible to form the normal equations, which involve a Toeplitz matrix, or to orthogonalize the column space in the least-squares problem, which also involves a Toeplitz matrix [4]. Methods for solving the Toeplitz normal equations efficiently have been known for some time [11], [23], [29], [34], [43] while more recently fast methods for orthogonalizing have been discovered also [3], [4], [9], [22], [31]. These fast orthogonalization methods have become known as "lattice" or "ladder" methods because of their form when implemented as circuits. It is for this reason that we call our procedure the "symmetric lattice" algorithm.

By contrast, the least-squares problem solved here involves normal equations with a Hankel matrix, which could be solved efficiently also [1], [33], [41] by essentially using the three-term recurrence relation for real orthogonal polynomials. The corresponding solution obtained by orthogonalizing the column space which is presented here uses the symmetric Lanczos algorithm. The advantages are quite evident—orthogonalization is numerically superior to forming normal equations for solving least-squares problems [17], [28], and the computation of the roots of the resulting polynomial is easily reduced to a symmetric tridiagonal eigenvalue problem for which efficient and accurate algorithms exist [13], [18], [36].

The correspondence between the Toeplitz and Hankel matrices is quite simple. Given the sequence, s_j, define

$$r_k = \sum_j s_j s_{j+k}$$

where we use zero when s_j or s_{j+k} is undefined (these are the autocorrelations for the time series) and define inductively

$$y^0 = s, \qquad y^{j+1} = Zy^j + Z^{-1}y^j,$$
$$h_k = \sum_j s_j y_j^k,$$

where we once again pad with zeros at the boundaries of definition (these parameters h_k can be referred to as "Markov" parameters). The Toeplitz matrix is then

$$T = \begin{bmatrix} r_0 & r_1 & \cdots & r_p \\ r_1 & r_0 & r_1 & \vdots \\ \vdots & \ddots & \ddots & r_1 \\ r_p & \cdots & r_1 & r_0 \end{bmatrix},$$

and the Hankel matrix is then

$$H = \begin{bmatrix} h_0 & h_1 & h_2 & \cdots & h_p \\ h_1 & h_2 & & & \\ h_2 & & & & \vdots \\ \vdots & & & & \\ h_p & & \cdots & & h_{2p} \end{bmatrix}.$$

The Markov parameters h_k are related to the autocorrelations r_k through the following expressions involving binomial coefficients:

$$h_0 = r_0,$$
$$h_1 = r_{-1} + r_1 = 2r_1,$$
$$h_2 = r_{-2} + 2r_0 + r_2 = 2(r_0 + r_2),$$
$$h_3 = r_{-3} + 3r_{-1} + 3r_1 + r_3,$$
$$\cdots.$$

In general, if we let $Zr_k = r_{k+1}$ and $Z^{-1}r_k = r_{k-1}$, then

$$h_k = (Z + Z^{-1})^k r_0.$$

Notice that both the Toeplitz and Hankel matrices are positive definite and symmetric.

The polynomials arising from the Toeplitz matrix are orthogonal with respect to a measure over the unit circle while the polynomials arising from the Hankel matrix are orthogonal with respect to a measure on the real interval $[-2, 2]$. The relationship between these two sets of polynomials goes as follows [14], [42]:

Let $q_j(x)$ and $p_j(z)$ be orthogonal polynomials with respect to the measure on the interval $[-2, 2]$ and the measure on the unit circle, respectively. Then

$$q_j(x) = C_j\left(p_{2j}(z) + z^{2p}p_{2j}\left(\frac{1}{z}\right)\right)z^{-j}$$

where C_j is a constant and $x = z + z^{-1}$.

It can be shown, using results from the theory of Padé approximation, that the minimizing polynomial sought to solve (2.1) has a closed-form expression in terms of values of the discrete Fourier transform of the data sequence s and Vandermonde determinants of the $n + p$ roots of unity (this involves a combination of results from [5], [20]).

4. Numerical examples and a priori analysis of the method. The essence of the described method comes from the symmetric Lanczos method [18], [36], which is typically used to estimate eigenvalues of symmetric matrices. In this case, the symmetric matrix is $Z+Z^{-1}$, which has a completely trivial eigenstructure—the eigenvectors are the columns of the discrete Fourier transform matrix and the eigenvalues are of the form $2\cos(2k/N)$ where $N=n+p$. The data vector s plays the role of the initial vector in the Lanczos algorithm.

Empirically, and to some extent analytically, it is known that the approximate eigenvalues that emerge from the Lanczos method roughly fall into two groups:

- For small values of p, the eigenvalues corresponding to eigenvectors of $Z+Z^{-1}$ that are "dominant" in s are estimated;
- For large values of p, the extreme eigenvalues of $Z+Z^{-1}$ emerge.

The meanings of "small" and "large" are relative to the problem, and what happens between large and small values is quite arbitrary. In our case, we must definitely consider the case where $p=5$ or less and $n=100$ or more as qualifying as circumstances where p is small. Hence, our applications suggest that as p gets too large, the information that begins to emerge is for the most part irrelevant, since the larger eigenvalues of $Z+Z^{-1}$ begin appearing and these are trivially known. The quantities of interest are precisely the harmonic components of the data vector s, and so p must be small for the method to work best.

In the following examples, $n=1{,}000$, and the first and last ten entries of s were 0 (to conform with the zero padding discussed earlier). For $10<j<990$,

$$s_j=5.5\cos(1.3j)+5.5\cos(0.2j)+1.7\cos(2.5j)+0.5w_j$$

where w_j were independent samples from the uniform distribution on the interval $[-1,1]$.

It is important to note that in these examples, the form of the data, s, is not meant to suggest a stochastic model, although that would be possible. The data has been selected to display some obvious dominant harmonic components.

Recall from § 2 that the reciprocal of the product of the β_j coefficients was a measure of the minimal error

$$\min\|r(\Omega)s\|^2=E_j,\qquad r\text{ monic},\quad \text{degree}(r)=j,$$

and that a heuristic monitor of the fit was monotone decay of E_j. When E_j begins to increase, the correct "order" of the fit has been surpassed.

We now list in Table 1 the results of the computations.

TABLE 1

Order $(=p)$	Arccosine (0.5 eigenvalues)	Relative E_p
1	0.93905	0.826
2	1.57031, 0.49533	0.985
3	2.50044, 1.30003, 0.20218	0.103
4	2.50456, 1.58907 1.29791, 0.20115	0.128
5	2.51664, 2.35310, 1.30101 1.00463, 0.20059	0.137
6	2.72209, 2.49510, 1.57326 1.29930, 0.73411, 0.20024	0.131

Note that the error is not monotonic and that at the correct model order, a steep drop in the error was observed. For $p = 3$, the dominant frequencies were obtained to three significant digits. As the order increases past 3, the frequencies computed gradually but distinctly cease being reasonable approximations to the three dominant frequencies in the data. This trend and the fluctuating value of E_j about the value 0.13 continues.

One feature of this example is that the frequencies enjoy significant separation and have amplitudes that are of the same order of magnitude. Our next example has two closely spaced frequencies, and we see that the method does not separate them well. Consider

$$s_j = 3.5 \cos(0.2j) + 4.0 \cos(2.7j) + 1.1 \cos(0.4j) + 0.3 w_j$$

where w_j are once again independent samples from the uniform distribution on $[-1, 1]$ (see Table 2).

TABLE 2

Order $(=p)$	Arccosine (0.5 eigenvalues)	Relative E_p
1	1.6178	1.87
2	2.6989, 0.2255	0.13
3	2.6999, 1.2371, 0.22136	0.16
4	2.7004, 2.1377	0.15
	0.6126, 0.2136	

The steep drop in the residual from $p = 1$ to $p = 2$ suggests that $p = 2$ is the correct model order, although the data itself was generated by three harmonic terms. The point is that 0.2 and 0.4 are quite close and have been observed as one component.

The final example differs from the above two in that the random component is large relative to the size of the data vector itself. In fact, the signal-to-noise ratio is

$$\frac{\|w\|}{\|s\|} = 0.17,$$

so that almost 20% of the data is noise (here w is the vector of white noise terms, w_j).

The data is (see Table 3)

$$s_j = \cos(1.3j) + \cos(2.5j) + 0.6 w_j;$$

TABLE 3

Order $(=p)$	Arccosine (0.5 eigenvalues)	Relative E_p
1	1.8248	1.08
2	2.4822, 1.2715	0.42
3	2.5017, 1.3083, 0.6245	0.31
4	2.5070, 1.0838	0.33
	1.2981, 0.4238	

In spite of the large noise component, the method has successfully detected two harmonic components with two significant digits after rounding. For an extremely fast and simple procedure, this must be regarded as significant evidence of the method's power.

More work on the analysis of this method is currently in progress. For example, most data of the type handled here is the result of discrete sampling of continuous data. How should the sampling rate be selected to optimize the behavior of the symmetric lattice algorithm? Although the Lanczos method for eigenvalue estimation has been the subject of extensive study, the behavior for small p, which is precisely the case of interest here, has never been adequately addressed. This is primarily because typically one is interested in spectral information about the underlying matrix and not the spectral content of the initial vector.

The theory described above suggests that any polynomial, say p, in $Z+Z^{-1}$ will define a "lattice algorithm" that gives information about the harmonic components of a data vector. The image of the unit circle under the polynomial then becomes the primary object of interest because this determines how the eigenvalues of $p(Z+Z^{-1})$, which are the images under p of the real parts of the Nth roots of unity, are distributed and the Lanczos method for small p becomes important again. Work in these directions is currently in progress.

5. Summary. We have presented a method for estimating the dominant harmonic components in a sequence of data. The method requires about $4np$ operations to compute a symmetric tridiagonal matrix whose eigenvalues are the cosines of twice the p approximating frequencies where the data consists of n sample values.

Section 1 motivates the form that the approximation takes, including a discussion of other approaches. Section 2 contains a detailed derivation of the method and an explicit listing of the main part of the algorithm. To summarize, let s be the n-vector of data padded with zeros as described in § 2. We then execute the algorithm in (2.2) for the desired value of p which is the number of frequencies sought. This gives a symmetric tridiagonal matrix whose eigenvalues are then computed. The desired frequencies are the arccosines of $\frac{1}{2}$ the computed eigenvalues.

Section 4 contains some numerical examples with comments on the behavior of the method.

REFERENCES

[1] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1969.

[2] G. E. Box and G. M. Jenkins, *Time Series Forecasting and Control*, Holden-Day, San Francisco, 1970.

[3] J. P. Burg, *Maximal entropy spectral analysis*, Ph.D. dissertation, Stanford Univ., Stanford, CA, 1975.

[4] G. Cybenko, *A general orthogonalization method with applications to time series analysis and signal processing*, Math. Comp., 40 (1983), pp. 323-366.

[5] ———, *Restrictions of normal operators, Padé approximation, and autoregressive time series*, SIAM J. Math. Anal., 15 (1984), to appear.

[6] ———, *Moment problems and low rank Toeplitz approximations*, Circuits, Systems Signal Process., 1 (1982), pp. 345-366.

[7] P. J. Davis, *Interpolation and Approximation*, Dover, New York, 1975.

[8] ———, *Circulant Matrices*, John Wiley, New York, 1976.

[9] R. De Meersman, *A method for least squares solutions of systems with a cyclic coefficient matrix*, J. Comp. Math., 1 (1975), pp. 51-54.

[10] R. E. Dubroff, *The effective autocorrelation function of maximum entropy spectra*, Proc. IEEE, 63 (1975), pp. 1622-1623.

[11] J. Durbin, *The fitting of time-series models*, Rev. Internat. Inst. Statist., 28 (1960), pp. 233-243.

[12] L. Fejer, *Über die Lage der Nullstellan von Polynomen, die aus Minimumforderungen gewisser Art entspringen*, Math. Ann., 85 (1922), pp. 41-48.

[13] B. S. Garbow et al., *Matrix Eigensystem Routines*—EISPACK *Guide*, Springer-Verlag, New York, 1977.

[14] L. Y. Geronimus, *Orthogonal Polynomials*, Consultants Bureau, New York, 1961.

[15] G. H. Golub and R. J. LeVeque, *Extensions and uses of the variable projection algorithm for solving nonlinear least squares problems*, Technical Report SU 236 P 30-60, Dept. Computer Science, Stanford Univ., Stanford, CA, 1979.

[16] G. H. GOLUB AND V. PEREYRA, *The differentiation of pseudoinverses and nonlinear least squares problems whose variables separate*, SIAM J. Numer. Anal., 12 (1975), pp. 571–592.

[17] G. H. GOLUB, *Numerical methods for solving least squares problems*, Numer. Math., 7 (1965), pp. 206–216.

[18] G. H. GOLUB AND C. VAN LOAN, *Advanced Matrix Computations*, Johns Hopkins Press, Baltimore, MD, 1983.

[19] D. M. GOODMAN AND E. K. MILLER, *A note on minimizing the prediction error when the zeros are restricted to the unit circle*, IEEE Trans. Acoust. Speech Signal Process., 30 (1982), pp. 503–505.

[20] W. B. GRAGG, *The Padé table and its relation to certain algorithms of numerical analysis*, SIAM Rev., 14 (1972), pp. 1–62.

[21] K. L. HIEBERT, *An evaluation of mathematical software that solves nonlinear least squares problems*, ACM Trans. Math. Software, 7 (1981), pp. 1–16.

[22] F. ITAKURA AND S. SAITO, *Digital filtering techniques for speech analysis and synthesis*, in Conference Record, 7th International Congress on Acoustics, Paper 25 C 1, 1971.

[23] T. KAILATH, S. Y. KUNG AND M. MORF, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.

[24] D. W. KAMMLER, *Prony's method for completely monotonic functions*, J. Math. Anal. Appl., 57 (1977), pp. 560–570.

[25] S. Y. KUNG, *A Toeplitz approximation method and some applications*, in International Symposium on Mathematical Theory of Networks and Systems, Santa Monica, CA, August 1981.

[26] C. LANCZOS, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards Sect. B, 45 (1950), pp. 225–280.

[27] ———, *Applied Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1956.

[28] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[29] N. LEVINSON, *The Wiener* RMS (*root mean square*) *error criterion in filter design and prediction*, J. Math. Phys., 25 (1947), pp. 261–278.

[30] K. MAKHOUL, *Linear prediction: A tutorial review*, Proc. IEEE, 63 (1975), pp. 561–580.

[31] ———, *A class of all-zero lattice digital filters: properties and applications*, IEEE Trans. Acoust. Speech Signal Process., 4 (1978), pp. 304–314.

[32] S. L. MARPLE, *Spectral analysis by Pisarenko and Prony methods*, in Proceedings of 1979 IEEE International Conference on Acoustics, Speech Signal Processing, Washington, DC, 1979, pp. 159–161.

[33] J. L. MASSEY, *Shift-register synthesis and* BCH *decoding*, IEEE Trans. Inform. Theory, 15 (1969), pp. 122–127.

[34] M. MORF, *Doubling algorithms for Toeplitz and related equations*, preprint.

[35] M. R. OSBORNE, *Some special nonlinear least squares problems*, SIAM J. Numer. Anal., 12 (1975), pp. 571–592.

[36] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[37] V. F. PISARENKO, *The retrieval of harmonics from a covariance function*, Geophys. J. Roy. Astronom. Soc., 33 (1973), pp. 347–366.

[38] R. DE PRONY, *Essai experimentale et analytique*, J. Ecole Polytech., 1 (1775), pp. 24–76.

[39] A. RALSTON AND P. RABINOWITZ, *A First Course in Numerical Analysis*, McGraw-Hill, New York, 1978.

[40] H. RAMSIN AND P. A. WEDIN, *A comparison of some algorithms for the nonlinear least squares problem*, BIT, 17 (1977), pp. 72–90.

[41] J. RISSANEN, *Algorithms for triangular decomposition of block Hankel and Toeplitz matrices*, Math. Comp., 27 (1973), pp. 147–154.

[42] G. SZEGO, *Orthogonal Polynomials*, AMS Colloquium Publications, vol. 23, American Mathematical Society, Providence, RI, 1959.

[43] W. F. TRENCH, *An algorithm for the inversion of finite Toeplitz matrices*, J. Soc. Indust. Appl. Math., 12 (1964), pp. 515–522.

[44] D. W. TUFTS AND R. KUMARESAN, *Improved Spectral Resolution* II, in Proceedings of 1980 IEEE International Conference on Acoustics, Speech and Signal Processing, Denver, CO, 1980, pp. 592–597.

[45] G. WALKER, *On periodicity in series of related terms*, Proc. Royal Soc. London Ser. A, 131A (1931), p. 518.

[46] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, London and New York, 1965.

[47] G. U. YULE, *On a method of investigating periodicities in disturbed series, with special reference to Wolfer's sunspot numbers*, Philos. Trans. Roy. Soc. London Ser. A, 226A (1927), pp. 267–298.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

AN EFFICIENT TECHNIQUE FOR THE COMPUTATION OF STABLE BIFURCATION BRANCHES*

H. WEBER†

Abstract. This paper deals with an application of the multi-grid iteration for large sparse linear systems to the problem of computing stable branches of solutions of nonlinear eigenvalue problems which bifurcate from simple eigenvalues. The theoretical background of the algorithm considered here is the *selective* Picard iteration. We present numerical results for difference approximations of nonlinear elliptic eigenvalue problems in one and two space dimensions. They confirm the efficiency of the algorithm proposed here.

Key words. multi-grid method, numerical solution, bifurcation, stability, nonlinear elliptic eigenvalue problem

1. Introduction. Recently there has been a widespread interest in the numerical solution of nonlinear eigenvalue and bifurcation problems, cf. e.g. [1], [19], [26]. Important results concerning the finite-dimensional approximation of nonlinear parameter-dependent equations were obtained e.g. in [3], [5], [22], [32]. These results give a good insight in the behavior of the solutions of finite element and finite difference approximations in the neighborhood of turning and bifurcation points. On the other hand there has been some progress in the computational solution of the finite-dimensional nonlinear systems arising via discretization near such singular points, cf. e.g. [12], [19], [31]. However, in this context, only few publications dealt with the efficient solution of large, sparse systems arising from discretizations of partial differential equations. We mention [8], [17], [23], [24], [25].

Georg's generalized inverse iteration [11] could also be used in connection with a nested iteration and a fast solver or a linear multi-grid code, i.e. in a manner similar to the approach discussed here.

In this paper we describe a multi-grid technique for the computation of stable branches of solutions of nonlinear eigenvalue problems which bifurcate from the trivial solution at simple eigenvalues. The algorithm is based on the selective iteration scheme analyzed by Scheurle [27], [28] and on the nested iteration together with some elementary bifurcation theory.

For the linear elliptic problems to be solved one can use a fast elliptic solver or a linear multi-grid iteration method. The class of multi-grid methods that we use here is based on work by Brandt [4] and Hackbusch [13], [15]. The multi-grid method has some very desirable properties. For certain elliptic operators on an n by n grid it computes the approximate solution to truncation error accuracy in $O(n^2)$ arithmetic operations and $O(n^2)$ storage. It seems natural to use multi-grid methods for solving nonlinear eigenvalue problems, too. However there are various different approaches possible, due to the variety of different problems arising in this area. Compare the papers [8], [17], [23], [25].

In § 2 of this paper we state some important facts concerning the selective iteration for bifurcation problems. The behavior of finite-dimensional approximations to simple bifurcation problems and some useful results of bifurcation theory are discussed in § 3. In § 4 we describe the multi-grid algorithm for computing stable branches and in § 5 we present numerical results.

* Received by the editors August 5, 1982, and in revised form February 17, 1983.

† Rechenzentrum und Fachbereich Mathematik der Johannes Gutenberg-Universität Mainz, Postfach 3980, D-6500 Mainz 1, Federal Republic of Germany.

2. The selective iteration procedure for bifurcation problems. Let X be a real Banach space. We consider a nonlinear mapping $F: \mathbb{R}\times X \to X$ satisfying

$$\text{(A1)} \qquad F \in C^2(I \times V, X),$$

where $I \subset \mathbb{R}$ is a suitable interval with $\lambda_0 \in I$ and V is a certain neighborhood of $0 \in X$. Assume that

$$\text{(A2)} \qquad F(\lambda, 0) = 0 \quad \text{for } \lambda \in I,$$

i.e. $(\lambda, 0)$ is the trivial solution of

$$\text{(2.1)} \qquad x = F(\lambda, x).$$

Furthermore, assume

(A3) 1 is a simple eigenvalue of $F_x(\lambda_0, 0)$. All other spectral points are contained in a ball with radius strictly less than 1 about the origin.

Let ϕ be a normed eigenvector of $F_x^0 := F_x(\lambda_0, 0)$ corresponding to the eigenvalue 1 and Z be a complement of span $\{\phi\}$ in X. Suppose that

$$\text{(A4)} \qquad F_{\lambda x}(\lambda_0, 0)\phi \notin R(F_x^0 - \text{id}).$$

Here R denotes the range of a linear operator. By N we shall denote the nullspace.

THEOREM 1. *Under the assumptions* (A1)–(A4) *the point* $(\lambda_0, 0)$ *is a simple bifurcation point of equation* (2.1). *The nontrivial solutions of* (2.1) *in* $I \times V$ *form a branch* $(\lambda(\varepsilon), x(\varepsilon))$, $|\varepsilon| \leqq \varepsilon_0$, *where* $x(\varepsilon) = \varepsilon\phi + \varepsilon\psi(\varepsilon)$ *holds and* $\lambda: [-\varepsilon_0, \varepsilon_0] \to \mathbb{R}$, $\psi: [-\varepsilon_0, \varepsilon_0] \to Z$ *are continuously differentiable functions,* $\lambda(0) = \lambda_0$, $\psi(0) = 0$.

For a proof see [9].

By $\gamma(\lambda)$ we denote the simple positive eigenvalue of $F_x(\lambda, 0)$ with $\gamma(\lambda_0) = 1$. It is well known (cf. [10]) that $\gamma(\lambda)$ crosses the unit circle with "nonvanishing velocity" $\gamma'(\lambda_0) \neq 0$, if (A4) is valid.

We call a fixed point z of a mapping $G: X \to X$ *stable*, if $\rho(F'(z)) < 1$ holds. By ρ we denote the spectral radius. A solution (λ, x) of (2.1) is called *stable* (*unstable*), if x is a (is not a) stable fixed point of $F(\lambda, \cdot)$. The natural iteration method (Picard-iteration) for solving (2.1) is

$$\text{(2.2)} \qquad x^{[n+1]} = F(\lambda, x^{[n]}), \qquad n = 0, 1, 2, \cdots.$$

THEOREM 2. *Let* (A1)–(A4) *be valid and assume that* $\gamma'(\lambda_0) > 0$. *Then the trivial solution of* (2.1) *is stable for* $\lambda < \lambda_0$ *and unstable for* $\lambda > \lambda_0$. *Suppose that* $\lambda'(\varepsilon) \neq 0$ *for* $\varepsilon \neq 0$. *Then the supercritical bifurcating solutions obtain stability from the trivial solution, whereas subcritical solutions are unstable.*

For the proof see [10] and [28].

For the convergence of (2.2) in the case of supercritical branching we have the following result of Scheurle [28].

THEOREM 3. *Let* (A1)–(A4) *be valid and* $\gamma'(\lambda_0) > 0$. *Assume that* $\lambda'(\varepsilon) > 0$ *for* $\varepsilon > 0$ *and* $\lambda'(\varepsilon) < 0$ *for* $\varepsilon < 0$. *Then, for small* $|\lambda - \lambda_0| > 0$ *there is a neighborhood* U *of* $0 \in X$, *depending on* λ, *such that the sequence* $\{x_n\}$ *defined by* (2.2) *converges to a stable fixed point of* $F(\lambda, \cdot)$ *for all* $x^{[0]} \in U \backslash D$, *where* D *is a proper submanifold of* U.

A similar result holds in the case of transcritical bifurcation where we have convergence of course only to the supercritical part of the branch.

In applications one frequently has the following setting. X and Y are Banach spaces, $X \subset Y$. Stationary solutions of many physical problems are described by equations of the type

$$Lu = G(\lambda, u), \qquad u \in D(L) \subset X. \tag{2.3}$$

Here $L: D(L) \to Y$ is a closed linear operator and one has $G: \mathbb{R} \times X \to Y$, $G(\lambda, 0) = 0$ for all λ. Often L has a compact or bounded inverse $L^{-1}: Y \to X$. Then (2.3) is equivalent to

$$u = L^{-1}G(\lambda, u) =: F(\lambda, u), \qquad u \in X. \tag{2.4}$$

The right-hand side defines a nonlinear operator on X satisfying the conditions of the above theorems. Thus in a neighborhood of a bifurcation point the iterative algorithm

$$\begin{aligned} &\text{Compute } u^{[n+1]} \text{ from} \\ &Lu^{[n+1]} = G(\lambda, u^{[n]}), \quad u^{[n+1]} \in D(L), \quad n = 0, 1, 2, \cdots \end{aligned} \tag{2.5}$$

yields the stable solutions of (2.3) in the sense indicated above.

A physically relevant criterion for stability is $\sigma(L - G_u(\lambda, u)) \subset \mathbb{C}^+ = \{z \in \mathbb{C} | \operatorname{Re}\{z\} > 0\}$. For ordinary differential equations it guarantees the stability of a solution u with respect to solutions of the evolution equation

$$\frac{du}{dt} + Lu - G(\lambda, u) = 0. \tag{2.6}$$

Generalizations to parabolic systems were given in [21]. In general, very little can be said about the spectrum of $L - G_u(\lambda, u)$ if $\sigma(L^{-1}G_u(\lambda, u))$ is known and vice versa. However, for an important class of nonlinear elliptic eigenvalue problems the numerical and physical notion of stability coincide, cf. [27]. The problems treated in this paper belong to this class.

From the arguments in [27], [28] it is clear that the iteration scheme (2.2) is also applicable to problems where bifurcation occurs from a nontrivial solution $u = u(\lambda)$ if the above stability conditions are satisfied. This holds of course for the discretized problems, too. However, it should be mentioned that discretized bifurcation problems generally exhibit numerically perturbed bifurcation, i.e. the true bifurcation is destroyed and one obtains two nonintersecting half branches, eventually having turning points, cf. [3], [5]. Under certain circumstances this will not cause difficulties for the iteration process (2.2) but it is a situation different from the one discussed below, where we focus on bifurcation from the trivial solution which is not destroyed by discretization.

3. Finite-dimensional approximations to bifurcation problems. Let X and Y be real Banach spaces. Consider the equation

$$Lu - f(\lambda, u) = 0, \tag{3.1}$$

where $L \in \mathscr{L}(X, Y)$, $f: \mathbb{R} \times X \to Y$ smooth, $f(\lambda, 0) = 0$ for all $\lambda \in \mathbb{R}$. We assume that

$$\begin{aligned} &N(L - f_u(\lambda_0, 0)) = \operatorname{span}\{\phi\}, \quad \phi \neq 0, \\ &R(L - f_u(\lambda_0, 0)) = \{y \in Y, \langle \psi, y \rangle = 0\}, \quad 0 \neq \psi \in Y', \\ &\langle \psi, f_{u\lambda}(\lambda_0, 0)\phi \rangle \neq 0. \end{aligned} \tag{3.2}$$

Y' denotes the dual space of Y and $\langle \cdot, \cdot \rangle$ the duality pairing. Under these hypotheses λ_0 is a simple bifurcation point of (3.1). The branching solutions have the form $u(\varepsilon) = \varepsilon\phi + \varepsilon^2 w(\varepsilon)$, $\lambda(\varepsilon) = \lambda_0 + \varepsilon\eta(\varepsilon)$, cf. [20].

Now consider a finite-dimensional approximation of (3.1) of the form, say

$$L_h u_h - f_h(\lambda, u_h) = 0, \tag{3.3}$$

where $L_h \in \mathcal{L}(X_h, Y_h)$, $f_h : \mathbb{R} \times X_h \to Y_h$ smooth, $f_h(\lambda, 0) = 0$. X_h and Y_h are finite dimensional Banach spaces with $\dim X_h = \dim Y_h$, h is a real discretization parameter tending to zero, cf. e.g. [3], [5]. Under appropriate consistency and stability requirements (cf. [3], [5], [22], [32]) one obtains that the discrete problems (3.3) also exhibit simple bifurcation from the trivial solution at points λ_{0h}, $\lim_{h\to 0} \lambda_{0h} = \lambda_0$. For the bifurcating curves, error estimates of the form

$$\begin{aligned} &\|\Delta_h u(\varepsilon) - u_h(\varepsilon)\|_h \leqq C_1 h^r, \\ &|\lambda(\varepsilon) - \lambda_h(\varepsilon)| \leqq C_2 h^r, \end{aligned} \qquad |\varepsilon| \leqq \varepsilon_0 \tag{3.4}$$

frequently are valid, where r is the order of consistency of the discretization scheme employed and $\Delta_h : X \to X_h$ is the usual matching operator. Figure 1 shows a typical diagram of the quantitative behavior of approximations of simple bifurcation problems.

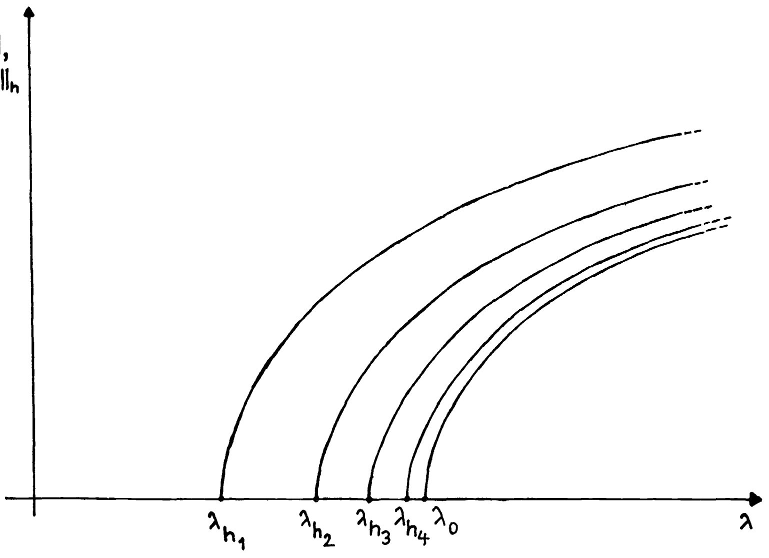


FIG. 1. *Bifurcation diagram of continuous and discrete problems for* $h_1 > h_2 > h_3 > h_4$.

We now turn to the asymptotic expansion of the solutions of a simple bifurcation problem which is useful also for numerical reasons. We assume that the nonlinearity f of problem (3.1) has the form

$$f(\lambda, u) = f_u(\lambda, 0)u + Q(\lambda, u) + R(\lambda, u), \tag{3.5}$$

with

$$Q(\lambda, u) = \frac{1}{(p+1)!} \frac{\partial^{p+1}}{\partial u^{p+1}} f(\lambda, 0)(u)^{p+1}$$

and

$$R(\lambda, u) = O(\|u\|^{p+2}), \quad \text{uniformly in } \lambda.$$

Let Z be a complement of span $\{\phi\}$ in X. Then we have the next theorem.

THEOREM 4. *Under the assumptions* (3.2) *and* (3.5) *the bifurcating solutions* $(u(\varepsilon), \lambda(\varepsilon))$ *satisfy*

$$u(\varepsilon) = \varepsilon(\phi + \varepsilon^p v_0) + O(\varepsilon^{p+2}), \qquad (3.6)$$
$$\lambda(\varepsilon) = \lambda_0 + \varepsilon^p \eta_0 + O(\varepsilon^{p+1}),$$

where

$$\eta_0 = -\frac{\langle \psi, Q(\lambda_0, \phi)\rangle}{\langle \psi, f_{u\lambda}(\lambda_0, 0)\phi\rangle}$$

and $v_0 \in Z$ *is uniquely determined by*

$$(L - f_u(\lambda_0, 0))v_0 = f_{u\lambda}(\lambda_0, 0)\phi\eta_0 + Q(\lambda_0, \phi).$$

Proof. This follows from the arguments in [20].

If the nonlinearity has special properties, the asymptotic expansion (3.6) may proceed in powers of ε^p, e.g. in the (scalar) case $f(\lambda, u) = \lambda \sin u$, where $p = 2$; we refer to [20].

We shall apply this result to the discrete problems (3.3). For the sake of simplicity we consider here only the case of symmetric operators $L_h - f_{hu_h}(\lambda_{0h}, 0)$, i.e. $\psi_h = \phi_h$, $\|\phi_h\|_h = 1$. If the discrete problems fulfill conditions corresponding to (3.2) and (3.5) (in most cases, these properties are inherited from the continuous problems), we have

$$u_h(\varepsilon) = \varepsilon(\phi_h + \varepsilon^p v_{0h}) + O(\varepsilon^{p+2}), \qquad (3.7)$$
$$\lambda_h(\varepsilon) = \lambda_{0h} + \varepsilon^p \eta_{0h} + O(\varepsilon^{p+1}),$$

with

$$\eta_{0h} = -\frac{\langle \phi_h, Q_h(\lambda_{0h}, \phi_h)\rangle_h}{\langle \phi_h, f_{hu_h\lambda}(\lambda_{0h}, 0)\phi_h\rangle_h}. \qquad (3.8)$$

Under moderate requirements concerning the discretizations one may obtain convergence of the η_{0h} to η_0.

4. The algorithm.

4.1. The nested iteration procedure. The nonlinear eigenvalue problem (3.1) is replaced by its finite-dimensional analogues (3.3). The kind of the discretization is arbitrary. One may use, for example, finite differences or finite elements. Referring to a sequence of stepsizes

$$h_0 > h_1 > h_2 > \cdots > h_{l-1} > h_l > \cdots \quad (l\text{: level number}), \qquad (4.1)$$

we also write

$$L_l u_l - f_l(\lambda, u_l) = 0, \qquad u_l \in X_l, \qquad (4.2)$$

for (3.3) with $h = h_l$. The stepsizes h_l may be chosen more general, but we consider the case

$$h_l = 2^{-l} h_0 \qquad (l \in \mathbb{N}). \qquad (4.3)$$

The norm in X_l will be denoted by $\|\cdot\|_l$.

In the following we assume that the linear operator L_l is symmetric, positive definite and (identifying it with a matrix) sparse. As a simple example consider the problem

$$-\Delta u = f(\lambda, u) \quad \text{on } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega,$$

where $\Omega = (0, 1) \times (0, 1)$, $f(\lambda, 0) = 0$ and its usual five-point discretization.

The discrete Picard iteration on level l has the form

$$L_l u_l^{[n+1]} = f_l(\lambda, u_l^{[n]}), \qquad n = 0, 1, 2, \cdots. \tag{4.4}$$

If the assumptions w.r.t. the eigenvalues of $L_l^{-1} f_{lu_l}(\lambda_{0l}, 0)$ are satisfied, cf. (A1)–(A3), then the iteration scheme (4.4) will be convergent for almost all initial values $u_l^{[0]}$.

For solving the linear equations arising in (4.11) the linear multi-grid algorithm could be used. However it is well known (cf. [28]) that the selective iteration (4.11)—in spite of its interesting theoretical properties—is not an effective tool for solving bifurcation problems computationally. Convergence will be linear and too slow, especially near the bifurcation point, since $\rho(L_l^{-1} f_{lu_l}(\lambda_l(\varepsilon), u_l(\varepsilon))) \to 1$ for $\varepsilon \to 0$, and of course if $|\lambda - \lambda_{0l}|$ increases.

For getting an effective algorithm one has to assure that only a very small number of calls of the linear multi-grid code on the finest grid, i.e. on level l, is necessary. This leads to the design of a *nested* iteration procedure.

(4.5)
λ fixed, $u_{l_0}^{[0]}$ given, $l_0 \geqq 0$,
solve (4.2) on level l_0 by (4.4), result: $u_{l_0}^*$,
for $k := l_0 + 1(1)l$: $u_k^{[0]} := q_k u_{k-1}^*$, solve (4.2) on level k by (4.4), result: u_k^*.

Here $q_k : X_{k-1} \to X_k$ is a higher order interpolation operator.

Algorithm (4.5) has been implemented using the multi-grid method for solving the linear problems in every step. It worked satisfactorily, but only relatively far from the bifurcation point. For λ in the vicinity of λ_0 it is not very useful, i.e. it is not faster than the simple iteration (4.4) on level l. The reason for this will become clear if we inspect Fig. 1: the parametrization of the different discrete branches by the same λ-scale is not adequate. This holds especially in the case of partial differential equations where the step size h cannot be made very small. If one wants to solve (4.2) on level l for a fixed λ, λ near λ_0, one has to use different values of λ, say λ^i, on the other, lower levels $i, i < l$; in the case of supercritical bifurcation as in Fig. 1 one should choose $\lambda^0 < \lambda^1 < \cdots < \lambda^l = \lambda$.

With the aid of the asymptotic representations of the bifurcating branches we can easily compute reasonable values of the λ^j's. For given $\lambda = \lambda^l$ we solve (see (3.6)) the equation

$$\lambda^l = \lambda_{0l} + \varepsilon^p \eta_{0l}$$

with respect to ε:

$$\varepsilon = \varepsilon_\lambda = \sqrt[p]{\frac{\lambda - \lambda_{0l}}{\eta_{0l}}}.$$

Here ε_λ has to be chosen real, of course. Set

$$\lambda^j = \lambda_{0j} + \varepsilon_\lambda^p \eta_{0j}, \qquad j = 0, 1, \cdots, l-1. \tag{4.6}$$

This is, of course, equivalent to the introduction of ε as a new (uniform) parameter valid near the bifurcation point.

It is not necessary to compute the η_{0j}'s very accurately, as experiments have shown. Additionally, one has a good starting vector

$$u_{l_0}^{[0]} = \varepsilon \phi_{l_0}$$

on the level l_0 which is the coarsest level for the nested iteration. If the distance $|\lambda - \lambda_0|$ increases we suggest using

$$\tilde{\lambda}^j = \lambda + \tau(\lambda^j - \lambda), \qquad 0 \leqq \tau < 1,$$

instead of λ^j in the nested iteration. This means that the distances between the $\tilde{\lambda}_j$ and λ become smaller now, according to the behavior of the discrete solutions for increasing $\lambda - \lambda_{l0}$. See Fig. 1. We summarize this approach as shown.

Nested iteration with λ-correction.

$$\lambda > \lambda_{0l},\ \phi_{l_0},\ l_0 \geqq 0,\ p \in \mathbb{N},\ \eta_{00}, \cdots, \eta_{0l} \text{ given}$$

$$\varepsilon_\lambda := \sqrt[p]{\frac{\lambda - \lambda_{0l}}{\eta_{0l}}}, \qquad \lambda^k := \lambda_{0k} + \varepsilon_\lambda^p \eta_{0k}, \quad k = l_0, l_0+1, \cdots, l \quad (\lambda^l = \lambda)$$

$$(4.7) \qquad \text{for } k := l_0(1)l: \begin{cases} u_k^{[0]} := \begin{cases} \varepsilon_\lambda \phi_{l_0}, & k = l_0, \\ q_k u_{k-1}^*, & k \neq l_0, \end{cases} \\ \text{for } i := 1, 2, \cdots : u_k^{[i+1]} := L_k^{-1} f_k(\lambda^k, u_k^{[i]}), \\ \text{result of the iteration: } u_k^*. \end{cases}$$

For the convergence of (4.7) we have the following result.

THEOREM 5.

(I) *Assume that for the problem $u = F(\lambda, u)$ with $F(\lambda, u) = L^{-1} f(\lambda, u)$ the hypotheses* (A1)–(A4) *are valid and that there is a supercritical stable branch of solutions of* (3.1) *corresponding to $\varepsilon > 0$.*

(II) *For the discrete problems* (3.2) *approximating* (3.1) *let the assumptions of* § 3 *be valid, especially assume that* (3.4) *and* (3.6) *hold uniformly with respect to $h = h_k$, $k = l_0, \cdots, l$, for $0 < \varepsilon \leqq \varepsilon_0$.*

Furthermore assume that:

(III) $\rho(L_k^{-1} f_{ku_k}(\lambda_k(\varepsilon), u_k(\varepsilon))) \leqq c_0(\varepsilon) < 1$ *for* $0 < \varepsilon \leqq \varepsilon_0$, *for all* $k, l_0 \leqq k \leqq l, c_0$ *independent of* k;

(IV) *for all* $u \in X$, $u^k := \Delta_k u \in X_k$ $(\Delta_k := \Delta_{h_k}: X \to X_k)$ *we have* $\|q_k u^{k-1} - u^k\|_k \leqq d_1 \|u\| \cdot h_k^\alpha, \alpha > 0$, *uniformly w.r.t.* k, $l_0 + 1 \leqq k \leqq l$;

(V) $\|q_k v_{k-1}\|_k \leqq d_2 \|v_{k-1}\|_{k-1}$ *for all* $v_{k-1} \in X_{k-1}$, $k = l_0 + 1, \cdots, l$, d_2 *independent of* k;

(VI) $\lambda_k(\varepsilon)$ *is strictly monotone increasing for* $0 < \varepsilon \leqq \varepsilon_0$, $l_0 \leqq k \leqq l$ *and* $\|u_k(\varepsilon_\lambda) - u_k[\lambda^k]\|_k \leqq d_3 \varepsilon_\lambda$ *uniformly w.r.t.* k, $l_0 \leqq k \leqq l$, *and* $0 < \varepsilon_\lambda \leqq \varepsilon_0$. *Here we use brackets* $[\cdot \cdots]$ *to indicate the λ-dependence of* u_k.

Then, if ε_λ and h_l are sufficiently small, the nested iteration process (4.7) *converges on all levels $l_0, \cdots, l$ and we have* $\lim_{i \to \infty} u_l^{[i]} = u_l[\lambda]$.

Proof. According to (I), (II) and (VI) the discrete problems possess supercritical branches emanating from $(0, \lambda_{0k})$ corresponding to $\varepsilon > 0$. Due to (III) and (VI) we have

$$(4.8) \quad \rho(L_k^{-1} f_{ku_k}(\lambda, u_k[\lambda])) \leqq c_0^*[\lambda] < 1 \quad \text{for } \lambda_{0k} < \lambda \leqq \lambda_{0k}^* := \lambda_{0k} + \varepsilon_0^p \eta_{0k}, \quad l_0 \leqq k \leqq l.$$

For the starting value $u_{l_0}^{[0]} = \varepsilon_\lambda \phi_{l_0}$ on the coarsest level l_0 of the nested iteration we find from (II) and (VI) the estimate

$$\|u_{l_0}^{[0]} - u_{l_0}[\lambda^{l_0}]\|_{l_0} \leqq \|u_{l_0}[\lambda^{l_0}] - u_{l_0}(\varepsilon_\lambda)\|_{l_0} + c_1 \varepsilon_\lambda^{p+1} \leqq d_3 \varepsilon_\lambda + c_1 \varepsilon_\lambda^{p+1}.$$

Ostrowski's theorem now implies the convergence on level l_0 for $\lambda - \lambda_{0l_0}$ sufficiently small, i.e. ε_λ sufficiently small.

From (4.8) it is also clear that the iteration process converges at the levels $l_0 + 1, \cdots, l$ if the starting values are sufficiently near to $u_k[\lambda^k]$, $k = l_0 + 1, \cdots, l$. This will be shown inductively. Due to (VI) and $u_{k-1}^* = u_{k-1}[\lambda^{k-1}]$ we have

$$\begin{aligned} \|u_k[\lambda^k] - u_k^{[0]}\|_k &\leqq d_3 \varepsilon_\lambda + \|u_k(\varepsilon_\lambda) - q_k u_{k-1}[\lambda^{k-1}]\|_k \\ &\leqq d_3 \varepsilon_\lambda + \|u_k(\varepsilon_\lambda) - \Delta_k u(\varepsilon_\lambda)\|_k + \|\Delta_k u(\varepsilon_\lambda) - q_k \Delta_{k-1} u(\varepsilon_\lambda)\|_k \\ &\quad + \|q_k\{\Delta_{k-1} u(\varepsilon_\lambda) - u_{k-1}(\varepsilon_\lambda)\}\|_k + \|q_k\{u_{k-1}(\varepsilon_\lambda) - u_{k-1}[\lambda^{k-1}]\}\|_k. \end{aligned}$$

Assumptions (II), (IV), (V) and (VI) then imply

$$\|u_k[\lambda^k]-u_k^{[0]}\|_k \leqq d_3\varepsilon_\lambda + C_1 h_k^r + d_1\|u(\varepsilon_\lambda)\| h_k^\alpha + d_2 2^r C_1 h_k^r + d_2 d_3 \varepsilon_\lambda$$

for $0<\varepsilon_\lambda \leqq \varepsilon_0$. Thus, for sufficiently small ε_λ and h_l, the iteration process converges at all levels k, $l_0<k\leqq l$. This finishes the proof.

The numerical solution of the linear systems with matrices of coefficients L_k may be performed by different choices of algorithms (we focus here on elliptic equations only): by *fast elliptic solvers*, such as Buneman's algorithm [7] or the reduction method of Schröder and Trottenberg [29], or on the other hand by *multi-grid* methods (cf. e.g. [4]).

In the author's opinion fast elliptic solvers are very valuable for elliptic equations on certain classes of simple domains, but multi-grid methods are more generally applicable, i.e. to elliptic equations on almost arbitrarily general domains, cf. Stüben [30], where a FORTRAN program for this purpose is described. In the following we will mention some basic facts and results on multi-grid methods for linear problems.

4.2. The multi-grid method for linear problems. Let us consider the linear problem

$$L_l u_l = f_l, \tag{4.9}$$

under the conditions (4.1) and (4.3). The levels l and $l-1$ are connected by the *restriction*

$$r_l: X_l \to X_{l-1}$$

and the *prolongation*

$$p_l: X_{l-1} \to X_l.$$

p_l is usually different from q_l which has been introduced above.

By $u \to G_l(u,f)$ we denote the result of a *smoothing* step. For elliptic problems there are different choices possible. We mention Jacobi and Gauss–Seidel iteration, SOR and the uncomplete LU-decomposition, cf. [15].

The characteristic feature of the multi-grid method is the combination of a smoothing step and *coarse-grid correction*. During the smoothing step the defect is not necessarily decreased but smoothed. By the following correction step the discrete solution is improved by means of an auxiliary equation on a coarser grid. In fact this equation has to be of the same structure and sparsity pattern. Well readable introductions to multi-grid methods are [4], [15], [18]. The following ALGOL-like program describes the multi-grid algorithm for the linear problem (4.9). It performs *one* iteration at the level l.

```
**procedure** multigrid($l, u, f$);
**value** $l$;
**integer** $l$; **array** $u, f$;
**comment** $l$: actual level number
          $\nu, \mu$: numbers of smoothing steps
          $\gamma$: number of calls of multigrid at level $l-1$
          $u = \overset{i}{u}_l$ as input, $u = \overset{i+1}{u}_l$ = next iterate as output
          $f = f_l$ right-hand side at level $l$;
**if** $l = 0$ **then** $u := L_0^{-1} * f$ **else**
  **begin integer** $j$; **array** $v, d$;
  **for** $j := 1$ **step** 1 **until** $\nu$ **do** $u := G_l(u, f)$;
  $d := r * (L_l * u - f_l)$; $v := 0$;
```

```
        for $j := 1$ step 1 until $\gamma$ do multigrid$(l-1, v, d)$;
        $u := u - p * v$;
        for $j := 1$ step 1 until $\mu$ do $u := G_l(u, f)$;
    end multigrid.
```

Usual values of (ν, μ) and γ are $(1, 0)$, $(2, 0)$, $(1, 2)$ and $\gamma = 1, 2$. The convergence of the multi-grid algorithm for linear elliptic problems has been analyzed, e.g. in [16].

We shall assume henceforth that the convergence of the multi-grid algorithm for the linear problem (4.9) is established.

For theoretical purposes a fixed strategy as in the above ALGOL-like program is useful. For the design of effective programs an adaptive strategy might be preferable. It could be as follows (cf. [4]). Transfer to a coarser grid when the ratio of the residual norm of the current iterate to the residual norm a sweep earlier is greater than some tolerance η, and transfer to a finer grid when the ratio of the residual norm of the current iterate to the residual norm on the next finer grid is less than another tolerance δ.

We are now ready to replace the application of L_k^{-1} in (4.7) by s steps of the multi-grid iteration, i.e. by s calls of multigrid, $s \geqq 1$. We formulate the final algorithm as an ALGOL-like program.

```
procedure multibif$(l, l_0, \tau, \text{nmax}, s, \lambda, \lambda_0, \eta_0, p, \phi_0, u)$;
integer $l, l_0, p, s$; real $\tau, \lambda$;
array $\lambda_0, \eta_0, \phi_0, u$; integer array nmax;
comment    $\eta_0$:    vector $(\eta_{0l_0}, \cdots, \eta_{0l})$
           $\phi_0$:    linearized discrete eigenfunction on level $l_0$
           nmax:  vector $(\text{nmax}_{l_0}, \cdots, \text{nmax}_l)$ of numbers of steps of
                  nested iteration on levels $l_0, \cdots, l$
           $u$:     contains result at the end of multibif;
begin
real $\varepsilon$, lam; array $v$; integer $k, i, n$;
$\varepsilon := \sqrt[p]{(\lambda - \lambda_0[l])/\eta_0[l]}$;
for $k := l_0$ step 1 until $l$ do
  begin
  lam $:= \lambda + \tau * (\lambda_0[k] + \varepsilon^p * \eta_0[k] - \lambda)$;
  if $k = l_0$ then $u := \varepsilon * \phi_0$ else $u := q * u$;
  for $n := 1$ step 1 until nmax$[k]$ do
    begin
    $v := u$; for $i := 1$ step 1 until $s$ do multigrid$(k, u, f_k(\text{lam}, v))$;
    end $n$;
  end $k$;
end multibif.
```

For the algorithm outlined above we need exactly the same storage as in the linear case, cf. [16]. If N_j denotes the number of grid points on level j we need

$$2 \sum_{k=0}^{l} N_k \leqq 2N_l/(1-2^{-n})$$

units. n is the number of space dimensions.

From the standard perturbation results valid for contraction mappings we can easily conclude that the iterative method described above results in iteration sequences which are slightly perturbed compared with those of (4.7). The distances between the corresponding iterates, however, can be made arbitrary small by choosing s, the number of multi-grid steps, sufficiently large. On the other hand, it does not make much sense to compute the discrete solutions more accurately then to the level of the discretization error.

A few words should be devoted to the problem of finding numerically λ_{0j} and ϕ_j. This may be done e.g. with the aid of Hackbusch's direct multi-grid approach [14] or by applying the multi-grid method of the second kind, see [15].

5. Numerical results. In this section we present results of calculations with algorithm multibif.

5.1. A one-dimensional example. Consider the parameter-dependent boundary value problem

$$-u''-g(\lambda,u)=0, \qquad u(0)=u(1)=0 \tag{5.1}$$

satisfying $g(\lambda,0)=0$, $\lambda\in\mathbb{R}$. If $g(\lambda,u)=\lambda\tilde{g}(u)$, $\tilde{g}'(0)=c\neq 0$, the points $\lambda_0^{[n]}=n^2\pi^2/c$ are bifurcation points and the corresponding linearized eigenfunctions are $\phi_0^{[n]}=\sin n\pi x$, cf. [9]. (5.1) is discretized as usual by

$$L_h u_h - g_h(\lambda,u_h)=0 \tag{5.2}$$

where

$$L_h = 1/h^2\begin{bmatrix} 2 & -1 & & & 0\\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1\\ 0 & & & -1 & 2\end{bmatrix}, \quad g_h=\begin{bmatrix} g(\lambda,u_1)\\ \vdots \\ g(\lambda,u_{N-1})\end{bmatrix},$$

Here $h=1/N$ denotes the step width, $u_h=(u_1,\cdots,u_{N-1})$, N is assumed to be an even integer. If $g(\lambda,u)=\lambda\tilde{g}(u)$ as above the discrete problem (5.2) has the bifurcation points $\lambda_{0h}^{[n]}=(4/h^2)\sin^2(n\pi h/2)/c$, $n=1,\cdots,1/h-1$ and the corresponding linearized discrete eigenfunctions $\phi_{0h}^{[n]}=(\sin n\pi x)_{x=h,2h,\cdots,1-h}$. We have $|\lambda_{0h}^{[n]}-\lambda_0^{[n]}|=O(h^2)$ for fixed n.

Weiss [32] gave a complete analysis of difference approximations to bifurcation problems for a class of ordinary boundary-value problems, to which (5.1) belongs. As an example for (5.1) we have treated numerically the pendulum equation

$$-u''=\lambda\sin u, \qquad u(0)=u(1)=0,$$

using its discretization (5.2). Weiss' theory [32] implies the existence of an error estimate of the form (3.4) with $r=2$, $\|\cdot\|(\|\cdot\|_h)$ being the (discrete) maximum norm, for the branches bifurcating at $\lambda_{0h}^{[n]}$ ($c=1$). All branches are supercritical. The first one, emanating from $\lambda_{0h}^{[1]}\approx\pi^2$ is stable, both in the physical and the numerical sense, compare paragraph 2. Thus our algorithm is applicable. The order p of the nonlinearity is $p=2$.

The details of the multi-grid method used are:
smoother G: Gauss–Seidel relaxation,
prolongation p: linear interpolation,
interpolation q: quadratic interpolation,
restriction r: weights $\frac14, \frac12, \frac14$.

We always used the fixed strategy $\nu=1$, $\mu=0$, $\gamma=1$. In Table 1 we present numerical results for $l=9$, $h=h_l=1/1024$, $h_0=1/2$, $l_0=3$, i.e. $\lambda_{0h}=9.8695967$, $\eta_{0h}\approx 1.234$ for small h. It turned out that $s=2$ calls of the linear multi-grid code were sufficient.

TABLE 1

λ	M_3	M_4	M_5	M_6	M_7	M_8	M_9	$u(0.5)$
9.9	309	36	4	1	1	1	1	0.1568288
10.0	212	4	2	2	1	1	1	0.3236076
11.0	46	19	15	10	2	5	1	0.9207691
12.0	26	14	12	9	6	4	2	1.2245260
15.0	7	7	6	6	5	4	3	1.7471018
20.0*	7	6	10	5	10	3	10	2.1906631
10.0*	15	3	21	100	31	97	110	0.3236089

Above we have M_i = number of iterates on level i. An asterisk indicates computation with fixed λ. The row 10* is included to demonstrate the benefit of using the λ-correction (4.14).

5.2. A two-dimensional example. We consider the nonlinear eigenvalue problem

$$(5.3) \qquad -\Delta u = \lambda u + f(u) \quad \text{on } \Omega = (0,1)\times(0,1), \qquad u = 0 \quad \text{on } \partial\Omega,$$

with $f(0) = 0$ and $f'(0) = 0$. Problems of this or a similar form arise in MHD-calculations, the theory of elasticity, or in the theory of chemical reactions (cf. e.g. [6]) where also numerical calculations are discussed.

The linearized equation $-\Delta u = \lambda u$, $u = 0$ on $\partial\Omega$, has the eigenvalues $\pi^2(m^2+n^2)$ and eigenfunctions

$$\phi_{mn}(x, y) = \sin m\pi x \cdot \sin n\pi y, \qquad (x, y) \in \Omega, \quad m, n \in \mathbb{N}.$$

We discretize (5.3) by the usual five-point difference star with uniform step width h, $h = 1/N$, N an even integer. It is well known that the eigenvalues of the discrete linearized problem $L_h u_h = \lambda_h u_h$ approximate the first $(N-1)^2$ eigenvalues of the continuous linearized eigenvalue problem. The first discrete eigenvalue is $\lambda_{11}^h = 4/h^2(1-\cos \pi h) = 2\pi^2 + O(h^2)$. As indicated by Beyn's [3] and Kikuchis's [22] results (here a linear finite element approximation leads to the same nonlinear system of equations), the discrete branch emanating from λ_{11}^h approximates the branch of the continuous problem bifurcating at λ_{11}. So it is quite clear that an error estimate of the form (3.4) with $r = 2$ holds, where $\|\cdot\|$ ($\|\cdot\|_h$) is the (discrete) L_2-norm. Choosing the nonlinearity

$$\lambda u - u^3,$$

now leads to a stable supercritical bifurcation from λ_{11} and our algorithm becomes applicable.

The details of the linear multi-grid code used here are:
smoother G: pointwise Gauss–Seidel relaxation,
prolongation p: linear interpolation,
interpolation q: quadratic interpolation,
restriction r: injection.
The program is based on Brandt's subroutines and uses the adaptive strategy, cf. [4].

Table 2 presents some typical results for $l = 6$, $h = h_l = 1/128$, $h_0 = 1/2$, $l_0 = 2$, $\lambda_{0h} = 19.738217$, $\eta_{0h} \approx 0.5625$ for small h.

W_i is the accumulated relaxation work of the iterations on level i where a sweep on the finest grid is taken as the work unit, cf. [4]. τ is the damping parameter introduced before (4.7). It is worthwhile to compare a typical value of the CPU-time (Honeywell–Bull HB 66/80, FORTRAN, single-precision), say for $\lambda = 25$: $\underline{23.35}$ sec

TABLE 2

λ	τ	W_2	W_3	W_4	W_5	W_6	$W=\sum W_i$	$u(0.5, 0.5)$
19.8	1	0.1	0.4	1.2	3.5	12.1	17.3	0.3310478
19.9	1	0.1	0.4	1.3	4.3	28.3	34.5	0.5353342
20.0	1	0.1	1.6	1.5	6.5	28.4	38.1	0.6808751
21.0	1	0.7	0.8	1.5	4.0	12.4	19.4	1.4887676
22.0	1	0.7	0.5	1.1	9.0	14.6	25.9	1.9842003
25.0	1	0.5	0.9	2.6	7.0	15.7	26.7	2.9860564
30.0	0.8	0.5	0.8	2.5	7.9	22.3	34.1	4.0855092

with the CPU-time required for solving Poisson's equation (with the right-hand side $\sin(\sin \pi x \cdot \cos \pi y))$ by the same linear multi-grid code ($h = 1/128$): 17.94 sec (16.7 work units).

The results are in good agreement with those given in [22]. Figure 2 shows the solutions $u(x, 0.5)$ for different λ's. A bifurcation diagram is given in Fig. 3.

5.3. Von Kármán's equations for the simply supported plate. As a less simple example we have treated numerically von Kármán's equations for the buckling of a thin elastic simply supported rectangular plate, which is subject to a compressive

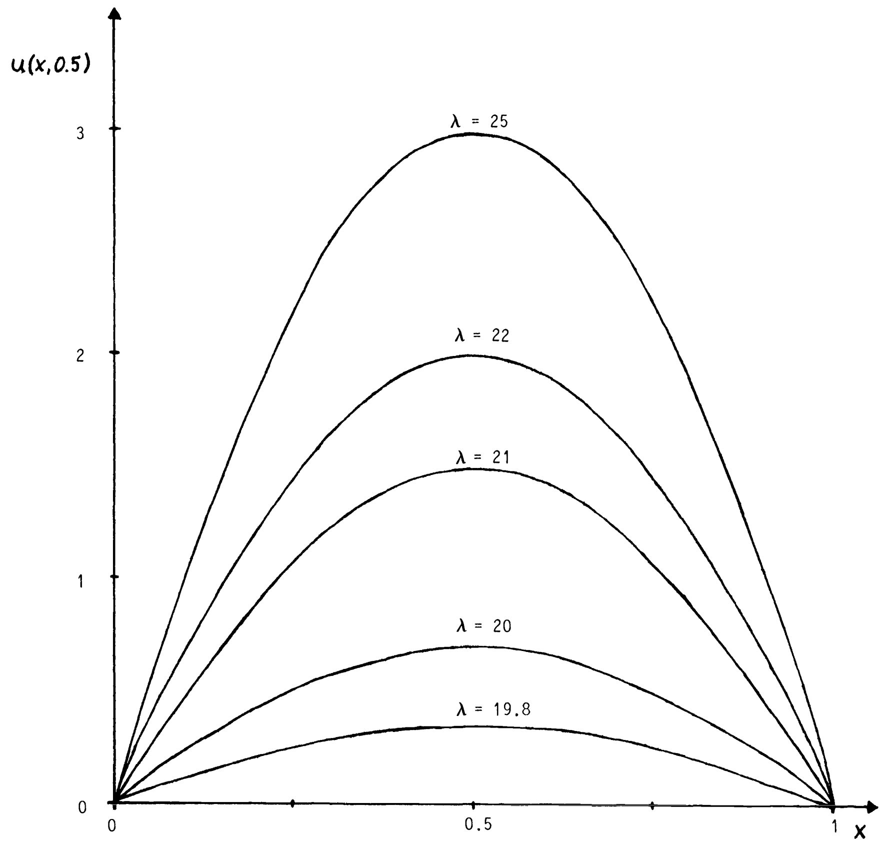


FIG. 2. *Plots of $u(x, 0.5)$ for different values of λ.*

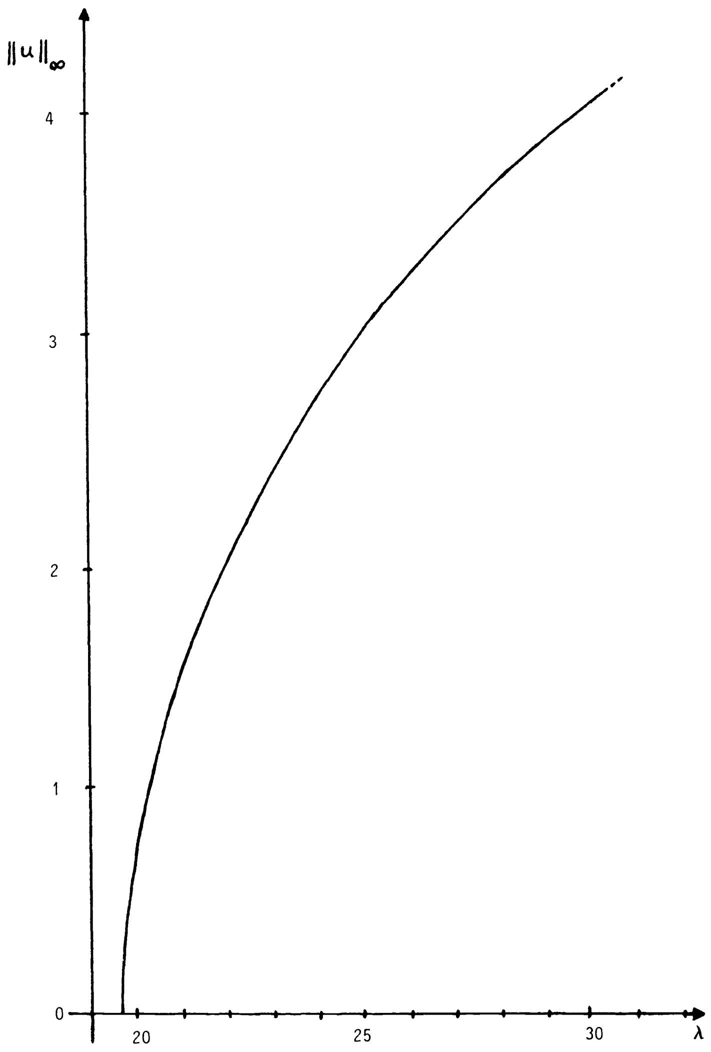


FIG. 3. *Bifurcation diagram* ($h = 1/128$).

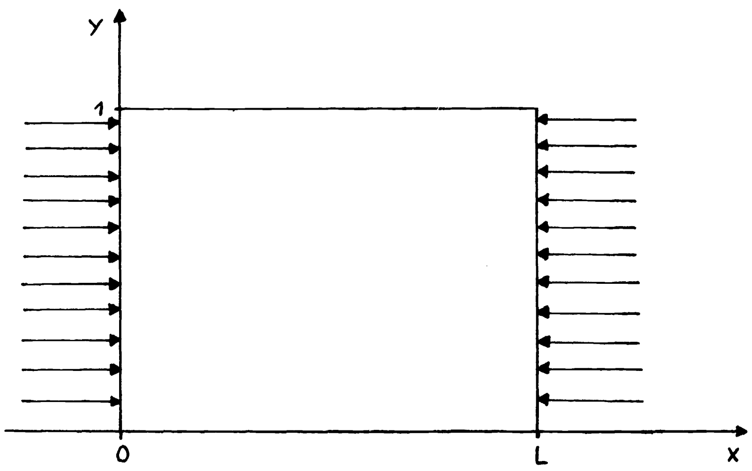


FIG. 4. *Plate problem.*

thrust applied along the short edges (see Fig. 4). In the dimensionless form von Kármán's equations for the deflection $w(x, y)$ and the stress function $f(x, y)$ are (cf. [2])

$$\begin{aligned} &\Delta^2 f = -\tfrac{1}{2}[w, w] \quad \text{on } \Omega, \qquad f = \Delta f = 0 \quad \text{on } \partial\Omega, \\ &\Delta^2 w + \lambda w_{xx} = [f, w] \quad \text{on } \Omega, \qquad w = \Delta w = 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{5.4}$$

where

$$[g, h] = g_{xx}h_{yy} + g_{yy}h_{xx} - 2g_{xy}h_{xy}.$$

λ is proportional to the compressive force. (5.4) may be written in the form

$$\Delta^2 w = -\lambda w_{xx} + C(w) \quad \text{on } \Omega, \qquad w = \Delta w = 0 \quad \text{on } \partial\Omega, \tag{5.5}$$

where C is a certain "cubic" operator. Thus (5.5) belongs to the class of equations treated here.

For numerical reasons, however, we introduce new variables

$$\Phi = \Delta f \quad \text{and} \quad \Psi = \Delta w$$

with Dirichlet boundary conditions. This leads to a mixed formulation of (12), consisting of four second order equations with zero boundary conditions. This problem may be solved iteratively by

$$\begin{aligned} &\Delta\Phi^{[i+1]} = -\tfrac{1}{2}[w^{[i]}, w^{[i]}] \quad \text{on } \Omega, \qquad \Phi^{[i+1]} = 0 \quad \text{on } \partial\Omega, \\ &\Delta f^{[i+1]} = \Phi^{[i+1]} \quad \text{on } \Omega, \qquad f^{[i+1]} = 0 \quad \text{on } \partial\Omega, \\ &\Delta\Psi^{[i+1]} = -\lambda w_{xx}^{[i]} + [f^{[i+1]}, w^{[i]}] \quad \text{on } \Omega, \qquad \Psi^{[i+1]} = 0 \quad \text{on } \partial\Omega, \\ &\Delta w^{[i+1]} = \Psi^{[i+1]} \quad \text{on } \Omega, \qquad w^{[i+1]} = 0 \quad \text{on } \partial\Omega, \quad i = 0, 1, \cdots. \end{aligned} \tag{5.6}$$

Of course a discrete version of (5.6) is actually used. We have again approximated the Laplace operator by the five-point difference star. The brackets [,] on the right-hand sides were evaluated by central differences. The same holds for w_{xx}. For the solution of the linear problems the same multi-grid code as in the above example was used. The eigenvalues and eigenfunctions of the linearized problem are

$$\lambda_{mn} = \frac{\pi^2}{L^2}\left[m + \frac{n^2L^2}{m}\right]^2, \quad w_{mn}(x, y) = \sin\frac{m\pi x}{L}\cdot \sin n\pi y, \quad f_{mn} = 0, \quad m, n \in \mathbb{N}.$$

For the square plate ($L = 1$) we have $\lambda_{11} = 4\pi^2$ and the corresponding discrete first eigenvalue is $\lambda_{11}^h = (4/h^2)(1 - \cos \pi h)^2/\sin^2(\pi h/2) = 4\pi^2 + O(h^2)$. An obvious generalization of multibif is applicable to the case of bifurcation from λ_{11}^h. The branch is table and supercritical. η_{0h} was determined experimentally to have a value near 1.6 for small h. The linear multi-grid code used here was the same as in the previous problem.

We present some typical results which where obtained for the square plate with $l = 6$, $h_l = 1/128$, $h_0 = 1/2$, $l_0 = 3$, $\lambda_{0l} = 39.476561$ (64516 unknowns!):

λ	39.6	39.8	40	41	43
$w(0.5, 0.5)$	0.27347502	0.44754198	0.58047975	1.0025525	1.5308431

45	50	60
1.9213904	2.6671492	3.7659858

The shape of the plate for $\lambda = 45$ is shown in Fig. 5. A typical value of the CPU-time is 97.56 sec for $\lambda = 43$ (HB 66/80).

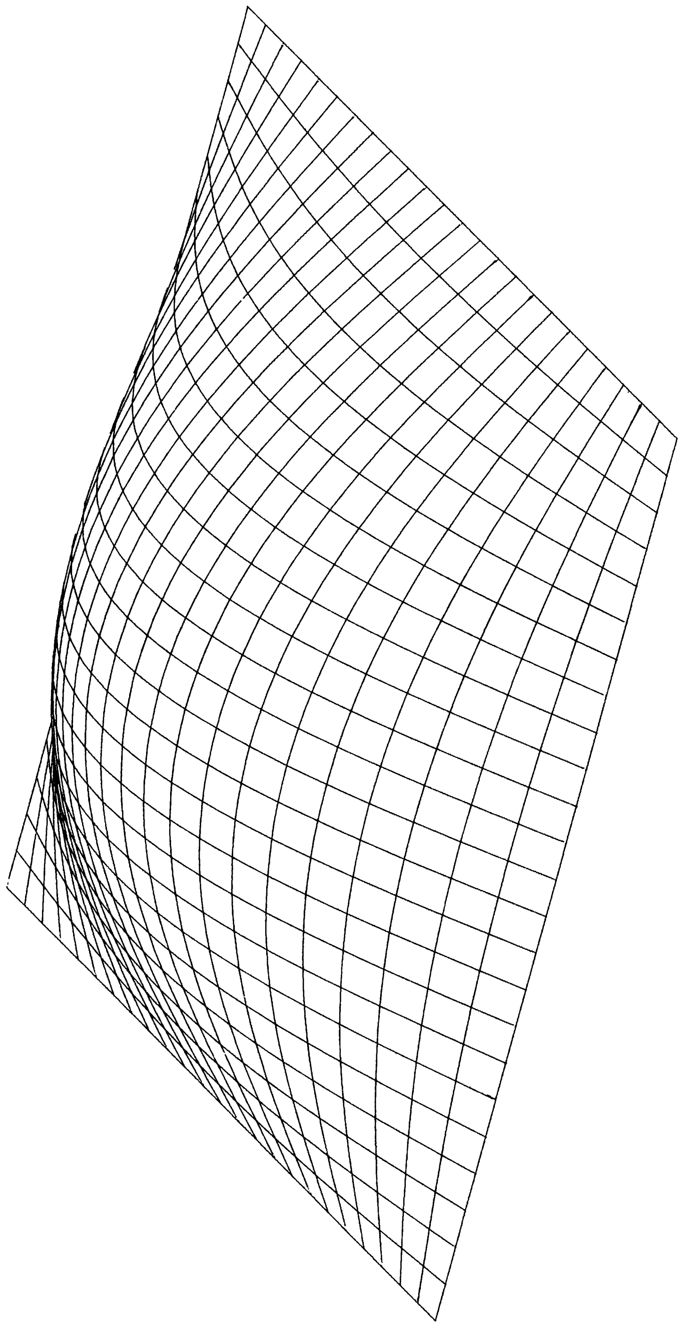


FIG. 5. *Shape of plate for* $\lambda = 45$.

6. Conclusion. We have analyzed a simple but effective method for computing stable branches of solutions of nonlinear elliptic eigenvalue problems. It needs only a multi-grid code for the corresponding linear problem and only the storage which is necessary for this. The CPU time required is of the order of the time necessary for solving Poisson's equation (on the same domain). The algorithm is limited to the computation of stable branches (in the sense of fixed points). For the problems under consideration, however, these are the physically interesting ones. Extensions to compute stable secondary branches and stable branches of perturbed bifurcation problems are possible.

REFERENCES

[1] E. L. ALLGOWER, *A survey of homotopy methods for smooth mappings*, Numerical Solution of Nonlinear Equations, Proceedings, Bremen 1980, E. L. Allgower, K. Glashoff, H.-O. Peitgen, eds., Lecture Notes in Mathematics, 878, Springer-Verlag, Berlin, 1981, pp. 1–29.

[2] L. BAUER AND E. L. REISS, *Nonlinear buckling of rectangular plates*, J. Soc. Ind. Appl. Math., 13 (1965), pp. 603–626.

[3] W.-J. BEYN, *On discretizations of bifurcation problems*, Bifurcation Problems and Their Numerical Solution, H. D. Mittelmann, H. Weber, eds., ISNM Vol. 54, Birkhäuser-Verlag, Basel, 1980, pp. 46–73.

[4] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comput., 31 (1977), pp. 333–390.

[5] F. BREZZI, J. RAPPAZ AND P. A. RAVIART, *Finite dimensional approximation of nonlinear problems, Part* III: *Bifurcation points*, Numer. Math., 38 (1981), pp. 1–30.

[6] P. J. BUDDEN AND J. NORBURY, *A nonlinear elliptic eigenvalue problem*, J. Inst. Math. Appl., 24 (1979), pp. 9–33.

[7] O. BUNEMAN, *A compact non-iterative Poisson solver*, SUIPR Report No. 294 Institute for Plasma Research, Stanford University, California, 1969.

[8] T. F. C. CHAN AND H. B. KELLER, *Arc-length continuation and multi-grid techniques for nonlinear elliptic eigenvalue problems*, SIAM J. Sci. Stat. Comp., 3 (1982), pp. 173–194.

[9] M. G. CRANDALL AND P. H. RABINOWITZ, *Bifurcation from simple eigenvalues*, J. Functional Anal., 8 (1971), pp. 321–340.

[10] ———, *Bifurcation, perturbation of simple eigenvalues and linearized stability*, Arch. Rational Mech. Anal., 52 (1973), pp. 161–180.

[11] K. GEORG, *On the convergence of an inverse iteration method for nonlinear elliptic eigenvalue problems*, Numer. Math., 32 (1979), pp. 60–74.

[12] ———, *On tracing an implicitly defined curve by quasi-Newton steps and calculating bifurcation by local perturbations*, SIAM J. Sci. Stat. Comp., 2 (1981), pp. 35–50.

[13] W. HACKBUSCH, *On the multi-grid method applied to difference equations*, Computing, 20 (1978), pp. 291–306.

[14] ———, *On the computation of approximate eigenvalues and eigenfunctions of elliptic operators by means of a multi-grid method*, SIAM J. Numer. Anal., 16 (1979), pp. 201–215.

[15] ———, *Introduction to multi-grid methods for the numerical solution of boundary value problems*, Lecture Series 1981-6, von Karman Institute for Fluid Dynamics, Rhode Saint Genese, Belgium, 1981.

[16] ———, *On the convergence of multigrid methods*, Beitr. Numer. Math., 9 (1981), pp. 213–239.

[17] ———, *Multigrid solution of continuation problems*, Iterative Solution of Nonlinear Systems of Equations, Proc. Oberwolfach 1982, R. Ansorge, Th. Meis and W. Törnig, eds., Lecture Notes in Mathematics, 953, Springer-Verlag, Berlin, 1982.

[18] P. W. HEMKER, *Introduction to multigrid methods*, Nieuw Archiev voor Wiskunde, 39 (1981), pp. 71–101.

[19] H. B. KELLER, *Numerical solution of bifurcation and nonlinear eigenvalue problems*, Applications of Bifurcation Theory, P. H. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359–384.

[20] H. B. KELLER AND W. F. LANGFORD, *Iterations, perturbations and multiplicities for nonlinear bifurcation problems*, Arch. Rational Mech. Anal., 48 (1972), pp. 83–108.

[21] H. J. KIELHÖFER, *Stability and semilinear evolution equations in Hilbert space*, Arch. Rational Mech. Anal., 59 (1974), pp. 150–165.

[22] F. KIKUCHI, *Finite element approximations of bifurcation problems*, Theoretical and Applied Mechanics, 26 (1976), Univ. Tokyo Press, Tokyo, pp. 37–51.

[23] TH. MEIS, H. LEHMANN AND H. MICHAEL, *Application of the multigrid method to a nonlinear indefinite problem*, Proc. Conference on Multi-Grid Methods, Cologne 1981, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics, 960, Springer-Verlag, Berlin, 1982.

[24] H. D. MITTELMANN, *Multi-grid methods for simple bifurcation problems*, Proc. Conference on Multi-Grid Methods, Cologne 1981, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics, 960, Springer-Verlag, Berlin, 1982.

[25] ———, *A fast solver for nonlinear eigenvalue problems*, Iterative Solution of Nonlinear Systems of Equations, Proc., Oberwolfach 1982, R. Ansorge, Th. Meis and W. Törnig, eds., Lecture Notes in Mathematics, 953, Springer-Verlag, Berlin, 1982, pp. 46–67.

[26] H. D. MITTELMANN AND H. WEBER, *A bibliography on numerical methods for bifurcation problems*, Preprint No. 56, Dept. Mathematics, Univ. Dortmund, 1982, submitted for publication.

[27] J. SCHEURLE, *Selective iteration and applications*, J. Math. Anal. Appl., 59 (1977), pp. 596–616.

[28] ———, *Ein selektives Projektions-Iterationsverfahren und Anwendungen auf Verzweigungsprobleme*, Numer. Math., 29 (1977), pp. 11–35.

[29] J. SCHRÖDER AND U. TROTTENBERG, *Reduktionsverfahren fur Differenzengleichungen bei Randwertaufgaben* I, Numer. Math., 22 (1973), pp. 37–68.

[30] K. STÜBEN, MG01: *A multi-grid program to solve* $\Delta U - c(x,y)U = f(x,y)$ (*in* Ω) $U = g(x,y)$ (*on* $\partial\Omega$) *on nonrectangular bounded domains* Ω, Tech. Rep. IMA 82.02.02, GMD/IMA, Bonn 1982.

[31] H. WEBER AND W. WERNER, *On the numerical solution of some finite-dimensional bifurcation problems*, Numer. Funct. Anal. Optimiz., 3 (1981), pp. 341–366.

[32] R. WEISS, *Bifurcation in difference approximations to two-point boundary value problems*, Math. Comput., 29 (1975), pp. 746–760.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

A FAST, EASILY IMPLEMENTED METHOD FOR SAMPLING FROM DECREASING OR SYMMETRIC UNIMODAL DENSITY FUNCTIONS*

GEORGE MARSAGLIA† AND WAI WAN TSANG†

Abstract. The fastest computer methods for sampling from a given density are those based on a mixture of a fast and slow part. This paper describes a new method of this type, suitable for any decreasing or symmetric unimodal density. It differs from others in that it is faster and more easily implemented, thereby providing a standard procedure for developing both the fast and the slow part for many given densities. It is called the ziggurat method, after the shape of a single, convenient density that provides for both the fast and the slow parts of the generating process. Examples are given for REXP and RNOR, subroutines that generate exponential and normal variates that, as assembler routines, are nearly twice as fast as the previous best assembler routines, and that, as Fortran subroutines, approach the limiting possible speed: the time for Fortran subroutine linkage conventions plus the time to generate one uniform variate.

Key words. random numbers, normal random variables, exponential random variables, ziggurat method, Monte Carlo, simulation

AMS(MOS) subject classifications, mr categories. 65C10, 65C05, 68A55

1. Introduction. It has long been known that, in principle, computer sampling from any given density can be made very rapid; if enough memory space is available and enough attention is paid to details that provide, most of the time, rapid return of the required variate by means of a few operations on a uniform variate [5].

Applications of this principle have led to very fast subroutines for normal and exponential variates, but these implementations have been ad hoc, tailored to the features of each density and each particular machine. The fastest seem to be those based on the rectangle-wedge-tail method of references [6], [8] and described in Knuth [3, p. 211].

This article seeks to improve on this situation by suggesting a general method that is easily applied to any decreasing or symmetric unimodal density $f(x)$. The method leads to exact, even faster, subroutines, using a table of $n+1$ entries, with the user free to choose n on the basis of time and complexity before setting up the table.

The time and space complexity of the algorithm may be inferred from the following outline, in which UNI means a uniform random variable and a table $x_0, x_1, \cdots, x_n$ and constants a, b, c are given:

```
ALGORITHM Z.
1. Choose $j$ uniformly from $\{1, 2, \cdots, n\}$.
2. Form $X = x_j^*$UNI; if $X < x_{j-1}$, return $X$.
   (Some 99% of the time when $n = 256$, 96% when $n = 64$.)
3. Else form $x = (X - x_{j-1})/(x_j - x_{j-1})$, $y =$ UNI.
4. If $y > c - af(b - bx)$, return $b - bx$.
5. If $f(x_j) + y/(nx_j) < f(X)$ return $X$.
6. Else return an $X$ from the tail density, $c'f(x), x > x_n$.
```

The algorithm returns the required variate some 96–99% of the time from step 2, which requires two table look-ups, one multiplication and one uniform variate (since a single uniform variate can provide j in step 1 and the UNI in step 2). If the fast

* Received by the editors November 17, 1981. This research was supported by the National Science Foundation under grant MCS80-05789.

† Computer Science Department, Washington State University, Pullman, Washington 99164.

part of the algorithm, steps 1 and 2, is coded as an assembler subroutine and the remaining, slower, part as a Fortran subroutine, the composite will be very fast—nearly twice as fast as the previous fastest.

The entire algorithm can also be easily implemented as a standard Fortran subroutine, to make it machine independent, but there is a time penalty, primarily because of Fortran linkage conventions, which are more costly than the generating procedure itself. But even as a standard Fortran subroutine the procedure is very fast and general, suited for decreasing or symmetric unimodal densities. The resulting average execution times approach the limit that a standard Fortran random variable subroutine can attain: the time for the linkage conventions plus the time to generate one UNI.

Development of the algorithm is in § 2, followed in § 3 by a straightforward procedure for setting up the table $x_0, x_1, \cdots, x_n$ and the three constants a, b, c needed in the algorithm. The set-up procedure accepts the table size n and the given decreasing density $f(x)$, $x \geqq 0$, or a symmetric unimodal density, in which case a uniform variate on $(-1, 1)$ is used.

Section 5 discusses implementations and gives Fortran programs for normal and exponential variates, but the method provides equally fast subroutines for decreasing or symmetric densities, such as Student's t for given degrees of freedom. Section 5 also gives a general method for handling the tail.

Section 6 discusses timing of subroutines and gives times for assembler and standard Fortran versions of the new method, followed by a summary in § 7.

2. The ziggurat method. This section will describe a method that leads to exceptionally fast algorithms for generating variates with decreasing or unimodal, symmetric density functions. Suppose we are given a decreasing $f(x)$ and want to generate a random variable X with density f. Choose a rectangle of unit area, one that crosses $f(x)$ twice, as pictured in Fig. 1.

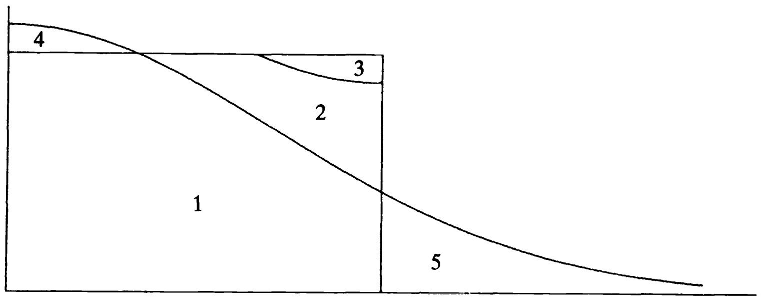


FIG. 1. *Efficient use of a uniform point in a rectangle to get a point under $y = f(x)$.*

This leads to five regions. Region 4 is called the *cap*. The cap is rotated and translated to get the corner region 3, having the same area. Then region 2 has the same area as region 5. (For densities unbounded at the origin, the exact-approximation method [9] may be used to provide a bounded cap.)

Now to generate a variate X with density $f(x)$: Choose (X, Y) uniformly from the rectangle. If (X, Y) is in region 1, return X; if in region 3, return $a - X$; else return a new X from the tail density $c'f(x), x > a$.

This idea is worth implementing if region 1 dominates the rectangle, and we can make it do so by means of what we call a ziggurat density, illustrated for the exponential density e^{-x}, $x > 0$, in Fig. 2. The ziggurat function $z(x)$ is the step function drawn with

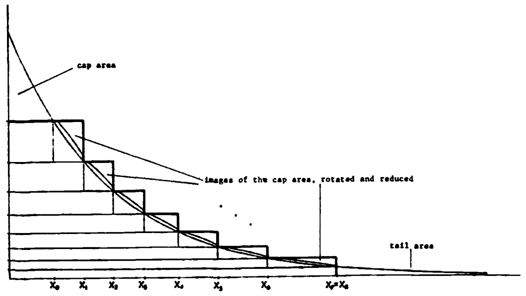


FIG. 2. *Crossing the exponential density twice: a ziggurat density with* 8 *layers of area* 1/8.

heavy lines. It is a density function, made up of n layers, each with area $1/n$. (Drawn with $n=8$—too small to be practical but chosen to provide detail.)

A point (X, Y) is easily generated uniformly under the ziggurat: Choose j uniformly from $\{1, 2, \cdots, n\}$, then (X, Y) uniformly from the jth layer. Most of the time, Y in the appropriate range will not be required, since $X<x_{j-1}$ ensures that the resulting (X, Y) will be under the curve $y=f(x)$.

Each layer of the ziggurat can be represented as in Fig. 3, and this leads to the outline of an algorithm with very fast average execution time:

1. Choose j uniformly from $\{1, 2, \cdots, n\}$.
2. Form $X=x_j^* UNI$; *if* $X<x_{j-1}$, return X.
3. Else choose Y uniformly in the range $f(x_j)<y<f(x_j)+p/x_j$. The resulting (X, Y) is uniform in $B\cup C\cup D$. If in B, return X; if in D, return a multiple of (x_j-X); if in C, return an X from the tail.

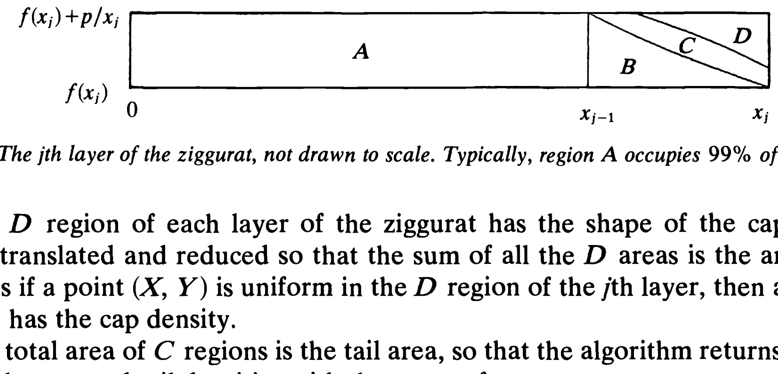


FIG. 3. *The jth layer of the ziggurat, not drawn to scale. Typically, region A occupies* 99% *of the layer.*

The D region of each layer of the ziggurat has the shape of the cap density, rotated, translated and reduced so that the sum of all the D areas is the area of the cap. Thus if a point (X, Y) is uniform in the D region of the jth layer, then a multiple of x_j-X has the cap density.

The total area of C regions is the tail area, so that the algorithm returns a variate X from the cap and tail densities with the proper frequency.

3. The set-up. The ziggurat density is a step function with n layers, each having area $p=1/n$. The abscissas of the steps are $x_0, x_1, \cdots, x_n$. They may be computed by letting x_n be the rightmost solution to $xf(x)=p$ (there will be two solutions); then $x_{n-1}=x_n$ and the remaining x's computed from $f(x_i)=f(x_{i+1})+p/x_{i+1}$ for $i=n-2, \cdots, 1$.

The generating process chooses one of the n layers of the ziggurat at random, say the jth, then forms $X=x_j^*$UNI and returns X if $X<x_{j-1}$. This happens most of the time (99% when $n=256$) and accounts for the speed of the method. Occasionally, further tests must be made after forming $Y=f(x_j)+\text{UNI}^*p/x_j$ so that (X, Y) is uniform in the rectangle at the end of the jth layer—region $B\cup C\cup D$ in Fig. 3.

Details of the additional tests are made easier if the point (X, Y), uniform in $B \cup C \cup D$, is made a point (x, y) uniform in the unit square by means of the transformation

$$x = (X - x_{j-1})/(x_j - x_{j-1}), \qquad y = (Y - f(x_j))x_j/p.$$

When transformed to the unit square, the test regions look like those in Fig. 4.

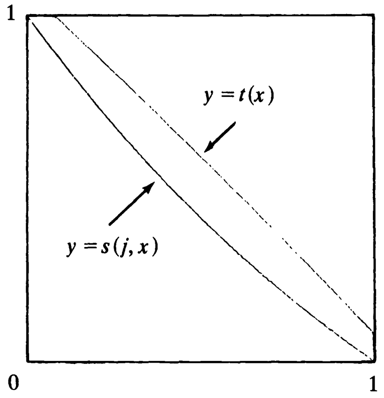


FIG. 4. *The rectangle at the end of the jth layer, transformed to the unit square.*

Then the test $(X, Y) \in B$ is equivalent to $y < s(j, x)$ and $(X, Y) \in D$ is equivalent to $y > t(x)$.

The region $y < s(j, x)$ changes with j:

$$s(j, x) = nx_j[f(x_{j-1} + x(x_j - x_{j-1})) - f(x_j)], \qquad 0 \leqq x \leqq 1.$$

The region $y > t(x)$ does not change with j:

$$t(x) = 1 - \frac{x_0[f(b - bx) - f(x_0)]}{b[f(0) - f(x_0)]}, \qquad 1 - x_0/b \leqq x \leqq 1,$$

with b to be determined so that the area above $y = t(x)$ is correct.

Let the area above $t(x)$ be r. Then r may be expressed in two different ways, providing a formula for the constant b in $t(x)$. Since each of the D regions is mapped into the region $y > t(x)$ of area r in the unit square, each D region must occupy the fraction r of the rectangle $B \cup C \cup D$. Thus the area of the jth D region is $r(1 - x_{j-1}/x_j)p$ and the sum of the D areas must be the cap area:

$$(1) \qquad r(1 - x_0/x_1)p + r(1 - x_1/x_2)p + \cdots + r(1 - x_{n-1}/x_n)p = \text{cap area}.$$

The cap area is $\int_0^{x_0} f(x)\,dx - x_0 f(x_0)$ and integration shows the area above $t(x)$ to be

$$(2) \qquad r = \int_{1-x_0/b}^{1} [1 - t(x)]\,dx = \frac{x_0(\text{cap area})}{b^2(f(0) - f(x_0))}.$$

Combining (1) and (2) provides the formula

$$b^2 = x_0\left(1 - p \sum_{i=1}^{n} \frac{x_{i-1}}{x_i}\right) \Big/ (f(0) - f(x_0))$$

as well as the interesting conclusion: *setting up the ziggurat does not require integration of f.*

We may summarize the ziggurat set-up procedure with this outline.

Given the decreasing density function $f(x)$, $x \geqq 0$, and $p = 1/n$, with n the number of layers:

1. Let x_n be the larger of the two solutions to $xf(x) = p$.
2. Put $x_{n-1} = x_n$ and solve for x_i from the relations

$$f(x_i) = f(x_{i+1}) + p/x_{i+1}, \; i = n-2, \cdots, 2, 1.$$

(See the note below.)

3. Compute

$$b = \left[x_0\left(1 - p \sum_{i=1}^{n} x_{i-1}/x_i\right) \Big/ (f(0) - f(x_0)) \right]^{1/2},$$

$$a = x_0/[b(f(0) - f(x_0))],$$

$$c = 1 + af(x_0).$$

Note. For some densities, putting $x_{n-1} = x_n$ then solving for the remaining x's in step 2 may leave the cap so large that regions B and D of Fig. 3 overlap. This difficulty may be overcome by stacking more than the first two layers of the ziggurat, such as by setting $x_{n-2} = x_{n-1} = x_n$. For example, the normal and some of the t densities require $x_{n-3} = x_{n-2} = x_{n-1} = x_n$ before computing the remaining x's. Thus for some densities the set-up procedure is not completely automatic, and will require user intervention. An alternative method is to transform the cap by the exact-approximation method [9], so that regions B and D never overlap and the boundary of D is nearly linear. Such a device also allows one to use the ziggurat method for unbounded densities.

4. The algorithm. With n, p, a, b, c and $x_0, x_1, \cdots, x_n$ provided by the ziggurat set-up and recalling that $p = 1/n$, Algorithm Z will return a variate with density $f(x)$.

Since the boundary functions in Fig. 4 are nearly linear, we may use a linear squeeze test in order to avoid the need to evaluate f in steps 4 and 5, most of the time. Define

$$c_1 = \min_{\substack{0 \leqq x \leqq 1 \\ 1 \leqq j \leqq n}} [x + s(j, x)], \qquad c_2 = \max_{a - x_0/b \leqq x \leqq 1} [x + t(x)].$$

Then $c_2 - x > t(x)$ and the linear test $x + y > c_2$ will imply $y > c_2 - x > t(x)$. Similarly, $x + y < c_1$ will imply $y < s(j, x)$.

Thus a faster algorithm results from inserting two steps between steps 3 and 4 of Algorithm Z:

```
1.
2.
3.
3.1. If $x + y > c_2$ return $b - bx$.
3.2. If $x + y < c_1$ return $X$.
4.
5.
6.
```

A quadratic squeeze test may speed up the algorithm even more, but the improvement does not justify the complication, unless f is exceedingly difficult to evaluate. When $n \geqq 256$, even the linear squeeze is barely worth it.

5. Implementation. As examples of the ziggurat method, we list below two Fortran subroutines, REXP and RNOR, that return exponential and normal random variables. They are based on the method with $n = 64$. The generating procedure is short, some 15–17 lines of virtually linear code that follows the outline above.

Both of these subroutines must themselves call random number generators (RNG's). We do not address here the problem of providing suitable RNG's but have assumed those based on the conventions set in the McGill Random Number Package "Super-Duper" [7]:

IUNI: a random integer in $[0, 2^{31})$,
IVNI: a random integer in $[-2^{31}, 2^{31})$,
UNI: a random real in $[0, 1)$.

Users who have different in-house RNG's, over possibly different ranges, should be able to replace calls to IUNI, IVNI, and UNI with suitable adaptation to their own system. Better yet is to insert Fortran code that generates the random IUNI and IVNI directly in the fast part of REXP and RNOR, for the subroutines are so fast that the time for the RNG in the fast part is a major determinant of the overall-speed.

EXPONENTIAL SUBROUTINE

```
      FUNCTION REXP(NULL)
C—RETURNS A RANDOM EXPONENTIAL VARIABLE, IGNORES ARGUMENT
      DATA A, B, C, RMAX/4.780222, .2339010, 4.807275, .4656613E-9/
      DATA C1, C2, P, XN/.9130147, 1.055764, .015625, 5.940712/
      REAL V(65)/.2275733, .2961199, .3568076, .4124534, .4645906,
     +.5141596,.5617859,.6079111,.6528617,.6968884,.7401897,
     +.7829269,.8252345,.8672267,.9090027,.9506499,.9922470,
     +1.033865,1.075572,1.117430,1.159497,1.201832,1.244491,
     +1.287529,1.331001,1.374964,1.419475,1.464591,1.510374,
     +1.556887,1.604196,1.652370,1.701488,1.751625,1.802871,
     +1.855318,1.909067,1.964230,2.020929,2.079300,2.139492,
     +2.201675,2.266037,2.332792,2.402185,2.474495,2.550045,
     +2.629211,2.712438,2.800248,2.893275,2.992284,3.098219,
     +3.212264,3.335930,3.471187,3.620674,3.788045,3.978562,
     +4.200208,4.465950,4.799011,5.247564,5.940712,5.940712/
C—FAST PART
      I=IUNI(0)
      J=MOD(I,64)+1
      REXP=I*RMAX*V(J+1)
      IF (REXP.LE.V(J)) RETURN
C————————SLOW PART
         X=(REXP-V(J))/(V(J+1)-V(J))
         Y=UNI(0)
         S=X+Y
         IF (S.GT.C2) GO TO 11
         IF (S.LE.C1) RETURN
         IF (Y.GT.C-A*EXP(B*X-B)) GO TO 11
         IF (EXP(-V(J+1))+Y*P/V(J+1).LE.EXP(-REXP)) RETURN
C——————————TAIL PART
              REXP=XN-ALOG(UNI(0))
              RETURN
11          REXP=B-B*X
            RETURN
      END
```

NORMAL SUBROUTINE

```
      FUNCTION RNOR(NULL)
C—RETURNS RANDOM NORMAL VARIABLE, IGNORES ARGUMENT.
      DATA AA,B,C,RMAX/12.37586, .4878992, 12.67706, .4656613E-9/
      DATA C1,C2,PC,XN/.9689279, 1.301198, .1958303E-1, 2.776994/
      REAL V(65)/.3409450, .4573146, .5397792, .6062427, .6631690,
     +.7136974, .7596124, .8020356, .8417227, .8792102, .9148948,
     +.9490791, .9820005, 1.013848, 1.044780, 1.074924, 1.104391,
     +1.133273, 1.161653, 1.189601, 1.217181, 1.244452, 1.271463,
     +1.298265, 1.324901, 1.351412, 1.377839, 1.404221, 1.430593,
     +1.456991, 1.483452, 1.510012, 1.536706, 1.563571, 1.590645,
     +1.617968, 1.645579, 1.673525, 1.701850, 1.730604, 1.759842,
     +1.789622, 1.820009, 1.851076, 1.882904, 1.915583, 1.949216,
     +1.983924, 2.019842, 2.057135, 2.095992, 2.136644, 2.179371,
     +2.224517, 2.272518, 2.323934, 2.379500, 2.440222, 2.507511,
     +2.583466, 2.671391, 2.776994, 2.776994, 2.776994, 2.776994/
C—FAST PART
      I=IVNI(0)
      J=MOD(IABS(I),64)+1
      RNOR=I*RMAX*V(J+1)
      IF (ABS(RNOR).LE.V(J)) RETURN
C—————SLOW PART; AA IS A*f(0)
      X=(ABS(RNOR)-V(J))/(V(J+1)-V(J))
      Y=UNI(0)
      S=X+Y
      IF (S.GT.C2) GO TO 11
      IF (S.LE.C1) RETURN
      IF (Y.GT.C-AA*EXP(-.5*(B-B*X)**2)) GO TO 11
      IF (EXP(-.5*V(J+1)**2)+Y*PC/V(J+1).LE.EXP(-.5*RNOR**2)) RETURN
C———————TAIL PART: .3601016 IS 1./XN
   22         X=.3601016*ALOG(UNI(0))
              IF (-2.*ALOG(UNI(0)).LE.X**2) GO TO 22
   33         RNOR=SIGN(XN-X,RNOR)
              RETURN
   11    RNOR=SIGN(B-B*X,RNOR)
         RETURN
         END
```

We have chosen to provide the tables in the subroutines through DATA statements, rather than generate them on first entry to the routine. By doing this we avoid the need to use or even look at the arguments of the functions REXP(NULL) and RNOR(NULL). The integer arguments are ignored in the subroutines, but they must be included, since Fortran conventions require that a function subprogram have at least one argument, used or not.

The values in the tables may be generated separately through recursive use of two double precision values DX and DY. Here are Fortran segments that generate the tables, given the initial value DX, the larger of the two solutions to $xf(x) = 1/n$, and DY = f(DX).

```
      DOUBLE PRECISION DX/5.940712090024669/,DY/.2630156076110249D-02/
      XN=DX
      DO 1 I=1,63
         IF (I.LE.2) V(63+I)=XN
         DY=DY+1.DO/(64.DO*DX)
         DX=-DLOG(DY)
1        V(64-I)=DX
```

```
     DOUBLE PRECISION DX/2.776994269662875/,DY/.1687976115474328D-1/
     XN=DX
     DO 2 I=1,61
         IF (I.LE.4) V(61+I)=XN
         DY=DY+1.DO/(64.DO*DX)
         DX=DSQRT(-2DO*DLOG(DY/.7978845608028653DO))
2        V(62-I)=DX
```

As the above examples show, there is a standard procedure for the set-up and generation parts of the ziggurat method, given a particular density—except for one thing: the tail part. This part of the overall procedure appears to require the user's intervention in what is otherwise an automatic process.

But even this part of the generating procedure can be made systematic, at least for most of the densities encountered in practice, in the following way: given the tail density $c'f(x)$, $x \geqq x_n$, choose a function $g(t)$ from the class $e^{-\beta t}$ or $(1+bt)^{-\beta}$ such that

$$f(x_n+t) \leqq f(x_n)g(t), \qquad t \geqq 0.$$

Then $G(x)=\int_x^\infty g(t)\,dt/D$ is easily inverted in terms of standard Fortran functions, where $D=\int_0^\infty g(t)\,dt$. To generate X from $c'f(x)$, $x \geqq x_n$, choose uniform U_1, U_2 until

$$U_2 f(x_n) g[G^{-1}(U_1)] \leqq f[x_n+G^{-1}(U_1)],$$

then return $X=x_n+G^{-1}(U_1)$.

The efficiency of this rejection method is $\int_{x_n}^\infty f(t)\,dt/[Df(x_n)]$, so the parameter(s) in $g(t)=e^{-\beta t}$ or $g(t)=(1+bt)^{-\beta}$ should be chosen to minimize D and still have g dominate f.

Examples. The normal tail:

$$f(x)=c'e^{-x^2/2}, \quad x \geqq x_n, \qquad g(t)=e^{-x_n t}.$$

Tail, Student's t with d degrees of freedom:

$$f(x)=c(1+x^2/d)^{-(d+1)/2}, \qquad x \geqq x_n,$$
$$g(t)=(1+bt)^{-d-1}, \qquad b=x_n/(d+x_n^2).$$

Since most of the densities encountered in practice—normal, exponential, gamma, beta, F, t, as well as the stable densities—may be asymptotically, and closely, dominated by a g of the form $e^{-\beta t}$ or $(1+bt)^{-\beta}$, this general method may serve as a uniform procedure for convenient and reasonably fast sampling from tail densities.

6. Timing. Most articles that propose a new method for generating random variables claim that the method is fast, and often specific times are quoted. But many questions arise in assessing such claims: Is the method implemented as a standard subroutine with linkage-convention overhead, or as a faster assembler routine? Are the necessary uniform variates generated within the subroutine or called from separate subroutines with linkage penalties? Is the method for generating uniform variates one of the fast ones, such as multiplicative congruential, or one of the safer, but slower, ones that combine two different methods? If exponential or normal variates are used within the subroutine, how are they generated? What is the effect of the computer used—not only because of possibly faster arithmetic operations and high-speed memory, but the architecture—for example, modular arithmetic for powers of 2 is inherently faster in 2's complement machines.

Some methods are so slow that answers to these questions do not make much difference, but they can make a great difference for methods that approach the limiting speed for a random variable generator.

What is the best possible speed for a method? If it is to be implemented as a standard Fortran subroutine, with linkage-convention overhead, we suggest that the best possible speed on a particular computer can be estimated by timing the subroutine

```
FUNCTION FAST(T)
FAST=UNI(1)
RETURN
END
```

with UNI the name of the uniform generator subroutine.

Such a limiting speed may be estimated by timing the loop

```
  DO 2 I=1,100000
2   X=FAST(1.)
```

then subtracting the time for the nugatory loop

```
  DO 3 I=1,100000
3   X=1.
```

(assuming that the compiler is not so smart that it will compress the latter loop).

Two questions still remain, however. Is the subroutine UNI itself a standard Fortran subroutine or a faster assembler routine, and what is the method for producing the uniform variates?

We contend that comparisons between two methods can be accurate only if both methods are implemented on the same computer, in the same way—as standard or as assembler subroutines—and using the same uniform generator.

Using this criterion, we will compare two methods for generating normal and exponential variates: the ziggurat method and the rectangle-wedge-tail method in the McGill package "Super-Duper" [7], chosen because it was the fastest we knew. None of the published methods such as in [1], [2], [4] provided implementations as simple or as fast as the ziggurat method. Readers may wish to compare these methods on their own systems.

Timing was done on an Amdahl V8, a computer with architecture patterned after that of the IBM 360/370. For the Amdahl V8, the limiting speed for a standard Fortran random variable generator was 6 microseconds (μs.), obtained by timing the above loops, with UNI the assembler routine in the McGill package [7] that generates uniform variates by combining a congruential and a shift-register generator.

The results are in Tables 1 and 2. No comparison could be made between standard Fortran versions, as the rectangle-wedge-tail method was available only as an assembler routine.

TABLE 1
Average speed, in microseconds, for the generation of normal and exponential variates using Fortran callable assembler routines, based on the ziggurat method and the rectangle-wedge-tail method in the Super-Duper package [7]. *For the Amdahl V8 computer.*

	Ziggurat		
	$n=64$	$n=256$	Super-Duper
RNOR	2.74	2.34	3.79
REXP	2.45	2.06	4.04

Table 1 compares the two methods with the fast parts implemented in assembler routines, the uniform variates generated within the subroutine. The slow parts are implemented as standard Fortran subroutines. The ziggurat method is faster and the choice of n has a greater effect than in Table 2.

Table 2 shows the speed, for $n = 64$ and 256, of the ziggurat method implemented as a standard Fortan subroutine. The times there, around 9 μs, should be compared to 6 μs, the limiting speed, on this machine, for generators as standard Fortran subroutines that call the subroutine UNI in the McGill Random Number Package [7].

TABLE 2

Average speed, in microseconds, for the generation of normal and exponential variates using the ziggurat method, implemented as a standard Fortran subroutine in the Amdahl V8.

	$n = 64$	$n = 256$
RNOR	9.53	9.11
REXP	9.49	9.20

Another means to put reported speeds of random variable subroutines into perspective is to compare them with the speeds of built-in Fortran functions for that machine. With the Fortran compiler for our Amdahl V8, the average times for SQRT, ALOG and EXP were 6.7, 7.1 and 8.6 microseconds, respectively, for arguments uniformly spread over (0, 1). Such built-in functions are, of course, written in assembly language, and do not have unnecessary linkage convention costs.

Still another measure of the speed for generating exponential variates is to compare REXP with -ALOG (UNI). For the Amdahl V8, the ziggurat speeds are 2.1 μs for REXP partially in assembler, 9.2 μs for REXP as a standard Fortran subroutine and 12 μs for -ALOG (UNI), with UNI itself a standard Fortran subroutine. If UNI is a fast assembler routine, then -ALOG (UNI) is as fast as the standard Fortran version of the ziggurat REXP, and much more convenient.

7. Summary. For repeated, very rapid sampling from a given decreasing or symmetric unimodal density, the ziggurat method described here is simple and faster than any other we know. If the fast part—$X = x_j^*$UNI; if $X < x_{j-1}$ return X—is implemented as an assembler routine, with the slower part in Fortran, the composite is nearly twice as fast as the previous fastest methods. If the entire procedure is implemented as a standard Fortran subroutine, to make it machine independent, the result, while slower, still approaches the limiting attainable speed: the time for subroutine linkage conventions plus the time to generate one uniform variate.

The method requires setting up a table of $n + 1$ values, which is slow but straightforward. The set-up is easily implemented as part of a Fortran program that accepts the density function and table size as input.

The method is not suited for rapid sampling from a family of densities with shape parameters changing from call to call.

Because of their importance, we think that UNI, RNOR and REXP should be available to Monte Carlo reserchers as very fast, reliable subroutines. These are the bread-and-butter functions for such research, as important to many as the SIN, COS, SQRT, ALOG, EXP functions available—as assembler routines—with every Fortran compiler. Because of the standardized set-up and generating procedure for the ziggurat method, assembler routines for the fast part, and Fortran routines for the remaining, are easily implemented. We urge serious users to consider doing so.

REFERENCES

[1] J. H. AHRENS AND K. D. KOHRT, *Computer methods for efficient sampling from largely arbitrary statistical distributions*, Computing, 26 (1981), pp. 19–31.

[2] A. J. KINDERMAN AND J. G. RAMAGE, *Computer generation of normal random variables*, J. Amer. Statist. Assoc., 71 (1976), pp. 893–896.

[3] D. E. KNUTH, *The Art of Computer Programming*, Vol. II: *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[4] R. A. KRONMAL AND A. V. PETERSON, JR. *Generating normal random variables using the alias-rejection-mixture method*, Proc. ASA 1979 Annual Meeting Computer Section, Washington, DC, 1980.

[5] G. MARSAGLIA, *Expressing a random variable in terms of uniform random variables*, Ann. Math. Statist., 32 (1961), pp. 894–899.

[6] G. MARSAGLIA, M. D. MACLAREN AND T. A. BRAY, *A fast procedure for generating normal random variables*, Comm. ACM, 7 (1964), pp. 4–10.

[7] G. MARSAGLIA et al., *The McGill Random Number Package "Super-Duper,"* School of Computer Science, McGill University, Montreal, 1972 (available from the present authors, together with *How to Use the McGill Random Number Package Super-Duper*).

[8] G. MARSAGLIA, K. ANANTHANARAYANAN AND N. J. PAUL, *Improvements on fast methods for generating normal random variables*, Inform. Process. Lett., 5 (1976), pp. 27–30.

[9] G. MARSAGLIA, *The exact-approximation method for generating random variables in a computer*, J. Amer. Statist. Assoc., to appear.

SIAM J. Sci. Stat. Comput.
Vol. 5, No. 2, June 1984

BIDIRECTIONAL SHOOTING: A STRATEGY TO IMPROVE THE RELIABILITY OF SHOOTING METHODS FOR ODE*

PIETER P. N. DE GROEN† AND MARTIN HERMANN‡

Abstract. A new shooting technique for the numerical treatment of boundary value problems in ordinary differential equations is proposed. It improves the accuracy and robustness of the simple (unidirectional) shooting method by combining the fundamental matrices originating from both endpoints of the interval. The combination is performed such that the smaller singular values of these matrices and the corresponding orthogonal projections are eliminated. The reason for this strategy is that perturbations of a matrix have a much stronger effect on the smaller singular values than on the larger ones.

Key words. two-point boundary value problem, shooting method, bidirectional shooting, singular value decomposition

AMS (MOS) classifications. Primary 65L10, secondary 65F15

1. Introduction. We study shooting methods for boundary value problems in ordinary differential equations from the point of view of how the accuracy and robustness of the basic (simple) shooting procedure can be enhanced. So we shall not deal with segmented shooting methods (multiple shooting), since this is only a repetition of the same argument (see e.g. Keller (1968), (1976), Osborne (1979), Hermann and Berndt (1981)), nor with nonlinear problems, since those generally are reduced to an iteration of linear ones by a linearization technique (see e.g. Roberts and Shipman (1972)).

We consider the boundary value problem

$$\mathscr{L}\mathbf{x}(t) := \dot{\mathbf{x}}(t) - F(t)\mathbf{x}(t) = \mathbf{0}, \qquad a \leqq t \leqq b, \tag{1.1a}$$

$$\mathscr{B}\mathbf{x}(t) := B_a\mathbf{x}(a) + B_b\mathbf{x}(b) = \boldsymbol{\beta}, \tag{1.1b}$$

where $\mathbf{x}(t)$, $\boldsymbol{\beta} \in \mathbb{R}^n$ and $F(t)$, B_a, B_b are $n \times n$ real matrices, and we assume that (1.1) possesses an isolated solution.

We study homogeneous differential equations only, since an inhomogeneous equation

$$\mathscr{L}\mathbf{y}(t) = \mathbf{f}(t) \tag{1.2}$$

can be reduced to homogeneous form. To this aim, we add a component y_{n+1} to the vector $\mathbf{y}$ and consider the system

$$\mathscr{L}\mathbf{y}(t) - \mathbf{f}(t)y_{n+1} = \mathbf{0}, \qquad \dot{y}_{n+1} = 0; \tag{1.3}$$

moreover, to the boundary conditions we add an equation of the form (α arbitrary)

$$\alpha y_{n+1}(a) + (1-\alpha)y_{n+1}(b) = 1. \tag{1.4}$$

The idea of shooting for linear problems with nonseparated boundary conditions is to generate the whole solution manifold of the differential equation and to select from it the solution of the boundary value problem by the associated algebraic equation resulting from the boundary conditions (cf. Keller (1968)). Effectively, in the simple

* Received by the editors June 28, 1982, and in revised form February 24, 1983.

† Departement Wiskunde, Vrije Universiteit Brussel, Pleinlaan 2, B-1050 Brussel, Belgium.

‡ Rechenzentrum, Friedrich-Schiller-Universität Jena, Humboldtstrasse 2, DDR-6900 Jena, German Democratic Republic. The research of this author was supported by a grant of the Belgian Ministry of National Education and Dutch Culture for a stay at the Vrije Universiteit, Brussel.

shooting method we compute from a boundary point, say a, the fundamental matrix $X(t;a)$ such that $X(a;a)=I$ (=identity) by an initial value problem solver. The solution of the boundary value problem is determined by applying the boundary conditions to $X(t;a)\mathbf{d}$ and solving the resulting linear system of algebraic equations for $\mathbf{d}$ by an appropriate numerical method.

For each t the columns of the fundamental matrix $X(t;a)$ form a basis for the solution manifold of the differential equation, and theoretically these columns remain independent forever, although the angles between them may become very small. In practice the columns become spoiled by truncation and roundoff errors, the more as $|t-a|$ grows larger or as the differences between the eigenvalues of the matrix $F(t)$ become bigger. Therefore the system of linear algebraic equations by which we select the solution of the boundary value problem may be perturbed so strongly that the approximation of the solution may become very poor. This puts a severe restriction on the length of an interval in which simple shooting is practicable, namely, that the angles between the columns of $X(t;a)$ not become too small.

Because of the fact that the fundamental matrix is a group with respect to t, it is possible to improve the stability of the simple shooting method by combining the fundamental matrices originating from both endpoints of the interval $[a,b]$. Given constant $n\times n$ matrices M_a and M_b, a matrix Y of the form

$$Y(t):=X(t;a)M_a+X(t;b)M_b \tag{1.5}$$

is again a fundamental matrix, provided M_a and M_b are chosen such that the rank of Y is maximal. Obviously this leaves much freedom in the choice of M_a and M_b to minimize the computational errors in $Y(t)$ and to optimize the condition of the resulting system of algebraic equations from which the solution of (1.1) is determined. Our choice of M_a and M_b is based on the singular value decomposition of $X(b;a)$ and $X(a;b)$, namely such that the smaller singular values of these matrices and the corresponding orthogonal projections are eliminated. The reason for this choice is that perturbations of a matrix have a much stronger effect on the smaller singular values than on the larger ones (cf. Lawson and Hanson (1974, Ch. 4–5)). From numerical experiments we can conclude that this technique of "bidirectional shooting" indeed gives better results and is more reliable than simple (unidirectional) shooting. Obviously the improved reliability has to be paid for by a doubling of the computational effort for the integration.

Although bidirectional shooting is more robust than unidirectional shooting, this method is doomed to fail also if the interval length becomes too large. An obvious generalization is then to use it as the basic algorithm in multiple shooting. Certainly the number of intermediate shooting points and hence the order of the resulting algebraic system will be less than when simple shooting is used. Moreover, the spreading of the singular values offers a good criterion for the selection of shooting points.

Finally we remark that shooting in both directions has been proposed earlier, e.g. in Fox (1962, pp. 61–62). However, there the solution manifolds are generated from both sides until they meet somewhere in the middle of the interval, where they are matched. Hence there is no redundant information generated on the solution manifold, from which the better part can be selected.

Outline of the paper. In § 2 of this paper we formulate the basic results on singular value decomposition of a matrix and perturbations thereof. In § 3 we describe precisely the bidirectional shooting method and in § 4 we report on numerical experiments.

2. Perturbations of the singular value decomposition. In this section we study the influence of perturbations of a matrix on its singular values and the corresponding orthogonal factors.

Each $n \times n$ matrix A can be decomposed (cf. Lawson and Hanson (1974, Ch. 4)) into the product $A = USV^H$, where $S := \operatorname{diag}(\sigma_i)$ is a nonnegative diagonal matrix consisting of the singular values of A and where U and V are orthogonal matrices that diagonalize AA^H $(= US^2U^H)$ and A^HA $(= VS^2V^H)$. Unless explicitly stated otherwise, we shall assume that the singular values in S are ordered in decreasing sense.

A perturbation E of A influences the singular values as follows. Let $\{\tilde{\sigma}_i\}$ denote the singular values of $\tilde{A} := A + E$. Then

$$|\sigma_i - \tilde{\sigma}_i| \leqq \|E\|, \tag{2.1}$$

where $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^n$ and the associated matrix norm; cf. Lawson and Hanson (1974, Ch. 5). The perturbation influences the orthogonal factors U and V too, but, since those matrices are not defined uniquely, the columns of the factors $\tilde{U}$ and $\tilde{V}$ of $\tilde{A}$ $(= \tilde{U}\tilde{S}\tilde{V}^H)$ need not be near to those of U and V. Therefore we must formulate those perturbations in terms of the corresponding orthogonal subspaces.

Let us define the $2n \times 2n$ matrices C, W, T and F by

$$C := \begin{pmatrix} 0 & A \\ A^H & 0 \end{pmatrix}, \quad W := \frac{1}{\sqrt{2}} \begin{pmatrix} U & -U \\ V & V \end{pmatrix}, \quad T := \begin{pmatrix} S & 0 \\ 0 & -S \end{pmatrix}, \quad F := \begin{pmatrix} 0 & E \\ E^H & 0 \end{pmatrix}.$$

Obviously, the symmetric matrix C satisfies the factorization

$$C = WTW^H, \tag{2.2}$$

where W is orthogonal and where T is a diagonal matrix with σ_i and $-\sigma_i$ $(i = 1, \cdots, n)$ on the diagonal. If

$$\sigma_{k-1} > \sigma_k \geqq \cdots \geqq \sigma_l > \sigma_{l+1} \qquad (\text{with } k \leqq l \leqq n),$$

then the kth up to the lth columns of W are an orthonormal basis in the joint eigenspace of C corresponding to the eigenvalues $\{\sigma_k, \cdots, \sigma_l\}$ of C. The orthogonal projection P on this eigenspace is given by

$$P := \frac{1}{2\pi i} \int_\Gamma (C - \lambda I)^{-1}\, d\lambda, \tag{2.3}$$

where Γ is a closed contour in the complex plane enclosing $\{\sigma_k \cdots \sigma_l\}$ and no other eigenvalues of C; cf. Kato (1966, § I 3.5). For the perturbed matrix $\tilde{C} := C + F$ we define analogously $\tilde{W}$, $\tilde{T}$ and $\tilde{P}$.

In order to estimate $P - \tilde{P}$ we define δ by

$$\delta := \tfrac{1}{2} \min \{\sigma_{k-1} - \sigma_k, \sigma_l - \sigma_{l+1}\},$$

which is half of the distance to the rest of the spectrum. We consider the strip $D \subset \mathbb{C}$.

$$D := \{z \in \mathbb{C} \mid \sigma_k + \delta > \operatorname{Re} z > \sigma_l - \delta, |\operatorname{Im} z| < R\}$$

which contains $\{\sigma_k \cdots \sigma_l\}$. If $\|E\| < \delta$, it contains also the eigenvalues $\{\tilde{\sigma}_k \cdots \tilde{\sigma}_l\}$ of the perturbed matrix $\tilde{C}$, according to (2.1). Hence, we can estimate the difference $P - \tilde{P}$

as follows:

$$P-\tilde{P}=\frac{1}{2\pi i}\int_{\partial D}\{(C-\lambda)^{-1}-(C+F-\lambda)^{-1}\}\,d\lambda,$$

$$\|P-\tilde{P}\|\leqq\frac{1}{2\pi}\int_{\partial D}\frac{\|(C-\lambda)^{-1}\|^2\|F\|}{1-\|(C-\lambda)^{-1}\|\|F\|}\,d\lambda.$$

Since $\|E\|=\|F\|$ and $\|(C-\lambda)^{-1}\|=\operatorname{dist}(\lambda,\sigma(C))$, the integrals over the horizontal parts of ∂D vanish in the limit for $R\to\infty$. For the integral on the line $\operatorname{Re}\lambda=\sigma_l-\delta$ we find

$$\int_{\sigma_l-\delta-iR}^{\sigma_l-\delta+iR}\|(C-\lambda)^{-1}\|^2\,d\lambda\leqq\int_{-\infty}^{\infty}\frac{dt}{\delta^2+t^2}=\frac{\pi}{\delta}$$

and on the line $\operatorname{Re}\lambda=\sigma_k+\delta$ we have the same estimate. Hence $P-\tilde{P}$ satisfies

$$\|P-\tilde{P}\|\leqq\frac{\|E\|}{\delta-\|E\|}, \tag{2.4}$$

provided $\|E\|<\delta$.

The orthogonal matrices W and $\tilde{W}$ that diagonalize C and $\tilde{C}$ are not uniquely determined; hence a straightforward estimate of the difference of corresponding columns of W and $\tilde{W}$ is not possible. However, if we denote the ith column of W by $\mathbf{w}_i$, we find from (2.4) the result

$$\left\|\sum_{i=k}^{l}(\mathbf{w}_i\mathbf{w}_i^T-\tilde{\mathbf{w}}_i\tilde{\mathbf{w}}_i^T)\mathbf{x}\right\|\leqq\frac{\|E\|\|\mathbf{x}\|}{\delta-\|E\|}, \tag{2.5}$$

for all $\mathbf{x}\in\mathbb{R}^n$, provided $\|E\|<\delta$. This implies that we do not make a big error in W if we replace the kth up to the lth columns of W by the corresponding columns of $\tilde{W}$, provided the eigenvalues $\{\sigma_k\cdots\sigma_l\}$ are well separated from the other ones.

The matrix W defines uniquely the orthogonal factors U and V in the singular value decomposition of A. Hence, denoting the jth columns of U, $\tilde{U}$, V and $\tilde{V}$ by $\mathbf{u}_i$, $\tilde{\mathbf{u}}_i$, $\mathbf{v}_i$ and $\tilde{\mathbf{v}}_i$ respectively, we find from (2.5) the estimate

$$\left\|\sum_{i=k}^{l}(\mathbf{u}_i\mathbf{v}_i^T-\tilde{\mathbf{u}}_i\tilde{\mathbf{v}}_i^T)\mathbf{x}\right\|\leqq\frac{\|E\|\|\mathbf{x}\|}{\delta-\|E\|} \tag{2.6}$$

for all $\mathbf{x}\in\mathbb{R}^n$. This implies that we do not make a big error, if we replace the kth up to the lth columns of U *and* V by those of $\tilde{U}$ and $\tilde{V}$, provided $\|E\|$ is small with respect to δ.

3. The bidirectional shooting method. Consider the boundary value problem

$$\mathcal{L}\mathbf{x}(t)=\mathbf{0},\quad a\leqq t\leqq b,\qquad \mathcal{B}\mathbf{x}(t)=\boldsymbol{\beta}. \tag{1.1}$$

Define the fundamental matrix $X(t;t_0)$ as the ($n\times n$ matrix) solution of the initial value problem

$$\mathcal{L}X=0,\qquad X(t_0;t_0)=I. \tag{3.1}$$

This fundamental matrix satisfies the group properties

$$\begin{aligned}&\text{(i)}\quad X(t_1;t_2)X(t_2;t_3)=X(t_1;t_3)\quad\text{and}\\&\text{(ii)}\quad X(t_1;t_2)=X(t_2;t_1)^{-1}\end{aligned} \tag{3.2}$$

provided $X(t;\tau)$ is nonsingular for all t and τ. Given constant $n\times n$ matrices M_a and

M_b, each constant vector $\mathbf{c} \in \mathbb{R}^n$ generates a solution of (1.1a) of the form

$$\mathbf{x}(t) = [X(t; a)M_a + X(t; b)M_b]\mathbf{c}, \tag{3.3}$$

provided $X(b; a)M_a + M_b$ (or equivalent $M_a + X(a; b)M_b$) is nonsingular. Moreover, each solution of (1.1a) can be represented in this way.

We select the solution of the differential equations which also satisfies the boundary conditions (1.1b) by inserting (3.3) into (1.1b) and by solving the resulting equation

$$[B_a(M_a + X(a; b)M_b) + B_b(X(b; a)M_a + M_b)]\mathbf{c} = \boldsymbol{\beta}. \tag{3.4}$$

From the analytic point of view M_a and M_b can be chosen arbitrarily if only the nonsingularity condition is satisfied. Numerically the choice influences the condition of the system of equations (3.4) drastically. In practice the matrices $X(b; a)$ and $X(a; b)$, which we shall abbreviate from now on by

$$X_a := X(b; a) \quad \text{and} \quad X_b := X(a; b) = X_a^{-1}, \tag{3.5}$$

are computed numerically and are perturbed by discretization and roundoff errors. If their singular values differ strongly in magnitude, the analysis of the previous section implies that the smaller ones and the corresponding orthogonal projections are approximated very poorly while the larger ones and their projections are still well determined. This motivates us to choose M_a and M_b such that they eliminate the parts of X_a and X_b in (3.4) corresponding to the smaller singular values.

Let $X_a = USV^H$ be the singular value decomposition of X_a, where U and V are orthogonal matrices and S is a positive diagonal matrix, and let us assume that the singular values of S are ordered in *decreasing* sense; from (3.2) we find $X_b = VS^{-1}U^H$. Inserting this into (3.4) we get

$$[B_a(M_a + VS^{-1}U^H M_b) + B_b(USV^H M_a + M_b)]\mathbf{c} = \boldsymbol{\beta}. \tag{3.6}$$

In the product $SV^H M_a$ we can eliminate the smaller singular values of S if we set $V^H M_a$ equal to a diagonal matrix with zeros on the corresponding diagonal places; in $S^{-1}U^H M_b$ we can do the same.

Let us now assume that we want to select with the aid of M_a the k larger singular values of S and to eliminate the others and to do the complementary for S^{-1} with the aid of M_b. Then we can choose

$$V^H M_a = \mu P, \quad U^H M_b = \nu(I - P), \quad P := \begin{pmatrix} I_k & 0 \\ 0 & 0 \end{pmatrix}, \tag{3.7}$$

where I_k is the identity in $\mathbb{R}^k$ and $\mu, \nu \in \mathbb{R}$. Inserting (3.7) into (3.6) we find

$$[B_a V(\mu P + \nu S^{-1}(I - P)) + B_b U(\mu SP + \nu(I - P))]\mathbf{c} = \boldsymbol{\beta}. \tag{3.8}$$

This choice of M_a and M_b implies that $\mu P + \nu S^{-1}(I - P)$ is invertible and that $(\mu P + \nu S^{-1}(I - P))^{-1}(\mu SP + \nu(I - P)) = S$. Hence, the left-hand side of (3.8) is equal to

$$(B_a + B_b X_a)[(\mu P + \nu S^{-1}(I - P))^{-1} V^H]^{-1}\mathbf{c}. \tag{3.9}$$

From the assumption about the unique solvability of (1.1), which implies that $B_a + B_b X_a$ is nonsingular, it follows that (3.8) is uniquely solvable with the choices (3.7) of M_a and M_b.

We remark that proper choices of μ and ν can improve the condition of the system (3.8) if the singular values of X_a differ largely in magnitude. They should be chosen such that the elements of the diagonal matrices

$$\mu P+\nu S^{-1}(I-P) \quad \text{and} \quad \mu SP+\nu(I-P)$$

in (3.8) have the same orders of magnitude.

Let us now assume that the k largest singular values of X_a are well separated from the other ones, and hence that the $n-k$ largest singular values of $X_b=X_a^{-1}$ are, too. Moreover, assume that we have computed approximations $\bar{X}_a$ and $\bar{X}_b$ and that the perturbations $\|\bar{X}_a-X_a\|$ and $\|\bar{X}_b-X_b\|$ are small with respect to this separation. According to the result (2.6) of the previous section, the corresponding projections are well determined. From these computed matrices we have to determine the unknown vector $\mathbf{c}$ in (3.4), and we can do this via the procedure stated above, which is designed to suppress the bad parts of $\bar{X}_a$ and $\bar{X}_b$.

4. Numerical examples. In this section we describe some numerical experiments in which we compare the performance of bidirectional shooting with unidirectional methods (Examples 1–5). All computations were executed on a CDC 170/750 computer in single precision arithmetic carrying a mantissa of 14 significant digits. For the singular value decomposition and the solution of the linear algebraic systems we used routines from the NAG-library.

For the integration we employed standard ODE solvers for nonstiff problems: in experiments 1, 2, 4 and 5, it was a routine from the NAG-library using a variable-stepsize and variable-order Adams method; in experiment 3 we used an extrapolation algorithm based on the midpoint rule with a stabilization proposed by Gragg (see e.g. Hermann and Berndt (1981)). A specialized stiff ODE solver might possibly enhance the results somewhat (following our opinion in the favour of bidirectional shooting). We used the same computed fundamental matrices in the evaluation of both the bidirectional and the unidirectional solutions. Hence, the experimental data give a fair comparison of both methods.

The equations as they are written in the examples are not implemented as such. We transformed them into first order systems, and we reduced inhomogeneous equations into homogeneous ones by using formulae (1.3)–(1.4). Moreover, we mixed the three uncoupled equations in experiments 4 and 5 by a linear transformation in order to simulate a coupling, at least numerically.

Our implementation of the bidirectional shooting method described in the previous section is as follows. Compute approximations $\bar{X}_a$ and $\bar{X}_b$ (cf. (3.5)) by an initial value problem solver (and store intermediate values). Make their singular value decompositions $\bar{X}_a=\bar{U}_a\bar{S}_a\bar{V}_a^H$ and $\bar{X}_b=\bar{V}_b\bar{S}_b\bar{U}_b^H$ such that the singular values of $\bar{S}_a$ are ordered decreasingly and those of $\bar{S}_b$ increasingly.

Select the index k used in the choice of M_a and M_b in (3.7) by the simplistic algorithm

```
$k := 2; l := n-1;$
while $k \leqq l$ do
    if $\frac{a_k}{a_1} > \frac{b_l}{b_n}$ then $k := k+1$ else $l := l-1;$
```

where $\{a_k\}$ and $\{b_k\}$ denote the singular values of $\bar{X}_a$ and $\bar{X}_b$ respectively. This algorithm does not care about clusters of neighboring singular values. However, since in our experiments the singular values appeared to lie far apart, it worked appropriately. Furthermore, we have chosen $\mu=\nu=1$ in (3.7). This we inserted into (3.8) and solved this system for $\mathbf{c}$. Finally, from the matrices M_a and M_b, the vector $\mathbf{c}$ and the stored intermediate fundamental matrices we computed the approximated solution by (3.3) in a number of points $t_k \in [a, b]$. We have tabulated the max-norm of the errors at those points in the column under head BD (*bid*irectional shooting). For comparison we give in the columns LR and RL the corresponding errors of the simple shooting from *l*eft to *r*ight and vice versa. At the end we give also the results of an experiment in which we have added random perturbations to the fundamental matrices $\bar{X}_a$ and $\bar{X}_b$ in order to demonstrate the better robustness of bidirectional shooting once more in a different manner.

Example 1 (see Table 1).

ODE: $\varepsilon x'' - (1+\varepsilon\lambda)x' + \lambda x = 0$.

BC: $x(0) = 1, x(1) = 1$.

Solution:

$$x(t) = \frac{e^{t/\varepsilon}(1+e^{\lambda}) - e^{\lambda t}(1+e^{1/\varepsilon})}{e^{\lambda} - e^{1/\varepsilon}}.$$

Transformation into first order system: $y_1 = x$, $y_2 = x'$.

TABLE 1
Numerical results for Example 1.

λ	$\frac{1}{\varepsilon}$	Exact solution: norm of y_2	Absolute error in y_2 of shooting method: BD	LR	RL
10	−10	1.0E1	3.2E−10	7.2E−6	7.2E−6
50	−50	5.0E1	2.2E−9	1.1E13	1.1E13
100	−100	1.0E2	1.7E−9	4.5E34	4.5E34
50	2	4.0E2	7.9E−7	6.7E7	9.0E−7
−50	−2	4.0E2	7.9E−7	9.0E−7	6.7E7
2	100	8.2E2	1.5E−5	3.2E29	1.4E−5

Example 2 (see Table 2).

ODE: $\varepsilon x'' - (1+\varepsilon\lambda)x' + \lambda x = e^{\eta(t-1/2)}(\varepsilon\eta - 1)(\eta - \lambda)$.

BC: $x(0) = 1, x(1) = -1$,

Solution:

$$x(t) = \frac{[(1-e^{-\eta/2})e^{\lambda} + (1-e^{\eta/2})]e^{t/\varepsilon} - [(1+e^{\eta/2}) + (1-e^{-\eta/2})e^{1/\varepsilon}]e^{\lambda t}}{e^{\lambda} - e^{1/\varepsilon}} + e^{\eta(t-1/2)}.$$

Transformation into first order system: $y_1 = x$, $y_2 = x'$.

TABLE 2
Numerical results for Example 2

λ	$\frac{1}{\varepsilon}$	η	Exact solution: norm of y_2	Absolute error in y_2 of shooting method: BD	LR	RL
2	.5	0	4.9E0	6.9E−13	9.7E−13	1.6E−12
10	5	0	2.0E1	1.1E−10	4.6E−9	9.7E−11
20	10	0	4.0E1	1.6E−8	6.9E−5	1.3E−8
40	10	7	6.4E5	8.0E−7	1.3E5	1.5E−6
−40	−10	5	8.7E6	2.8E−4	4.8E−5	3.2E6
−40	10	5	7.1E1	8.7E−10	3.8E−9	2.6E5

Example 3 (Mattheij and Staarink (1981) (see Table 3).
ODE:

$$\frac{d\mathbf{y}}{dt}=\begin{bmatrix}1-\lambda\cos 2t & 0 & 1+\lambda\sin 2t\\ 0 & \lambda & 0\\ -1+\lambda\sin 2t & 0 & 1+\lambda\cos 2t\end{bmatrix}\mathbf{y}(t)-\begin{bmatrix}2+\lambda(\sin 2t-\cos 2t)\\ \lambda\\ \lambda(\sin 2t+\cos 2t)\end{bmatrix}.$$

BC:

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\mathbf{y}(0)+\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\mathbf{y}(\pi)=\begin{bmatrix}2\\2\\2\end{bmatrix}.$$

Solution:

$$\mathbf{y}(t)=[1,1,1]^T.$$

TABLE 3
Numerical results for Example 3.

λ	Exact solution: norm of y_1	Absolute error in y_1 of shooting method: BD	LR	RL
5	1.0E0	9.4E−9	5.0E−7	1.0E−9
6	1.0E0	5.5E−8	1.1E−5	6.0E−8
7	1.0E0	2.5E−6	3.1E−5	9.5E−7
8	1.0E0	1.3E−5	1.7E−3	3.0E−5
9	1.0E0	7.3E−5	3.5E−2	4.9E−4
10	1.0E0	6.8E−3	1.0E0	1.6E−2

Example 4 (see Table 4).
ODE:

$$u''-\alpha^2u=-\delta(\alpha^2+\eta^2)\cos\eta t+\delta\alpha^2\cos\eta,\quad v''+2t\beta^2v'=0,\quad w'-\gamma w=0.$$

BC:

$$u(1)=1,\quad u(-1)=-1,\quad v(1)=1,\quad v(-1)=-1,\quad w(1)+w(-1)=1.$$

Solution:

$$u(t)=\frac{\sinh\alpha t}{\sinh\alpha}+\delta(\cos\eta t-\cos\eta),\qquad v(t)=\frac{\operatorname{erf}\beta t}{\operatorname{erf}\beta},\qquad w(t)=\frac{1}{2}\frac{e^{\gamma t}}{\cosh\gamma}.$$

Transformation into first order system:

$$y_1=u+v+w,\quad y_2=u+v-w,\quad y_3=u-v-w,\quad y_4=u'+v',\quad y_5=u'-v'.$$

TABLE 4
Numerical results for Example 4.

α	β	γ	δ	η	Exact solution: norm of y_1	Absolute error in g_1 of shooting method: BD	LR	RL
5	3	20	1	10	3.0E0	1.1E−8	2.1E5	4.8E−7
10	1	20	1	20	3.0E0	2.4E−3	5.7E5	1.4E−2
10	1	20	1	1	3.0E0	2.9E−3	1.1E6	7.0E−3
5	−3	−20	2	−5	3.7E0	4.6E−9	1.1E−6	3.4E2
5	−3	−30	2	−5	3.7E0	9.0E−9	1.4E−6	2.0E11
2	−1	−32	1	−2	2.0E0	3.4E−10	1.6E−9	2.3E13

Example 5 (see Tables 5a, 5b).
ODE:

$$u''-\alpha^2u=0,\quad v''+2\beta^2tv'=0,\quad w''+\gamma\{4tw'+(4\gamma t^2+2)w\}=0.$$

BC:

$$u(1)=1,\quad u(-1)=-1,\qquad v(1)=1,\quad v(-1)=-1,$$

$$w(1)=0,\quad w(-1)=1.$$

Solution:

$$u(t)=\frac{\sinh \alpha t}{\sinh \alpha},\quad v(t)=\frac{\operatorname{erf}\beta t}{\operatorname{erf}\beta},\quad w(t)=\tfrac{1}{2}(1-t)\,e^{\gamma(1-t^2)}.$$

Transformation into first order system:

$$y_1=u+v+w,\quad y_2=u+v-w,\quad y_3=u-v-w,$$

$$y_4=u'+w',\qquad y_5=v'+u',\qquad y_6=w'+v'.$$

TABLE 5a
Numerical results for Example 5

			Exact solution:	Absolute error in y_1 of shooting method		
α	β	γ	norm of y_1	BD	LR	RL
10	3	5	7.4E1	2.0E−5	1.3E−3	1.3E−3
10	5	10	1.1E4	1.2E−1	1.1E1	2.9E0
3	6	3	1.0E1	8.0E−1	1.2E3	2.9E2
10	2	15	1.6E6	3.8E0	3.6E1	3.5E0
8	4	8	1.5E3	4.9E−5	2.7E−4	2.8E−4
6	4	6	2.0E2	4.0E−7	5.9E−5	5.1E−6

TABLE 5b
Numerical results for Example 5 *in which the entries of $\bar{X}_a$ and $\bar{X}_b$ are perturbed by adding a small number ε multiplied by the largest singular value and by a random number equally distributed in* $[-1, 1]$.

			Exact solution:		Absolute error in y_1 of shooting method		
α	β	γ	norm of y_1	ε	BD	LR	RL
10	5	10	1.1E4	0	1.2E−1	1.1E1	2.9E0
				1.0E−11	8.9E1	3.1E4	1.3E3
				1.0E−10	8.0E2	7.9E4	2.9E4
10	2	15	1.6E6	0	3.8E0	3.6E1	3.5E0
				1.0E−10	4.7E2	1.1E6	3.2E4
				1.0E−9	2.1E4	3.9E6	6.0E5
8	4	8	1.5E3	0	4.9E−5	2.7E−4	2.8E−4
				1.0E−10	3.6E−2	2.9E0	9.6E−1
				1.0E−9	3.4E−1	1.0E2	5.1E0

REFERENCES

L. FOX (1962), *Numerical Solution of Ordinary and Partial Differential Equations*, Pergamon Press, Oxford.

M. HERMANN AND H. BERNDT (1981), RWPM: *A multiple shooting code for nonlinear two-point boundary value problems*, Preprint 27 and 57, Friedrich-Schiller-Universität Jena, Jena, German Democratic Republic.

T. KATO (1966), *Perturbation Theory for Linear Operators*, Springer-Verlag, Berlin.

H. B. KELLER (1968), *Numerical Methods for Two-Point Boundary Value Problems*, Ginn-Blaisdell, Waltham, MA.

——— (1976), *Numerical Solution of Two Point Boundary Value Problems*, CBMS Regional Conference Series in Applied Mathematics 24, Society for Industrial and Applied Mathematics, Philadelphia.

C. L. LAWSON AND R. J. HANSON (1974). *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ.

R. M. M. MATTHEIJ AND G. W. M. STAARINK (1981), *On optimal shooting intervals*, Preprint 23, Katholieke Universiteit Nijmegen, Nijmegen, the Netherlands, Math. of Comp. (1984), to appear.

M. R. OSBORNE (1979), *The stabilised march is stable*, SIAM J. Numer. Anal., 16, pp. 923–933.

S. M. ROBERTS AND J. S. SHIPMAN (1972), *Two-Point Boundary Value Problems: Shooting Methods*, American Elsevier, New York.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

NUMERICAL SIMULATION OF TIME-DEPENDENT CONTACT AND FRICTION PROBLEMS IN RIGID BODY MECHANICS*

PER LÖTSTEDT†

Abstract. A numerical method is given for the solution of a system of ordinary differential equations and algebraic, unilateral constraints. The equations govern the motion of a mechanical system of rigid bodies, where contacts between the bodies are created and disappear in the time interval of interest. The ordinary differential equations are discretized by linear multistep methods. In order to satisfy the constraints, a quadratic programming problem is solved at each time step. The fact that the variation of the objective function is small from step to step is utilized to save computing time. A discrete friction model, based on Coulomb's law of friction and suitable for efficient computation, is proposed for planar problems where dry friction cannot be neglected. The normal forces and the friction forces are the optimal solution to a quadratic programming problem. The methods are tested on four model problems. A data structure and possible generalizations are discussed.

Key words. ordinary differential equations, numerical methods, mechanical systems, rigid body, contact problems, Coulomb friction

1. Introduction. In many engineering applications there is an interest in the simulation on computers of the large-scale motion of mechanical systems. These systems are often well approximated by a number of rigid bodies interacting with each other. The bodies may be interconnected by different kinds of joints, e.g. ball-and-socket joints or hinge joints, or they may be in contact with each other at a point, along a curve, or on a surface. The reason to choose a rigid body model instead of a more elaborate and perhaps more accurate elastic or elastic-plastic model is that the governing equations are a system of ordinary differential equations and the number of degrees of freedom is much lower after discretization. Examples of areas of application where rigid body systems have been simulated are

(i) mechanisms and machinery [33], [34], [38],
(ii) satellites [12], [37],
(iii) biomechanics: the human body represented by 10–15 rigid bodies in sports activities [36], vehicle crash simulations [20],
(iv) vehicles on the road [11] and on rails [24],
(v) rock mechanics: the motion of assemblages of rock blocks [7],
(vi) building construction, the collapse of a concrete building [28].

In the first four examples the joints between the bodies are assumed to be permanent during the time interval of interest. Friction is usually ignored in the first three categories. These systems are simulated in both two and three space dimensions. In examples (v) and (vi) it is of importance to use a realistic model for the creation and disappearance of contacts between the bodies and the friction between bodies sliding on each other. Due to the complicated geometry of the bodies in three space dimensions these systems have so far only been treated as planar systems.

The standard numerical methods for the solution of ordinary differential equations can often be applied directly to the systems in (i)–(iv), possibly with a modification to allow for algebraic constraints (Gear [15, p. 226]). There are methods in analytical mechanics to reduce the dimension of the system of ordinary differential equations

* Received by the editors October 7, 1982, and in revised form March 22, 1983. This work was supported by The National Swedish Board for Technical Development under contract dnr 80-3341.

† Department of Numerical Analysis and Computing Science, The Royal Institute of Technology, S-100 44 Stockholm 70, Sweden. Present address: KLUB-2, Aerospace Division, SAAB-Scania, S-581 88 Linköping, Sweden.

to the degree of freedom of the mechanical system; see Paul [34]. This technique is used in Sheth and Uicker [38], while in Orlandea and Calahan [33] extra variables, Lagrange multipliers, and associated algebraic constraints are introduced. For problems where the number of contacts is small, say 1–3, it is possible to reduce the size of the differential equation system as described above and to reformulate the system at each time a new contact is established or an old contact disappears. For mechanical systems with several contacts and dry friction, the Lagrange multiplier approach is much more flexible and efficient. Furthermore, the multipliers are directly proportional to the contact and friction forces.

Coulomb's law is a simple and common model for dry friction between sliding bodies. In this paper we develop a numerical method for the simulation of time-dependent contact and Coulomb friction problems for rigid body systems based on a Lagrange multiplier approach. The method was outlined in Lötstedt [27]. An improved version is here analyzed and tested on model problems inspired by rock mechanical applications. Cundall [7], in his simulation of rock blocks, introduces springs and dampers at the points of contact to prevent the blocks from penetrating each other. In our method the Lagrange multipliers serve the same purpose.

Cundall's idea is related to the penalty function method in nonlinear optimization and is analyzed by Lötstedt [26]. The analysis of two deformable bodies in contact is reviewed in Kalker [21] and methods of solution are indicated in Cottle [6]. Numerical methods for the elastostatical contact problem have been devised e.g. by Haug, Chand and Pan [19] for many bodies without friction and by Fredriksson [14] for two bodies with friction.

In the next section the system of equations and inequalities satisfied by the coordinates of the bodies and the Lagrange multipliers is formulated. Friction-free contact problems are treated in § 3. The system of ordinary differential equations is solved by a linear multistep method and the Lagrange multipliers are the optimal solution to a quadratic programming (QP) problem. A discrete interpretation of Coulomb's law of friction is made in § 4. The friction forces are also the solution to a QP problem. The performance of the method described in § 3 and § 4 is illustrated in four examples. In § 5 a data structure for the representation of the mechanical system in the computer is proposed. Possible generalizations are discussed in the final section.

2. The governing equations.

The governing equations for planar contact and Coulomb friction problems are derived in Lötstedt [29], [30]. The equations are stated and explained here for completeness and the notation used throughout the paper is introduced. The necessary modifications in three space dimensions are discussed in § 6.

The system of ordinary differential equations satisfied by $q(t) \in \mathbb{R}^m$, the vector of the coordinates of the bodies, is for $t \geqq 0$

$$M\ddot{q} = f(q, \dot{q}, t) + G(q)\lambda(t). \tag{2.1}$$

The position of a body is represented in q by the Cartesian coordinates (x, y) of its center of gravity and an angle of direction ψ. $M \in \mathbb{R}^{m \times m}$ is the mass matrix, a constant, diagonal matrix with positive diagonal elements. M has the factorization

$$M = D^T D. \tag{2.2}$$

$f \in \mathbb{R}^m$ contains the explicity given driving forces and $G\lambda$ represents the contribution of the contact and friction forces to the equations of motion. λ is the Lagrange multiplier vector which is determined such that certain algebraic constraints and inequalities are fulfilled.

Let the components of $\phi(q) \in \mathbb{R}^p$ be the holonomic, unilateral constraint functions. ϕ_j may e.g. be the perpendicular distance between the corner of one body and the edge of another body. The unilateral algebraic constraints in the mechanical system are

$$\phi(q) \geqq 0. \tag{2.3}$$

To simplify the presentation, only holonomic constraints are discussed here. In the computational procedures in § 3 holonomic and nonholonomic constraints are given equal treatment. Let $g_{Ni} = \partial\phi_i/\partial q$ and let the transposition of the vector or matrix C be denoted by C^T. If ϕ_j is the distance between a corner and an edge then $d\phi_j/dt = g_{Nj}^T\dot{q}$ is the relative velocity between the same corner and edge in a direction normal to the edge. λ_{Ni} is the nonnegative Lagrange multiplier associated with ϕ_i. Define δ_N as follows:

$$\delta_{Ni} = \frac{d^2\phi_i}{dt^2} = g_{Ni}^T\ddot{q} + \dot{g}_{Ni}^T\dot{q}. \tag{2.4}$$

Introduce the time-dependent, active constraint set J_N

$$J_N = \{i \mid \phi_i = 0\}. \tag{2.5}$$

The complementarity relations satisfied by ϕ_i, $g_{Ni}^T\dot{q}$, δ_{Ni} and λ_{Ni} are

$$\phi_i \geqq 0, \quad \lambda_{Ni} \geqq 0, \quad \lambda_{Ni}\phi_i = 0, \tag{2.6a}$$

$$\text{if } i \in J_N \text{ then } g_{Ni}^T\dot{q} \geqq 0, \lambda_{Ni}g_{Ni}^T\dot{q} = 0, \tag{2.6b}$$

$$\text{if } g_{Ni}^T\dot{q} = 0 \text{ then } \delta_{Ni} \geqq 0, \lambda_{Ni}\delta_{Ni} = 0. \tag{2.6c}$$

Since ϕ_i and λ_{Ni} are nonnegative, the conditions $\lambda_{Ni}\phi_i = 0$, $i = 1, 2, \cdots, p$, are equivalent to $\lambda_N^T\phi = 0$. The physical interpretation of (2.6a) is that if there is no contact between the corner and the edge then $\phi_j > 0$, and the multiplier proportional to the normal force at the point of contact λ_{Nj} has vanished. If $\lambda_{Nj} > 0$ then we have a contact, $\phi_j = 0$.

The vectors $g_{Fi} \in \mathbb{R}^m$, $i = 1, 2, \cdots, k \leqq p$, are determined such that $g_{Fi}^T\dot{q}$ is the relative velocity in the transverse direction between two bodies in contact at a point or along two edges. Let λ_{Fi} be the associated Lagrange multiplier proportional to the friction force acting on the bodies. δ_F has the definition

$$\delta_{Fi} = \frac{d(g_{Fi}^T\dot{q})}{dt} = g_{Fi}^T\ddot{q} + \dot{g}_{Fi}^T\dot{q}. \tag{2.7}$$

To each friction multiplier λ_{Fi} there is a corresponding normal multiplier λ_{Ni}. Introduce three index sets

$$J_{F0} = \{i \mid i \in J_N, g_{Fi}^T\dot{q} = 0\}, \quad J_{FW} = \{i \mid i \in J_N, g_{Fi}^T\dot{q} \neq 0\}, \quad J_F = (J_{FW} \cup J_{F0}) \subset J_N. \tag{2.8}$$

All the sets in (2.8) are time-dependent and are altered at discrete time points. If $i \in J_{F0}$ then no sliding takes place at this particular contact. According to Coulomb's law of friction the following relations are satisfied by $g_{Fi}^T\dot{q}$, δ_{Fi}, λ_{Ni} and λ_{Fi}:

$$\text{if } i \in J_F \text{ then } |\lambda_{Fi}| \leqq \mu\lambda_{Ni}, \tag{2.9a}$$

$$\text{if } i \in J_{FW} \text{ then } |\lambda_{Fi}| = \mu\lambda_{Ni}, \lambda_{Fi}g_{Fi}^T\dot{q} \leqq 0, \tag{2.9b}$$

$$\text{if } i \in J_{F0} \text{ then } (\mu\lambda_{Ni} - |\lambda_{Fi}|)\delta_{Fi} = 0, \lambda_{Fi}\delta_{Fi} \leqq 0. \tag{2.9c}$$

The friction coefficient is μ, $\mu \geqq 0$. The condition (2.9b) expresses the fact that if two bodies slide upon each other, then the friction force has the direction opposite to the relative velocity. It follows from (2.9c) that when there is no sliding then $\delta_{Fi}=0$ and $|\lambda_{Fi}| \leqq \mu\lambda_{Ni}$, but when the sliding has just begun then $\delta_{Fi} \neq 0$ and $|\lambda_{Fi}| = \mu\lambda_{Ni}$.

The columns of G in (2.1) are at least those vectors g_{Ni} and g_{Fi} for which $i \in J_N$ at a given time point. The elements in λ in (2.1) are the corresponding normal and friction multipliers λ_{Ni} and λ_{Fi}.

If $i \in J_{FW}$ then λ_{Fi} can be written in terms of λ_{Ni}, μ and $s_i = \pm 1$:

(2.10)
$$\lambda_{Fi} = s_i \mu \lambda_{Ni}.$$

For friction problems it is convenient to split G into two parts, G_1 and H. Let λ_1 contain the normal multipliers λ_{Ni} and the friction multipliers λ_{Fj}, $j \in J_{F0}$. Moreover, let the columns of G_1 consist of those g_{Ni} and g_{Fj} corresponding to the components in λ_1 and collect the remaining vectors g_{Fj}, $j \in J_{FW}$, in H. By (2.10) there exists a matrix U such that

(2.11)
$$\sum_{i \in J_{FW}} g_{Fi}\lambda_{Fi} = \sum_{i \in J_{FW}} g_{Fi} s_i \mu \lambda_{Ni} = HU\lambda_1.$$

The contribution of the working friction forces to the equations of motion (2.1) is (2.11). Thus, with the new notation, (2.1) is equivalent to

(2.12)
$$M\ddot{q} = f + (G_1 + HU)\lambda_1, \qquad t \geqq 0.$$

Suppose that a new constraint j becomes active at t_*. Then $J_N(t \geqq t_*) = J_N(t < t_*) \cup \{j\}$ and $\phi_j(t) > 0$, $t < t_*$, but $\phi_j(t) = 0$, $t \geqq t_*$. In general, there is an impulse Λ in the system at t_* and $\dot{q}$ is discontinuous. Let $\dot{q}_+$ and $\dot{q}_-$ denote $\dot{q}(t+0)$ and $\dot{q}(t-0)$, respectively. F is the vector of external impulses. Then one relation between $\dot{q}_+$, $\dot{q}_-$ and Λ is

(2.13)
$$M(\dot{q}_+ - \dot{q}_-) = F + G(q)\Lambda.$$

G consists of the columns g_{Ni} and g_{Fi} such that $i \in J_N(t_* + 0)$. A component Λ_{Ni} of Λ corresponds to a contact constraint ϕ_i and Λ_{Fj} to a friction constraint $g_{Fj}^T\dot{q}$. The complementarity relations for $\dot{q}_+$ and Λ depend on the material in the colliding bodies and on the impulse friction model. Here the collision is assumed to be inelastic and Λ is the optimal solution to the QP problem

(2.14)
$$\begin{gathered} \min \tfrac{1}{2}\Lambda^T G^T M^{-1} G \Lambda + \Lambda^T G^T(\dot{q}_- + M^{-1}F), \\ \Lambda_{Ni} \geqq 0, \qquad -\mu_I \Lambda_{Ni} \leqq \Lambda_{Fi} \leqq \mu_I \Lambda_{Ni}. \end{gathered}$$

The impulse friction coefficient is μ_I, $\mu_I \geqq 0$. For a friction-free system $\mu_I = 0$ and $\Lambda_{Fi} = 0$. By (2.13) and the Kuhn–Tucker conditions [13, p. 51] at the optimum of (2.14) we have

(2.15)
$$g_{Ni}^T \dot{q}_+ \geqq 0, \quad \Lambda_{Ni} \geqq 0, \quad \Lambda_{Ni} g_{Ni}^T \dot{q}_+ = 0.$$

The implications of (2.14) on Λ_{Fi} and $g_{Fi}^T\dot{q}_+$ when $\mu_I > 0$ are discussed in § 4.3.

Another source of discontinuities is when a friction constraint j is transferred from J_{FW} to J_{F0}, i.e. when $g_{Fj}^T\dot{q} \to 0$. Then $\ddot{q}$ and λ are in general discontinuous. The time derivatives $\dddot{q}$ and $\dot{\lambda}$ are usually discontinuous when λ_{Nj} reaches its lower bound 0 and j is removed from J_N, and when $|\lambda_{Fj}|$ attains its upper bound $\mu\lambda_{Nj}$ and j is transferred from J_{F0} to J_{FW}.

In order to solve (2.1) or (2.12) in combination with the relations (2.6) and (2.9), the problem is discretized by two linear multistep methods (Gear [15]). An approximation $(q_i, \dot{q}_i)$ to $(q(t_i), \dot{q}(t_i))$ is determined at $t = t_i$, $i = 0, 1, \cdots$, where $t_0 = 0$ and $t_{i+1} = t_i + h_{i+1}$. Assume that $(q_i, \dot{q}_i)$, $i = 0, 1, \cdots, n-1$, are known. Then the formulas for computation of $(q_n, \dot{q}_n)$ with a constant step size $h_i = h$ and two r-step methods are

$$\sum_{i=0}^{r} \alpha_i^1 q_{n-i} = h \sum_{i=0}^{r} \beta_i^1 \dot{q}_{n-i}, \tag{2.16a}$$

$$\sum_{i=0}^{r} \alpha_i^2 \dot{q}_{n-i} = h \sum_{i=0}^{r} \beta_i^2 M^{-1}(f_{n-i} + G_{n-i}\lambda_{n-i}), \tag{2.16b}$$

where $f_{n-i} = f(q_{n-i}, \dot{q}_{n-i}, t_{n-i})$, $G_{n-i} = G(q_{n-i})$ and the coefficients α_i^j, β_i^j, $j = 1, 2$, are constant. Sum all the quantities in (2.16a) and (2.16b) that are known from previous steps in b_n^1 and b_n^2, respectively, so that q_n and $\dot{q}_n$ are the solutions to the nonlinear equations

$$q_n = \frac{1}{\alpha_0^1}(h\beta_0^1 \dot{q}_n + b_n^1), \qquad \dot{q}_n = \frac{1}{\alpha_0^2}(h\beta_0^2 M^{-1}(f_n + G_n\lambda_n) + b_n^2). \tag{2.17}$$

In a friction free problem, $\mu = 0$, q_n and λ_n satisfy in addition to (2.17) the discrete version of (2.6a)

$$\phi(q_n) \geqq 0, \quad \lambda_n \geqq 0, \quad \lambda_n^T \phi(q_n) = 0. \tag{2.18}$$

The system (2.17), (2.18) is referred to as a nonlinear complementarity problem (NLCP) in Cottle [6]. The prospects of having to solve an NLCP at each time step are not so encouraging. Therefore, in the next section, by a suitable choice of linear multistep methods in (2.16) and by replacing the condition (2.6a) by (2.6b), we arrive at a linear complementarity (LCP) problem whose solution is calculated with a QP algorithm.

The norm $\|\cdot\|$ in the sequel always denotes the Euclidean vector norm or the subordinate, spectral matrix norm. The weighted norm $\|x\|_M$ of the vector x has the definition

$$\|x\|_M^2 = x^T M x = \|Dx\|^2. \tag{2.19}$$

3. Contact problems. The difference equations for friction-free contact problems fulfilled by $(q_n, \dot{q}_n)$ are chosen in this section. A numerical procedure for obtaining λ_n is suggested for time points t_n where the active constraint set is constant, $J_N(t_{n-1}) = J_N(t_n)$, and when contacts disappear, $J_N(t_{n-1}) \supset J_N(t_n)$, $\phi_j(t_{n-1}) = 0$, but $\phi_j(t_n) > 0$ for some j. When a new contact is established at t_*, the impulse in the system and the jump in $\dot{q}$ satisfy (2.13) and (2.14). The section is concluded by two examples.

3.1. Advancing the solution in time. The coefficients α_i^1, β_i^1 in (2.16a) are determined by a member of the Adams–Bashforth family of explicit formulas denoted by AB-r [15, p. 104]. The method in (2.16b) is a backward difference formula [15, p. 217] with $\beta_0^2 = 1$, $\beta_j^2 = 0$, $j = 1, 2, \cdots, r$, abbreviated BDF-r. Replace (2.18) by the condition (2.6b) in intervals where J_N is constant or indices are dropped from J_N. Let g_{Ni}, $i \in J_N$, form G. Since $\beta_0^1 = 0$, we have by (2.17) and (2.6b) that q_n and $\dot{q}_n$ are the solution to the system

$$q_n = \frac{1}{\alpha_0^1} b_n^1, \tag{3.1a}$$

$$\dot{q}_n = \frac{1}{\alpha_0^2}(h_n M^{-1}(f_n + G_n\lambda_n) + b_n^2), \tag{3.1b}$$

$$G_n^T \dot{q}_n \geqq 0, \quad \lambda_n \geqq 0, \quad \lambda_n^T G_n^T \dot{q}_n = 0. \tag{3.1c}$$

First compute q_n in (3.1a). If we let c_n denote $b_n^2+h_nM^{-1}f_n$ and temporarily assume that $f=f(q,t)$, then ((3.1b), (3.1c)) is a linear complementarity problem. The system ((3.1b), (3.1c)) is equivalent to the QP problem (Cottle [6])

$$\min \tfrac{1}{2}\lambda_n^TG_n^TM^{-1}G_n\lambda_n+h_n^{-1}\lambda_n^TG_n^Tc_n, \qquad \lambda_n\geqq 0. \tag{3.2}$$

Since

$$c_n=h_nM^{-1}f_n-\sum_{i=1}^{r}\alpha_i^2\dot q_{n-i},$$

we have $\|G_n^Tc_n\|=O(h_n)$ if h_i varies smoothly and $\|\lambda_n\|=O(1)$ in (3.2). The dual problem to (3.2) is

$$\min \tfrac{1}{2}v_n^TG_n^TM^{-1}G_nv_n, \qquad h_nG_n^TM^{-1}G_nv_n+G_n^Tc_n\geqq 0, \tag{3.3}$$

where $G_n\lambda_n=G_nv_n$ (see Dorn [10]). Substitute

$$G_n\lambda_n=h_n^{-1}M(\alpha_0^2\dot q_n-c_n)$$

in (3.3) and after dropping unnecessary constants in the objective function and the constraints we have

$$\min \left\|\sum_{i=0}^{r}\alpha_i^2\dot q_{n-i}-h_nM^{-1}f_n\right\|_M, \qquad G_n^T\dot q_n\geqq 0. \tag{3.4}$$

There are two advantages to using the complementarity condition on the derivative of the constraint (2.6b) or (3.1c) instead of (2.6a) or (2.18). The nilpotency of the system (3.1) is lower (see Petzold [35] for a treatment of these matters) and the constraint function $G^T\dot q$ is linear in $\dot q$. The disadvantage is that $\phi_i(q)=0$, $i\in J_N$, is not satisfied exactly. A possible remedy is the stabilization procedure developed by Baumgarte [1].

How to compute λ_n is treated in § 3.2. Here we shall continue with a discussion of the properties of the time discretization. In most steps from t_{n-1} to t_n the active constraint set J_N remains unaltered. Then the constraint on $\dot q_n$ in (3.1c) is an equality constraint, $G_n^T\dot q_n=0$. It is shown in [27] that the global accuracy in q_n, $\dot q_n$, λ_n and $\phi(q_n)$ for a constant step size h is $O(h^r)$ when $G_n^T\dot q_n=0$. The recursion in (3.1) is restarted using the first order method (AB-1, BDF-1) for two steps after a discontinuity in $\dot q$ or $\ddot q$. In order to simplify the solution process, the discontinuities in $\ddot q$ stemming from the removal of a constraint from J_N are ignored. Errors proportional to h^3 and h^2 are introduced in q_n and $\dot q_n$, respectively. It does not pay to use methods of higher order than two, $r=2$, if the order of the global error is to be the same at points when J_N is constant and when constraints are dropped from J_N. We assume that the alterations of J_N are independent of h. Since these take place at discrete points, the global accuracy of $O(h^2)$ in $\dot q_n$ is maintained.

The stepsize h_n is chosen such that a weighted norm of the local error l_n in q_n per unit step is less than a prescribed tolerance ε,

$$\|l_n\|_M\leqq h_n\varepsilon. \tag{3.5}$$

The reason for choosing this error condition is that we can allow certain errors in $\dot q_n$ to be dealt with later to be of $O(\varepsilon)$. The local error at t_n in AB-1 (the forward Euler method) is proportional to $h_n^2\ddot q_n$ and in AB-2 to $h_n^3\dddot q_{n-1}=h_n^2(\ddot q_{n-1}-\ddot q_{n-2})+O(h_n^4)$ if $h_n=h_{n-1}$. The estimate of $\dddot q_{n-1}$ is less accurate when a constraint is dropped from J_N, but is for simplicity used there also.

The stability of the compound methods (AB-1, BDF-1) and (AB-2, BDF-2) is studied by means of the scalar test equation

$$\ddot{u} + d\dot{u} + ku = 0 \tag{3.6}$$

in [27]. This spring-and-damper equation has stable solutions if and only if $k > 0$, $d \geqq 0$ or $d > 0$, $k \geqq 0$. The stability regions for the first and second order methods are depicted in Fig. 1.

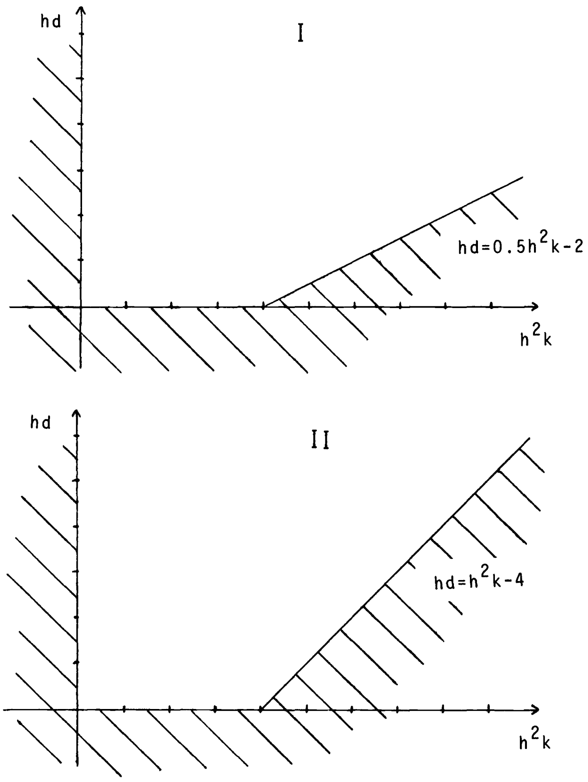


FIG. 1. *Stability regions for the methods* (AB-1, BDF-1) *and* (AB-2, BDF-2) *applied to the test equation* (3.6) *are plotted in pictures* I *and* II. *The methods are stable in the unshaded areas except for the corners where the difference equation has a double root* $\zeta_1 = \zeta_2$, $|\zeta_1| = 1$.

Suppose that $\phi_j(t_{n-1}) > 0$ but $\phi_j(t_n) \leqq 0$ for some j. Then a collision has occurred at t_*, $\phi_j(t_*) = 0$, $t_{n-1} < t_* \leqq t_n$, where $\dot{q}$ is discontinuous. The point t_* is determined by inverse linear interpolation between $\phi_j(t_{n-1})$ and $\phi_j(t_n)$. The error introduced by this approximate computation of t_* is of $O(h_n^2)$. An example where $\ddot{q}$ is discontinuous is when one body in contact with another body slides off the support of the other body. Another reason for a jump in $\ddot{q}$ is when f is discontinuous. The crucial point t_* is also here determined by inverse linear interpolation.

The assumption $f = f(q, t)$ is not always satisfied. A linear damper gives rise to an external force proportional to $\dot{q}$, $f = f(q, \dot{q}, t)$. Sufficient conditions for the existence of a unique solution to ((3.1b), (3.1c)) when f is velocity dependent are stated below.

PROPOSITION 1. *Let f be continuously differentiable in $\dot{q}$ and let A be defined by*

$$A(\dot{q}) = M - \frac{h_n}{\alpha_0^2} \frac{\partial f}{\partial \dot{q}}(q_n, \dot{q}, t_n).$$

Assume that A is invertible in a sufficiently large neighborhood Q of $\dot q_{n-1}$ and let

$$B(\dot q)=G_n^TA^{-1}G_n.$$

If

$$x^TBx\geqq \kappa x^Tx,\qquad \kappa>0,\quad \dot q\in Q, \tag{3.7}$$

then there is a unique solution $(\dot q_n,\lambda_n)$ to ((3.1b), (3.1c)).

Proof. Since A^{-1} exists when $\dot q\in Q$, $\dot q_n$ in (3.1b) can be expressed uniquely as $\dot q_n=u_n(\lambda_n)$, [9, Thm. 10.2.2].

Furthermore,

$$\frac{\partial \dot q_n}{\partial\lambda_n}=\frac{h_n}{\alpha_0^2}A^{-1}(u_n(\lambda_n))G_n=\frac{h_n}{\alpha_0^2}A_n^{-1}G_n.$$

Consider the NLCP

$$\lambda_n\geqq 0,\quad G_n^T\dot q_n=G_n^Tu_n(\lambda_n)\geqq 0,\quad \lambda_n^TG_n^Tu_n=0. \tag{3.8}$$

We find that

$$\frac{\partial}{\partial\lambda_n}G_n^T\dot q_n=\frac{h_n}{\alpha_0^2}G_n^TA_n^{-1}G_n=\frac{h_n}{\alpha_0^2}B.$$

If B has the property (3.7), then there exists a unique λ_n solving (3.8) according to [22, Thm. 3.2]. Hence, $\dot q_n$ is also unique. □

Remark. Assume that G_n has full column rank. If h_n is sufficiently small then $A^{-1}=M^{-1}+h_nC$, $\|C\|=O(1)$, $x^TG_n^TM^{-1}G_nx\geqq\kappa_0x^Tx$, $\kappa_0>0$, and (3.7) is satisfied. It can be shown that if $\partial f/\partial\dot q$ is symmetric and the eigenvalues λ_i of $D^{-T}\,\partial f/\partial\dot q\,D^{-1}$ satisfy $1>\kappa_1\geqq h_n\lambda_i/\alpha_0^2$, then the sufficient conditions in the proposition are fulfilled.

The proposed solution process for ((3.1b), (3.1c)) when $f=f(q,\dot q,t)$ is based on functional iteration. The value $\dot q_n^{(0)}$ is predicted by $\dot q_{n-1}$, the AB-1 or the AB-2 formula depending on the number of $\dot q_i$ available with sufficient continuity. Then $\dot q_n^{(j)}$, $j=1,2,\cdots$, is determined by the corrector equation

$$\begin{gathered}\dot q_n^{(j)}=\frac{1}{\alpha_0^2}(h_nM^{-1}(f(q_n,\dot q_n^{(j-1)},t_n)+G_n\lambda_n^{(j)})+b_n^2),\qquad G_n^T\dot q_n^{(j)}\geqq 0,\quad \lambda_n^{(j)}\geqq 0,\\ \lambda_n^{(j)}G_n^T\dot q^{(j)}=0.\end{gathered} \tag{3.9}$$

The criterion for interruption of the iteration is discussed after the following proposition.

PROPOSITION 2. *Let $f_n=f(q_n,\dot q,t_n)$ be continuous and differentiable in $\dot q$. Let c_1 and c_2 be constants such that $\|M^{-1}\,\partial f_n/\partial\dot q\|_M\leqq c_1$ in $\|\dot q-\dot q_n^{(0)}\|_M<c_2$. If h_n is so small that $h_nc_1/\alpha_0^2\leqq 1$ and $\|\dot q_n^{(1)}-\dot q_n^{(0)}\|_M<c_2(1-h_nc_1/\alpha_0^2)$, then the proposed iterative process* (3.9) *converges to the unique solution $\dot q_n$ of* ((3.1b), (3.1c)).

Proof. Define F_0 and F_1 by

$$F_0(y)=\tfrac12\alpha_0^2y^TMy-h_ny^Tf_n^0-y^TMb_n^2,\qquad F_1(y)=\tfrac12\alpha_0^2y^TMy-h_ny^Tf_n^1-y^TMb_n^2,$$

where $f_n^i=f(q_n,x_i,t_n)$. Define the minimization problems

$$\min F_0(y),\qquad G_n^Ty\geqq 0, \tag{3.10a}$$

$$\min F_1(y),\qquad G_n^Ty\geqq 0. \tag{3.10b}$$

Let y_0 be the solution to (3.10a) and $y_1=y_0+\delta y$ the solution to (3.10b). It is shown in [31, (3.5)–(3.7)] that y_0 and y_1 satisfy

$$\delta y^T(\nabla F_1(y_1)-\nabla F_1(y_0))\leqq\delta y^T(\nabla F_0(y_0)-\nabla F_1(y_0)), \tag{3.11}$$

where ∇F_i, $i=1,2$, denotes the gradient of F_i. Then

$$\|\delta y\|_M^2 \leqq h_n \delta y^T(f_n^1-f_n^0)/\alpha_0^2 = h_n(D\delta y)^T DM^{-1}(f_n^1-f_n^0)/\alpha_0^2.$$

Therefore, by (2.19)

$$\|\delta y\|_M \leqq h_n\|M^{-1}(f_n^1-f_n^0)\|_M/\alpha_0^2.$$

It follows from [9, 8.5.4] that if $\|x_0-\dot q_n^{(0)}\|_M<c_2$ and $\|x_1-\dot q_n^{(0)}\|_M<c_2$, then

(3.12) $$\|y_1-y_0\|_M \leqq h_n c_1\|x_1-x_0\|_M/\alpha_0^2 < \|x_1-x_0\|_M.$$

If $x_i=\dot q_n^{(j-1+i)}$ then by (3.4) $\dot q_n^{(j)}=y_i$ in the iteration process (3.9). The fixed point theorems [9, Thms. 10.1.1 and 10.1.2] and (3.12) prove the convergence of the procedure (3.9) to the unique solution of ((3.1b), (3.1c)). □

Using the suggested predictor methods, it can be shown that for the difference between the predictor $\dot q_n^{(0)}$ and the first corrector solution $\dot q_n^{(1)}$ we have

$$\|\dot q_n^{(1)}-\dot q_n^{(0)}\|_M = O(h_n^k), \qquad k\geqq 1.$$

The sufficient conditions in the proposition can always be met by choosing h_n small enough. However, for problems where c_1 is large the step size h_n may be intolerably short. The conditions are the same in a problem without the constraints $G^T q\geqq 0$ (cf. Gear [15, p. 114]).

The error $\|\dot q_n-\dot q_n^{(s)}\|_M$ in the accepted solution $\dot q_n^{(s)}$ of (3.9) is allowed to be at most 0.1ε. This error is multiplied by $\beta_1^1 h$ in the computation of q_{n+1}. The factor 0.1 is chosen in order to make $0.1\beta_1^1 h_{n+1}\varepsilon$ small in comparison with the local error (3.5) in q_{n+1}. Let the error $G_n\delta\lambda_n^{(j)}$ in $G_n\lambda_n^{(j)}$ satisfy

(3.13) $$\|D^{-T}G_n\delta\lambda_n^{(j)}\| = \|M^{-1}G_n\delta\lambda_n^{(j)}\|_M \leqq \varepsilon_0.$$

The iterative error in $\dot q_n^{(j)}$ is estimated by

(3.14) $$\|\dot q_n-\dot q_n^{(j)}\|_M < \frac{\rho\|\dot q_n^{(j)}-\dot q_n^{(j-1)}\|_M+\varepsilon_0}{1-\rho},$$

Dahlquist and Björck [8, p. 239]. An approximate rate of convergence $\rho\approx h_n c_1/\alpha_0^2$ is determined during the iteration. When the error bound in (3.14) is less than 0.1ε for $j=s$, then the iteration is terminated. If f is dependent on the velocity $\dot q$, then $\varepsilon_0=0.01\varepsilon$. Otherwise, $\dot q_n$ is computed directly in (3.1b), and the required accuracy ε_0 in $G_n\lambda_n$ is 0.1ε.

In conclusion, we give the algorithm for advancing the solution one step from t_{n-1} to t_n.

Algorithm 1.

1. Compute q_n by AB-1 or AB-2 with a step length h_n such that the local error satisfies (3.5).
2. If necessary, predict the value $\dot q_n^{(0)}$ and then compute $f_n^{(0)}=f_n(q_n,\dot q_n^{(0)},t_n)$. Determine $G_n\lambda_n$ to the prescribed accuracy ε_0 and calculate $\ddot q_n = M^{-1}(f_n+G_n\lambda_n)$ and $\dot q_n$ by BDF-1 or BDF-2. If f is a function of $\dot q$, this step is performed iteratively (3.9).
3. Test whether
   (i) $\dot q$ is discontinuous between t_{n-1} and t_n because there is an external impulse $F\neq 0$ in (2.13), or a new constraint becomes active, i.e. $\phi_j(t_{n-1})>0$, but $\phi_j(t_n)\leqq 0$ for some j;
   (ii) $\ddot q$ is discontinuous between t_{n-1} and t_n because f or $G\lambda$ is discontinuous, e.g. due to the loss of support of a body.

If (i) or (ii) has occurred, then the time t_* where $\dot{q}$ or $\ddot{q}$ is discontinuous is computed by inverse linear interpolation. If there are several detected jumps in $\dot{q}$ or $\ddot{q}$ in the interval at $t_*^1, t_*^2, \cdots, t_*^j$, then $t_* = \min(t_*^1, \cdots, t_*^j)$. After a discontinuity in $\dot{q}$ the new velocity $\dot{q}_+$ in (2.13) is first determined. Then restart the integration at Step 1 by AB-1 with $q(t_*)$, $\dot{q}(t_*+0)$ and the new active constraint set $J_N(t_*+0)$.

4. Test whether there are any elements λ_{nj} in λ_n such that $\lambda_{nj}=0$ and the corresponding Δ_{nj} in
$$\Delta_n = G_n^T\dot{q}_n$$
satisfies $\Delta_{nj} > \varepsilon$. If that is the case then remove j from J_N.
5. End of algorithm.

Initially, J_N at $t=0$ is defined by $\phi(q(0))$ and $G^T\dot{q}(0)$ as follows:
$$J_N = \{i \mid \phi_i = 0 \text{ and } g_i^T\dot{q} = 0 \text{ at } t=0\}.$$
We exclude the possibility of a j such that $\phi_j(q(0))<0$, i.e. the initial conditions are compatible with the constraints. It may be necessary to introduce an impulse in the system at $t=0$ so that $g_j^T\dot{q} \geqq 0$ for every j with $\phi_j = 0$. In the restart step $\ddot{q}_{n-1}$ is not available for the local error estimate. The first step size h_n in Step 1 is instead taken to be h_{n-1}, the old step size, and when Step 2 has been completed, the local error in q_n is estimated using $\ddot{q}_n$. If (3.5) is not satisfied, then Steps 1 and 2 are repeated with a new step size. The threshold value ε in Step 4 is chosen to avoid unnecessary zigzagging; i.e., a constraint that left J_N at t_{n-1} is reintroduced in J_N between t_{n-1} and t_n(cf. nonlinear optimization in Fletcher [13, p. 113]). If $\Delta_{nj} \sim \varepsilon$, then $\lambda_{nj}=0$ and λ_{nj} does not affect the equations of motion, even if j is not dropped from J_N.

3.2. The quadratic programming problem. The velocity $\dot{q}_n$ can be calculated directly as the solution to (3.4) without introducing λ_n. It is, however, advantageous to solve (3.2) for λ_n first and then insert the result into (3.1b) to obtain $\dot{q}_n$. The merits of (3.2) in comparison with (3.4) are

1. An initial, feasible λ_n^0 in an active set algorithm for QP problems is easily found.
2. The QP problem for friction-free systems is a special case of the QP problem for systems with Coulomb friction.
3. The multiplier vector λ_n is sometimes of interest itself, e.g. in friction problems.

It is well known from classical mechanics that in time-independent problems, the contact forces cannot be uniquely determined due to the fact that the columns of G are linearly dependent ("statical indeterminacy"). This can of course also be the case in dynamical problems. In Lötstedt [30] it is shown that $G\lambda$ in (2.1) is always unique even if G does not have full column rank.

We shall consider a slightly generalized QP problem (3.2) and study the uniqueness of the solution and the sensitivity to perturbations. The generalization is motivated by our Coulomb friction model in § 4. The new QP problem is
$$\begin{aligned} &\min \tfrac{1}{2}x^TA^TAx + x^TA^Td, \qquad a_i \leqq x_i \leqq b_i, \quad i=1,2,\cdots,l, \\ &\qquad\qquad a_i \leqq x_i, \ i=l+1, l+2, \cdots, k, \end{aligned} \tag{3.15}$$
where $A \in \mathbb{R}^{m\times k}$, $d \in \mathbb{R}^m$, a, $x \in \mathbb{R}^k$ and $b \in \mathbb{R}^l$. If the objective function in (3.15) is replaced by $\|Ax+d\|$, we have transformed (3.15) to a linear least squares problem with the same solution x_*. If x_* is not unique then on the manifold of solutions choose x_* satisfying
$$\min \tfrac{1}{2}x^Tx, \qquad x \text{ solves (3.15)}. \tag{3.16}$$

In the perturbed problem ((3.15), (3.16)), denoted by ($(3.15)_\delta$, $(3.16)_\delta$), the quantities A, d, a and b are perturbed by δA, δd, δa and δb. The next proposition characterizes the solution to ((3.15), (3.16)) and ($(3.15)_\delta$, $(3.16)_\delta$). In order to state the proposition we need some notation. $R(B)$ denotes the range and $N(B)$ the null space of the matrix B. P_X is the orthogonal projector on the subspace X. Furthermore, let S and S_δ denote the sets of constraints on $x_N \in N(A)$ that *must* be satisfied as *equalities* at the solutions x_* and $x_* + \delta x_*$ to (3.15) and $(3.15)_\delta$, respectively. Let us consider an example. Suppose that $l = 0$, $n = 2$, $a^T = (0, 0)$, $x_*^T = (0, 0)$,

$$R(A^T) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} z_1 \quad \text{and} \quad N(A) = \begin{pmatrix} 1 \\ -1 \end{pmatrix} z_2, \tag{3.17}$$

where z_1 and z_2 are arbitrary. Then $S = \{1, 2\}$. Conversely, if the definitions of $R(A^T)$ and $N(A)$ in (3.17) are interchanged, then $S = \varnothing$. If $N(A) = \varnothing$ then of course $S = \varnothing$.

PROPOSITION 3. *A solution x_* to* (3.15) *always exists and $P_{R(A^T)}x_*$ and Ax_* are unique. If A has full column rank or if x_* also satisfies* (3.16), *then x_* is unique. Define $\eta = \max\{\|\delta A\|, \|\delta d\|, \|\delta a\|, \|\delta b\|\}$. Assume that*

$$\operatorname{rank}(A) = \operatorname{rank}(A + \delta A),$$

$a_i < b_i$, $i = 1, 2, \cdots, l$, and $S = S_\delta$. Then there exist positive c and η_0 such that if $\eta \leqq \eta_0$ then

$$\|\delta x_*\| \leqq c\eta. \tag{3.18}$$

Proof. The existence and uniqueness results follow from [31, Thm. 1]. The conditions of [31, Thm. 3(i) and (iii)] are satisfied and the perturbation bound (3.18) is a consequence of the theorem. □

The proposition is directly applicable to (3.2) with $A = D^{-T}G_n$, $x = \lambda_n$, $d = Dc_n/h_n$, $l = 0$ and $a = 0$. In contact problems where we are not explicitly interested in a unique λ_n, it is sufficient to compute the unique $G_n\lambda_n$.

The QP problem (3.2) is solved by an algorithm exploiting the fact that the objective function does not vary much from step to step. The active set method by Gill and Murray [18] is modified to handle a positive semidefinite A^TA in the step where a new search direction is calculated.

An algorithm similar to that in [18] is described by Lawson and Hanson [25]. The procedure for solving (3.15) with $a = 0$ and $l = 0$ is presented below. The algorithm is justified and a comparison is made in a separate paper (Lötstedt [32]). The index of a variable belongs to one of the sets F, X, or P. The restriction A_F consists of the columns of A corresponding to an index in F.

F is the set of free variables, $x_i \geqq 0$, such that a QR decomposition of A_F with a nonsingular R is available. X is the index set of the fixed variables $x_i = 0$, and P, the passive set, contains the rest of the indices. The parameter ε_1 is an error tolerance.

ALGORITHM 2.
Initialization.
1. If necessary compute a QR decomposition of the maximal number of linearly independent columns in A. Let F contain the indices of the columns represented by the QR decomposition and P the remaining indices.
2. Compute the residual $r = Ax + d$.

Main iteration loop.
3. If $F \neq \varnothing$ then compute a new descent direction p by solving

$$A_F^TA_Fp + A_F^Tr = 0. \tag{3.19}$$

Otherwise go to 7.

4. Let θp, $0 \leqq \theta \leqq 1$, be the maximal step possible in the direction p without violating any constraints. θ satisfies
$$\tau_i = -\frac{x_i}{p_i}, \qquad i \in F_* = \{j \mid j \in F, p_j < 0\}, \quad \theta = \min\left(\min_{i \in F_*} \tau_i, 1\right).$$
5. Update x and r
$$x_i := x_i + \theta p_i, \quad i \in F, \qquad r := r + \theta A_F p.$$
6. If $\tau_i > 1$, $i \in F_*$, in Step 4 then go to 7. For each x_j reaching its lower bound in this step, transfer j from F to X and remove the corresponding column in the QR decomposition of A_F.
Go to 3.
7. Compute the gradient of the objective function
$$\Delta = A^T r = A^T A x + A^T d.$$
8. Find
$$\gamma = \max\left(-\min_{i \in X} \Delta_i, \max_{i \in P} |\Delta_i|\right).$$
and an index j such that $\gamma = -\Delta_j$, $j \in X$, or $\gamma = |\Delta_j|$, $j \in P$. If $\gamma > \varepsilon_1$ then the objective function can be decreased further if x_j becomes a free variable. Transfer j from X or P to F and add the corresponding column to the QR decomposition. Go to 3. If $\gamma \leqq \varepsilon_1$ then an approximate optimum has been found.
9. End of algorithm.

On termination of the algorithm for the problem (3.2) we have
$$r = D^{-T} G_n \lambda_n + h_n^{-1} D c_n.$$
Thus,
$$\dot{q}_n = h_n D^{-1} r / \alpha_0^2 \quad \text{and} \quad G_n^T \dot{q}_n = h_n \Delta / \alpha_0^2.$$

The algorithm is initialized with $x = \lambda_{n-1}$, the feasible solution from the previous step. Another possibility is to use polynomial extrapolation to obtain a start value. At most time points t_n, the optimum is found after the computation of one descent direction in (3.19). Only in exceptional steps is the lower bound on a variable x_i attained and more than one system (3.19) has to be solved.

The matrix $A = D^{-T} G_n$ varies continuously in time and not very much from step to step. Therefore, A_F is not factorized at each step, but the old R-matrix is used to accelerate a conjugate gradient algorithm for solution of (3.19) as described by Björck and Elfving [3, p. 156]. The convergence rate for $A_F R^{-1}$ in (3.19) is superior to that of A_F if R is part of a recent factorization. The old QR decomposition is updated in Steps 6 and 8. The column added in Step 8 is a fresh column of A computed at t_n. If $j \in X$ at Step 9, then the corresponding constraint ϕ_j is most likely dropped from J_N at Step 4 in Algorithm 1. The QR decomposition is already prepared in Algorithm 2 for this change in J_N. The factorization of A is obtained by Householder transformations (Businger and Golub [4]), and the determination of its rank is based on Karasalo [23]. Methods for updating the decomposition are described in Gill et al. [16].

The criterion for computing a new factorization is derived as follows. Denote by T the total number of calls of Algorithm 2, i.e. the number of steps taken, the cost in floating point operations of a factorization by C_{QR} and the total number of decompositions performed by n. Assume that the cost due to the iterative solution of (3.19) is $C_0+C_{IT}\cdot s^r$, C_0, C_{IT}, $r>0$, where s is the number of steps since the last decomposition and s^r is the number of cg-iterations per step. This is a reasonable assumption if (3.19) is solved only once per step. In a steady-state situation, we have that the cost C_c in one cycle from one factorization up to the next is

$$C_c \approx C_{QR} + \int_0^{T/n} (C_0 + C_{IT}\cdot s^r)\,ds.$$

Thus, the total cost is

$$C(n) \approx C_{QR} n + C_0 T + \frac{C_{IT} T^{r+1} n^{-r}}{r+1}.$$

The optimal number of steps per cycle is

$$\frac{T}{n} \approx \left(\frac{r+1}{r}\frac{C_{QR}}{C_{IT}}\right)^{1/(r+1)},$$

rendering $C(n)$ a minimum. Here we also assume that $r=1$. A new QR decomposition is computed when the number of iterations in the cg-algorithm exceeds $(2C_{QR}/C_{IT})^{1/2}$.

The conjugate gradient iterations are terminated when the residual $v = A_F^T A_F p + A_F^T r$ in (3.19) is sufficiently small. The error δp in p satisfies

$$R\delta p \approx R^{-T} A_F^T A_F \delta p = -R^{-T} v.$$

If the loop in Algorithm 2 is repeated only once, then

$$\|M^{-1} G_n \delta\lambda_n\|_M = \|D^{-T} G_n \delta\lambda_n\| = \|A_F \delta p\| \approx \|QR\delta p\| \leqq \|v\|\,\|R^{-1}\|. \tag{3.20}$$

Hence, if $\|v\|$ in (3.20) is less than $\varepsilon_0/\|R^{-1}\|$ then the accuracy requirements in (3.13) are satisfied. $\|R^{-1}\|$ is estimated according to Cline et al. [5]. The parameter ε_1 in Step 8 is taken to be $\varepsilon_0/\|R^{-1}\|$.

If the number of active constraints is always small, say 1–3, there is of course the straightforward method to compute λ_n by guessing which of the components will attain 0 and then solving a system of linear equations for the remaining multipliers. If the complementarity conditions (3.1c) are not satisfied, then we try another probable combination. This approach can be very slow for larger problems without more information on which multipliers we can expect to reach 0.

3.3. Computation of impulses. The treatment of the friction-free impulse problem is almost equal to the treatment of the time-dependent contact problem. The QP problem (2.14) with $\mu_I = 0$ corresponds to (3.2), and (2.13) to (3.1b). An impulse Λ is computed in Step 3 of Algorithm 1 at each time the external impulse vector $F \neq 0$ or a constraint j is included in J_N. A modification of Algorithm 2 that can handle the constraints in (2.14), when $\mu_I > 0$, is used to determine Λ (Gill and Murray [18]). This modified version is for $\mu_I = 0$, equivalent to Algorithm 2. The algorithm always works with the exact QR decomposition. A constraint j is dropped from J_N if $g_{Nj}^T q_+ > \varepsilon$, the same ε as in (3.5).

3.4. Examples. The proposed Algorithms 1 and 2 have been tested on two friction-free model problems. More details about the implementation are given in § 5. In the figures the x-axis is horizontal and x increases to the right. The y-axis is vertical and y is increasing upwards. In the first example in Fig. 2, the boxes are of dimension 1.5×1 and their density is 1. Since the only external force is the gravity force, f in (2.1) is constant. The bodies have no initial velocities except for body 2 where

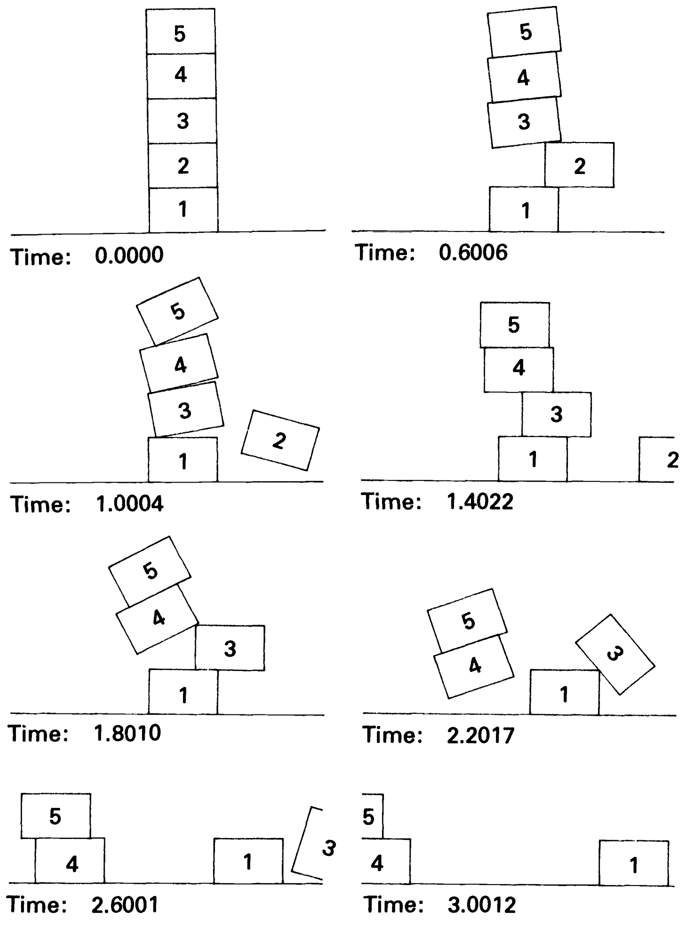


FIG. 2. *The motion of five boxes piled on each other is simulated when the initial velocity of box* 2 *is* 2.0 *in the horizontal x-direction,* $\mu = \mu_I = 0$, $\varepsilon = 0.001$.

$\dot{x}(0) = 2$. The numbers below each picture indicate the time passed from the initial state. When the simulation was interrupted, body 1 moved slowly to the right and the bodies 4 and 5, slowly to the left. In Fig. 3, a homogeneous hexagon is suspended in a linear spring and damper. Its shortest side is 1.5 and its mass is 7.5. The spring and damper constants (cf. (3.6)) are both 6. The external forces are the gravity force and the spring and damper force, $f = f(q, \dot{q})$. The body has no initial velocity. On the ground the hexagon oscillates back and forth until all kinetic energy has been dissipated by the damper. The velocity in the x-direction at $t = 4.60$ is 0.69. The error tolerance ε was 0.001 in both examples.

4. Coulomb friction problems. Coulomb's friction model for the transverse force at a contact between two rigid bodies has several undesirable properties (Lötstedt [29]). The existence of a solution to (2.1), (2.6) and (2.9) cannot be guaranteed, and if a solution exists it is not always unique even if the columns of G are linearly independent. A solution may also be very sensitive to small perturbations in the data. We shall here develop a numerical friction model based on Coulomb's law. The equations in our model always possess a unique solution which is reasonably insensitive to small perturbations. The Lagrange multipliers are determined by a QP problem

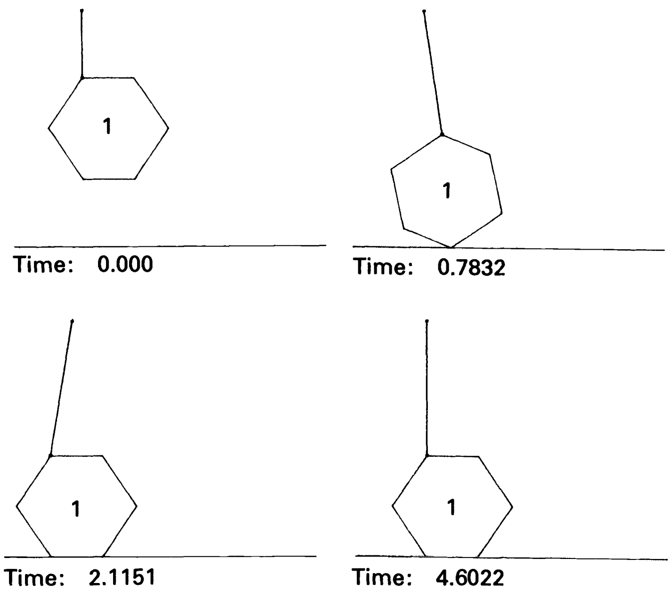


FIG. 3. *The motion of a hexagon is studied. The body is suspended in a linear spring and damper with one end fixed in the inertial frame,* $\mu = \mu_I = 0$, $\varepsilon = 0.001$.

such that the friction-free problem is a special case of the friction problem. A model for friction impulses is also presented. Two examples illustrate the method.

4.1. Advancing the solution in time. The discretization methods for the numerical solution of the system of ordinary differential equations are the same as those selected in § 3. Apply AB-r and BDF-r, $r = 1, 2$, to (2.12) to obtain

$$q_n = \frac{1}{\alpha_0^1} b_n^1, \tag{4.1a}$$

$$\dot{q}_n = \frac{1}{\alpha_0^2}(h_n M^{-1}(f_n + (G_{1n} + H_n U_n)\lambda_{1n}) + b_n^2). \tag{4.1b}$$

where b_n^i, $i = 1, 2$, are defined as in (2.17). The discrete equations (4.1) and the relations (2.6) and (2.9) suffer from the same inconsistencies as the analytical equations (2.12) and the relations (2.6), (2.9), i.e. nonexistence and nonuniqueness of solutions and sensitivity to small perturbations. The sufficient conditions in [29] for existence and uniqueness are not always satisfied by physical systems.

In order to circumvent the difficulties, the computation of $(G_{1n} + H_n U_n)\lambda_{1n}$ is split into two parts.

The first part, $G_{1n}\lambda_{1n}$, is the contribution to the equations of motion of the contact forces and the friction forces at contacts where no sliding takes place. We describe how $G_{1n}\lambda_{1n}$ is determined by a QP algorithm in § 4.2. The second part, $H_n U_n \lambda_{1n}$, is the term of the working friction forces in the equations of motion. In the step from t_{n-1} to t_n the columns of H_n are those $g_{Fj}(t_n)$ for which $j \in J_{FW}$ at t_{n-1}. The signs s_i in (2.10) are such that (2.9b) is satisfied at t_{n-1}. It follows from (2.10) and (2.11) that the only components of λ_{1n} involved in the computation of $H_n U_n \lambda_{1n}$ are those normal multipliers associated with a sliding contact. They are computed by polynomial extrapolation. If $\ddot{q}$ is continuous then λ can be chosen continuous [30]. Let λ_{1j}^i denote the approximate value of $\lambda_{Ni}(t_j)$, $i \in J_F$, solving the QP problem at t_j. If $\ddot{q}$ is not discontinuous between t_{n-1} and t_n and $\lambda^i_{1,n-1}$, and if $\lambda^i_{l,n-2}$ are available, then λ^i_{1n} is approximated by λ^i_{*n} in

$$\lambda^i_{*n} = \lambda^i_{1,n-1} + \frac{h_n(\lambda^i_{1,n-1} - \lambda^i_{1,n-2})}{h_{n-1}}. \tag{4.2}$$

If only $\lambda^i_{1,n-1}$ is available due to a restart of the integration algorithm at t_{n-2} (see Algorithm 2), then

$$\lambda^i_{*n} = \lambda^i_{1,n-1}. \tag{4.3}$$

In the very first step or in the first step after a restart because of a discontinuity in $\dot{q}$, the procedure to obtain λ^i_{*n} is as follows:

1. Compute approximate positions at t_n of the bodies involved in sliding contacts, ignoring the contact constraints, by the formula:

$$q_* = q_{n-1} + h_n\dot{q}_{n-1} + 0.5h_n^2 f_{n-1}.$$

2. Compute the corresponding constraint functions $\phi_i(q_*)$ at t_n.

We have for $\phi_i(q_*)$ that

$$\phi_i(q_*) = \phi_i + h_n g^T_{Ni}\dot{q}_{n-1} + 0.5h_n^2 g^T_{Ni} f_{n-1} + 0.5h_n^2\, \partial^2 g_{Ni}/\partial q^2\, \dot{q}_{n-1}\dot{q}_{n-1} + O(h_n^3),$$

where ϕ_i, g_{Ni} and $\partial^2 g_{Ni}/\partial q^2$ are evaluated at q_{n-1}. Since $\phi_i = 0$ and $g^T_{Ni}\dot{q}_{n-1} = 0$, the third term in the expression for q_* is necessary to include all terms of $O(h_n^2)$ in $\phi_i(q_*)$. Then let λ^i_{*n} be the force from a spring and damper at the point of contact

$$\lambda^i_{*n} = \max\{0, -(dg^T_{Ni}(q_*)\dot{q}_{n-1} + k\phi_i(q_*))\}, \tag{4.4}$$

with suitably chosen spring and damper constants k and d. After a jump in $\ddot{q}$, (4.3) is used. Now that we have λ^i_{*n} from (4.2), (4.3) or (4.4), substitute

$$\sum_{i\in J_{FW}} g_{Fi}(q_n)s_i\mu\lambda^i_{*n}, \qquad s_i = -\text{sign}\,(g^T_{Fi}\dot{q}(t_{n-1})),$$

for $H_nU_n\lambda_{1n}$ in (4.1b). If this term from the working friction forces is included in the sum of the terms explicitly given by the previous time steps b_n^2, then (4.1b) has the same form as (3.1b):

$$\dot{q}_n = \frac{1}{\alpha_0^2}(h_nM^{-1}(f_n + G_{1n}\lambda_{1n}) + b_n^2). \tag{4.5}$$

Assuming that $\lambda_1(t)$ and λ_{1n} are uniquely determined by (2.12), (4.1), (2.6) and (2.9) initially or after a discontinuity, we shall examine the errors introduced by replacing λ_{1n} by λ^i_{*n}, $i = 1, 2, \cdots$. Observe that if both $\lambda_1(t)$ and $\lambda_2(t)$ fulfill (2.12), (2.6) and (2.9) and $\lambda_1 - \lambda_2 \notin N(G_1 + HU)$, then there are two different solutions q_1 and q_2 to (2.12). If $\ddot{q}$ and λ are continuous, then the additional local errors in $\dot{q}_n$ caused by λ^i_{*n} in (4.2) are of $O(h_n^3)$ when the quotient h_{n-1}/h_n is of $O(1)$. The additional local error in $\dot{q}_n$ is of $O(h_n^2)$ when (4.3) is used or when (4.2) is used and λ is discontinuous. The order of the local error due to the discretization methods employed in combination with (4.2) or (4.3) is the same as the order of the local error due to the approximations (4.2) or (4.3). Hence, the order of the local and global errors in $\dot{q}_n$ and q_n is not altered. The local errors introduced in $\dot{q}_n$ immediately after a discontinuity in $\dot{q}$ or $\ddot{q}$ are of $O(h_n)$. The inconsistencies in the analytical problem exemplified in [29] occurred after a discontinuity in $\dot{q}$. Since the solution may fail to exist after such an event, it is not always meaningful to discuss rates of convergence for the numerical solution there.

The time t_*, $t_* \leqq t_n$, such that $g_{Fj}^T\dot{q}(t_{n-1}) \neq 0$, $j \in J_{FW}(t_{n-1})$, and $g_{Fj}^T\dot{q}(t_*) = 0$, $j \in J_{F0}(t_*)$, is calculated by inverse linear interpolation. Since $\ddot{q}$ in general has a jump at t_*, the integration algorithm is restarted from that point. The additions to Algorithm 1 in Steps 3 and 4 motivated by the friction algorithm are:

3. Test whether
   (ii)′ $\ddot{q}$ is discontinuous between t_{n-1} and t_n because $g_{Fj}^T\dot{q}$ has reached 0 in the interval.
   If (ii)′ has occurred then update J_{FW} and J_{F0}.
4. Test whether there is a multiplier λ_{Fj} in λ_{1n} such that $j \in J_{F0}(t_{n-1})$, but $|\lambda_{Fj}| = \mu\lambda_{Nj}$ and $|g_{Fj}^T\dot{q}_n| > \varepsilon$. If that is the case then transfer j from J_{F0} to J_{FW}.

4.2. The quadratic programming problem. The remaining task in the time-marching algorithm is to show how to compute $G_{1n}\lambda_{1n}$ in (4.5). Since we wish the friction-free algorithm to be a special case of the friction algorithm, we accept λ_{1n} as the solution to

$$\begin{aligned} &\min \tfrac{1}{2}\lambda^T G_{1n}^T M^{-1} G_{1n}\lambda + h_n^{-1}\lambda^T G_{1n}^T c_n, \\ &\lambda_{Ni} \geqq 0, \quad i \in J_N(t_{n-1}), \qquad -a_i \leqq \lambda_{Fi} \leqq a_i, \quad i \in J_{F0}(t_{n-1}). \end{aligned} \tag{4.6}$$

The bound on $|\lambda_{Fi}|$ is $a_i = \mu\lambda_{*n}^i$. The Kuhn–Tucker conditions [13] satisfied at the optimum of (4.6) are

$$\begin{aligned} &\lambda_{Ni} \geqq 0, \quad g_{Ni}^T\dot{q}_n \geqq 0, \quad \lambda_{Ni}g_{Ni}^T\dot{q}_n = 0, \quad |\lambda_{Fi}| \leqq a_i, \quad \lambda_{Fi}g_{Fi}^T\dot{q}_n \leqq 0, \\ &(a_i - |\lambda_{Fi}|)g_{Fi}^T\dot{q}_n = 0. \end{aligned} \tag{4.7}$$

These conditions on the friction multipliers are the same as those imposed by Coulomb's law in (2.9b) and (2.9c) except for the approximate upper bound on $|\lambda_{Fi}|$.

It follows from Proposition 3 that if λ_{1n} solves (4.6) then $G_{1n}\lambda_{1n}$ is unique. Unfortunately, if G_{1n} does not have full column rank then the components λ_{Nj} of λ_{1n} are not necessarily unique. Any solution λ_{2n}, satisfying the constraints and such that $\lambda_{1n} - \lambda_{2n} \in N(G_{1n})$, is a possible minimizer. Thus, by (4.2) and (4.3), a_i in (4.6) and the substitution for $H_nU_n\lambda_{1n}$ are not necessarily unique. As a remedy we introduce the *principle* of choosing the λ_{1n} among the solutions of (4.6) rendering $\lambda_{1n}^T\lambda_{1n}$ a minimum. The actual computation is performed in two stages. Firstly, a λ_* solving (4.6) is determined by Algorithm 2 modified in Steps 4, 6 and 8 to allow for upper and lower bounds on the variables (see Gill and Murray [17]). Secondly, with λ_* as the initial feasible starting point, $\lambda^T\lambda$ is minimized by reducing the component of λ_* in $N(G_{1n})$. This is achieved by the following method.

Compute the QR decomposition of G_{1n}^T by Householder transformations [4]. Let G_{1n} have the dimension $m \times l$ and $r = \text{rank}(G_{1n})$. If $r < l$ then Q can be partitioned $Q = (Q_1, Q_2)$, $Q_1 \in \mathbb{R}^{l \times r}$, $Q_2 \in \mathbb{R}^{l \times (l-r)}$, where Q_2 spans $N(G_{1n})$ [18]. Let $Q_2z = P_{N(G_{1n})}\lambda$ and $\lambda_R = P_{R(G_{1n}^T)}\lambda_*$. The projection λ_R is unique according to Proposition 3. Then seek the optimum of

$$\begin{aligned} &\min \tfrac{1}{2}z^T Q_2^T Q_2 z = \tfrac{1}{2}z^T z, \\ &\lambda = Q_2z + \lambda_R \text{ satisfies the constraints in (4.6).} \end{aligned} \tag{4.8}$$

The problem (4.8) is a QP problem in z termed the "least-distance programming problem" [25, p. 159]. The solution z_* is calculated by Gill and Murray's method [18] adapted to the special properties of the problem. Note that the initial $z = Q_2^T\lambda_*$ is consistent with the constraints. The final solution λ_{1n} to (4.6) and (4.8) is obtained directly from (4.8) as $Q_2z_* + \lambda_R$. The quantities $G_{1n}\lambda_{1n}$ and $\dot{q}_n$ in (4.5) are results available on termination of the algorithm for solving (4.6); cf. the friction-free case. The formulation (4.8) of the problem of minimizing $\lambda^T\lambda$ requires one QR decomposition per time step. A method where old decompositions are utilized to accelerate a conjugate gradient procedure for the problem, is described and compared with the approach in (4.8) in Lötstedt [32].

If $f=f(q,\dot{q},t)$ in (4.5), then an iterative procedure for solution of (4.5) and (4.6) corresponding to (3.9) in the friction free case converges to the unique solution $\dot{q}_n$ of (4.5) under the assumptions of Proposition 2. We shall only sketch the proof of this assertion here. Let y_0, $y_1=y_0+\delta y$, f_n^0, and $f_n^1=f_n^0+\delta f$ be defined by

$$y_i=\frac{1}{\alpha_0^2}(h_nM^{-1}(f_n^i+G_{1n}\lambda_{1n}^i)+b_n^2),\quad f_n^i=f(q_n,x_i,t_n),\quad i=0,1,$$

where λ_{1n}^0 and $\lambda_{1n}^1=\lambda_{1n}^0+\delta\lambda$ solve the associated problems (4.6). Then apply (3.11) to (4.6) to obtain

$$(D^{-T}G_{1n}\delta\lambda)^T(D^{-T}G_{1n}\delta\lambda)\leqq-(D^{-T}G_{1n}\delta\lambda)^TD^{-T}\delta f.$$

It follows from the definition of δy and the above inequality that

$$\|\delta y\|_M\leqq h_n\|M^{-1}\delta f\|_M/\alpha_0^2.$$

Hence, (3.12) holds true and the rest of the proof is identical to the last part of the proof of Proposition 2.

The conclusion from Proposition 3 applied to (4.6) and (4.8) is that under reasonable assumptions, the solution λ_{1n} is well behaved when the data is subject to small perturbations. It is clear from the definition of λ_{*n}^i in (4.4) after a discontinuity in $\dot{q}$ at t_*, (4.6) and (4.8) that the solution $\dot{q}_n$ at $t_n=t_*+h_n$ does not blow up as $h_n\to 0$ even if the analytical solution fails to exist for $t>t_*$. Finally, it should be pointed out that the algorithm presented above for computation of time-dependent systems obeying Coulomb's law is not the only possible modification of the law.

4.3. Computation of impulses. The model for friction impulses is defined by the QP problem (2.14). By [31, Thm. 1] $G\Lambda$ and $\dot{q}_+$ in (2.13) are unique. The problem (2.14) is solved by the modified version of Algorithm 2 mentioned in § 3.3.

We shall study the optimal solution of (2.14) by means of the Kuhn–Tucker conditions when $\mu_I\geqq 0$. Suppose that $J_N=\{1,2,\cdots,n\}$ and $J_F=\{1,2,\cdots,k\}$, $k\leqq n$. Let the variables Λ_{Ni}, Λ_{Fi} be ordered in Λ such that

$$\Lambda^T=(\Lambda_{N1},\Lambda_{N2},\cdots,\Lambda_{Nn},\Lambda_{F1},\Lambda_{F2},\cdots,\Lambda_{Fk}).$$

Then the constraints in (2.14) can be written in matrix form,

$$W\Lambda\geqq 0,\qquad W=\begin{pmatrix} I_1 & 0\\ \mu_IE & -I_2\\ \mu_IE & I_2\end{pmatrix},$$

where I_1 and I_2 are the $n\times n$ and $k\times k$ identity matrices and $E\in\mathbb{R}^{k\times n}$, $E_{ij}=\delta_{ij}$, the Kronecker delta. The dual problem to (2.14) is (Dorn [10]),

$$\text{(4.9)}\quad \min\tfrac{1}{2}u^TG^TM^{-1}Gu,\qquad W^Tv=G^TM^{-1}Gu+G^T\dot{q}_-+G^TM^{-1}F,\quad v\geqq 0.$$

Since $Gu=G\Lambda$ by [10], (4.9) is equivalent to

$$\text{(4.10a)}\qquad \min\tfrac{1}{2}(\dot{q}_+-(\dot{q}_-+M^{-1}F))^TM(\dot{q}_+-(\dot{q}_-+M^{-1}F)),$$

$$\text{(4.10b, c)}\qquad W^Tv=G^T\dot{q}_+,\qquad v\geqq 0.$$

By virtue of (4.10b) and the Kuhn–Tucker conditions [13], Λ and $\dot{q}_+$ fulfill

$$v^TW\Lambda=\Lambda^TG^T\dot{q}_+=0. \tag{4.11}$$

Let us interpret (4.10b) and (4.11) for the individual contacts. For $i=k+1, k+2, \cdots, n$, the relations (2.15) are obtained.

For $i\leqq k$, $\Lambda_{Ni}>0$ and $|\Lambda_{Fi}|<\mu_I\Lambda_{Ni}$, we have $v_i=v_{n+i}=v_{n+k+i}=0$ and $g_{Ni}^T\dot{q}_+=g_{Fi}^T\dot{q}_+=0$. If $\Lambda_{Ni}=0$ then $v_i\geqq 0$ and $g_{Ni}^T\dot{q}_+\geqq 0$. The case that differs from the friction force case in § 4.2 is when $|\Lambda_{Fi}|=\mu_I\Lambda_{Ni}>0$. If $\Lambda_{Fi}=\mu_I\Lambda_{Ni}$ then $v_i=v_{n+k+i}=0, v_{n+i}\geqq 0$, and $g_{Fi}^T\dot{q}_+=-v_{n+i}\leqq 0$, $g_{Ni}^T\dot{q}_+=\mu_Iv_{n+i}\geqq 0$. If $\Lambda_{Fi}=-\mu_I\Lambda_{Ni}$ then $v_i=v_{n+i}=0$, $v_{n+k+i}\geqq 0$, and $g_{Fi}^T\dot{q}_+=v_{n+k+i}\geqq 0$, $g_{Ni}^T\dot{q}_+=\mu_Iv_{n+k+i}\geqq 0$. Hence, if $\mu_I>0$ and the relative velocity in the tangential direction between the bodies involved in a contact is nonzero after the collision, then the contact disappears immediately. The bodies merely "bounce" on each other. The effect on the kinetic energy $T=\frac{1}{2}\dot{q}^TM\dot{q}$ before and after the collision is, taking (4.11) into account,

$$\begin{aligned}2\Delta T&=2(T_+-T_-)=\dot{q}_+^TM\dot{q}_+-\dot{q}_-^TM\dot{q}_-=(\dot{q}_++\dot{q}_-)^TM(\dot{q}_+-\dot{q}_-)\\&=(\dot{q}_++\dot{q}_-)^T(F+G\Lambda)\\&=\dot{q}_-^TG\Lambda+(\dot{q}_++\dot{q}_-)^TF=\sum_{i\in J_N}\Lambda_{Ni}g_{Ni}^T\dot{q}_-+\sum_{i\in J_F}\Lambda_{Fi}g_{Fi}^T\dot{q}_-+(\dot{q}_++\dot{q}_-)^TF.\end{aligned} \tag{4.12}$$

The first term in the resulting expression for $2\Delta T$ in (4.12) is always nonpositive since $\Lambda_{Ni}\geqq 0$, $g_{Ni}^T\dot{q}_-\leqq 0$, $i\in J_N$. The second term can be positive if there is a j such that $g_{Fj}^T\dot{q}_-\neq 0$ and sign $(g_{Fj}^T\dot{q}_-)=$ sign (Λ_{Fj}), e.g. if sign $(g_{Fj}^T\dot{q}_+)=-$sign $(g_{Fj}^T\dot{q}_-)$. However, if $g_{Fi}^T\dot{q}_-=0$ or sign $(g_{Fi}^T\dot{q}_-)=-$sign (Λ_{Fi}), $i\in J_F$, and $F=0$ then $\Delta T\leqq 0$.

This friction impulse model was chosen for reasons of simplicity and consistency with the other contact and friction models in §§ 3.2, 3.3 and 4.2. It depends on the particular application if it is acceptable or not. The choice of model here is important since the difference in $\dot{q}_+$ for different models is of $O(1)$.

4.4. Examples. The method for systems obeying Coulomb's friction law has been tested on two model problems. The dimension of the boxes in Fig. 4 is 3×1 and their density is 1. The friction coefficient $\mu=0.5$ and there is no friction impulse $\mu_I=0$. The gravity is the only external force in both Figs. 4 and 5. Initially, an external impulse $F_1^T=(F_{1x}, F_{1y}, F_{1\psi})=(10, 0, -10)$ is applied to body 1 simulating a push in the positive x-direction in the upper left corner. The bodies were at rest in the final picture in Fig. 4. The system at $t=2.0073$ is statically indeterminate, since the G-matrix has 9 rows (three coordinates per body) and 11 columns (two multipliers per point contact and three per edge-to-edge contact; see § 5). The configurations in the first and the last pictures of Fig. 5 are also indeterminate: G does not have full column rank. The dimension of the boxes is 2×1 and their density is 1. The values of the friction coefficients are $\mu=0.5$ and $\mu_I=0.8$. The initial velocity of body 2 is $\dot{x}(0)=4$. All other initial velocities are zero. The system is at rest in the last picture in Fig. 5. The error tolerance ε was 0.001 in both examples. The first example was also run with $\mu_I=0.8$. With this choice of μ_I more zigzagging was observed due to the effects of the friction impulse model explained in § 4.3.

5. Practical considerations. In this section we shall discuss two details of general interest in the program that produced Figs. 2, 3, 4 and 5. The first issue is how the constraint functions ϕ_i and $g_{Fi}^T\dot{q}$ are defined. Then the data structure is described for the system of rigid bodies subject to contact and friction constraints.

Suppose that the number of corners of each body is k. Since there are k edges on each body, there are k^2 potential point contacts corner-to-edge for each body with

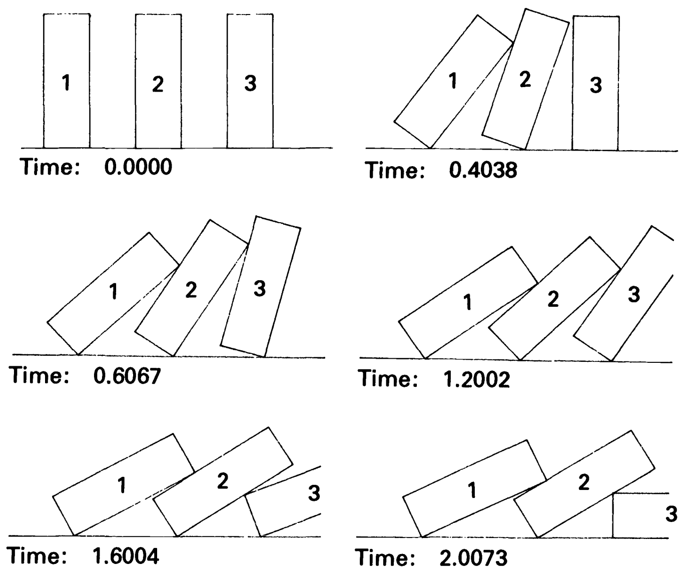


FIG. 4. *The motion of three boxes colliding with each other is simulated when body* 1 *is given an initial push in the upper left corner,* $\mu = 0.5$, $\mu_I = 0$, $\varepsilon = 0.001$.

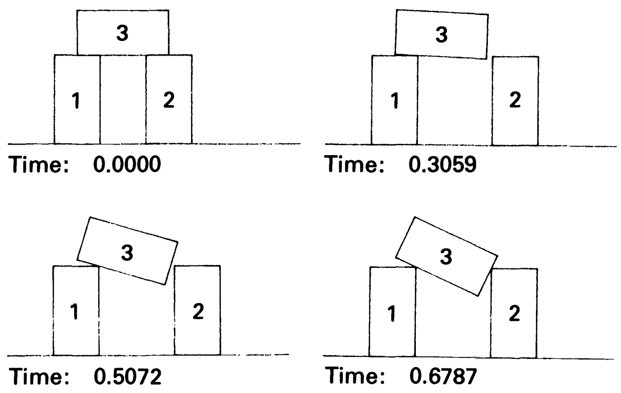


FIG. 5. *Body* 2 *has the initial velocity* 4.0 *in the horizontal x-direction. The motion of the system is simulated with* $\mu = 0.5$, $\mu_I = 0.8$ *and* $\varepsilon = 0.001$.

respect to every other body. In a system with n bodies, there are $k^2n(n-1)$ possible constraint functions ϕ_i between the bodies and then there are kn possible ϕ_i between the bodies and the ground. Therefore, the introduction of a constraint in the program must be done with selection. Initially, one constraint function ϕ_i is defined for each point of contact between a corner and an edge. A contact between an edge of one body and an edge of another body is consequently described by ϕ_i at the corner in one end of the line of contact and by ϕ_{i+1} at the other corner. The user is monitoring the progress of the simulation interactively on a graphics terminal and introduces a new constraint $\phi_i > 0$ by a simple command when he discovers a collision risk. A constraint j is removed only when ϕ_j passes a positive threshold value. The bookkeeping of the constraints is done automatically in Cundall [7]. In order to register a collision a corner must hit the line defining an edge within the two ends of the edge. Observe that the condition $\phi_j(t_*) = 0$ is only a necessary condition for collision. If $\mu > 0$ and two bodies have collided at t_*, $\phi_j(t) > 0$ for $t < t_*$ but $\phi_j(t_*) = 0$, then the corresponding friction constraint function $g_{Fj}^T\dot{q}$ is introduced by the program at t_*. In an edge-to-edge contact the friction constraint j is associated with two contact constraints $\phi_j = \phi_{j+1} = 0$. The upper bound a_j for $|\lambda_{Fj}|$ in (4.6) is $\mu(\lambda_{Nj} + \lambda_{N,j+1})$.

The data structure for rigid body systems such as those in the examples must contain easily accessible information on the properties of the bodies, the properties of the active constraints and relations between the bodies and the constraints. It should also be flexible enough to allow for simple creation and deletion of active constraints. From this data the discretized equations of motion (3.1) and (4.1) are assembled at each step. The data structure for a simple system is displayed in Fig. 6. Each body and each active constraint are represented by a vertex in a directed graph. The ground is a separate vertex. The arcs in the graph correspond to connections between the bodies and the constraints. The vertex A is the first vertex in the linked list of body vertices. Similarly, the vertex B is the head of the list of active constraints. For every constraint vertex there is a Lagrange multiplier to compute in (3.1) or (4.1). Constraints belonging to the same contact are connected by arcs. There are two arcs from a constraint to the two bodies involved in the constraint. It is a simple task to update the data structure when constraints are to be inserted or deleted. The program was implemented in SIMULA [2]. SIMULA is a general purpose language where graph structures are manipulated conveniently. With the class concept it is natural to represent the bodies and the constraints as instances of different classes. Each vertex in Fig. 6 contains information on the body (geometry, mass, moment of inertia, etc.) or the constraint (geometry of the contact, sign of μ, etc.). The arcs are implemented as reference variables.

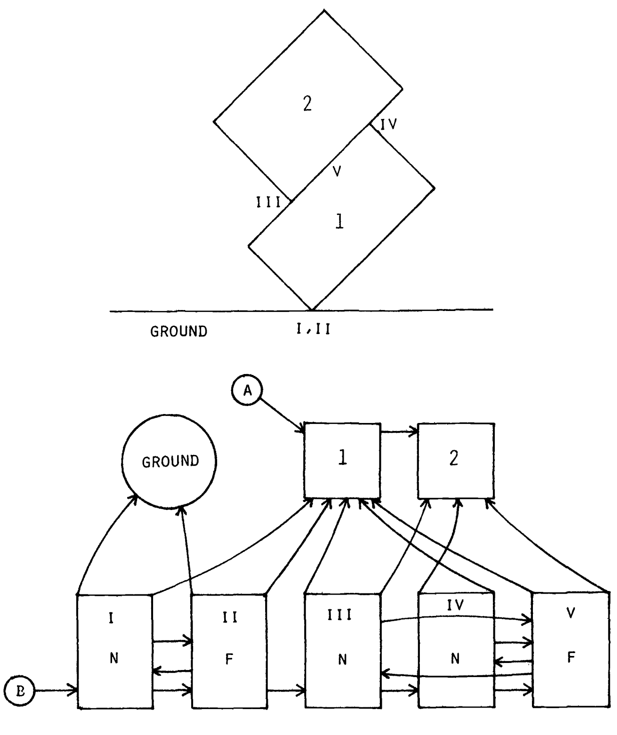


FIG. 6. *The data structure in the lower half of the figure represents the mechanical system in the upper half. The constraints are numbered by roman numerals. The contact constraints are denoted by N and the friction constraints by F.*

6. Generalizations. The numerical procedure developed in §§ 3 and 4 was tested on two-dimensional rigid body systems where the bodies had polyhedral boundary. It is a straightforward generalization to introduce wheels into the systems. Then the constraint function ϕ_i denotes the shortest distance between the periphery of the wheel and a corner or an edge of a polyhedral body or the periphery of another wheel. The friction constraint function $g_F^T\dot{q}$ is the relative velocity in the direction tangential to the periphery of the wheel between the points of contact on the wheel and the other body. Extensions to arbitrary bodies are of course possible, but from a computational point of view, bodies with a simple geometry of their boundary are preferred.

Other kinds of constraints are possible in the framework of §§ 3 and 4. Since the holonomic constraint function ϕ_i never appears explicitly in the numerical formulation (3.1), we can obtain solutions to systems with nonholonomic constraint functions $g_i^T(q)\dot{q}$ by the same method. There are no upper or lower bounds on the Lagrange multipliers λ_i corresponding to bilateral constraints $\dot{\phi}_i = g_i^T\dot{q} = 0$, $t \geqq 0$. In the simple plasticity model for reinforcement steel bars in [28], there is only an upper bound on the Lagrange multiplier proportional to the force in the bar. After a few minor modifications of the algorithms in §§ 3.2 and 4.2, the last two examples of constraints can also be treated. The third space dimension is essential in many applications. In three dimensions, M in (2.1) is still symmetric and positive definite almost everywhere but not constant, $M = M(q)$. The algorithms in § 3 rely, not on M being constant and diagonal, but on the existence of a factorization (2.2). Wittenburg [39] has developed a computational method for assembling M and f in (2.1) such that multipliers corresponding to bilateral constraints are eliminated. The method can be extended to include also systems with unilateral constraints. The constraint functions ϕ_i are defined in a way similar to the planar case, but the geometry of the bodies in three dimensions is more complex. An example of such a mechanical system is the human body; see § 1. The angles between the upper arm and the forearm and between the thigh and the lower part of the leg belong to the interval $[0, \pi)$, and the chin cannot penetrate the chest. Another potential area is the simulation of industrial robots. Coulomb friction in three space dimensions requires a nontrivial extension of Algorithm 2. The natural way to generalize (2.9) is to introduce $\lambda_{Fi} = (\lambda_{Fi}^1, \lambda_{Fi}^2) \in \mathbb{R}^2$. λ_{Fi}^1 and λ_{Fi}^2 are proportional to the two orthogonal components of the friction force on the slip surface. The condition on the modulus of λ_{Fi} that is reduced to (2.9a) in two dimensions is

$$(\lambda_{Fi}^1)^2 + (\lambda_{Fi}^2)^2 \leqq \mu^2 \lambda_{Ni}^2. \tag{6.1}$$

The constraint (6.1) on λ_{Fi} is quadratic, whereas the constraints in (4.6) are linear. In special cases, the structure of the problem is such that the introduction of quadratic constraints on the multipliers is not necessary.

Finally, we wish to mention another area of engineering where the governing equations resemble the equations and relations (2.1) and (2.6): the simulation of electrical networks. As an example, a simple model for a diode is considered. Suppose that in a branch of a network we have a diode, a resistor R and an inductance L. The branch voltage is v and i is the branch current. Then i satisfies

$$L\frac{di}{dt} + Ri = v + \lambda, \qquad i \geqq 0, \quad \lambda \geqq 0, \quad \lambda i = 0. \tag{6.2}$$

The Lagrange multiplier λ can be regarded as the voltage drop over the diode. Electrical networks, whose components have this and similar behavior, can be simulated by the methods in §§ 3 and 4. In (6.2) the complementarity condition is particularly simple.

REFERENCES

[1] J. BAUMGARTE, *Stabilization of constraints and integrals of motion in dynamical systems*, Comput. Methods Appl. Mech. Engrg., 1 (1972), pp. 1–16.

[2] G. M. BIRTWHISTLE, O.-J. DAHL, B. MYRHAUG AND K. NYGAARD, SIMULA BEGIN, Auerbach, Philadelphia, 1973.

[3] Å. BJÖRCK AND T. ELFVING, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[4] P. BUSINGER AND G. H. GOLUB, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.

[5] 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.

[6] R. W. COTTLE, *Numerical methods for complementarity problems in engineering and applied science*, Computing Methods in Applied Sciences and Engineering, 1977, I, R. Glowinski and J. L. Lions, eds., Springer, Berlin, 1979, pp. 37–52.

[7] P. A. CUNDALL, *Rational design of tunnel supports*, AD/A-00162, Dept. Civil and Mining Engineering, Univ. Minnesota, Minneapolis, 1974.

[8] G. DAHLQUIST AND Å. BJÖRCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[9] J. DIEUDONNÉ, *Foundations of Modern Analysis*, Academic Press, New York, 1969.

[10] W. S. DORN, *Duality in quadratic programming*, Quart. Appl. Math., 18 (1960), pp. 155–162.

[11] H. DUGOFF AND R. W. MURPHY, *The dynamic performance of articulated highway vehicles—A review of the state of the art*, SAE paper 710223, 1971.

[12] G. E. FLEISCHER, *Multibody system applications and simulations at the Jet Propulsion Laboratory*, in Dynamics of Multibody Systems, K. Magnus, ed., Springer, Berlin, 1978, pp. 36–47.

[13] R. FLETCHER, *Practical Methods of Optimization, Vol. 2, Constrained Optimization*, John Wiley, Chichester–New York, 1981.

[14] B. FREDRIKSSON, *On elastostatic contact problems with friction*, Ph.D. thesis, Dept. of Mechanical Engineering, Linköping Institute of Technology, Linköping, Sweden, 1976.

[15] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[16] P. E. GILL, G. H. GOLUB, W. MURRAY AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.

[17] P. E. GILL AND W. MURRAY, *Minimization of a non-linear function subject to bounds on the variables*, NAC-72, National Physical Laboratories, Teddington, Middlesex, 1976.

[18] ———, *Numerically stable methods for quadratic programming*, Math. Programming, 14 (1978), pp. 349–372.

[19] E. HAUG, R. CHAND AND K. PAN, *Multibody elastic contact analysis by quadratic programming*, J. Optim. Theory Appl., 21 (1977), pp. 189–198.

[20] R. L. HUSTON, R. E. HESSEL AND J. M. WINGET, *Dynamics of a crash victim—A finite segment model*, AIAA J., 14 (1976), pp. 173–178.

[21] J. J. KALKER, *A survey of the mechanics of contact between solid bodies*, Z. Angew. Math. Mech., 57 (1977), pp. T3–T17.

[22] S. KARAMARDIAN, *The nonlinear complementarity problem with applications, Part* I, J. Optim. Theory Appl., 4 (1969), pp. 87–98.

[23] I. KARASALO, *A criterion for truncation of the* QR-*decomposition algorithm for the singular linear least squares problem*, BIT, 14 (1974), pp. 156–166.

[24] A. KERR AND J. L. BLAIR, *Simulation of the longitudinal dynamics of a train*, in Dynamics of Multibody Systems, K. Magnus, ed., Springer, Berlin, 1978, pp. 108–119.

[25] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[26] P. LÖTSTEDT, *On a penalty function method for the simulation of mechanical systems subject to constraints*, TRITA-NA-7919, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1979.

[27] ———, *A numerical method for the simulation of mechanical systems with unilateral constraints*, TRITA-NA-7920, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1979.

[28] ———, *Interactive simulation of the progressive collapse of a building, revisited*, TRITA-NA-7921, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1979.

[29] ———, *Coulomb friction in two-dimensional rigid body systems*, Z. Angew. Math. Mech., 61 (1981), pp. 605–615.

[30] ———, *Mechanical systems of rigid bodies subject to unilateral constraints*, SIAM J. Appl. Math., 42 (1982), pp. 281–296.

[31] ———, *Perturbation bounds for the linear least squares problem subject to linear inequality constraints*, BIT, 23 (1983), pp. 500–519.

[32] ———, *Solving the minimal least squares problem subject to bounds on the variables*, TRITA-NA-8215, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1982; BIT, to appear.

[33] N. ORLANDEA AND D. A. CALAHAN, *A sparsity-oriented approach to the design of mechanical systems*, in Problem Analysis in Science and Engineering, F. H. Branin, Jr. and K. Huseyin, eds., Academic Press, New York, 1977, pp. 361–389.

[34] B. PAUL, *Analytical dynamics of mechanisms—A computer oriented overview*, Mech. Mach. Theory, 10 (1975), pp. 481–507.

[35] L. PETZOLD, *Differential/algebraic equations are not* ODEs, this Journal, 3 (1982), pp. 367–384.

[36] M. R. RAMEY AND A. T. YANG, *A simulation procedure for human motion studies*, J. Biomechanics, 14 (1981), pp. 203–213.

[37] R. SCHWERTASSEK, *Der Roberson/Wittenburg Formalismus and das Programmsystem* MULTIBODY *zur Rechnersimulation von Mehrkörpersystemen*, DFVLR-FB 78-08, Deutsche Forschungs und Versuchsanstalt für Luft und Raumfahrt, Oberpfaffenhofen, West Germany, 1978.

[38] P. N. SHETH AND J. J. UICKER, JR., IMP (*Integrated Mechanisms Program*), *A computer-aided design analysis system for mechanisms and linkage*, Trans. ASME, Ser. B, J. Eng. Indust., 94 (1972), pp. 454–464.

[39] J. WITTENBURG, *Dynamics of Systems of Rigid Bodies*, Teubner, Stuttgart, 1977.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

A GENERAL UPDATING ALGORITHM FOR CONSTRAINED LINEAR LEAST SQUARES PROBLEMS*

ÅKE BJÖRCK†

Abstract. Linear least squares problems which are sparse except for a small subset of dense equations can be efficiently solved by an updating method. Often the least squares solution is also required to satisfy a set of linear constraints, which again can be divided into sparse and dense subsets. This paper develops an updating algorithm for the solution of such problems. The algorithm is completely general in that no restrictive assumption on the rank of any subset of equations or constraints is made.

Key words. sparse least squares, updating methods, constrained least squares

1. Introduction.

Updating algorithms for adjusting a least squares solution when new equations are added have a long history and date back to Gauss, see [4]. Recently there has been a renewed interest in such algorithms for applications to certain sparse east squares problems.

Consider the linear least squares problem

$$\min_x \|b - Ax\|_2, \tag{1.1}$$

where A is a sparse $m \times n$ matrix. The Cholesky factor R defined by $A^TA = R^TR$ is essentially unique (apart from row multipliers of modulus one). In particular, the sparsity structure of R is unique. Therefore the performance of both the method of normal equations and methods which compute R by orthogonal transformations, e.g., [5], depends on the sparsity of R.

A realistic assumption is that $R + R^T$ is at least as full as A^TA, see [1] and [5]. If A contains some nonsparse rows, then from

$$A^TA = \sum_{i=1}^{m} a_i a_i^T, \qquad a_i = i\text{th row of } A,$$

it is seen that these rows will cause catastrophic fill in A^TA and thus also in R. This is a fundamental difficulty in the solution of sparse least squares problems. A corresponding complete loss of sparsity does not take place when solving a sparse system of linear equations $Ax = b$ with a few dense rows.

To solve (1.1) when A has a few rows with a relatively large number of nonzeros it is desirable initially to withhold the nonsparse rows, and then update the solution taking the omitted rows into effect. Note that in this context it is essential that only the solution and not the factorization be updated.

Sometimes, e.g., due to theoretically known relationships among the variables, the solution x is subject to a set of linear constraints. We then have a problem of the form

$$\min_x \|b - Ax\|_2, \quad \text{subject to } Cx = d. \tag{1.2}$$

Again we wish to be able to cope with the situation where some of the constraints are sparse and some dense. Then we also withhold initially the processing of the set of dense constraints.

* Received by the editors March 1, 1982, and in revised form March 15, 1983. This research was supported by the Department of Energy under grant DE-AT03-76ER71030.

† Department of Mathematics, University of Linköping, S-581 83 Linköping, Sweden.

In [5] an updating algorithm is derived for unconstrained problems, where the sparse subproblem has full rank. In [6] several other cases are treated, e.g., updating of an unconstrained sparse problem of full rank when both equations and constraints are added. A more general updating algorithm is described in [2], where both sparse and dense constraints are allowed and where the sparse subproblem may have rank less than n. However, this algorithm is developed using the Peters–Wilkinson algorithm as the basic method and is therefore more complicated than otherwise necessary. Also, even this algorithm is not completely general in that the constraints are assumed to be consistent.

In this paper we will develop a completely general updating algorithm for the constrained linear least squares problem, based on the use of the *QR* decomposition and direct elimination (see e.g. [7, Chap. 21]). Such full generality is perhaps not often required and no attempt has yet been made to implement the algorithm. However, motivation enough is that updating algorithms for any desired special case can easily be derived in a uniform way from the general algorithm given here. The number of such special cases of practical interest is large—there are four sets of equations (corresponding to the classifications sparse/dense and equation/constraint), some of which may be empty, restrictive assumptions about rank and consistency may be made, etc. As an example, several of the six algorithms given in [6] can be derived in this way.

2. Processing the sparse subsystem. A robust algorithm for the constrained least squares problem (1.2) should include a check on the consistency of the constraints $Cx = d$. Therefore it is more practical to allow inconsistency in the problem formulation. We reformulate the problem (1.2) as suggested in [8] as a hierarchical set of minimization problems

$$\min_{x\in S} \|b - Ax\|_2, \qquad S = \{x;\text{ minimizing } \|d - Cx\|_2\},$$
$$A \;\; m\times n, \quad C \;\; p\times n, \quad r = \operatorname{rank}\begin{pmatrix} C \\ A \end{pmatrix}. \tag{2.1}$$

If $r = n$, then (2.1) has a unique solution. In the case when $r < n$, there is a unique solution which minimizes $\|x\|_2$. For many practical purposes it suffices to compute a (nonunique) basic solution, i.e., one with a minimum number of nonzero components.

In the following we refer to $Ax = b$ as "the equations" and $Cx = d$ as "the constraints." We now split the equations and constraints into sparse and dense parts

$$\begin{array}{l} m_1\{ \\ m_2\{ \end{array}\!\begin{pmatrix} A_s \\ A_d \end{pmatrix} x = \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \qquad \begin{array}{l} p_1\{ \\ p_2\{ \end{array}\!\begin{pmatrix} C_s \\ C_d \end{pmatrix} x = \begin{pmatrix} d_s \\ d_d \end{pmatrix}, \tag{2.2}$$

where the subscripts s and d stand for sparse and dense respectively. We assume in the following that the dimensions m_2 and p_2 are small enough so that dense problems of size $(m_2 + p_2) \times n$ can be readily solved by standard algorithms.

We now describe the processing of the sparse subsystems. Our basic approach is to use a sequence of elementary orthogonal and elimination transformations.

Step 1.1. *Processing of the sparse constraints.* Transform the sparse constraints by a sequence of orthogonal transformations and column interchanges into upper trapezoidal form,

$$Q_1 C_s P_1 = \begin{pmatrix} R_{s1} & S_{s1} \\ 0 & 0 \end{pmatrix}\}p_1', \qquad Q_1 d_s - \begin{pmatrix} c_{1s} \\ c_{2s} \end{pmatrix}, \tag{2.3}$$

where

$$p_1' = \operatorname{rank}(C_s), \qquad R_{s1} \text{ upper triangular } p_1' \times p_1'.$$

If the dense constraints are consistent with the first p_1' transformed constraints in (2.3), then we can proceed to step 1.3. In general we must first determine the residual of the transformed sparse constraints before processing the sparse equations. We put

$$u_1 = c_{1s} - (R_{s1} \quad S_{s1})\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{2.4}$$

where we have partitioned the solution x consistently with (2.3).

Step 1.2. *Computing the residuals of the sparse constraints.* Combining (2.4) and the dense constraints we have

$$\begin{pmatrix} I & R_{s1} & S_{s1} \\ 0 & C_{d1} & C_{d2} \end{pmatrix}\begin{pmatrix} u_1 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} c_{1s} \\ d_d \end{pmatrix}.$$

We now zero the block C_{d1} in the system above by a sequence of elimination transformations. The dense constraints are then transformed into

$$(\tilde{C}_{d0} \quad \tilde{C}_{d2})\begin{pmatrix} u_1 \\ x_2 \end{pmatrix} = \tilde{d}_d, \qquad p_2' = \operatorname{rank}(\tilde{C}_{d2}).$$

We next transform this system by a sequence of orthogonal transformations from the left, without any column interchanges, into

$$\begin{pmatrix} E_{d1} & B_d \\ E_{d2} & 0 \end{pmatrix}\begin{pmatrix} u_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} e_{1d} \\ e_{2d} \end{pmatrix}\}p_2', \tag{2.5}$$

where B_d is $p_2' \times (n - p_1')$ and has full row rank. We now see that x_2 can always be chosen so that the first p_2' constraints in (2.5) are exactly satisfied. Moreover, since orthogonal transformations were used in step 1.1 and in the reduction to (2.5) we have

$$\|d - Cx\|_2^2 = \left\|\begin{pmatrix} u_1 \\ c_{2s} \end{pmatrix}\right\|_2^2 + \|e_{2d} - E_{d2}u_1\|_2^2.$$

It follows that $\|d - Cx\|_2$ is minimized, when u_1 is the solution to the least squares problem

$$\min_{u_1} \left\|\begin{pmatrix} e_{2d} \\ 0 \end{pmatrix} - \begin{pmatrix} E_{d2} \\ I \end{pmatrix}u_1\right\|_2. \tag{2.6}$$

The solution of this problem can be expressed in terms of the SVD of the small dense matrix E_{d2}. We now compute modified right-hand sides for the reduced constraints (2.4) and (2.5)

$$(R_{s1} \quad S_{s1})\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = c_{1s}', \qquad c_{1s}' = c_{1s} - u_1, \tag{2.7}$$

and

$$B_d x_2 = e_{d1}', \qquad e_{d1}' = e_{d1} - E_{d1}u_1. \tag{2.8}$$

Step 1.3. *Processing of the sparse equations.* We first apply the permutations from step 1.1 to A_s and partition conformally with $P_1^T x = (x_1 x_2)^T$ to get

$$(A_{s1} \quad A_{s2})\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = b_s.$$

Combining this with the reduced sparse constraints (2.7), we have

$$\begin{pmatrix} R_{s1} & S_{s1} \\ A_{s1} & A_{s2} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} c'_{1s} \\ b_s \end{pmatrix}.$$

We now zero the block A_{s1} in the system above by a sequence of elimination transformations, which transform the sparse equations into

$$\tilde{A}_{s2} x_2 = \tilde{b}_s, \qquad \tilde{A}_{s2} \ m_1 \times (n - p'_1).$$

We next reduce $\tilde{A}_{s2}$ to upper trapezoidal form by a sequence of orthogonal transformations and column interchanges to get

$$(2.9) \qquad Q_2 \tilde{A}_{s2} P_2 = \begin{pmatrix} R_{s2} & S_{s2} \\ 0 & 0 \end{pmatrix} \}m'_1, \qquad Q_2 \tilde{b}_s = \begin{pmatrix} c_{3s} \\ c_{4s} \end{pmatrix},$$

where

$$m'_1 = r_1 - p'_1, \quad \text{and} \quad r_1 = \operatorname{rank}\begin{pmatrix} C_s \\ A_s \end{pmatrix}$$

is the rank of the sparse subsystem. The permutations P_2 are applied also to S_{s1} and x_2, and we put

$$(2.10) \qquad S'_{s1} = S_{s1} P_2, \qquad P_2^T x_2 = \begin{pmatrix} x_{21} \\ x_{22} \end{pmatrix}.$$

After steps 1.1 and 1.3 the sparse subsystem has been reduced to the form shown in Fig. 1.

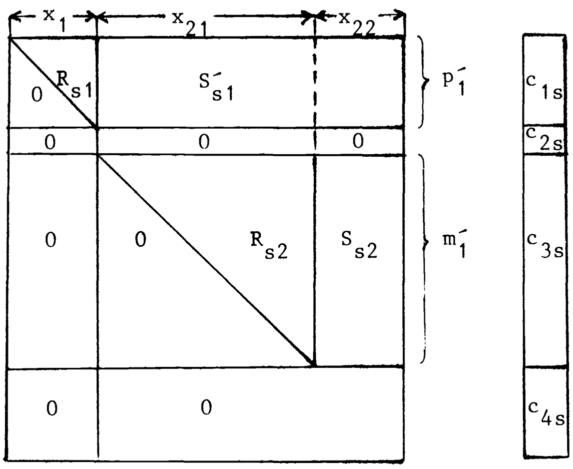


FIG. 1. *Reduced sparse subsystem after steps* 1.1, 1.3.

An efficient way to perform the transformations of the sparse subsystem in steps 1.1 and 1.3 is to process the rows of C_s and A_s sequentially one by one, using Givens rotations and elementary eliminations. For sparse systems of full rank and without constraints, such an algorithm has been developed by George and Heath [5]. For rank deficient sparse systems, column interchanges may be needed to get the upper triangular form of the reduced system (see Fig. 1). Such interchanges would destroy the fixed data structure essential for this method. However, Heath [6] has recently shown an ingenious way to avoid the column interchanges. This applies in a straightforward way also to the constrained problem considered here. The rows of C_s and A_s are processed one by one in that order. Let j be the subscript of the first nonzero component of the row being processed. If row j of the data structure is still vacant, then the row is placed into row j of the data structure. Otherwise, row j is used to annihilate the first nonzero. For this elimination a Givens rotation is used, except when row j is a constraint and the row being processed is an equation; then an elementary elimination is used.

If the above method of Heath is used, then the sparse subsystem will not be reduced to the exact form in Fig. 1. Instead, it will have a structure schematically depicted in Fig. 2. The transformed constraints will now not necessarily be located at the top of the triangle, but may be scattered. Also the $(n-r_1)$ zero rows are in general not at the bottom. However, this does not present any real complications in the updating algorithm we derive. It is sufficient to keep the indices of the three different groups of rows corresponding to constraints, equations and zero rows, respectively. For simplicity, we therefore continue to assume that the reduced system has the form in Fig. 1.

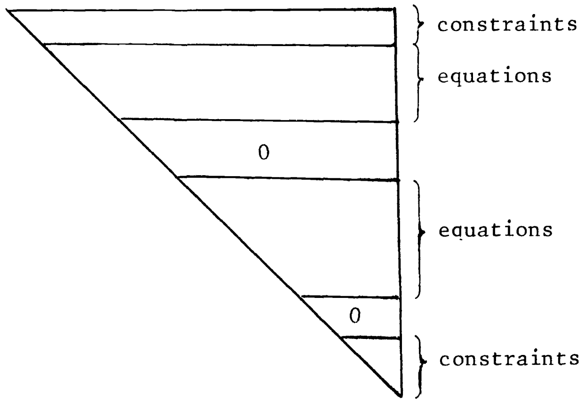


FIG. 2

3. Processing the dense subsystem. In step 1.3 the sparse equations were reduced to the form (2.9). We now introduce the residual vector

$$u_2 = c_{3s} - (R_{s2} \quad S_{s2})\begin{pmatrix} x_{21} \\ x_{22} \end{pmatrix} \}m_1' \tag{3.1}$$

of the m_1' first of these equations. The dense constraints were transformed in step 1.2 to the system (2.8), which we now write

$$(B_{d1} \quad B_{d2})\begin{pmatrix} x_{21} \\ x_{22} \end{pmatrix} = e_{d1}'. \tag{3.2}$$

Step 2.1. *Reduction of dense constraints.* Combining (3.1) and (3.2) we have

$$\begin{pmatrix} I & R_{s2} & S_{s2} \\ 0 & B_{d1} & B_{d2} \end{pmatrix}\begin{pmatrix} u_2 \\ x_{21} \\ x_{22} \end{pmatrix} = \begin{pmatrix} c_{3s} \\ e_{d1}' \end{pmatrix}.$$

We now zero the block B_{d1} by a sequence of elimination transformations. The dense constraints are then transformed into

$$(3.3)\qquad (\tilde{B}_{d0}\quad \tilde{B}_{d2})\begin{pmatrix} u_2 \\ x_{22} \end{pmatrix} = \tilde{e}_{d1}.$$

We now reduce this system further by a sequence of orthogonal transformations from the left:

(i) Reduce the block $\tilde{B}_{d0}$ to upper trapezoidal form, using column interchanges only within that block.

(ii) Continue the reduction of the block $\tilde{B}_{d2}$, transforming the lower part of that block into upper trapezoidal form, using column interchanges only within that block.

After these two steps the reduced system (3.3) has the following form (for simplicity we suppress here and in the following the above column permutations from our notations):

$$(3.4)\qquad \begin{pmatrix} T_{d1} & S_{d1} \\ 0 & T_{d2} \end{pmatrix}\begin{pmatrix} u_2 \\ x_{22} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}\}p_2'', \qquad p_2'' = \operatorname{rank}(\tilde{B}_{d0}),$$

where T_{d1} and T_{d2} are upper trapezoidal matrices.

The next step in the algorithm is the transformation of the dense equations $A_d x = b_d$, which we want to express in the variables u_2 and x_{22}.

Step 2.2. *Transformation of the dense equations.* This step is similar to the first part of step 1.2 and the first part of step 1.3. For notational convenience we describe the transformation as two separate steps. Combining (2.7) and the dense equations we have

$$\begin{pmatrix} R_{s1} & S_{s1} \\ A_{d1} & A_{d2} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} c_1' \\ b_d \end{pmatrix}.$$

By a sequence of elimination transformations, we zero block A_{d1} and partition and permute the resulting transformed block A_{d2} to get

$$(A_{d1}'\quad A_{d2}')\begin{pmatrix} x_{21} \\ x_{22} \end{pmatrix} = b_d'.$$

We adjoin this system to (3.1) to get

$$\begin{pmatrix} I & R_{s2} & S_{s2} \\ 0 & A_{d1}' & A_{d2}' \end{pmatrix}\begin{pmatrix} u_2 \\ x_{21} \\ x_{22} \end{pmatrix} = \begin{pmatrix} c_{3s} \\ b_d' \end{pmatrix},$$

and continue to eliminate the block A_{d1}'. We finally get the transformed dense equations

$$(3.5)\qquad (\tilde{A}_{d0}\quad \tilde{A}_{d2})\begin{pmatrix} u_2 \\ x_{22} \end{pmatrix} = \tilde{b}_d.$$

The form of the dense subsystems after the transformation made in steps 2.1 and 2.2 is depicted in Fig. 3.

4. The updating algorithm. We now formulate the original constrained least squares problem (2.1) in terms of the transformed system. Since orthogonal transformations were used in the last part of step 1.3, we have

$$(4.1)\qquad \|b_s - A_s x\|_2^2 = \|u_2\|_2^2 + \|c_{4s}\|_2^2,$$

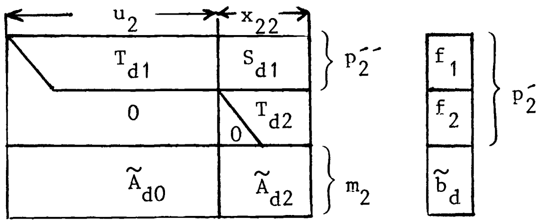


FIG. 3. *Reduced dense subsystem after steps* 2.1, 2.2.

where u_2 is defined by (3.1). The residuals of the dense equations $r_d = b_d - A_d x$ are by (3.5)

$$r_d = \tilde{b}_d - (\tilde{A}_{d0} \quad \tilde{A}_{d2})\begin{pmatrix} u_2 \\ x_{22} \end{pmatrix}. \tag{4.2}$$

Finally, to constrain the solution to minimize $\|d - Cx\|_2$, the equations (3.4) must be satisfied.

Now, assume that x_{22} is known. Then it follows that v_2 must be a solution to the following problem:

$$\text{minimize} \left\|\begin{pmatrix} r_d \\ u_2 \end{pmatrix}\right\|_2 \text{ subject to}$$

$$\begin{pmatrix} I & \tilde{A}_{d0} \\ 0 & T_{d1} \end{pmatrix}\begin{pmatrix} r_d \\ u_2 \end{pmatrix} = \begin{pmatrix} \tilde{b}_d - \tilde{A}_{d2}x_{22} \\ f_1 - S_{d1}x_{22} \end{pmatrix}\begin{matrix} \}m_2 \\ \}p_2'' \end{matrix}. \tag{4.3}$$

This problem is a minimum norm problem, where the matrix in the constraints (4.3) has full row rank. Therefore this problem has a unique solution $(r_d, u_2)^T$, which can be computed from the orthogonal factorization of a matrix with $(m_2 + p_2'')$ rows

$$\begin{pmatrix} I & \tilde{A}_{d0} \\ 0 & T_{d1} \end{pmatrix} Q_3 = (L \quad 0), \tag{4.4}$$

where Q_3 is orthogonal and L is lower triangular, nonsingular and of dimension $(m_2 + p_2'')$. Using (4.4) the solution becomes

$$\begin{pmatrix} r_d \\ u_2 \end{pmatrix} = Q_3 \begin{pmatrix} I \\ 0 \end{pmatrix} L^{-1} \begin{pmatrix} \tilde{b}_d - \tilde{A}_{d2}x_{22} \\ f_1 - S_{d1}x_{22} \end{pmatrix}, \tag{4.5}$$

and its norm as a function of x_{22} is

$$\left\|\begin{pmatrix} r_d \\ u_2 \end{pmatrix}\right\|_2 = \left\|L^{-1}\begin{pmatrix} \tilde{b}_d \\ f_1 \end{pmatrix} - L^{-1}\begin{pmatrix} \tilde{A}_{d2} \\ S_{d1} \end{pmatrix} x_{22}\right\|_2. \tag{4.6}$$

We should now choose x_{22} to minimize (4.6), and thus x_{22} can be computed as a solution to the following constrained least squares problem of dimension $(m_2 + p_2'') \times (n - r_1)$:

$$\min_{x_{22}} \|g - Fx_{22}\|_2, \qquad \text{subject to } T_{d2}x_{22} = f_2, \tag{4.7}$$

where

$$F = L^{-1}\begin{pmatrix} \tilde{A}_{d2} \\ S_{d1} \end{pmatrix}, \qquad g = L^{-1}\begin{pmatrix} \tilde{b}_d \\ f_1 \end{pmatrix}, \tag{4.8}$$

and the constraints in (4.7) come from (3.4). This small dense problem can be solved by direct elimination and orthogonal factorization.

We remark that we must expect to get x_{22}, the free variables of the sparse problem, as the solution of a constrained least squares problem. Note that in the special case when the sparse subsystem is void (we have made no assumptions about the rank of A_s and C_s!), then we have

$$L = I, \quad F = A_d, \quad g = b_d,$$

and the constraints $T_{d2}x_{22} = f_2$ are reduced to $C_d x = d_d$. Thus, in this case our proposed algorithm simplifies in the natural way.

When the problem (4.7)–(4.8) has been solved, we get r_d and u_2 from (4.5), which also can be written

$$\begin{pmatrix} r_d \\ u_2 \end{pmatrix} = Q_3^T \begin{pmatrix} s_2 \\ 0 \end{pmatrix}, \qquad s_2 = g - Fx_{22}. \tag{4.9}$$

We finally compute x_{21} and x_1 by backsubstitution in (3.1) and (2.7) respectively,

$$\begin{aligned} &R_{s2}x_{21} = c_{3s} - u_2 - S_{s2}x_{22}, \\ &R_{s1}x_1 = c'_{1s} - S_{s1}x_2, \qquad x_2 = P_2\begin{pmatrix} x_{21} \\ x_{22} \end{pmatrix}. \end{aligned} \tag{4.10}$$

The solution x_{22} of the problem (4.7) may not be unique. In that case a minimum norm solution x to (2.1) may be computed by solving another small least squares problem of dimension $n \times (n - r)$.

We remark that the given algorithm is an updating algorithm in its pure sense only when the constraints $Cx = d$ are consistent. In that case $c'_{1s} = c_{1s}$, and the solution of the sparse subproblem is given by (4.10) with $u_2 = 0$ and x_{22} arbitrary.

5. Conclusions. We have derived a completely general updating algorithm for constrained least squares problems, without any restricting assumptions about rank or consistency. Algorithms for special cases are easily derived by specialization. These algorithms may also be of interest for applications in computational statistics.

An implementation aspect, which we have not discussed here, is that in almost all steps of the algorithm the numerical rank of certain submatrices has to be estimated. As pointed out in [6], although this in general is a delicate problem, heuristic rules provide adequate answers in most contexts. We also refer to the discussion of rank determination in [3]. However, it is important to stress that an updating algorithm cannot be expected to be stable in all cases. Stability will be a problem whenever the sparse subproblem is much more ill-conditioned than the full problem. However such a situation can be detected during the solution.

We finally point out another possible stability problem, which may occur if the sparse subproblem is reduced by the method of George and Heath to the form shown in Fig. 2. This scheme avoids any column interchanges during the reduction. However, one can show that in general column interchanges are needed when equations of greatly different weights or constraints are present.

Acknowledgments. This paper was conceived and worked out when the author had the privilege of enjoying the stimulating hospitality of the Computer Science Department at Stanford University. The author gratefully acknowledges helpful discussions with Mike Heath and is also indebted to two referees whose comments greatly improved the presentation.

REFERENCES

[1] Å. BJÖRCK, *Methods for sparse linear least squares problems*, in Sparse Matrix Computations, J. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 177–199.

[2] Å. BJÖRCK AND I. S. DUFF, *A direct method for the solution of sparse linear least squares problems*, Linear Algebra Appl., 34 (1980), pp. 43–67.

[3] P. DEUFLHARD AND W. SAUTTER, *On rank deficient pseudoinverses*, Linear Algebra Appl., 29 (1980), pp. 91–111.

[4] R. W. FAREBROTHER, *A historical note on the least squares updating formulas*, Rep., Univ. Manchester, December 1978.

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

[6] M. T. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, this Journal, 3 (1982), pp. 223–237.

[7] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[8] P. Å. WEDIN, *Notes on the constrained linear least squares problem.* Report UMINF-75.79, Univ. Umeå, Sweden, 1979.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

RANK DEGENERACY*

G. W. STEWART†

Abstract. At some point in many applications a decision must be made about the rank of a matrix. The decision is frequently complicated by the fact that the matrix in question is contaminated with errors. This paper surveys some of the more commonly used methods for approximating rank, with particular attention being paid to the effects of errors.

Key words. rank, QR decomposition, singular value decomposition, inverse, cross product, matrix perturbation theory

1. Introduction. The problem treated in this paper is the following. Let X be an $n \times p$ matrix that satisfies

$$k \equiv \operatorname{rank}(X) \leqq p \leqq n.$$

Suppose further that X itself cannot be observed; rather we are given a perturbed matrix

$$\tilde{X} = X + E, \tag{1.1}$$

where E represents a matrix of errors. From the matrix $\tilde{X}$ we wish to

1. determine k

and

2. find an approximation $\hat{X}$ to X that is of rank k.

There are at least two reasons for considering this problem. First, if $k < p$ the attempt to use $\tilde{X}$ instead of X in a computational procedure may result in violently inaccurate results. For example, regression coefficients calculated from $\tilde{X}$ may be inaccurate owing to the discontinuity of the pseudo-inverse around a rank deficient matrix [13]. If, however, one can determine k, the use of a rank deficient approximation $\hat{X}$ may restore the lost accuracy. Thus one reason for solving the problem is to obtain a regularization procedure for certain ill-posed problems.

The second application occurs in areas, such as bifurcation theory, where the matrix X depends on a parameter λ. In computations for such problems, the normal state of affairs is that X is of full rank p. However, at certain critical values of λ the rank drops below p, and special computational action must be taken, action that usually involves, at least implicitly, a rank degenerate approximation $\hat{X}$ (e.g. see [10]).

It is important to realize that the problem as stated may be unsolvable, even when a fair amount is known about the matrix E. For example, suppose that the elements of E are known to be bounded by 10^{-2} and consider the following two decompositions of the form (1.1):

$$\begin{bmatrix} 1.010 & -0.005 \\ 0.005 & 0.005 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0.010 & -0.005 \\ 0.005 & 0.005 \end{bmatrix},$$

$$\begin{bmatrix} 1.010 & -0.005 \\ 0.005 & 0.005 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0.01 \end{bmatrix} + \begin{bmatrix} 0.010 & -0.005 \\ 0.005 & -0.005 \end{bmatrix}.$$

In both cases the matrix $\tilde{X}$ is the same and the elements of E are bounded by 10^{-2}.

* Received by the editors June 4, 1982, and in revised form April 6, 1983.

† Computer Science Department, University of Maryland, College Park, Maryland 20742.

However, in the first case $k=1$, whereas in the second $k=2$. On the other hand if

$$\tilde{X}=\begin{bmatrix}1.010 & 0.005\\ 0.005 & 0.100\end{bmatrix},$$

then under the assumption that the elements of E are bounded by 10^{-2} we can say definitely that $k=2$.

There are two conclusions to be drawn from these examples. First, we must know something about E to say anything at all. Second, unless we have very precise information about E, we can only obtain lower bounds on k, usually by showing that $\tilde{X}$ could have come from a matrix X whose rank is equal to the lower bound. *The decision to accept this lower bound as a "numerical rank" must be made by considering its consequences for the specific problem being solved.* The failure to appreciate this simple fact has, in the author's opinion, resulted in many misguided attempts to determine rank automatically, perhaps the most notorious examples being pseudo-inverse programs with built-in tolerances that are invisible to the user.

Because of the difficulties introduced by the matrix E, the next four sections of this paper will be devoted to describing how to compute indicators of possible rank deficiency. Throughout these sections, we shall use the terms like "sufficiently small" without specifying precisely what we mean. Only in the last section will an attempt be made to say how small is small.

Throughout this paper we shall let $\|\cdot\|_2$ denote the ordinary Euclidean vector norm defined by

$$\|x\|_2^2=x^Tx$$

or the spectral matrix norm defined by

$$\|X\|_2=\max_{\|x\|_2=1}\|Xx\|_2.$$

The symbol $\|\cdot\|_F$ will denote the Frobenius matrix norm defined by

$$\|X\|_F^2=\sum x_{ij}^2=\text{trace}\,(X^TX).$$

For more on matrix and vector norms see [12].

2. The singular value decomposition. Perhaps the most widely recommended tool for detecting rank degeneracies is the singular value decomposition. It can be shown (e.g. see [12]) that there are orthogonal matrices U and V such that[1]

$$U^TXV=\begin{bmatrix}\Psi\\ 0\end{bmatrix}, \tag{2.1a}$$

where

$$\Psi=\text{diag}\,(\psi_1,\psi_2,\cdots,\psi_p) \tag{2.1b}$$

with

$$\psi_1\geqq\psi_2\geqq\cdots\geqq\psi_p\geqq 0. \tag{2.1c}$$

It is sometimes more convenient to write the decomposition in the factored form

$$X=U_X\Psi V^T, \tag{2.2}$$

[1] Most numerical analysts would write Σ where we have written Ψ. However, this usage is impossible for statisticians, to whom Σ means a variance. Since Ψ is little used in either camp, we have adopted it here. As a mnemonic take ψ to stand for "psingular" value.

where U_X comes from the partition

$$U = (U_X \quad U_P) \tag{2.3}$$

with U_X having p columns.

The singular value decomposition is closely related to the spectral decomposition of the *cross-product matrix*

$$A = X^T X. \tag{2.4}$$

Specifically, from (2.2) and the fact that $U_X^T U_X = I$ it follows that

$$A = V \Psi^2 V^T. \tag{2.5}$$

Since A is symmetric and V is orthogonal, the squares of the singular values of X are the eigenvalues of A. The eigenvectors are the corresponding columns of V.

From (2.1) and (2.2) it follows that $\operatorname{rank}(X) = \operatorname{rank}(\Psi)$. Consequently if ψ_{k+1}, $\psi_{k+2}, \cdots, \psi_p$ are sufficiently small and ψ_k is sufficiently large, it is natural to accept a rank k approximation to X. The best such approximation may be obtained as follows. Let

$$\hat{\Psi} = \operatorname{diag}(\psi_1, \psi_2, \cdots, \psi_k, 0, \cdots, 0),$$

and in analogy with (2.2) let

$$\hat{X} = U_X \Psi \hat{V}^T. \tag{2.6}$$

Then a classical result of Eckart and Young [8] states that

$$\psi_{k+1}^2 + \cdots + \psi_p^2 = \|X - \hat{X}\|_F^2 = \min_{\operatorname{rank}(\bar{X}) \leq k} \|X - \bar{X}\|_F^2.$$

Mirsky [9] has proved the corresponding result for the spectral norm:

$$\psi_{k+1} = \|X - \hat{X}\|_2 = \min_{\operatorname{rank}(\bar{X}) \leq k} \|X - \bar{X}\|.$$

In applications one must compute the singular value decomposition of $\tilde{X}$ not X. Even if X is exactly rank degenerate, $\tilde{X}$ will not be, and the sizes of the singular values of $\tilde{X}$ corresponding to the zero singular values of X will depend on the properties of E (e.g., none of them can be larger than $\|E\|_2$). Thus something must be known about E before we can decide which singular values of $\tilde{X}$ to ignore.

Once a decision has been made about the rank of X, that is, once one has decided on a value of k, one can work with the approximation $\hat{X}$ in (2.6). In practice it is important to remember that $\hat{X}$ will be the best approximation to $\tilde{X}$, from which it has been computed, not to X, which is unobserved. Nonetheless, this approximation may be good enough in many cases.

In most problems it will be unnecessary to compute $\hat{X}$ explicitly; rather it can be manipulated in the factored form (2.6), with considerable savings in computations.

3. The QR decomposition. A second very effective tool for detecting rank degeneracy is the QR decomposition. Specifically, it can be shown [12] that given any permutation matrix J, there is an orthogonal matrix Q such that

$$Q^T X J = \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where R is an upper triangular matrix with nonnegative diagonal elements. Moreover,

J may be chosen by the method of *column pivoting* [3] so that

$$r_{jj}^2 \geqq \sum_{i+1}^{k} r_{ik}^2 \qquad (k = j, j+1, \cdots, p). \tag{3.1}$$

As with the singular decomposition, the QR decomposition can be written in a factored form

$$XJ = Q_X R, \tag{3.2}$$

where Q_X is taken from the partition

$$Q = (Q_X \quad Q_P)$$

with Q_X having the same dimensions as X.

The QR decomposition can also be related to the cross-product matrix A in (2.4). Specifically, from (3.2), it follows that

$$J^T A J = R^T R. \tag{3.3}$$

Since R is upper triangular, $R^T R$ is the *Cholesky factorization* [12] of A with its rows and columns symmetrically permuted according to J.

The application of the QR decomposition to the problem of detecting rank degeneracy goes as follows. Let R be partitioned in the form

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix},$$

where R_{11} is $k \times k$. If $r_{k+1,k+1}$ is negligible, then (3.1) assures us that each column of R_{22} also has negligible norm; indeed

$$\rho \equiv \|R_{22}\|_F \leqq (p-k)^{1/2} r_{k+1,k+1}. \tag{3.4}$$

Moreover, if $r_{kk} \neq 0$ and we set

$$\hat{R} = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}, \tag{3.5a}$$

then

$$\hat{X} = Q_X \hat{R} J^T \tag{3.5b}$$

is an approximation to X that has the following properties [15]:

1. $\hat{X}$ is of rank k.
2. $\|X - \hat{X}\|_F = \|R_{22}\|_F$.
3. XJ and $\hat{X}J$ differ only in their last $p-k$ columns.
4. If $\bar{X}$ is any matrix satisfying 1 and 3, then $\|X - \bar{X}\|_F \geqq \|R_{22}\|_F$.

According to the fourth property, which is an analogue of the Eckart–Young–Mirsky result, the rank degenerate approximation (3.5) is the best that can be obtained by altering only the last $p-k$ columns of X. When $k = p-1$, the number ρ in (3.4) has a particularly nice interpretation: it is the norm of the smallest perturbation of the last column of XJ that will make X exactly degenerate. By computing $\rho = r_{pp}$ for a sequence of permutations J that moves each column of X to the last, one obtains a set of numbers

$$\{\rho_1, \rho_2, \cdots, \rho_p\}, \tag{3.6}$$

corresponding to the columns of X, which tell how much each column must be altered to make X degenerate. These numbers can be efficiently computed from the upper triangular factor of the QR factorization of X by means of the LINPACK subroutine SCHEX [3].

It is unfortunate that there seems to be no simple, sharp relation between the number ρ in (3.4) for the QR decomposition with column pivoting and the last $p-k$ singular values of $\tilde{X}$. When $k=p-1$, there is empirical evidence that will be of the same order of magnitude as ψ_p [14]. Moreover, it can be shown that

$$\psi_p \leqq \min \rho_i \leqq p\psi_p, \tag{3.7}$$

where the ρ_i are the numbers appearing in (3.6). The general folklore says that the QR decomposition with column pivoting is about as good as the singular value decomposition.

The practical considerations that arise in using the QR decomposition are similar to the ones discussed in the section on the singular value decomposition. There is, however, one additional point. It frequently happens that once a value of k for which ρ is negligible has been determined, the problem can be recast entirely in terms of the first k columns of XJ, with a potentially large saving in work.

4. The inverse cross-product matrix. We have seen [cf. (2.5) and (3.3)] that the singular value decomposition and the QR decomposition are intimately related to the cross-product matrix $A=X^TX$. Since many procedures for solving problems involving X form A^{-1} explicitly (e.g. the solution of least squares problems by means of inverting the normal equations [11]), it is natural to ask what can be learned about rank degeneracy from A^{-1}.

The principal result is that

$$a_{jj}^{(-1)} = \rho_j^{-2}, \tag{4.1}$$

where $a_{jj}^{(-1)}$ denotes the jth diagonal of A^{-1} and ρ_j is the jth member of (3.6). To see this, suppose that J has been chosen so that the jth column of X is the last column of XJ. If R is partitioned in the form

$$R=\begin{bmatrix} R_* & r \\ 0 & \rho_j \end{bmatrix},$$

then from (3.3)

$$(J^TAJ)^{-1}=\begin{bmatrix} R_* & r \\ 0 & \rho_j \end{bmatrix}^{-1}\begin{bmatrix} R_* & r \\ 0 & \rho_j \end{bmatrix}^{-T}.$$

From this it is easily seen that the (p,p)-element of $(J^TAJ)^{-1}$, which is just $a_{jj}^{(-1)}$, is equal to ρ_j^{-2}.

In view of (3.7) and (4.1) it is possible to detect rank degeneracy by examining the diagonal elements of A^{-1}. However, one cannot determine a numerical rank without further calculation. One possibility is to determine an index j for which ρ_j is minimal and then use a Gauss–Jordan SWEEP operator [1], [6], to compute the inverse cross-product matrix for a matrix corresponding to X with its jth column deleted. Iterating this process leads to a sort of reverse pivoting algorithm, whose properties are not well understood at this time. (For anyone who wishes to pursue this matter, we note that this algorithm is essentially equivalent to calculating the Cholesky decomposition of A^{-1} with pivoting. We also note that there is an equivalent, but more stable algorithm that works with R instead of A^{-1}.)

5. Computational properties. There are two aspects to the algorithmic realization of a general procedure: the amount of work required and the effects of rounding error. We shall consider each in turn.

Most algorithms for computing the singular value decomposition are based on a preliminary reduction to bidiagonal form followed by an iteration for the singular values [3], [5]. Although the amount of work required is $O(np^2)$, the order constant is rather large, so that, in the author's opinion, the singular value decomposition should not be calculated explicitly unless there is a specific need for it.

In fact, the QR decomposition can often be used in place of the singular value decomposition. There are essentially three methods for computing a QR decomposition: the Gram–Schmidt algorithm with reorthogonalization [7], the Golub–Householder method [4], and a method based on plane rotations [3], [4]. The first two require that X be maintained in high-speed memory. The last permits the formation of R by bringing in X a row at a time; however, column pivoting is not possible, at least directly, and the storage of the rotations requires as much memory as the storage of X. All the methods require $O(np^2)$ work, but the constant is much smaller than the one for the singular value decomposition.

An important composite algorithm for the singular value decomposition can considerably reduce the amount of work required when $n \gg p$. Specifically, it follows from (2.5) and (3.3) (with $J = I$) that the singular value decomposition of R is given by

$$W^T R V = \Psi,$$

where V and Ψ are as in (2.1). It further follows from (3.2) that

$$X = (Q_X W)\Psi V^T$$

is the singular value factorization of X. This suggests that to get the singular value factorization of X one first compute the QR factorization of X and then the singular value decomposition of the small $p \times p$ matrix R. A program implementing this approach is given in [2].

A similar approach can be used to compute the QR factorization *with column pivoting* of X when n is so large that X must be brought into memory by rows (or groups of rows). Namely, the R-factor of the QR factorization without column pivoting is first computed, say by plane rotations. Then the QR decomposition of R is computed with pivoting; i.e., $W^T R J = R'$. It then follows that

$$XJ = Q_X W R'$$

is the QR factorization, with pivoting, of X.

The computation of the cross-product matrix A requires less work than the computation of either the QR or the singular value decompositions. Moreover, it can be computed in the form

$$A = \sum_{i=1}^{n} x_i x_i^T, \tag{5.1}$$

which allows the rows x_i^T of X to be brought into main storage one at a time. Great savings can be effected when X is sparse, since if x_{ij} is zero, the row $x_{ij} x_i^T$ in the sum (5.1) need not be accumulated in A.

Once A has been computed, its spectral decomposition may be computed to get the singular values Ψ and the singular vectors V [cf. (2.5)]. Alternatively, the Cholesky factorization of A may be computed to get the R-factor of X (cf. (3.3)). Finally, A

may be inverted. Of the three alternatives, the last is the most common; however, the author has a predilection for the first or second for reasons that will be given shortly.

In discussing the effects of rounding error on the decompositions, we assume that the calculations are carried out in floating-point arithmetic and that no overflows or destructive underflows occur. Using standard techniques from rounding-error analysis [17], it can be shown that the *computed* QR decomposition of X satisfies

$$Q^T(X+F)J=\begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{5.2a}$$

where

$$\|F\|_F \leqq \phi(n,p)\varepsilon_M. \tag{5.2b}$$

In (5.2b) the number ε_M is the rounding unit for the arithmetic in question; e.g., if thirteen decimal digits are carried in the calculation, then ε_M will be about 10^{-13}. The function ϕ is a slowly growing function of n and p. A similar result holds for the singular value decomposition.

The kind of result embodied in (5.2) has two important implications. First, it may be possible to choose ε_M so that F is much smaller than the error matrix E. If this is done, then the entire effects of rounding error can be regarded as coming from an insignificant perturbation of E. Since any reasonable procedure must be insensitive to minor changes in E, about which not much is known, the effects of rounding error can be ignored.

The second implication is that when X is known exactly, one can tell a true rank degeneracy from a spurious one by increasing the precision. For example, suppose that the singular values of X are computed at precision ε_M and ψ_p is of order ε_M. Then there is some question as to whether ψ_p is nonzero or if the value observed is due to rounding error. If computation is reperformed in double precision ($\varepsilon'_M \approx \varepsilon^2_M$) and the resulting singular value ψ'_p is of order ε_M, then X cannot have been degenerate. On the other hand, if ψ'_p is of order ε'_M, then there is strong reason to believe that the true singular value is zero and that the computed nonzero values are due to rounding error. It is very important to remember that in applying this technique the elements of X must be computed to the same or greater precision as that in which the singular values are computed; otherwise errors made in computing X will comprise the larger part of the errors in the singular values.

An example will make this point clear. Consider the matrix

$$X=\begin{bmatrix} 3.142 & 6.285 \\ 2.718 & 5.436 \end{bmatrix}, \tag{5.3}$$

which is assumed to be known exactly. If ρ from the QR decomposition of X is computed in four decimal digit, floating-point arithmetic, the computed value is $\rho=10^{-3}$. This is near enough $\varepsilon_M \approx 10^{-4}$ so that we cannot tell whether it is zero or not. If the calculations are repeated with $\varepsilon'_M \approx 10^{-8}$, the results are $\rho'=6.542 \cdot 10^{-4}$. This is nowhere nearly as small as ε'_M and hence X is of full rank.

On the other hand, let X in (5.3) be regarded as a four-digit approximation to the matrix

$$\begin{bmatrix} \pi & 2\cdot\pi \\ e & 2\cdot e \end{bmatrix}. \tag{5.4}$$

When the procedure is applied in four digit arithmetic, the result is the same as before.

However, when the procedure is repeated in eight digit arithmetic, we must work with the matrix

$$\begin{bmatrix} 3.1415927 & 6.2831853 \\ 2.7182818 & 5.4365637 \end{bmatrix},$$

which is an eight digit approximation to (5.4). The result is $\rho' = 10^{-8}$, which is convincing, although not rigorous evidence that the rank of the matrix (5.4) is one—not two.

Rounding error has more serious effects on the cross-product matrix than on the singular value decomposition or the QR decomposition. To see why this should be so, assume that all the elements of X are roughly the same size, and consider the effects on the singular values of rounding the elements of X with rounding unit ε_M. The resulting matrix $X' = X + F$ will have an error matrix F with $\|F\|_2 \approx \psi_1 \varepsilon_M$, and this will introduce a perturbation of the same size in ψ_p [12]. Thus as long as

$$(5.5) \qquad \frac{\psi_p}{\psi_1} > \varepsilon_M,$$

rounding errors will leave some accuracy in ψ_p.

Now suppose that $A = X^T X$ is rounded at the same precision ε_M, giving $A' = A + G$, where $\|G\|_2 \approx \psi_1^2 \varepsilon_M$ (since $\|A\|_2 = \psi_1^2$). The corresponding perturbation in the smallest eigenvalue ψ_p^2 of A will be of order $\|G\|_2$. Hence to retain any accuracy in ψ_p, as computed from A', we must have

$$(5.6) \qquad \left|\frac{\psi_p}{\psi_1}\right|^2 > \varepsilon_M.$$

A comparison of (5.5) and (5.6) shows that the second condition will be violated before the first. For example, if $\varepsilon_M = 10^{-4}$, then the condition (5.5) requires that $\psi_p/\psi_1 > 10^{-4}$, whereas condition (5.6) requires that $\psi_p/\psi_1 > 10^{-2}$. The general conclusion to be derived from this is that it requires twice the precision to accommodate the same range of singular values when one works with A instead of X.

The matrix X of (5.3) furnishes a nice example. Its smallest singular value is about $3 \cdot 10^{-4}$. On the other hand, the smallest eigenvalue of the matrix

$$(5.7) \qquad \bar{A} = \begin{bmatrix} 17.26 & 34.52 \\ 34.52 & 69.05 \end{bmatrix},$$

which is $X^T X$ rounded to four digits, is $2 \cdot 10^{-3}$. This gives the unacceptably large value of $4.5 \cdot 10^{-2}$ as an approximation to ψ_2.

If passing from X to A can loose some of the information about X, passing from A to A^{-1} can loose all of it. For example, the inverse, rounded to four digits, of the matrix $\bar{A}$ in (5.7) is

$$B = \begin{bmatrix} 400.1 & -200.0 \\ -200.0 & 100.0 \end{bmatrix}.$$

The exact inverse of B is

$$B^{-1} = \begin{bmatrix} 10 & 20 \\ 20 & 40.01 \end{bmatrix},$$

which bears only a passing resemblance to $\bar{A}$, which it is supposed to approximate. Phenomena like this will occur whenever (5.6) comes near to being violated. In general, one should use the inverse cross-product matrix only if one is prepared to compute in high precision, and even then only with safeguards.

6. How small is small. In this section we shall treat the problem of determining when a singular value is negligible. The principal problem is that we must work with the singular values $\tilde{\psi}_1, \tilde{\psi}_2, \cdots, \tilde{\psi}_p$ of $\tilde{X}$, not those of X. In particular, we should like to know what values of $\tilde{\psi}_p$ are consistent with the hypothesis that $\psi_p = 0$.

The basic result that will be used is a characterization of $\tilde{\psi}_p$ by means of a perturbation expansion. Suppose that ψ_{p-1} and ψ_p are well separated relative to E, say $\psi_{p-1} - \psi_p > 5\|E\|_2$. Then [16]

$$\tilde{\psi}_p^2 = (\psi_p + u_p^T E v_p)^2 + \|U_P^T E v_p\|_2^2 + O(\psi_p \|E\|_2^2), \tag{6.1}$$

where U_P is the matrix in the partition (2.3).

In order to apply this result, something must be known about E. We shall examine a simple but revealing model in which the elements of E are assumed to be independent random variables with mean zero and standard deviation σ. It is also assumed that a rough estimate of the size of σ is available.

The expected value of the second term in (6.1) is

$$E(\|U_P^T E v_p\|_2^2) = (n-p)\sigma^2.$$

Hence if ψ_p^2 is significantly greater than $(n-p)\sigma^2$, then

$$\tilde{\psi}_p \approx \psi_p + u_p^T E v_p.$$

The term $u_p^T E v_p$ has standard deviation σ, which is a fortiori small compared to ψ_p. Thus in this case, ψ_p and $\tilde{\psi}_p$ are not likely to differ by much, and we are justified in concluding that $\psi_p \neq 0$ if we observe a value of $\tilde{\psi}_p^2$ that is significantly greater than $(n-p)\sigma^2$.

On the other hand, if $\psi_p = 0$, then ignoring the $O(\psi_p\|E\|_2^2)$ term in (6.1), we have

$$E(\tilde{\psi}_p^2) = (n-p+1)\sigma^2.$$

Thus values of $\tilde{\psi}_p^2$ near $n\sigma^2$ are not inconsistent with ψ_p being zero. However, any decision to treat ψ_p as if it were zero must be made in the context of the problem being solved—as we were at pains to point out in the introduction.

The assumption of a common standard deviation in the above analysis has important implications for the scaling of $\tilde{X}$. Specifically, *if the elements of E are assumed to be independent, then the rows and columns of X must be scaled so that all the elements of E are roughly equal.* The failure to observe this dictum can result in spurious indications of degeneracy, a phenomenon sometimes referred to as artificial ill conditioning.

To see how artificial ill conditioning arises, consider the behavior of the matrix

$$X_t = (X_* \, tx)$$

as t approaches zero. Let

$$R_t = \begin{bmatrix} R_* & tr \\ 0 & t\rho \end{bmatrix}$$

be the R-factor of X_t. Then it can be shown that the smallest singular value $\psi_p^{(t)}$ of X_t is asymptotic to $t\rho$:

$$\psi_p^{(t)} \sim t\rho. \tag{6.2}$$

Moreover the right singular vector $v_p^{(t)}$ satisfies

$$v_p^t \sim \begin{bmatrix} -tR_*^{-1}r \\ 1 \end{bmatrix}, \tag{6.3}$$

and consequently the left singular vector $u_p^{(t)} = X_t v_p^{(t)}/\|X_t v_p^{(t)}\|_2$ satisfies

$$\lim_{t\to 0} u_p^{(t)} = \frac{x - X_* R_*^{-1} r}{\|x - X_* R_*^{-1} r\|_2} \equiv u_p^{(0)}.$$

Now the error matrix E_t associated with X_t inherits the scaling of X; that is, E_t may be written in the form

$$E_t = (E_* \; te). \tag{6.4}$$

Assuming $E_* \neq 0$, we have

$$\lim_{t\to 0} \|E_t\|_2 = \|E_*\|_2 > 0. \tag{6.5}$$

It follows from (6.2) and (6.5) that by taking t small enough we can make $\psi_p^{(t)}$ arbitrarily smaller than $\|E_t\|_2$. The same kind of analysis shows that $\tilde{\psi}_p^{(t)}$ will also be small compared with $\|E_t\|_2$, from which an inexperienced person might conclude that X_t is degenerate.

The fallacy in so concluding may be exposed by considering the perturbation expansion (6.1). From (6.3) and (6.4) it follows that

$$E_t v_p^{(t)} = t(e - E_* R_*^{-1} r) \equiv t\bar{e}.$$

Consequently,

$$\psi_p^{(t)2} \sim (\psi_p^{(t)} + u_p^{(t)T} E_t v_p^{(t)})^2 + \|U^T E_t v_p^{(t)}\|_2^2 = t^2[(\rho + u_p^{(0)T}\bar{e})^2 + \|U_P^T \bar{e}\|_2^2].$$

Now the components of $\bar{e}$ are independent with variance $\sigma^2(1 + \|R_*^{-1} r\|_2^2)$. Hence if

$$\rho \gg [(n-p)(1 + \|R_*^{-1} r\|_2^2)]^{1/2},$$

then the perturbation introduced in $\psi_p^{(t)}$ by E_t is small relative to $\psi_p^{(t)}$, and it is not reasonable to conclude from a small value of $\tilde{\psi}_p^{(t)}$ that $\psi_p^{(t)}$ could have been zero.

We conclude this section with a caveat. Although the analysis given here is quite successful as far as it goes, there are many problems to which it does not apply. The sticky point is the assumption of the independence of the components of E, which is patently false in many applications. Consider, for example, a polynomial regression problem in which a polynomial of degree $p-1$ is to be fit to ordinates $y_1, y_2, \cdots, y_n$ observed at distinct abscissas $t_1, t_2, \cdots, t_n$. The resulting regression matrix X has rows of the form

$$x_i^T = (1, t_i, t_i^2, \cdots, t_i^{p-1}) \qquad (i = 1, 2, \cdots, n).$$

Now suppose the t_i are determined with errors e_i. The resulting regression matrix will have rows of the form

$$\tilde{x}_i^T = [1, \; t_i + e_i, \; (t_i + e_i)^2, \; \cdots, \; (t_i + e_i)^{p-1}].$$

Clearly the errors in the ith row, depending as they do solely on e_i, are correlated, and the analysis given above does not apply. This can be verified independently from the fact that, no matter how small the singular values of X are, the matrix $\tilde{X}$ cannot become degenerate unless the errors e_i are large enough to allow the t_i to cluster into a set of fewer than p distinct points. The problem of how to make decisions about rank in the presence of highly structured errors is an open question that needs further research.

REFERENCES

[1] F. L. BAUER AND C. REINSCH, *Inversion of positive definite matrices by the Gauss–Jordan method*, in Handbook for Automatic Computation II. Linear Algebra, J. H. Wilkinson and C. Reinsch, eds., Springer, New York, 1971, pp. 45–49.

[2] TONY CHAN, *An improved algorithm for computing the singular value decomposition*, ACM Trans. Math. Software, 8 (1982), pp. 72–83.

[3] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER AND G. W. STEWART, *The* LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[4] G. H. GOLUB, *Numerical methods for solving least squares problems*, Numer. Math., 7 (1965), pp. 206–216.

[5] G. H. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solution*, Numer. Math., 14 (1970), pp. 403–420.

[6] J. H. GOODNIGHT, *A tutorial on the* SWEEP *operator*, Amer. Statistician, 33 (1979), pp. 149–158.

[7] J. W. DANIEL, W. B. GRAGG, L. KAUFMAN AND G. W. STEWART, *Reorthogonalization and stable algorithms for updating the Gram–Schmidt* QR *factorization*, Math. Comp., 30 (1976), pp. 772–795.

[8] C. ECKART AND G. YOUNG, *The approximation of one matrix by another of lower rank*, Psychometrica, 1 (1936), pp. 211–218.

[9] L. MIRSKY, *Symmetric gauge functions and unitarily invariant norms*, Quart. J. Math., 11 (1960), pp. 50–59.

[10] W. C. RHEINBOLDT, *Numerical methods for a class of finite dimensional bifurcation problems*, SIAM J. Numer. Anal., 15 (1978), pp. 1–11.

[11] G. A. F. SEBER, *Linear Regression Analysis*, John Wiley, New York, 1977.

[12] G. W. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1974.

[13] ———, *On the perturbation of pseudo-inverses, projections, and linear least squares problems*, SIAM Rev., 19 (1977), pp. 634–662.

[14] ———, *The efficient generation of random orthogonal matrices with an application to condition estimators*, SIAM J. Numer. Anal., 17 (1980), pp. 403–409.

[15] ———, *Assessing the effects of variable error in linear regression*, University of Maryland Computer Science Technical Report No. 818, 1979.

[16] ———, *A second order perturbation expansion for small singular values*, University of Maryland Computer Science Technical Report No. 1241, 1982.

[17] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

OPEN BOUNDARY CONDITIONS FOR FORCED WAVES IN A ROTATING FLUID*

LARS PETTER RØED† AND OLE MARTIN SMEDSTAD‡

Abstract. Previous studies on open boundary conditions in unbounded rotating fluid flows have concentrated on how to accurately simulate the outflow of *free* waves through an open boundary. In most limited area integrations of rotating fluids, however, the generated waves are *forced* rather than *free*, in which case the Sommerfeld radiation condition, applied in previous studies, is not valid. A new open boundary condition applicable to rotating stratified fluid flows is suggested below, which allows the fluid to be forced everywhere, including the open boundaries. To prove the applicability of the suggested open boundary conditions, some numerical experiments with a fluid contained within an infinitely long channel are considered. The new open boundary conditions are applied at two cross sections along the channel.

Key words. open boundary conditions, rotating fluid, channel flow

1. Introduction.

In numerical simulations of geophysical fluid motion the conditions used at "open" boundaries may play a crucial role. An open boundary differs from a physical boundary, like a rigid wall, etc., in that it does not confine the fluid to be contained within the imposed boundaries. Thus, the waves propagate and the fluid is free to advect through the boundary. This paper considers in what way one can accurately simulate the outflow of *forced* dispersive long waves from a domain with open boundaries.

Because numerical simulations of geophysical fluid motion by necessity have to be confined to a finite domain, open boundary conditions frequently occur. One of the first realistic studies was made by Charney et al. (1950), integrating the barotropic vorticity equation for a limited area.

The problem considered here is of hyperbolic nature, which makes it natural to apply the Sommerfeld radiation condition at the open boundaries, viz.,

$$\frac{\partial \phi}{\partial t} + c_\phi \frac{\partial \phi}{\partial s} = 0. \tag{1.1}$$

Here c_ϕ is the local phase speed associated with the dependent variable ϕ. The derivatives are with respect to time, t, and the coordinate perpendicular to the open boundary, s, respectively. This type of boundary condition has been widely used in analytical studies as well as numerical integrations of hyperbolic flows. See for instance Elvius and Sundstrøm (1973) and also the thorough discussion by Engquist and Majda (1977), (1979). A practical and realistic implementation of (1.1) is given by Orlanski (1976) (henceforth IO). He used the Sommerfeld condition on a prescribed open boundary. To demonstrate the applicability of the implemented condition he integrated two models: the collapsing bubble and the spatially growing Kelvin–Helmholtz instability. Camerlengo and O'Brien (1980) (henceforth CO) used a modified version of IO's boundary condition to simulate the outflow of Rossby and Kelvin waves from an equatorial domain with one open boundary. The idea of IO and CO is to evaluate the local phase speed, for each dependent variable, close to the boundary by means of (1.1). If the local phase speed is positive, a boundary value is extrapolated from the

* Received by the editors May 17, 1982, and in revised form February 4, 1983. This research was supported by the Norwegian Office of Naval Research.

† Institute of Geophysics, University of Oslo, P.O. Box 1022, Blindern, N-Oslo 3, Norway. Present affiliation: Det norske Veritas, Research Division, P.O. Box 300, N-1322 Høvik, Oslo, Norway.

‡ Institute of Geophysics, University of Oslo, P.O. Box 1022, Blindern, N-Oslo 3, Norway.

interior values close to the boundary. If the local phase speed is negative, the boundary value is either specified or unchanged.

For a review on open boundary conditions see Haltiner and Williams (1980), Reid et al. (1975) and CO.

Open boundary conditions for oceanic flows have been used by Hurlburt (1974) and also by Busalacchi and O'Brien (1981) who used the CO condition for their ten-year simulation of the equatorial Pacific forced by observed winds. However, they had to use explicit smoothing at the boundary in order to prevent noise from being generated at the boundary (A. J. Busalacchi, private communication).

The Sommerfeld condition (1.1) is only valid for *free* waves, and as such does not work properly when external forcing is present at the open boundary. A new condition which allows forcing to be present even at the boundary itself is proposed, and some results from successful numerical experiments applying the new condition are presented.

The motivation for this kind of work mainly comes from considerations of ocean models where long dispersive and nondispersive edge waves are present. The results reported below are confined to numerical studies of the response of an ocean, confined between two parallel walls, to wind forcing. The integration is limited to a finite portion of the channel, applying open boundary conditions at two cross sections defining the limit of the integration area.

2. Definition of sample problem. To be considered is the response, to wind forcing, of a stratified fluid in an infinitely long channel of constant width. The depth of the channel will be a function of the cross channel coordinates only, so as to generate dispersive as well as nondispersive long waves. The perturbations from the rest state, where pressure and density are functions of depth only, are assumed to be small, so that a linear analysis can be applied. Also the perturbations are assumed to have horizontal scales large enough for the "hydrostatic" approximation to be valid. Thus, the motion can be separated into normal modes (see Gill and Clarke (1974)). The response of each mode is the same as for a homogeneous fluid with an appropriate "equivalent depth." It is therefore sufficient to consider the response of a barotropic model. It is important to realize that each mode has to be treated similarly; that is, the open boundary conditions must be applied to each mode separately. Some of the modes may propagate toward the open boundary and some inward, away from it. When summed, they provide the complex boundary condition applicable to the total solution.

Let (x, y, z) be rectangular coordinates such that the z-axis points upwards and the sides of the channel are parallel to the y-axis (Fig. 1). The axes are fixed in a frame that rotates uniformly about the vertical axis with angular velocity $\frac{1}{2}f$. Thus f is the Coriolis parameter. Let η be the upward displacement of the free surface from its equilibrium position and g, the gravitational acceleration. Within the framework of the above mentioned assumptions, the velocity is then independent of depth. The governing equations may therefore be integrated to yield equations for the volume flux components (U, V) corresponding to the coordinates (x, y). If the undisturbed depth of the fluid columns are represented by $H(x)$, the governing equations are the momentum equations

$$\frac{\partial U}{\partial t} = fV - gH\frac{\partial \eta}{\partial x} + \frac{\tau^x}{\rho}, \tag{2.1}$$

$$\frac{\partial V}{\partial t} = -fU - gH\frac{\partial \eta}{\partial y} + \frac{\tau^y}{\rho} \tag{2.2}$$

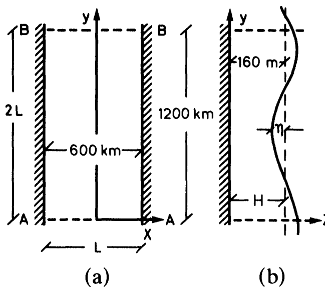


FIG. 1. *Sketch of channel with coordinate system.* (a) *Top view,* (b) *side view. Open boundary conditions are applied at two cross sections along the channel axis marked A–A and B–B in* (a).

and the continuity equation

$$\frac{\partial \eta}{\partial t} = -\frac{\partial U}{\partial x} - \frac{\partial V}{\partial y}. \tag{2.3}$$

Here (τ^x, τ^y) are the components of the stress exerted by the wind on the free surface. For the present study the bottom stress has been neglected.

These equations are the familiar linear "shallow water" equations extensively used in both analytical and numerical models, and it therefore seems proper to apply them in a study on open boundary conditions. Note that each normal mode satisfies a similar set of equations, with the equivalent depth being H. Initially the fluid is at rest with a prescribed free surface deviation, and the response of the fluid will be studied by means of a numerical integration of (2.1)–(2.3) for a specified bottom topography, H, and wind stress, (τ^x, τ^y). Open boundary conditions will be applied at two cross sections separated by a distance equal to twice the width of the channel.

Equations (2.1)–(2.3) are approximated by a finite difference scheme, where a staggered grid for the discretization in space is used. The grid corresponds to lattice C of Mesinger and Arakawa (1976, p. 47). A forward difference in time is used in the continuity equation (2.3), and a backward time differencing in (2.1) and (2.2) except for the Coriolis term in (2.1) (Sielecki (1968)). This scheme is stable even when there is no bottom friction and has no numerical dissipation, provided the *CFL* condition is satisfied. There is some numerical dispersion, but for waves which are well resolved the dispersion is small. For a detailed description of the scheme the reader is referred to Martinsen et al. (1979). The actual bottom topography and wind will be specified in each case below.

3. Open boundary conditions.

3.1. Theory. The condition (1.1) is appropriate for *free* waves. Thus it is valid only for a subset of (2.1)–(2.3) with the forcing terms excluded, i.e. the terms containing τ^x and τ^y. The terms in (2.1)–(2.3) which contains derivatives with respect to y carry information across the open boundary. Thus these are the terms which require a boundary condition to be specified at the artificial boundaries. It is therefore suggested to split the dependent variables into two modes, one mode which contains the local contribution, including the local forcing by the wind, and one mode which carries information about events generated at nonlocal positions. Formally, this may be done by defining

$$U = U_1 + U_2, \quad V = V_1 + V_2, \quad \eta = \eta_1 + \eta_2, \tag{3.1}$$

where subscripts 1 and 2 denote "local" and "global" modes, respectively. The local mode is defined so as to satisfy a local system of equations. Let the open boundary be perpendicular to the y-axis. Then the local mode is defined by

(3.2) $$\frac{\partial U_1}{\partial t} = fV_1 - gH\frac{\partial \eta_1}{\partial x} + \frac{\tau^x}{\rho},$$

(3.3) $$\frac{\partial V_1}{\partial t} = -fU_1 + \frac{\tau^y}{\rho}$$

and

(3.4) $$\frac{\partial \eta_1}{\partial t} = -\frac{\partial U_1}{\partial x}.$$

These equations do not require *any boundary conditions to be applied* at any section perpendicular to the y or channel axis. Their solutions can be computed at any section, including the open boundary section, independently.

The global parts may now be found by subtracting (3.2)–(3.4) from the original set of equations (2.1)–(2.3) to give

(3.5) $$\frac{\partial U_2}{\partial t} = fV_2 - gH\frac{\partial \eta_2}{\partial x},$$

(3.6) $$\frac{\partial V_2}{\partial t} = -fU_2 - gH\frac{\partial \eta_2}{\partial y} - \left[gH\frac{\partial \eta_1}{\partial y}\right]$$

and

(3.7) $$\frac{\partial \eta_2}{\partial t} = -\frac{\partial U_2}{\partial x} - \frac{\partial V_2}{\partial y} - \left[\frac{\partial V_1}{\partial y}\right].$$

If the forcing (τ^x, τ^y) is independent of y, then the *local* solution is independent of y from (3.2)–(3.4), and the bracketed terms in (3.5) and (3.7) are zero. In these circumstances, the set of equations (3.5)–(3.7) constitutes the *free wave* equations. Hence, the appropriate boundary condition required for the *global part* is the Sommerfeld condition (1.1).

In order to find the value of the dependent variables at the open boundaries, Sommerfeld radiation condition (1.1) is applied to the *global part*, for each time step, in a manner similar to that described by IO or CO. The local values at the boundaries are computed by means of (3.2)–(3.4). The "correct" boundary values for the dependent variables are found by simply adding the so-computed global and local values.

First, when the forcing is zero, the *local* parts are identically equal to zero and the solutions at every point, including the boundary points, are given by the global values alone. Second, when the forcing is everywhere independent of the along-channel coordinate, y, the governing equations degenerate to the local equations, and thus the global parts of the solution are zero. This is to be expected, since by definition they should carry information about events propagating towards the boundary from interior points.

The separation into local and global modes also has a slightly different interpretation. The terms containing derivatives tangential to the open boundary are retained in the definition of the local mode. Thus, that part of the wave which travels along the boundary is part of the local mode. In this sense the term "local" applies to whole cross sections or strips parallel to the open boundary rather than to a single point. Only that part of the wave which propagates across these strips is termed global. Note that this global part is also unforced except for the bracketed terms in (3.6) and (3.7). As will be described below, the global modes are *not* computed by means of a numerical integration of (3.5)–(3.7), thus avoiding the evaluation of the bracketed terms in (3.6) and (3.7).

3.2. Numerical implementation. The local fields are computed by means of a finite difference representation of (3.2)–(3.4) similar to that used to represent (2.1)–(2.3) (Martinsen et al. (1979)). Figure 2 provides a sketch of the staggered grid close

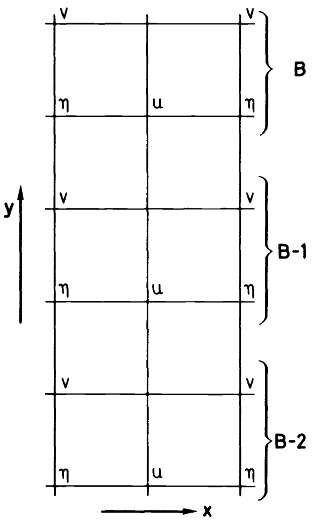


FIG. 2. *Grid stencil close to open boundaries. Grid points marked B are boundary points while grid points marked B–1 and B–2 are interior grid points adjacent to the open boundary.*

to the boundary. The local fields are only computed at the boundary point, B, and at the two adjacent points, i.e. B-1 and B-2. At the two interior points B-1 and B-2 the global fields are found simply by subtracting the local values from the total values. At the boundary the global values are computed by applying the method of IO or CO. Let ϕ denote the global dependent variable in question. Then from a finite difference representation of (1.1) consistent with the finite difference approximation of (2.1)–(2.3), it follows that the local phase speed associated with ϕ is given by the formula

(3.8)
$$c_\phi = -\frac{\Delta x}{\Delta t}\frac{\phi_{B-1}^n - \phi_{B-1}^{n-1}}{\phi_{B-1}^{n-1} - \phi_{B-2}^{n-1}},$$

where the notations are given in Fig. 3. If c_ϕ is positive, then the boundary value of the dependent variable, ϕ_B^{n+1}, is either extrapolated by repeated use of (3.8) with B-1 replaced by B and n by $n+1$ as in IO, or simply by the value at the point next to the boundary at the previous time step as in CO, viz.,

(3.9)
$$c_\phi > 0 \Rightarrow \phi_B^{n+1} = \begin{cases} \phi_{B-1}^n & \text{(CO)}, \\ \phi_B^n\left(1 - c_\phi \dfrac{\Delta t}{\Delta x}\right) + \dfrac{c_\phi \Delta t}{\Delta x}\phi_{B-1}^n & \text{(IO)}. \end{cases}$$

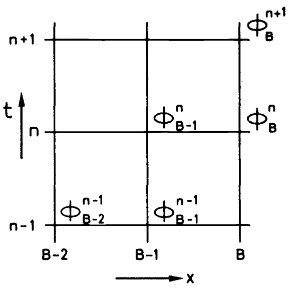


FIG. 3. *The time levels and grid close to the open boundary conveniently defining some of the notations used in the text.*

If c_ϕ is negative, no change in the boundary value is experienced, i.e.

$$c_\phi < 0 \Rightarrow \phi_B^{n+1} = \phi_B^n, \tag{3.10}$$

which is in accordance with both IO and CO. The discrimination between a positive and negative value of the local phase speed is based on the different physics involved. When c_ϕ is positive, it entails advection or propagation of interior events *toward* the boundary. When c_ϕ is negative, information is carried *away* from the boundary, in which case the dependent variable in question must either retain its value at the boundary or be specified according to a given exterior solution. This applies only to the global part of the solution.

4. Adjustment under gravity in a rotating channel. Dispersive as well as nondispersive waves can be set up in a number of different ways. In this first simulation, no forcing is applied and the adjustment towards a geostrophic equilibrium from an initial state with a single jump in the surface elevation is considered. This problem has been treated analytically by Gill (1976). It therefore conveniently serves as a test of the applicability of the chosen finite difference scheme and the implementation of the Sommerfeld condition (3.9) and (3.10). Gill showed that adjustment was accomplished by dispersive Poincaré waves and nondispersive Kelvin waves propagating outwards from the initial jump in surface elevation. The actual values given to the parameters of this simulation and those to follow are listed in Table 1. The channel has a constant

TABLE 1

Parameter	Symbol	Value assigned
Width of channel	L	600 km
Viewport of channel	$2L$	1200 km
Depth of deep channel	H	1600 m
Depth of shallow channel	H	160 km
Coriolis parameter	f	$1.32 \cdot 10^{-4}\,\mathrm{s}^{-1}$
Density of water	ρ	$1.0 \cdot 10^3\,\mathrm{kg\,m}^{-3}$
Gravitational acceleration	g	$9.8\,\mathrm{m/s}^2$
Gridsize	Δs	20 km
Time step {deep channel	Δt	100 s
Time step shallow channel}	Δt	300 s
Rossby deformation radius:		
a) for deep channel	$\frac{(gH)^{1/2}}{f}$	$950\,\mathrm{km} = 1.6L$
b) for shallow channel	$\frac{(gH)^{1/2}}{f}$	$300\,\mathrm{km} = 0.5L$

depth and is fairly shallow. The ratio of the channel width to the radius of deformation approximates two, so the channel is wide enough for rotational effects to be appreciable. Since no forcing is applied, only the global mode contributes to the solution, and hence the Sommerfeld condition can be applied directly. Only the CO condition is used, since some initial test simulations show no appreciable difference in the physical solution when the channel has a constant depth. An illuminating comparison between the methods of IO and CO will be discussed in the next section.

The solutions are represented by Figs. 4a and b, which depict the deviation of the surface elevation from its equilibrium position at times $t = 3$ and 32 hours. Initially the jump was 0.1 meters and Fig. 4a shows the front, made up of the Kelvin wave and the fastest Poincaré waves, moving toward the artificial boundaries. The front is followed by the slower moving waves. Figure 4b depicts the almost adjusted solution, where most of the waves have passed the open boundary. The striking similarity between these solutions and those of Gill (1976) is satisfying. Thus, the chosen scheme and the implementations of the open boundary conditions work satisfactorily.

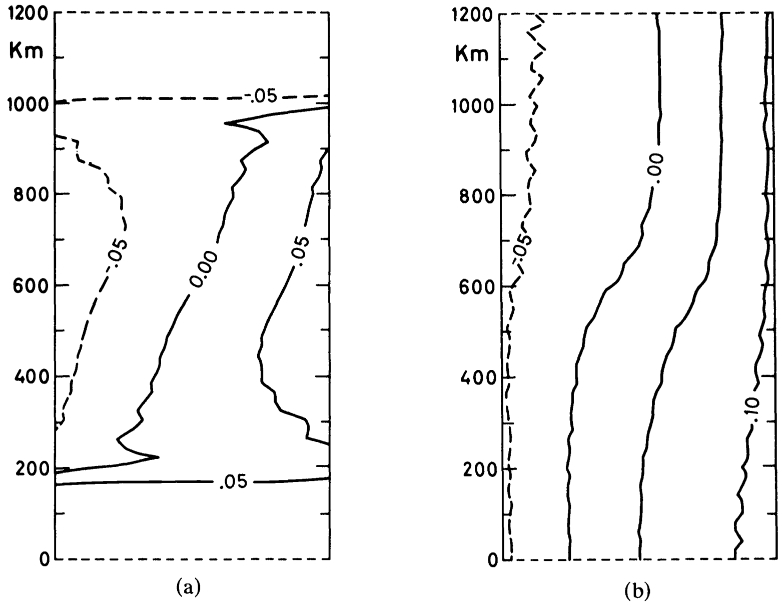


FIG. 4. *Solid (positive) and dashed (negative) curves give the deviations of the surface elevation away from its equilibrium position in meters. The open boundaries are shown by straight dashed lines whereas the parallel channel walls are solid straight lines. Initially the jump was* 0.1 m *and positioned in the middle of the viewport.* (a) *The solution after time* $t = 3$ *hours,* (b) $t = 32$ *hours. For details see text. Contour interval is* 0.05 m.

5. Wind generated waves and shelf topography. In this case the channel has a fairly shallow shelf extending about one-third of the channel width from the right-hand channel wall, followed by a deep part extending all the way to the left wall (look at Fig. 1). When the ocean is impulsively forced, this topography generates continental shelf waves, which are trapped to the shelf and are highly dispersive (Martinsen et al. (1979)). Also a nondispersive Kelvin wave mode is generated, but it does not "see" the shelf. For the deep channel the ratio of the channel width to the radius of deformation is only about 0.6, which entails a fairly narrow channel. The Kelvin wave is therefore not appreciably affected by rotation, a fact evident in the solutions below.

Because of the presence of the highly dispersive shelf waves, this setup constitutes a suitable test case for comparing different open boundary conditions for *free* waves. With this purpose in mind the lower boundary section is closed by means of a rigid wall, while the upper boundary section is kept "open." The waves are generated by applying a stress along the channel axis over the lower half portion of the channel, viz.,

$$\tau^x = 0, \qquad \tau^y = \begin{cases} 0, & y \geqq L, \\ \tau_0 \tanh \alpha(y-L), & y < L. \end{cases} \tag{5.1}$$

Here $\tau_0 = 2.0\,\mathrm{Nm}^{-2}$, $L = 600$ km and $\alpha \Delta s = 0.2$. The wind stress is turned impulsively on for 8 hours and then abruptly set to zero. For comparison reasons the model is first run with a closed channel basin extended along the channel axis. The closed channel is so long that the fastest wave and its reflection will not disturb the solutions within the region of interest during the computations.

The sequence of events is illustrated by Fig. 5. In order to adjust to the new environments, a Kelvin wave is generated when the wind stress is turned on. This Kelvin wave rapidly propagates away from the forced region. In Fig. 5a, which depicts the solution after 6 hours, the Kelvin wave has almost propagated out of the viewport region. Shelf waves have steadily been generated in the wake of the Kelvin wave and are clearly visible in Fig. 5a. As is evident from Fig. 5b, after 8 hours these shelf waves are dispersive. Hence the region they "cover" is stretched as time progresses. At 8 hours the wind forcing is turned off and a new Kelvin wave is formed which rapidly propagates through the shelf waves and out of the viewport region. The Kelvin waves may clearly be seen in Fig. 6. In Fig. 5c, after 11 hours, the shelf waves covers the whole shelf within the viewport, while at Fig. 5d after 17 hours, a substantial portion of the shelf waves have propagated out of the region of interest. Finally, in Fig. 5e displaying the solution after 32 hours, all the waves have propagated out, and most of the disturbances initiated by the impulsive wind forcing have levelled out.

Two more runs with the same forcing and geometry, but with an open boundary at $y = 2L$, are depicted in Figs. 7 and 8. For comparison reasons only the deviations of the surface elevations from the previous are shown. Figures 7a and b show these anomalies after 32 hours with the open boundary condition adapted from IO and implemented as described in § 3. As is evident from Figs. 7a and b, the IO boundary condition tends to remove fluid from the viewport basin. However, the solution compares favorably up to about 14 hours, after which fluid is steadily removed from the integration area compared to the "true" solution.

The open boundary conditions suggested by CO behaves similarly (Figs. 8a and b), but recovers and does not remove fluid appreciably. It is therefore concluded that the simpler version of the open boundary condition suggested by CO, with this particular setup, more accurately simulates the outflow of dispersive waves on an f-plane.

6. Forced waves at the open boundaries. In order to test the applicability of the new open boundary conditions suggested in § 3, the final experiment was set up in much the same way as the experiments described in the previous sections. The channel depth is constant and the lower $(y = 0)$ as well as the upper $(y = 2L)$ boundary are open. The forcing is adapted from § 5. Afterwards the fluid was allowed to adjust towards its geostrophic equilibrium. Again, in order to adjust to the new environment brought to the fore by the wind, *free* Poincaré and Kelvin waves propagate away from the forced region towards the open boundary at $y = 2L$. At the same time *forced* Poincaré and Kelvin waves propagate towards the boundary at $y = 0$. The sequence of events is illustrated by Figs. 9a–d. In the forced region, $0 \leqq y \leqq L$, the surface tilts

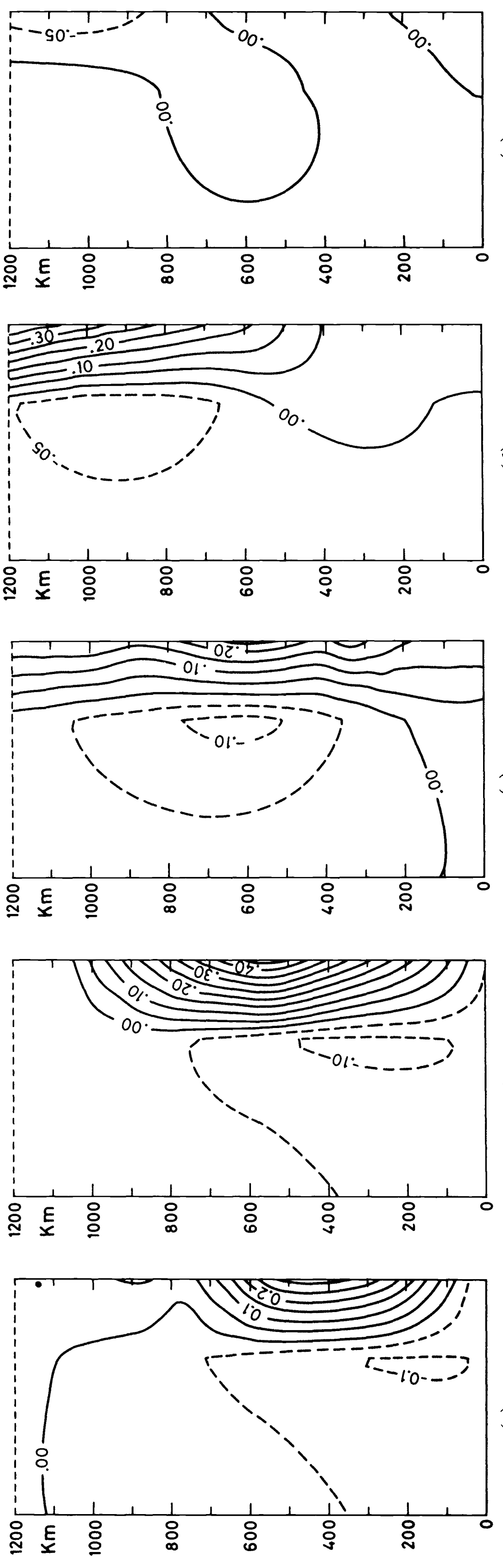


FIG. 5. *Same as Fig. 4. In this case the lower boundary is closed, indicated by a solid line replacing the dashed line of Fig. 4. The grid point (29, 57) is marked by a black spot in Fig. 5a. (a) The solution after t = 6 hours, (b) t = 8 hours, (c) 11 hours, (d) 17 hours and (e) t = 32 hours.*

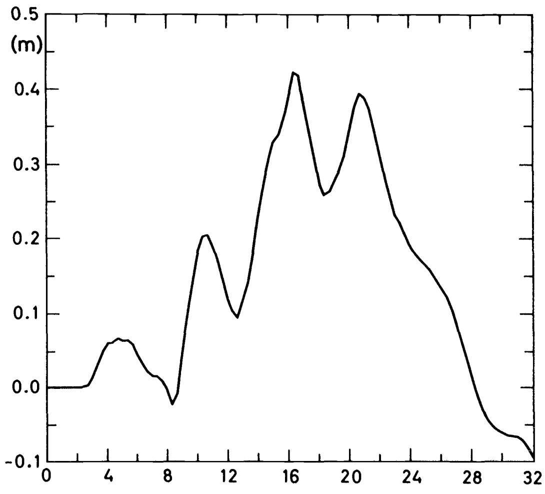


FIG. 6. *The solid curve shows the time dependence of the deviation of the surface elevation away from its equilibrium position at the grid point* (29, 57) *in meters* (*see Fig.* 5a). *Time is indicated along horizontal axis in hours.*

in order to establish a pressure gradient to balance the wind forcing. The development of this tilt is a combination of Ekman drift and the forced Kelvin and Poincaré waves. As time progresses (look at Figs. 9a–c) this gradient grows almost linearly with time. In the upper unforced region the *free* Kelvin wave is clearly depicted and carries the information of the wind in the lower half portion of the channel towards the upper open boundary. As revealed by Figs. 9a–c, even the forced waves are accurately simulated to propagate out at the lower open boundary. At time $t = 8$ hours, the wind is turned off and the adjustment is allowed to take place. After time $t = 16$ hours (Fig. 9d) a geostrophic flow has developed which is stationary except for inertial oscillations. This is to be expected. When the wind is turned off the motion is governed by the equations

$$\frac{\partial U}{\partial t} - fV = -gH\frac{\partial \eta}{\partial x}, \tag{6.1}$$

$$\frac{\partial V}{\partial t} + fU = -gH\frac{\partial \eta}{\partial y} \tag{6.2}$$

and

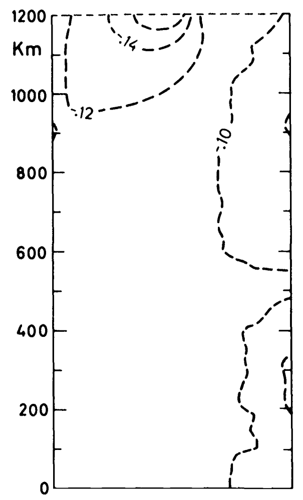
$$\frac{\partial \eta}{\partial t} = -\frac{\partial U}{\partial x} - \frac{\partial V}{\partial y}. \tag{6.3}$$

As shown by Gill (1976), the potential vorticity, as given by

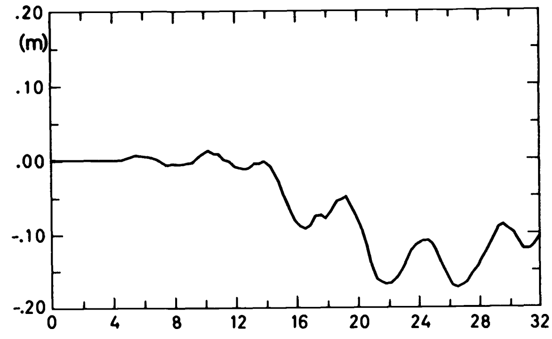
$$P = \frac{-\frac{\partial U}{\partial y} + \frac{\partial V}{\partial x} + fH}{H(\eta + H)}, \tag{6.4}$$

is conserved for each fluid column individually; i.e., each column conserves its initial potential vorticity. For large times as $t \to \infty$ it may be shown that $U \equiv 0$, and thus from (6.1) follows

$$V = -\frac{gH}{f}\frac{d\eta}{dx}. \tag{6.5}$$

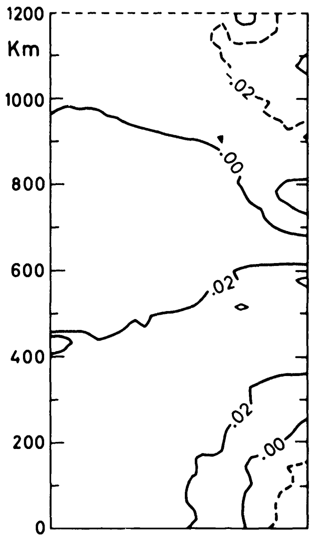


(a)

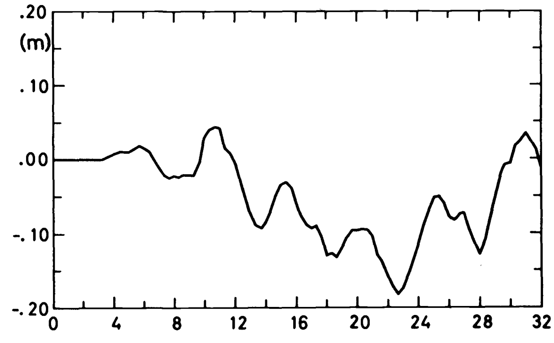


(b)

FIG. 7. (a) *Solid (positive) and dashed (negative) curves show the anomaly of the surface elevation compared to that shown by Fig.* 5e, *i.e. after time* $t = 32$ *hours. Dashed curves indicate a relative lower surface elevation. In this case, the* IO *open boundary condition has been used. Contour interval is* 0.02 m. (b) *Solid curve shows the time dependent anomaly of the surface elevation as compared to that of Fig.* 6 *for the grid point* (29, 57) (*see Fig.* 5a). *The* IO *open boundary condition has been applied.*



(a)



(b)

FIG. 8. (a) *Same as Fig.* 7a. *However, in this case the* CO *open boundary condition has been used.* (b) *Same as Fig.* 7b. *However, in this case the* CO *open boundary condition has been used.*

The apparent ambiguity in the choice of V and η is resolved by the conservation of potential vorticity (6.4) which gives a unique solution for V and η, provided the initial potential vorticity is known. Figure 9d depicts a solution to (6.5) such that the potential vorticity given to the fluid by the wind during the first 8 hours of the simulation is conserved.

7. Discussion. A new open boundary condition for hyperbolic flows of an unbounded stratified rotating fluid is suggested which allows forcing to be present at

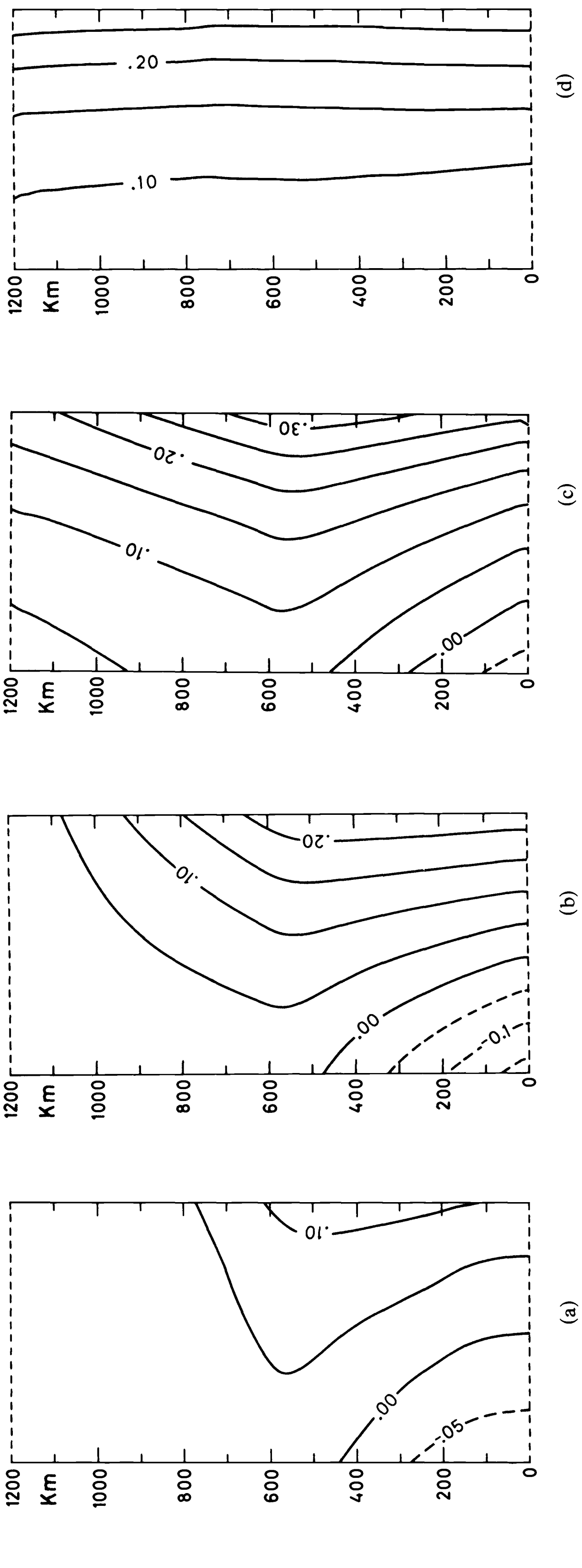


FIG. 9. *Same as Fig. 4. Solutions are shown after time* (a) $t = 3$ *hours,* (b) $t = 5$ *hours,* (c) $t = 8$ *hours and* (d) $t = 16$ *hours. The lower open boundary is forced, the upper one unforced, the forcing being confined to the lower half portion of the channel.*

the open boundaries. This is accomplished by splitting the dependent variables into two modes, termed local and global modes. The local mode does not require any boundary conditions to be imposed and may be computed everywhere, even at the boundaries. The global modes consist of *free* waves and may therefore be assessed using the Sommerfeld radiation condition (1.1) as previously considered by Orlanski (1976) and Camerlengo and O'Brien (1980). The stratification is handled by a separation of the dependent variables into vertical modes (Gill and Clarke (1974)) and thus only the barotropic mode is considered. However, it is important to realize that each vertical mode has to be treated separately in order to satisfy the conditions at an open boundary.

Also considered is a comparison between the method of Orlanski (1976) and the modification of this method made by Camerlengo and O'Brien (1980). The comparison for long dispersive edge waves seems to be in favor of the simpler version of Camerlengo and O'Brien (1980), especially for long simulations.

The applicability of the suggested open boundary condition is shown in a sample problem. The sample problem considers the response of a fluid, contained within an infinitely long channel, to wind forcing. The new conditions are applied at two cross sections along the channel. Both the *forced* waves and *free* waves are reasonably accurately simulated to propagate out of the region.

Acknowledgment. Valuable comments from Jim O'Brien and Tony Busalacchi are appreciated. We thank the Environmental Data Center at the Norwegian Meteorological Institute for providing graphic facilities and Mrs. I. Lange for preparation and typing help.

REFERENCES

[1] A. J. Busalacchi and J. J. O'Brien (1980), *Interannual variability of the Equatorial Pacific in the 1960's*, J. Geophys. Res., 86, pp. 10901–10907.

[2] (CO) A. L. Camerlengo and J. J. O'Brien (1980), *Open boundary conditions in rotating fluids*, J. Comp. Phys., 35, pp. 12–35.

[3] J. G. Charney, R. Fjørtoft and J. von Neumann (1950), *Numerical integration of the barotropic vorticity equation*, Tellus, 2, pp. 237–254.

[4] T. Elvius and A. Sundstrøm (1973), *Computationally efficient schemes and boundary conditions for a fine-mesh barotropic model based on the shallow-water equations*, Tellus, 25, pp. 132–256.

[5] B. Engquist and A. Majda (1977), *Absorbing boundary conditions for the numerical simulation of waves*, Math. Comp., 31, pp. 629–651.

[6] ———, (1979), *Radiation boundary conditions for acoustic and elastic wave calculations*, Comm. Pure Appl. Math., 32, pp. 313–357.

[7] A. E. Gill and A. J. Clarke (1974), *Wind-induced upwelling, coastal currents and sea level changes*, Deep Sea Res., 21, pp. 325–345.

[8] A. E. Gill (1976), *Adjustment under gravity in a rotating channel*, J. Fluid Mech., 79, pp. 603–621.

[9] G. J. Haltiner and R. T. Williams (1980), *Numerical Prediction and Dynamic Meteorology*, John Wiley & Sons, New York, p. 250.

[10] H. E. Hurlburt (1974), *The influence of coastline geometry and bottom topography on the eastern ocean circulation*, Ph.D. dissertation, Florida State Univ., Tallahassee, FL.

[11] E. A. Martinsen, B. Gjevik and L. P. Røed (1979), *A numerical model for long barotropic waves and storm surges along the western coast of Norway*, J. Phys. Oceanogr., 9, pp. 1126–1138.

[12] F. Mesinger and A. Arakawa (1976), *Numerical Methods used in Atmospheric Models*, GARP Publ. Ser. no. 17, Vol. 1, WMO-ICSU Joint Organizing Committee.

[13] (IO) I. Orlanski (1976), *A simple boundary condition for unbounded hyperbolic flows*, J. Comp. Phys., 21, pp. 251–269.

[14] R. O. Reid, A. R. Robinson and K. Bryan (1975), *Numerical Models of Ocean Circulation*, National Academy of Sciences, Washington, DC.

[15] A. Sielecki (1968), *An energy-conserving difference scheme for storm surge equations*, Mon. Weath. Rev., 96, pp. 150–156.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

AN INTERVAL ARITHMETIC APPROACH FOR THE CONSTRUCTION OF AN ALMOST GLOBALLY CONVERGENT METHOD FOR THE SOLUTION OF THE NONLINEAR POISSON EQUATION ON THE UNIT SQUARE*

HARTMUT SCHWANDT†

Abstract. The discretization of the nonlinear Poisson equation on the unit square with Dirichlet boundary conditions leads to very large systems of nonlinear equations for small mesh sizes. The use of interval arithmetic enables us to develop a Newton-like method with an interval "fast Poisson solver." This method converges to the solution of the discretized problem provided an initial inclusion vector is known. The latter is easy to compute. We therefore speak of almost global convergence. Our method competes very well with known algorithms like the generalized conjugate gradient method and others with regard to global convergence, storage requirement and computation time.

Key words. nonlinear Dirichlet problem, fast Poisson solvers, interval arithmetic, Newton-like methods

Introduction. We consider the nonlinear Dirichlet problem

$$\begin{aligned} &\Delta u = f(u) \quad \text{on } I=[0,1]\times[0,1],\\ (0.1)\qquad &u=g \quad \text{on the boundary } \Gamma \text{ of } I,\\ &g \text{ continuous on } \Gamma,\ f_u \text{ continuous and } f_u \geqq 0 \text{ on } I\times\vartheta,\\ &\text{where } \vartheta\subseteq\mathbb{R} \text{ is an open interval.}\end{aligned}$$

It is well known that (0.1) has a unique solution (see for example [3, Chap. II, § 7]). The discretization of this boundary value problem (BVP) even in the case of the simplest five-point formula leads to very large systems of nonlinear equations if one uses realistically small mesh sizes. In this paper we discuss an iterative method that converges under the above conditions to the unique solution—see [14, Thm. 4.4.1]—of the discretized problem. For this purpose we need an interval vector including the solution. As this vector is easy to compute and is furthermore the only prerequisite for the convergence to the solution, we could even speak of global convergence. This particular aspect distinguishes the method to be discussed here from well-known methods for the numerical solution of (0.1) that often suffer from convergence problems (e.g., most of the Newton-like methods are only locally convergent). Most of the well-known globally convergent methods are either time consuming or need too much storage (see [14, Chaps. 10, 12–13], [2, Chaps. 19 and 22] or [18]). One of the first successful algorithms combining a faster method with the property of global convergence was the nonlinear block SOR method (NBSOR) where convergence was controlled by a numerical strategy [12]. The simple structure of problems like the Poisson equation on rectangles permits us, however, to incorporate fast direct methods into algorithms for nonlinear equations. Concus, Golub and O'Leary developed a very efficient generalized conjugate gradient (CG) method for nonlinear equations where the "residual" equation can be solved approximately by a fast direct method. Convergence can be ensured by periodically restarting the iteration after a minimization along a line [7]. The range of applications of that method goes far beyond the Poisson equation discussed here.

* Received by the editors July 7, 1982, and in revised form April 20, 1983.

† Technische Universität Berlin, Fachbereich 3-Mathematik, Berlin, West Germany.

We should also mention the multigrid methods developed in the last years which have also been applied to nonlinear equations—but without a proof of global convergence [10], [11]. The method presented in this paper profits from the use of interval arithmetic. The latter guarantees not only the convergence to the solution of (0.1) by consecutive inclusion, but also permits the design of a fast Poisson solver applicable in the present nonlinear problem as part of an iterative method. This method converges rapidly to the solution of the discretized problem without an excessive storage requirement.

Section 1 contains the notation and some fundamental rules of interval arithmetic. In § 2 we discretize (0.1) to obtain a system of nonlinear equations. In § 3 we present some results for the interval Gauss algorithm needed in later sections. In § 4 we develop a special "poisson solver" for systems of equations with interval coefficients and linear form (see § 3). This method is an important part of the iterative method developed and discussed in § 5. The methods and results of §§ 4 and 5 can be extended to more general situations. This will be done in a subsequent paper.

In § 6 we consider both our interval method and a few other (well-known) methods. We examine their implementation on a computer and discuss their applicability to the BVP (0.1). More specifically we compare our method with some other interval methods, the nonlinear block SOR method (NBSOR) in a version of Hageman and Porsching [12] and the generalized CG method of Concus, Golub and O'Leary [7] (the latter being to our knowledge the fastest of the known algorithms for the numerical solution of (0.1)). Section 7 presents results of some numerical experiments. We computed systems up to a size of 16129×16129. Section 7 is followed by a glossary of abbreviations for algorithms used in this paper.

1. Notation and basic results. Our notation of interval arithmetic is very similar to that in [2, Chaps. 1–4 and 10]. We refer therefore to that source.

We denote by:

$I(\mathbb{R})$	the set of:	the compact intervals in $\mathbb{R}$,
$V_N(\mathbb{R})$		the point vectors with N real components,
$V_N(I(\mathbb{R}))$		the interval vectors with N components belonging to $I(\mathbb{R})$,
$M_{NN}(\mathbb{R})$		the real $N \times N$-point matrices,
$M_{NN}(I(\mathbb{R}))$		the $N \times N$-interval matrices with coefficients belonging to $I(\mathbb{R})$.

We denote by $a, \cdots, z$ the elements of $\mathbb{R}$, by $A, \cdots, Z$ those of $I(\mathbb{R})$, by $\underset{\cdot}{\mathscr{a}}, \cdots, \underset{\cdot}{\mathscr{z}}$ those of $V_N(\mathbb{R})$, by $\mathscr{a}, \cdots, \mathscr{z}$ those of $V_N(I(\mathbb{R}))$, by $\underset{\cdot}{\mathscr{A}}, \cdots, \underset{\cdot}{\mathscr{Z}}$ those of $M_{NN}(\mathbb{R})$ and by $\mathscr{A}, \cdots, \mathscr{Z}$ those of $M_{NN}(I(\mathbb{R}))$.

Point methods use only real numbers and real coefficients. In interval methods real numbers may be replaced by intervals, and real operations are replaced by the corresponding interval operations.

For elements of $I(\mathbb{R})$, $V_N(I(\mathbb{R}))$ and $M_{NN}(I(\mathbb{R}))$ we denote by $d(\cdot)$ the (vector, matrix of) diameter(s), by $|\cdot|$ (that of) the absolute value(s), by $m(\cdot)$ (that of) the midpoint(s) and by $q(\cdot,\cdot)$ (that of) the distance(s). $\rho(\cdot)$ denotes the spectral radius of a real matrix.

Coefficients of $N \times N$-matrices and vectors with N components are specified by

$$\underset{\cdot}{\mathscr{a}} = (a_i)_{i=1}^N, \quad \mathscr{a} = (A_i)_{i=1}^N, \quad \underset{\cdot}{\mathscr{A}} = (a_{ij})_{i=1,j=1}^{N\ \ N}, \quad \mathscr{A} = (A_{ij})_{i=1,j=1}^{N\ \ N}.$$

Intervals $A \in I(\mathbb{R})$ are written as $[\underset{\sim}{A}, \bar{A}]$, a real number a will be identified with the interval $[a, a]$, $\Gamma(A)$ describes the boundary $\{\underset{\sim}{A}, \bar{A}\}$ and A^0 the interior $(\underset{\sim}{A}, \bar{A})$ of an interval A. For $\mathcal{A} \in M_{NN}(I(\mathbb{R}))$ we also write $\mathcal{A} = [\underline{\mathcal{A}}, \overline{\mathcal{A}}]$ with $\underline{\mathcal{A}} = (\underline{A_{ij}})_{i=1,j=1}^{N\ \ N}$; analogously for $\overline{\mathcal{A}}$ and for vectors. The elements of sequences $(x^n)_{n\in\mathbb{N}_0}$ and $(\mathcal{M}^{(n)})_{n\in\mathbb{N}_0}$ are given by $x^n = (X_k^{(n)})_{k\in\mathbb{N}_0}$ and $\mathcal{M}^{(n)} = (M_{ij}^{(n)})_{i=1,j=1}^{N\ \ N}$. We define convergence by

$$\mathcal{M}^{(n)} \to \mathcal{M}^* \in M_{NN}(I(\mathbb{R}))(n \to \infty)$$
$$:\Leftrightarrow \overline{\mathcal{M}^{(n)}} \to \overline{\mathcal{M}^*}, \quad \underline{\mathcal{M}^{(n)}} \to \underline{\mathcal{M}^*}(n \to \infty) \text{ and } \mathcal{M}^* \subseteq \mathcal{M}^{(n+1)} \subseteq \mathcal{M}^{(n)} \quad \forall n \in \mathbb{N}_0$$

and analogously for vector sequences. A vector $x \in V_N(I(\mathbb{R}))$ can be decomposed as follows:

$$N = pq, \quad x = (x_1, \cdots, x_q), \quad x_i = (X_{i1}, \cdots, X_{ip}) \quad \forall i \in \{1, \cdots, p\}.$$

Writing $\mathcal{D} - \mathcal{L} - \mathcal{U}$ or $\mathcal{D}(\cdot) - \mathcal{L} - \mathcal{U}$ for a matrix $\mathcal{M}$ or $\mathcal{M}(\cdot)$, we mean the decomposition into its diagonal strictly lower left respectively strictly upper right part.

$\mathcal{I}$ is the identity matrix, $\mathcal{O}$ (o) the matrix (vector) of zeros. Block tridiagonal interval matrices are described by (with $N = pq$)

$$\mathcal{M} = \begin{bmatrix} \mathcal{A}_1 & \mathcal{C}_1 & & & \\ \mathcal{B}_1 & \mathcal{A}_2 & \mathcal{C}_2 & & \mathcal{O} \\ & \ddots & \ddots & \ddots & \\ \mathcal{O} & & \mathcal{B}_{q-2} & \mathcal{A}_{q-1} & \mathcal{C}_{q-1} \\ & & & \mathcal{B}_{q-1} & \mathcal{A}_q \end{bmatrix} = (\mathcal{B}_{j-1}, \mathcal{A}_j, \mathcal{C}_j)_q.$$

We use the natural (componentwise) partial ordering on $V_N(\mathbb{R})$ and $M_{NN}(\mathbb{R})$. For example, $x > y$ means $x_i > y_i$ for all $i \in \{1, \cdots, N\}$. A matrix $\mathcal{A} \in M_{NN}(\mathbb{R})$ is called an
L-matrix if $a_{ij} \leqq 0$ for $i \neq j$ and $a_{ii} > 0$ for $i = j$,
M-matrix if $a_{ij} \leqq 0$ for $i \neq j$, $\mathcal{A}^{-1}$ exists and $\mathcal{A}^{-1} \geqq \mathcal{O}$.

(1.1) An L-matrix $\mathcal{A}$ is an M-matrix iff there is a vector $u > o$ such that $\mathcal{A}u > o$ [9].

A matrix $\mathcal{A}$ is called an interval M-(L-)matrix if all $\mathcal{A} \in \mathcal{A}$ are M-(L-)matrices.

(1.2) An interval L-matrix $\mathcal{M}$ is an interval M-matrix if there exists an M-matrix $\mathcal{C}$ such that $\mathcal{C} \leqq \underline{\mathcal{M}}$ [16, p. 15].

A map from $V_N(\mathbb{R})$ into $V_N(\mathbb{R})$ is denoted by a point, for example $\dot{f}$, and its values are specified for all $x \in V_N(\mathbb{R})$ by

$$\dot{f}(x) = (f_i(x_1, \cdots, x_N))_{i=1}^N.$$

If we replace operands by intervals, standard functions by interval standard functions, if possible, and operations by the corresponding interval operations, and if the resulting vector belongs to $V_N(I(\mathbb{R}))$, i.e. all operations provide values in $I(\mathbb{R})$, then we write this vector as

$$\dot{f}(x) = (f_i(X_1, \cdots, X_N))_{i=1}^N$$

and call it the interval extension of $\dot{f}$ in x. We further need for $x, y \in V_N(\mathbb{R})$

$$\dot{f}(x/y) = (f_i(x_1, \cdots, x_{i-1}, y_i, \cdots, y_N))_{i=1}^N.$$

The following relations (1.3)–(1.8) are easily deduced from the rules in interval arithmetic,

subset property

(1.3) $\mathcal{A} \subseteq \mathcal{B},\ \mathcal{C} \subseteq \mathcal{D},\ * \in \{+, -, \cdot\} \Rightarrow \mathcal{A} * \mathcal{C} \subseteq \mathcal{B} * \mathcal{D},$

$A \subseteq B,\ C \subseteq D,\ 0 \notin D \Rightarrow A/C \subseteq B/D;$

(1.4) $(\mathcal{A} + \mathcal{B})\mathcal{C} \subseteq \mathcal{A}\mathcal{C} + \mathcal{B}\mathcal{C},\ \mathcal{C}(\mathcal{A} + \mathcal{B}) \subseteq \mathcal{C}\mathcal{A} + \mathcal{C}\mathcal{B},$

$(\mathcal{A} + \mathcal{B})\underset{\cdot}{\mathcal{C}} = \mathcal{A}\underset{\cdot}{\mathcal{C}} + \mathcal{B}\underset{\cdot}{\mathcal{C}},\ \underset{\cdot}{\mathcal{C}}(\mathcal{A} + \mathcal{B}) = \underset{\cdot}{\mathcal{C}}\mathcal{A} + \underset{\cdot}{\mathcal{C}}\mathcal{B};$

(1.5) $\underset{\cdot}{O} \in \mathcal{A},\ \underline{\mathcal{B}} \geqq \underset{\cdot}{O}$ or $\bar{\mathcal{B}} \leqq \underset{\cdot}{O} \Rightarrow d(\mathcal{A}\mathcal{B}) = |\mathcal{A}|d(\mathcal{B}),\ d(\mathcal{B}\mathcal{A}) = d(\mathcal{B})|\mathcal{A}|;$

(1.6) $\mathcal{A} \subseteq \mathcal{B} \Rightarrow d(\mathcal{A}) \leqq d(\mathcal{B});$

(1.7) $d(\underset{\cdot}{\mathcal{A}}\mathcal{B}) = |\underset{\cdot}{\mathcal{A}}|d(\mathcal{B}),\ d(\mathcal{B}\underset{\cdot}{\mathcal{A}}) = d(\mathcal{B})|\underset{\cdot}{\mathcal{A}}|;$

(1.8) $\underset{\cdot}{\mathcal{A}} \in \mathcal{B} \subseteq \mathcal{C} \Rightarrow \underline{\mathcal{C}} \leqq \underline{\mathcal{B}} \leqq \underset{\cdot}{\mathcal{A}} \leqq \bar{\mathcal{B}} \leqq \bar{\mathcal{C}}$ and $|\underset{\cdot}{\mathcal{A}}| \leqq |\mathcal{B}| \leqq |\mathcal{C}|;$

(1.9) $|\mathcal{B}| \leqq \underset{\cdot}{\mathcal{A}} \Rightarrow \rho(\mathcal{B}) \leqq \rho(\underset{\cdot}{\mathcal{A}})$ ([19, Thm. 2.8]).

Let I be an iterative method producing the set $\{(\underset{\cdot}{e}^k)_{k \in \mathbb{N}_0} \in I\}$ of sequences converging to the solution $\underset{\cdot}{y}$ of a given problem. We measure the asymptotic convergence rate of I by the R_1-factor

$$R_1(I, \underset{\cdot}{y}) = \sup \{\limsup_{k \to \infty} \|\underset{\cdot}{e}^k\|^{1/k} | (\underset{\cdot}{e}^k)_{k \in N_0} \in I\}.$$

(See [2, App. A] and [14, Chap. 9].).

The relations (1.3) and (1.4) play the most important role in interval arithmetic. (1.3) enables one to explore inclusion properties by means of interval arithmetic; (1.4) can be regarded as an important deviation from the ordinary real arithmetic. It explains why the unreflected use of intervals often yields too large intervals, i.e. useless results. The main difficulty of all interval methods is therefore the appropriate application of the interval arithmetic.

2. Five-point discretization of the boundary value problem. The discretization of (0.1) is carried out by using the central difference quotient and the mesh size $h = 1/(m+1)$, $m \in \mathbb{N}$, for both coordinate directions:

(2.1)
$$\frac{x_{ij-1} - 2x_{ij} + x_{ij+1}}{h^2} + \frac{x_{i-1j} - 2x_{ij} + x_{i+1j}}{h^2} = f_{ij},$$
$$x_{0j} = g_{0j} = g(jh, 0), \quad x_{m+1j} = g_{m+1j} = g(jh, 1),$$
$$x_{i0} = g_{i0} = g(0, ih), \quad x_{im+1} = g_{im+1} = g(1, ih),$$

with $\underset{\cdot}{x} = (\underset{\cdot}{x}_1, \cdots, \underset{\cdot}{x}_m)$, $\underset{\cdot}{x}_i = (x_{i1}, \cdots, x_i)$ and $f_{ij} = f(jh, ih, x_{ij})$

for $1 \leqq i, j \leqq m$.

This yields the system

(2.2) $\underset{\cdot}{f}(\underset{\cdot}{x}) = \underset{\cdot}{o}$ where $\underset{\cdot}{f}(\underset{\cdot}{x}) = \underset{\cdot}{\mathcal{M}}\underset{\cdot}{x} + \Phi(\underset{\cdot}{x})\ \forall \underset{\cdot}{x} \in V_N(\mathbb{R})$,

$\underset{\cdot}{\mathcal{M}} = (-\underset{\cdot}{\mathcal{I}}, \underset{\cdot}{\mathcal{A}}, -\underset{\cdot}{\mathcal{I}})_m,\ \underset{\cdot}{\mathcal{A}} = (-1, 4, -1)_m,$

and $\Phi(\underset{\cdot}{x}) = (\phi_{kj}(x_{kj}))_{k,j=1}^m$ is a vector with the $N = m^2$ components, $\phi_{kj}(x_{kj}) = h^2 f_{kj} - b_{kj}$ where b_{kj} stands for a boundary value belonging to (k, j) and for 0 otherwise.

Thus we have

$$\mathscr{f}'(\mathscr{x}) = \mathscr{M} + \Phi'(\mathscr{x}) = (-\mathscr{I}, \mathscr{A}_j(\mathscr{x}), -\mathscr{I})_m$$

where $\Phi'(\mathscr{x})$ is a diagonal matrix with diagonal coefficients $\partial\phi_{kj}(x_{kj})/\partial x_{kj}$ and $\mathscr{M}$ is an M-matrix. (2.2) has a unique solution $\mathscr{y} \in \mathscr{x}^0$ (see the introduction). We further have $\Phi'(\mathscr{x}) \geqq \mathscr{O}\ \forall \mathscr{x} \in \vartheta$ because of $f_u \geqq 0$ on ϑ and therefore $\underline{\Phi'(\mathscr{x}) \geqq \mathscr{O}}$ for $\mathscr{x} \subset \vartheta$ if we use a suitable interval extension of $\Phi'(\cdot)$ in $\mathscr{x}$. $\Phi'(\cdot)$ is a diagonal matrix. Therefore $\mathscr{f}'(\mathscr{x})$ is an interval M-matrix for $\mathscr{x} \subset \vartheta$ by (1.2).

3. Interval-arithmetic Gauss algorithm (IGA).

The interval-arithmetic Gauss algorithm (IGA) plays an important role in § 4. We therefore collect some properties (see also [1] and [2, Chap. 15]). IGA solves the following problem

(3.1) *Given*

$\mathscr{A} \in M_{NN}(I(\mathbb{R}))$, $\mathscr{y} \in V_N(I(\mathbb{R}))$, $\mathscr{A}^{-1}$ exists for all $\mathscr{A} \in \mathscr{A}$,

we *seek* an inclusion $\mathscr{x} \in V_N(I(\mathbb{R}))$ of the set of solutions

$$LM = \{\mathscr{x} \in V_N(\mathbb{R}) \mid \mathscr{A}\mathscr{x} = \mathscr{y},\ \mathscr{A} \in \mathscr{A},\ \mathscr{y} \in \mathscr{y}\}.$$

$(\mathscr{A}, \mathscr{y})$ defines what we will call a system of equations with interval coefficients and linear form.

In this paper we need the following version of IGA, where the nonvanishing coefficients of $\mathscr{A}$ are stored in an $N \times (2m+1)$-matrix $\mathscr{D}$ defined by

$$D_{ij} := \begin{cases} A_{ij+i}, & 1 \leqq i \leqq N, \quad -m \leqq j \leqq m, \quad j+i>0, \\ 0, & \text{otherwise.} \end{cases}$$

(3.2) IGA *for band matrices* (*bandwidth* $2m+1$)

```
reduction part
k := 1(1)N − 1
    i := k + 1(1) min (k + m, N)
        $C := D_{i,k-i}/D_{k0}$
        $Y_i := Y_i - Y_k \cdot C$
        $j := k+1-i(1) \min(k+m, N) - i$
            $D_{ij} := D_{ij} - D_{k,j-k+i} \cdot C$
solution part
$Y_N := Y_N/D_{N0}$
$i := N-1(-1)1$
  $j := 1(1) \min(m, N-i)$
    $Y_i := Y_i - D_{ij}Y_{j+i}$
  $Y_i := Y_i/D_{i0}$.
```

The vector $\mathscr{y}$ gives us the desired inclusion $\mathscr{x}$ of LM, for which we write

$$\mathscr{x} = \text{IGA}(\mathscr{A}, \mathscr{y}).$$

For our iterative method in § 5 we apply a simplified version of (3.2) which can be found in § 6. The division in IGA are well defined if $0 \notin D_{k0} = A_{kk}$. As the D_{k0} are altered, this condition is not automatically fulfilled. But we may apply the following criterion to ensure that IGA can be carried out, even without pivoting. Let $\mathcal{B}$ be the matrix defined by

$$b_{ij} = \begin{cases} m(A_{ii}) - d(A_{ii})/2, & i = j, \\ -|A_{ij}|, & i \neq j, \end{cases} \quad 1 \leqq i, j \leqq N.$$

THEOREM 3.1 [1]. *If $\mathcal{B}$ is an M-matrix, the interval-arithmetic Gauss algorithm can be carried out with $\mathcal{A}$. No pivoting is necessary.*

If IGA can be carried out then we immediately deduce from the subset property (1.3):

LEMMA 3.2. *If $\mathcal{A}, \mathcal{B} \in M_{NN}(I(\mathbb{R}))$, $x, y \in V_N(I(\mathbb{R}))$, then*

$$\mathcal{A} \subseteq \mathcal{B}, x \subseteq y \Rightarrow \text{IGA}(\mathcal{A}, x) \subseteq \text{IGA}(\mathcal{B}, y).$$

Remark 3.3. Due to the continuity of the interval operations $(+, -, *, /)$, IGA itself is continuous: assuming that IGA can be carried out, we have

$$\mathcal{A}^{(k)} \to \mathcal{A}^*, \ell^k \to \ell^*(k \to \infty) \Rightarrow \text{IGA}(\mathcal{A}^{(k)}, \ell^k) \to \text{IGA}(\mathcal{A}^*, \ell^*)(k \to \infty).$$

4. Reduction method for interval matrices. While solving (2.2) we have to operate with large matrices of a special structure. We now develop a reduction method for the computation of an inclusion vector for the set of solutions of a system with interval coefficients and linear form whose matrix has the given structure. Comparing this method with the Gauss algorithm we will notice a remarkable reduction of the number of arithmetic operations and of the storage requirement (see the numbers, for problem (2.2), of the iterative methods using IGA and of those using the reduction method (Tables 2 and 3, § 7)). Our reduction method is an important part of the iterative method to be presented in the next section.

In (3.1) we replace $\mathcal{A}$ by the matrix

$$\mathcal{M} = \begin{bmatrix} \mathcal{A} & \mathcal{T} & & & \\ \mathcal{T} & \mathcal{A} & \mathcal{T} & & \mathcal{O} \\ & \ddots & \ddots & \ddots & \\ & \mathcal{O} & \mathcal{T} & \mathcal{A} & \mathcal{T} \\ & & & \mathcal{T} & \mathcal{A} \end{bmatrix} \in M_{NN}(I(\mathbb{R})) \tag{4.1}$$

with $\mathcal{A} \in M_{pp}(I(\mathbb{R}))$, $\mathcal{T} \in M_{pp}(\mathbb{R})$,

$\mathcal{A}\mathcal{T} = \mathcal{T}\mathcal{A}$ for all $\mathcal{A} \in \mathcal{A}$,

$N = pq$, $q = 2^{k+1} - 1$, $k, p \in \mathbb{N}$.

We will develop an algorithm formally similar to the Buneman algorithm for linear point systems (see [5] and [17, § 8.8]).

We first choose $\mathcal{M} \in \mathcal{M}$ and $y \in y$ and consider the system

$$\mathcal{M}x = y.$$

With $\mathscr{A}^{(0)} := \mathscr{A}$, $\mathscr{T}^{(0)} := \mathscr{T}$, $\mathscr{y}^{(0)} := \mathscr{y}$ this system can be written as follows:

$$\begin{pmatrix} \mathscr{A}^{(0)} & \mathscr{T}^{(0)} & & \mathscr{O} \\ \mathscr{T}^{(0)} & \mathscr{A}^{(0)} & \mathscr{T}^{(0)} & \\ \ddots & \ddots & \ddots & \\ \mathscr{O} & \mathscr{T}^{(0)} & \mathscr{A}^{(0)} & \mathscr{T}^{(0)} \\ & & \mathscr{T}^{(0)} & \mathscr{A}^{(0)} \end{pmatrix} \begin{pmatrix} \mathscr{x}_1 \\ \mathscr{x}_2 \\ \vdots \\ \mathscr{x}_{2^{k+1}-2} \\ \mathscr{x}_{2^{k+1}-1} \end{pmatrix} = \begin{pmatrix} \mathscr{y}_1^{0} \\ \mathscr{y}_2^{0} \\ \vdots \\ \mathscr{y}_{2^{k+1}-2}^{0} \\ \mathscr{y}_{2^{k+1}-1}^{0} \end{pmatrix}.$$

With $\mathscr{x}_0 := \mathscr{x}_{2^{k+1}} := \mathscr{o}$ an equivalent formulation is

$$\begin{aligned} &\mathscr{T}^{(0)}\mathscr{x}_{j-2} + \mathscr{A}^{(0)}\mathscr{x}_{j-1} + \mathscr{T}^{(0)}\mathscr{x}_j && = \mathscr{y}_{j-1}^{0}, \\ &\qquad \mathscr{T}^{(0)}\mathscr{x}_{j-1} + \mathscr{A}^{(0)}\mathscr{x}_j + \mathscr{T}^{(0)}\mathscr{x}_{j+1} && = \mathscr{y}_j^{0}, \qquad 2 \leqq j \leqq 2^{k+1}-2, \\ &\qquad\qquad \mathscr{T}^{(0)}\mathscr{x}_j + \mathscr{A}^{(0)}\mathscr{x}_{j+1} + \mathscr{T}^{(0)}\mathscr{x}_{j+2} && = \mathscr{y}_{j+1}^{0}. \end{aligned}$$

Then we multiply the equations $j-1$ and $j+1$ by $\mathscr{T}^{(0)}$ and equation j by $-\mathscr{A}^{(0)}$, and add. For $\mathscr{T}^{(1)} := (\mathscr{T}^{(0)})^2$ and $\mathscr{A}^{(1)} := 2\mathscr{T}^{(1)} - (\mathscr{A}^{(0)})^2$, the relation $\mathscr{A}\mathscr{T} = \mathscr{T}\mathscr{A}$ gives $\mathscr{A}^{(1)}\mathscr{T}^{(1)} = \mathscr{T}^{(1)}\mathscr{A}^{(1)}$. With

$$\mathscr{y}_j^{1} := \mathscr{T}^{(0)}(\mathscr{y}_{j-1}^{0} + \mathscr{y}_{j+1}^{0}) - \mathscr{A}^{(0)}\mathscr{y}_j^{0}, \qquad j := 2(2)2^{k+1}-2,$$

we obtain the reduced system

$$\begin{pmatrix} \mathscr{A}^{(1)} & \mathscr{T}^{(1)} & & \\ \mathscr{T}^{(1)} & \mathscr{A}^{(1)} & \mathscr{T}^{(1)} & \mathscr{O} \\ \ddots & \ddots & \ddots & \\ \mathscr{O} & \mathscr{T}^{(1)} & \mathscr{A}^{(1)} & \mathscr{T}^{(1)} \\ & & \mathscr{T}^{(1)} & \mathscr{A}^{(1)} \end{pmatrix} \begin{pmatrix} \mathscr{x}_2 \\ \mathscr{x}_4 \\ \vdots \\ \mathscr{x}_{2^{k+1}-4} \\ \mathscr{x}_{2^{k+1}-2} \end{pmatrix} = \begin{pmatrix} \mathscr{y}_2^{1} \\ \mathscr{y}_4^{1} \\ \vdots \\ \mathscr{y}_{2^{k+1}-4}^{1} \\ \mathscr{y}_{2^{k+1}-2}^{1} \end{pmatrix},$$

which can again be reduced as described above.

We now suppose that in the $(r-1)$st step, $2 \leqq r \leqq k$, we have obtained the equations

$$\mathscr{T}^{(r-1)}\mathscr{x}_{j-2^{r-1}} + \mathscr{A}^{(r-1)}\mathscr{x}_j + \mathscr{T}^{(r-1)}\mathscr{x}_{j+2^{r-1}} = \mathscr{y}_j^{r-1}, \qquad j := 2^{r-1}(2^{r-1})2^{k+1} - 2^{r-1}.$$

We multiply the equations $j-2^{r-1}$ and $j+2^{r-1}$ by $\mathscr{T}^{(r-1)}$ and equation j by $-\mathscr{A}^{(r-1)}$, and we add these equations. We define

$$\begin{aligned} &\mathscr{T}^{(r)} := (\mathscr{T}^{(r-1)})^2, \qquad \mathscr{A}^{(r)} := 2\mathscr{T}^{(r)} - (\mathscr{A}^{(r-1)})^2, \\ &\mathscr{y}_j^{r} := \mathscr{T}^{(r-1)}(\mathscr{y}_{j-2^{r-1}}^{r-1} + \mathscr{y}_{j+2^{r-1}}^{r-1}) - \mathscr{A}^{(r-1)}\mathscr{y}_j^{r-1}, \qquad j := 2^r(2^r)2^{k+1} - 2^r. \end{aligned}$$

In view of $\mathscr{A}^{(r-1)}\mathscr{T}^{(r-1)} = \mathscr{T}^{(r-1)}\mathscr{A}^{(r-1)}$ we deduce the relation

$$\mathscr{A}^{(r)}\mathscr{T}^{(r)} = \mathscr{T}^{(r)}\mathscr{A}^{(r)}$$

and obtain again a reduced system:

$$\begin{pmatrix} \mathscr{A}^{(r)} & \mathscr{T}^{(r)} & & \\ \mathscr{T}^{(r)} & \mathscr{A}^{(r)} & \mathscr{T}^{(r)} & \mathscr{O} \\ \ddots & \ddots & \ddots & \\ \mathscr{O} & \mathscr{T}^{(r)} & \mathscr{A}^{(r)} & \mathscr{T}^{(r)} \\ & & \mathscr{T}^{(r)} & \mathscr{A}^{(r)} \end{pmatrix} \begin{pmatrix} \mathscr{x}_{2^r} \\ \mathscr{x}_{2^{r+1}} \\ \vdots \\ \mathscr{x}_{2^{k+1}-2^{r+1}} \\ \mathscr{x}_{2^{k+1}-2^r} \end{pmatrix} = \begin{pmatrix} \mathscr{y}_{2^r}^{r} \\ \mathscr{y}_{2^{r+1}}^{r} \\ \vdots \\ \mathscr{y}_{2^{k+1}-2^{r+1}}^{r} \\ \mathscr{y}_{2^{k+1}-2^r}^{r} \end{pmatrix}.$$

For $r=k$ the system

$$\mathcal{A}^{(k)}x_{2^k}=y^k_{2^k}$$

remains to be solved. The reduced systems of the steps $r=0,\cdots,k-1$ can be solved in reversed order with respect to the subvectors x_j composing the solution x: we solve for x_j

$$\text{(4.2)}\qquad \begin{gathered} r:=k-1(-1)0,\qquad j:=2^r(2^r)2^{k+1}-2^r: \\ \mathcal{T}^{(r)}x_{j-2^r}+\mathcal{A}^{(r)}x_j+\mathcal{T}^{(r)}x_{j+2^r}=y^r_j. \end{gathered}$$

In order to avoid multiplications with the matrices $\mathcal{A}^{(r)}$, we replace the vectors y^r_j by new vectors. The computation of the new vectors only requires the solving of systems of linear equations with the matrices $\mathcal{A}^{(r)}$. Instead of computing the y^r_j, we write

$$p^0_j:=o,\quad q^0_j:=y_j,\quad j:=1(1)2^{k+1}-1,$$

and define

$$\begin{aligned} p^{r+1}_j&:=p^r_j-(\mathcal{A}^{(r)})^{-1}\{\mathcal{T}^{(r)}(p^r_{j-2^r}+p^r_{j+2^r})=q^r_j\}, \\ q^{r+1}_j&:=\mathcal{T}^{(r)}\{q^r_{j-2^r}+q^r_{j+2^r}-2\mathcal{T}^{(r)}p^{r+1}_j\},\qquad r:=0(1)k-1,\qquad j:=2^r(2^r)2^{k+1}-2^r. \end{aligned}$$

Using (4.2) it can be shown by induction that

$$\text{(4.3)}\qquad r:=0(1)k,\ j:=2^r(2^r)2^{k+1}-2^r:\quad y^r_j=\mathcal{A}^{(r)}p^r_j+q^r_j.$$

Later we will substitute (4.3) into (4.2). Multiplications by $\mathcal{A}^{(r)}$ will be replaced by the solution of linear systems of equations. We therefore factorize

$$\text{(4.4)}\qquad \mathcal{A}^{(r)}=-\prod_{j=1}^{2^r}\left(\mathcal{A}+\left\{2\cos\left(\frac{2j-1}{2^{r+1}}\pi\right)\right\}\mathcal{T}\right).$$

The way of solving the system $\mathcal{M}x=y$ as done above leads us to the interval-arithmetic reduction algorithm for the system $(\mathcal{M},y)$:

(4.5) *Reduction algorithm for interval matrices* (IBU).

```
$\mathcal{T}^{(0)}:=\mathcal{T},\ \mathcal{A}^{(0)}:=\mathcal{A},\ p^0_j:=o,\ q^0_j:=y,$          $j:=1(1)2^{k+1}-1,$
$\mathcal{G}^{(r)}_j:=\mathcal{A}+\{2\cos((2j-1)\pi/2^{r+1})\}\mathcal{T},$          $r:=0(1)k,\ j:=1(1)2^r.$
reduction part
$r:=1(1)k:$
    $\mathcal{T}^{(r-1)}:=(\mathcal{T}^{(r-2)})^2$          (if $r>1$)
    $j:=2^r(2^r)2^{k+1}-2^r:$
        $z^{2^{r-1}}:=\mathrm{IGA}\,(\mathcal{G}^{(r-1)}_{2^{r-1}},\ \mathcal{T}^{(r-1)}(p^{r-1}_{j-2^{r-1}}+p^{r-1}_{j+2^{r-1}})-q^{r-1}_j)$
        $l:=2^{r-1}-1(-1)1:$
            $z^l:=\mathrm{IGA}\,(\mathcal{G}^{(r-1)}_l,\ z^{l+1})$
        $p^r_j:=p^{r-1}_j+z^1$
        $q^r_j:=\mathcal{T}^{(r-1)}(q^{r-1}_{j-2^{r-1}}+q^{r-1}_{j+2^{r-1}}-2\mathcal{T}^{(r-1)}p^r_j).$
```

```
*solution part*
$\mathscr{x}_0 := \mathscr{x}_{q+1} := \mathscr{o}$
$r := k(-1)0$:
    $j := 2^r(2^{r+1})2^{k+1} - 2^r$
        $\mathscr{z}^{2^r} := \mathrm{IGA}\,(\mathscr{G}_{2^r}^{(r)}, \mathscr{q}_j^r - \mathscr{T}^{(r)}(\mathscr{x}_{j-2^r} + \mathscr{x}_{j+2^r}))$
        $l := 2^r - 1(-1)1$
            $\mathscr{z}^l := \mathrm{IGA}\,(\mathscr{G}_l^{(r)}, \mathscr{z}^{l+1})$
        $\mathscr{x}_j := \mathscr{p}_j^r - \mathscr{z}^1$.
```

For point systems $\underset{\cdot}{\mathscr{M}}\underset{\cdot}{\mathscr{x}} = \underset{\cdot}{\mathscr{y}}$ this algorithm degenerates to an algorithm similar to Buzbee's extension to the case $\mathscr{A}\mathscr{T} = \mathscr{T}\mathscr{A}$ of Buneman's algorithm [6]. However, our algorithm needs fewer arithmetic operations.

First of all we show that the vector $\mathscr{x} = (\mathscr{x}_1, \cdots, \mathscr{x}_q)$ includes the set LM of solutions of $(\mathscr{M}, \mathscr{y})$. We write

$$\mathscr{x} = \mathrm{IBU}\,(\mathscr{M}, \mathscr{y}).$$

THEOREM 4.1. *Suppose* IBU *is applicable to problem* (3.1) *modified by* (4.1). *Then*

$$LM \subseteq \mathscr{x} = \mathrm{IBU}\,(\mathscr{M}, \mathscr{y}).$$

Proof. Let $\underset{\cdot}{\mathscr{M}} \in \mathscr{M}$, $\underset{\cdot}{\mathscr{y}} \in \mathscr{y}$ and $\underset{\cdot}{\mathscr{M}}\underset{\cdot}{\mathscr{x}} = \underset{\cdot}{\mathscr{y}}$. We apply IBU simultaneously to $(\underset{\cdot}{\mathscr{M}}, \underset{\cdot}{\mathscr{y}})$ and $(\mathscr{M}, \mathscr{y})$. Of course the representation (4.5) and the factorization (4.4) are also valid for point matrices and point vectors. Step by step we show for all indices

$$\underset{\cdot}{\mathscr{p}}_j^r \in \mathscr{p}_j^r, \quad \underset{\cdot}{\mathscr{q}}_j^r \in \mathscr{q}_j^r, \quad \underset{\cdot}{\mathscr{x}}_j \in \mathscr{x}_j \tag{4.6}$$

using Lemma 3.2 and the subset property (1.3). We then have

$$\underset{\cdot}{\mathscr{x}} = \mathrm{IBU}\,(\underset{\cdot}{\mathscr{M}}, \underset{\cdot}{\mathscr{y}}) = \underset{\cdot}{\mathscr{M}}^{-1}\underset{\cdot}{\mathscr{y}}$$

and finally

$$\underset{\cdot}{\mathscr{x}} = \mathrm{IBU}\,(\underset{\cdot}{\mathscr{M}}, \underset{\cdot}{\mathscr{y}}) \in \mathrm{IBU}\,(\mathscr{M}, \mathscr{y}) = \mathscr{x}.$$

As $\underset{\cdot}{\mathscr{M}} \in \mathscr{M}$ and $\underset{\cdot}{\mathscr{y}} \in \mathscr{y}$ are arbitrary, we conclude

$$LM \subseteq \mathscr{x} = \mathrm{IBU}\,(\mathscr{M}, \mathscr{y}). \qquad \square$$

The next theorem gives a criterion for deciding if IBU can be carried out.

THEOREM 4.2. *Let the matrix $\mathscr{B} \in M_{pp}(\mathbb{R})$ be defined by*

$$b_{ij} = \begin{cases} \underline{A}_{ii} + 2\{\cos((1-2^{-(k+1)})\pi)\}t_{ii}, & i=j, \quad \underline{A}_{ii} > 0, \quad t_{ii} \geqq 0, \\ \underline{A}_{ii} + 2\{\cos(2^{-(k+1)}\pi)\}t_{ii}, & i=j, \quad \underline{A}_{ii} > 0, \quad t_{ii} < 0, \\ -\max\limits_{1 \leqq r \leqq k} \max\limits_{1 \leqq l \leqq 2^r} \left\{ \left| A_{ij} + 2\cos\left(\dfrac{2l-1}{2^{r+1}}\pi\right) t_{ij} \right| \right\}, & i \neq j. \end{cases}$$

If $\mathscr{B}$ is an M-matrix, then IBU *can be applied to* $(\mathscr{M}, \mathscr{y})$. *No pivoting is necessary for the applications of* IGA *in* IBU.

Proof. The question of the applicability of IBU only arises where IGA has to be used. $\mathcal{B}$ being an M-matrix, we have $b_{ii} > 0$ for $1 \leqq i \leqq p$. With

$$-2 < 2\cos\left\{\left(1-\frac{1}{2^{k+1}}\right)\pi\right\} \leqq 2\cos\left\{\left(\frac{2j-1}{2^{r+1}}\right)\pi\right\} \leqq 2\cos\left(\frac{\pi}{2^{k+1}}\right) < 2$$

for $j \in \{1, \cdots, 2^r\}$, $r \in \{1, \cdots, k\}$, we conclude that for $i \in \{1, \cdots, p\}$ and each $\mathscr{A} \in \mathscr{A}$

$$0 < \underline{A_{ii}} + 2\left\{\cos\left(\left(1-\frac{1}{2^{k+1}}\right)\pi\right)\right\} t_{ii} \leqq a_{ii} + 2\left\{\cos\left(\frac{2j-1}{2^{r+1}}\pi\right)\right\} t_{ii} \quad \text{if } \underline{A_{ii}} \geqq 0, \quad t_{ii} \geqq 0$$

and

$$0 < \underline{A_{ii}} + 2\left\{\cos\left(\frac{\pi}{2^{k+1}}\right)\right\} t_{ii} \leqq a_{ii} + 2\left\{\cos\left(\frac{2j-1}{2^{r+1}}\pi\right)\right\} t_{ii} \quad \text{if } \underline{A_{ii}} \geqq 0, \quad t_{ii} < 0.$$

For $i \neq j$ and $1 \leqq l \leqq 2^r$, $1 \leqq r \leqq k$, we have

$$|b_{ij}| \geqq \left| A_{ij} + 2\left\{\cos\left(\frac{2l-1}{2^{r+1}}\pi\right)\right\} t_{ij} \right| \geqq \left| a_{ij} + 2\left\{\cos\left(\frac{2l-1}{2^{r+1}}\pi\right)\right\} t_{ij} \right|.$$

$\mathcal{B}$ being an M-matrix, there is a vector $u > o$ such that $\mathcal{B}u > o$. This means for all $i \in \{1, \cdots, p\}$, $r \in \{1, \cdots, k\}$, $l \in \{1, \cdots, 2^r\}$

$$\begin{aligned}\left(a_{ii} + 2\left\{\cos\left(\frac{2l-1}{2^{r+1}}\pi\right)\right\} t_{ii}\right) u_i &\geqq \left(\underline{A_{ii}} + 2\left\{\cos\left(\left(1-\frac{1}{2^{k+1}}\right)\pi\right)\right\} t_{ii}\right) u_i \\ &\geqq \sum_{\substack{j=1 \\ j\neq i}}^{p} \max_{1\leqq r\leqq k} \max_{1\leqq s\leqq 2^r} \left\{ \left| A_{ij} + 2\left\{\cos\left(\frac{2s-1}{2^{r+1}}\pi\right)\right\} t_{ij} \right| \right\} u_j \\ &\geqq \sum_{\substack{j=1 \\ j\neq i}}^{p} \left| a_{ij} + 2\left\{\cos\left(\frac{2l-1}{2^{r+1}}\pi\right)\right\} t_{ij} \right| u_j\end{aligned}$$

if $\underline{A_{ii}} > 0$, $t_{ii} > 0$ (analogously for the other case). Following [1, proof of (1.2) and Thm. 1], we conclude from the above chain of inequalities that IGA can be carried out with the matrices $\mathscr{A}$ and $\mathscr{G}_j^{(r)}$ without pivoting. Consequently IBU can be carried out.

Remark 4.3. Like IGA, IBU is continuous in the following sense:

$$\mathcal{M}^{(k)} \to \mathcal{M}^*, \; y^k \to y^*(k\to\infty) \Rightarrow \text{IBU}\,(\mathcal{M}^{(k)}, y^k) \to \text{IBU}\,(\mathcal{M}^*, y^*)(k\to\infty).$$

Remark 4.4. If $\mathcal{M}$ is an interval M-matrix, then it can be shown that all coefficient matrices arising in the reduction part are also interval M-matrices. We therefore know the signs of many interval bounds while carrying out IBU. This fact immensely simplifies interval operations. The applications in § 6 will lead us to interval M-matrices.

5. An iterative method for systems of nonlinear equations. The system (2.2) can be imbedded in a more general framework for §§ 5–7:

(5.1) *Given* a function $f: \vartheta \to V_N(\mathbb{R})$, $\vartheta \subseteq V_N(\mathbb{R})$, f continuously differentiable in ϑ, and an interval vector $x^0 \in V_N(I(\mathbb{R}))$ contained in ϑ and including exactly one zero y of f:

$$y \in x^0 \subseteq \vartheta,$$

we *seek* y.

Then, for all $\mathbf{x} \subseteq \mathbf{x}^0$ and all $x, z \in \mathbf{x}$, there is a vector $u \in \mathbf{x}$ such that

(5.2) $$f(x) - f(z) = f'(u)(x - z).$$

(5.3) We further require the existence of an interval extension of $f'(\cdot)$ in $\mathbf{x}^0$.

If $f'(x)$ is invertible for all $x \in \mathbf{x}^0$, we can transform (5.2) into

$$y = x - f'(u)^{-1} f(x)$$

with $z = y$. If IGA can be applied to $f'(\mathbf{x}^0)$, the subset property (1.3), Lemma 3.2 and the fact that $u \in \mathbf{x}$ show that

(5.4) $$y \in x - \mathrm{IGA}\,(f'(\mathbf{x}), f(x)).$$

We can define the

(5.5) *Interval-arithmetic Newton method* (ING)

$$\mathbf{x}^{k+1} := \{m(\mathbf{x}^k) - \mathrm{IGA}\,(f'(\mathbf{x}^k), f(m(\mathbf{x}^k)))\} \cap \mathbf{x}^k \quad \text{for all } k \in \mathbb{N}_0.$$

Decomposing $f'(\cdot) = \mathcal{D}(\cdot) - \mathcal{L}(\cdot) - \mathcal{U}(\cdot)$ (see § 1), a similar development leads to the

(5.6) *Interval-arithmetic Newton–Gauss–Seidel method with intersection after each component* (INESVKD)

$i := 1(1)N,$

$$X_i^{(k+1)} := \left\{ m(\mathbf{x}^k) - \left(\sum_{j=1}^{i-1} L_{ij}(\mathbf{x}^k)(m(X_j^{(k)}) - X_j^{(k+1)}) \right.\right.$$
$$\left. + \sum_{j=i+1}^{N} U_{ij}(\mathbf{x}^k)(m(X_j^{(k)}) - X_j^{(k)}) + f_i(m(\mathbf{x}^k)) \right)$$
$$\left. \Big/ D_{ii}(\mathbf{x}^k) \right\} \cap X_i^{(k)} \quad \text{for all } k \in \mathbb{N}_0.$$

Under certain conditions INESVKD and ING converge to y (see [2, §§ 19, 22]). There the convergence of ING to y is ensured by using an auxiliary sequence. Applied to the system (2.2), INESVKD requires a relatively small number of arithmetic operations per step, but the total computation time is much too high. If ING converges to y, the convergence is quadratic. However, the iteration steps are too expensive for large systems.

In order to be able to compete in computation speed with methods like NBSOR or the generalized CG method, we replace IGA by IBU in ING. But $f'(\cdot)$ does not have the form required by IBU—see (2.2) and (4.1). We therefore define

$$\forall \mathbf{x} \subseteq \mathbf{x}^0$$

(5.7) $$\mathcal{A}(\mathbf{x}) = (A_{ik}(\mathbf{x}))_{i=1}^{p}{}_{k=1}^{p} \quad \text{by}$$

$$A_{ik}(\mathbf{x}) = [\min\{\underline{A_{ikj}(\mathbf{x})} \mid 1 \leq j \leq q\}, \quad \max\{\overline{A_{ikj}(\mathbf{x})} \mid 1 \leq j \leq q\}]$$

and replace $f'(\mathbf{x}) = (\mathcal{T}, \mathcal{A}_j(\mathbf{x}), \mathcal{T})_q$ (with $\mathcal{T} = -\mathcal{I}$ for (2.2)) by

(5.8) $$\hat{\mathcal{F}}(\mathbf{x}) = (\mathcal{T}, \mathcal{A}(\mathbf{x}), \mathcal{T})_q.$$

Hence

(5.9) $$f'(\mathbf{x}) \subseteq \hat{\mathcal{F}}(\mathbf{x}) \quad \forall \mathbf{x} \subseteq \mathbf{x}^0.$$

The q diagonal blocks $\mathcal{A}(x)$ of $\hat{\mathcal{F}}(x)$ are now identical. So we can take advantage of the results of § 4 in order to apply IBU:

(5.10) *Interval-arithmetic Newton-like method with* IBU (INB)

$$\forall k \in N_0: m^k \in x^k$$

$$x^{k+1} := \{m^k - \text{IBU}(\hat{\mathcal{F}}(x^k), f(m^k))\} \cap x^k.$$

Analogously to (5.3) we conclude from (5.2) and (5.9) that

(5.11) $$y \in x - \text{IBU}(\hat{\mathcal{F}}(x), f(x)) \quad \forall x \subseteq x^0, \quad \forall x \in x.$$

We therefore can show by induction:

LEMMA 5.1. *If* INB *can be carried out, all the components of the sequence* $(x^k)_{k\in\mathbb{N}_0}$ *computed by* INB *include* y:

$$y \in x^k \quad \forall k \in \mathbb{N}_0.$$

Due to the intersection in each step we thus have a sequence with

$$y \in x^k \subseteq x^{k-1} \subseteq \cdots \subseteq x^0 \quad \forall k \in \mathbb{N}_0.$$

This guarantees the existence of

$$x^* = \lim_{k\to\infty} x^k = \bigcap_{k\in\mathbb{N}_0} x^k$$

with $y \in x^*$. The following considerations will lead to conditions that guarantee $x^* \equiv x^*$, i.e. $y = x^*$. An example shows that the latter equality is not obvious for methods like INB or ING.

Example 5.2. The map $f: \mathbb{R}^2 \to \mathbb{R}^2$ defined by

$$f(x) = (-x^2 + y^2 - 1, x^2 - y) \quad \forall x \in \mathbb{R}^2$$

has the real zeros $(\pm\sqrt{(1+\sqrt{5})/2}, (1+\sqrt{5})/2)$. Thus the starting interval

$$x^0 = ([1.1, 1.9], [1.1, 1.9])$$

for the positive zero does not include another zero. We obtain

$$([-3/88, 90771/12584], [7/8, 5801/1144]) = m(x^0) - \text{IGA}(f'(x^0), f(m(x^0))) \supseteq x^0,$$

hence $x^0 = x^1$. ING terminates on the starting vector.

In order to force the converge of ING and INB to y, we add auxiliary steps (but only if necessary). After a finite number of steps the diameter of each component of the iterates falls below a given bound eps. To terminate the iteration we therefore use the criterion

(5.12) $$\|d(x^k)\|_\infty < \text{eps}.$$

Auxiliary steps are necessary only if the diameters of the components of two consecutive iterates do not differ enough. As a decision criterion in the $(k+1)$th main iterative step, we take:

(5.13) $$\exists i \in \{1, \cdots, N\}: d(\tilde{X}_i^{(k+1)}) \geq \alpha d(X_i^{(k)}),$$
$$\text{with given } \alpha < 1, \alpha \approx 1 \quad \text{(see (5.14))}.$$

Before we can apply (5.13), we have to know if the condition $d(\tilde{X}_i^{(k+1)}) < \text{eps}$ is already fulfilled. In this case a further decrease of the diameter is either not desirable or not possible.

If, however, auxiliary steps become necessary after the kth main step, we set $\mathfrak{z}^0 := \tilde{\mathfrak{x}}^{k+1}$ and compute a number l of auxiliary iterates. Then we set $\mathfrak{x}^{k+1} = \mathfrak{z}^l$. We require $l \geqq g$, where g is an estimate for the quotient of the number of arithmetic operations of a main and an auxiliary step. The auxiliary steps are computed by a modification of INESVKD (5.6), in which the interval expressions computed for the partial derivatives needed for a main iterate $\mathfrak{x}^k$ are used for all corresponding auxiliary iterates $\mathfrak{z}^j$ with $\mathfrak{z}^0 = \tilde{\mathfrak{x}}^{k+1}$. This is done in order to reduce the computation time.

As we consider the special system (2.2) in this paper, we have $\mathfrak{f}'(\mathfrak{x}) = \mathcal{M} + \Phi'(\mathfrak{x})\ \forall \mathfrak{x} \in V_N(\mathbb{R})$ with the diagonal matrix $\Phi'(\mathfrak{x})$. Hence $\mathcal{L}(\cdot) = \mathcal{L}$, $\mathcal{U}(\cdot) = \mathcal{U}$ are independent of $\mathfrak{x}$. Using the second part of (1.4) the modification of INESVKD can be simplified. The number l of auxiliary steps is determined by $l \geqq g$ and the condition $d(\mathfrak{z}^{l+1}) < \alpha d(\mathfrak{x}^k)$.

In order to avoid unnecessary steps, the condition $\|\mathfrak{z}^i\|_\infty < \text{eps}$ with $i < g$ also terminates the whole process. We now define

(5.14) INB *with a modified version* HINESVKD *of* INESVKD *as auxiliary method* (INBHIN)

```
$\mathfrak{m}^0 \in \mathfrak{x}^0$;
$\forall k \in \mathbb{N}_0$: $\tilde{\mathfrak{x}}^{k+1} := \{\mathfrak{m}^k - \mathrm{IBU}\,(\mathfrak{F}(\mathfrak{x}^k), \mathfrak{f}(\mathfrak{m}^k))\} \cap \mathfrak{x}^k$; $\tilde{\mathfrak{m}}^{k+1} \in \tilde{\mathfrak{x}}^{k+1}$;
    **If** no auxiliary steps are necessary,
    **then** $\mathfrak{x}^{k+1} := \tilde{\mathfrak{x}}^{k+1}$; $\mathfrak{m}^{k+1} := \tilde{\mathfrak{m}}^{k+1}$
    **else** $\mathfrak{z}^0 := \tilde{\mathfrak{x}}^{k+1}$; $\tilde{\mathfrak{m}}^0 := \tilde{\mathfrak{m}}^{k+1}$; $i := 0$;
        **repeat**
            $j := 1(1)N$
            $Z_j^{(i+1)} := \left\{\tilde{m}_j^{(i)} - \left(-\sum_{r=1}^{j-1} l_{jr} Z_r^{(i+1)} - \sum_{r=j+1}^{N} u_{jr} Z_r^{(i)} + d_j \tilde{m}_j^{(i)} + \phi_j(\tilde{m}_j^{(i)})\right) \Big/ (d_j + \phi_j'(X_j^{(k)}))\right\} \cap Z_j^{(i)}$; $\tilde{m}_j^{(i+1)} \in Z_j^{i+1}$;
            $i := i+1$
        **until** ($i > g$ **and** $d(\mathfrak{z}^i) < \alpha d(\mathfrak{x}^k)$) **or** $\|d(\mathfrak{z}^i)\|_\infty < \text{eps}$;
        $\mathfrak{x}^{k+1} := \mathfrak{z}^i$; $\mathfrak{m}^{k+1} := \tilde{\mathfrak{m}}^i$.
```

Computing $\tilde{\mathfrak{x}}^{k+1}$ as in ING, we obtain the method INGHIN.[1] The next theorem concerns inclusion and convergence properties of INBHIN.

THEOREM 5.3. *Given problem* (2.2) *and* (5.1), *the following assertions are valid for the sequence* $(\mathfrak{x}^k)_{k \in \mathbb{N}_0}$ *computed by* INBHIN *with the starting vector* $\mathfrak{x}^0$:

1) $\mathfrak{y} \in \mathfrak{x}^k \quad \forall k \in \mathbb{N}_0$,

2) $\mathfrak{x}^k \to \mathfrak{y} \quad (k \to \infty)$.

The $\mathfrak{m}^k \in \mathfrak{x}^k (k \in \mathbb{N}_0)$ *are arbitrary.*

Proof. Assertion 1. Because of (5.1) we have $\mathfrak{y} \in \mathfrak{x}^0$. We assume that for a $k \in \mathbb{N}_0$ we have shown $\mathfrak{y} \in \mathfrak{x}^i$ for all $i \leqq k$. Hence $\mathfrak{y} \in \tilde{\mathfrak{x}}^{k+1}$ by (5.11). If no auxiliary steps are necessary in the $(k+1)$th step, we have $\mathfrak{x}^{k+1} = \tilde{\mathfrak{x}}^{k+1}$, thus $\mathfrak{y} \in \mathfrak{x}^{k+1}$.

[1] See list of abbreviations at end of paper.

Otherwise we note that $\boldsymbol{y} \in \mathfrak{z}^0 = \tilde{\mathfrak{x}}^{k+1}$. Then, because of $\tilde{\boldsymbol{m}}^0, \boldsymbol{y} \in \mathfrak{z}^0$, there is a vector $\boldsymbol{u} \in \mathfrak{z}^0$ such that

$$\boldsymbol{f}(\tilde{\boldsymbol{m}}^0) = \boldsymbol{f}'(\boldsymbol{u})(\tilde{\boldsymbol{m}}^0 - \boldsymbol{y}),$$

and we conclude step by step

(5.15) $j := 1(1)N$

$$y_j = \tilde{m}_j^{(0)} - \left(-\sum_{r=1}^{j-1} l_{jr} y_j - \sum_{r=j+1}^{N} u_{jr} y_j + d_j \tilde{m}_j^{(0)} + \phi_j(\tilde{m}_j^{(0)}) \right) \Big/ (d_j + \phi_j'(u_j))$$

$$\in \left\{ \tilde{m}_j^{(0)} - \left(-\sum_{r=1}^{j-1} l_{jr} Z_r^{(1)} - \sum_{r=j+1}^{N} u_{jr} Z_r^{(0)} + d_j \tilde{m}_j^{(0)} + \phi_j(\tilde{m}_j^{(0)}) \right) \Big/ (d_j + \phi_j'(X_j^{(k)})) \right\} \cap Z_j^{(0)} = Z_j^{(1)};$$

thus $\boldsymbol{y} \in \mathfrak{z}^1, \tilde{\boldsymbol{m}}^1 \in \mathfrak{z}^1 \subseteq \mathfrak{z}^0 = \tilde{\mathfrak{x}}^{k+1}$. If $\boldsymbol{y} \in \mathfrak{z}^i$ for an $i \in \mathbb{N}$, then we replace in (5.15) 0 by i and get $\boldsymbol{y} \in \mathfrak{z}^{i+1}$ in view of

$$\boldsymbol{y} \in \mathfrak{z}^i \subseteq \mathfrak{z}^{i-1} \subseteq \cdots \subseteq \mathfrak{z}^0 = \tilde{\mathfrak{x}}^{k+1} \subseteq \mathfrak{x}^k.$$

This finally means that $\boldsymbol{y} \in \mathfrak{x}^{k+1}$.

Assertion 2. If $d(\tilde{\mathfrak{x}}^{k+1}) < \alpha d(\mathfrak{x}^k)$ for all $k \in \mathbb{N}_0$, then $\mathfrak{x}^{k+1} = \tilde{\mathfrak{x}}^{k+1}$; thus

$$d(\mathfrak{x}^{k+1}) < \alpha d(\mathfrak{x}^k) \quad \forall k \in \mathbb{N}_0. \tag{5.16}$$

Otherwise there exists a $k \in \mathbb{N}_0$ such that auxiliary steps have to be added with $\mathfrak{z}^0 = \tilde{\mathfrak{x}}^{k+1}$. Then we show that

$$\begin{aligned} d(\mathfrak{z}^{i+1}) &\leqq (\mathcal{D} + \underline{\Phi'(\mathfrak{x}^k)})^{-1}(\mathcal{L} + \mathcal{U}) d(\mathfrak{z}^i) \\ &\leqq (\mathcal{D} + \underline{\Phi'(\mathfrak{x}^k)})^{-1}(\mathcal{L} + \mathcal{U}) d(\mathfrak{x}^k). \end{aligned} \tag{5.17}$$

Since $\mathfrak{z}^i \subseteq \mathfrak{x}^k \subseteq \mathfrak{x}^0$ we further show with (1.3) and (1.9) (see [17, p. 256]) that

$$\rho((\mathcal{D} + \underline{\Phi'(\mathfrak{x}^k)})^{-1}(\mathcal{L} + \mathcal{U})) \leq \rho(\mathcal{D}^{-1}(\mathcal{L} + \mathcal{U})) = \cos \frac{\pi}{m+1} < 1 \quad \forall k, m \in \mathbb{N}_0. \tag{5.18}$$

The diagonal elements of $\boldsymbol{f}'(\mathfrak{x}^0)$ have the form

$$D_{ii}(\mathfrak{x}^0) = 4 + h^2 f_u(jh, ih, X_{ij}^{(0)}).$$

Since $f_u \geqq 0$, this means $0 \notin D_{ii}(\mathfrak{x}^0)\ \forall i \in \{1, \cdots, N\}$. By arguments very similar to those in [2, § 22, proof of Thm. 1], we now obtain $\mathfrak{z}^i \to \boldsymbol{y}$ $(i \to \infty)$, hence

$$\exists i \in \mathbb{N}_0 \quad \forall j \in \mathbb{N}_0, \quad j \geqq i: \qquad d(\mathfrak{z}^j) < \alpha d(\mathfrak{x}^k). \tag{5.19}$$

With $j := \max\{g, i\}$ we can write $\mathfrak{x}^{k+1} := \mathfrak{z}^j$ and conclude that

$$d(\mathfrak{x}^{k+1}) < \alpha d(\mathfrak{x}^k). \tag{5.20}$$

(5.16) and (5.20) show that

$$d(\mathfrak{x}^{k+1}) < \alpha d(\mathfrak{x}^k) \quad \forall k \in \mathbb{N}_0. \tag{5.21}$$

Because $\alpha < 1$, this means $d(\mathfrak{x}^k) \to \boldsymbol{o}$ $(k \to \infty)$ and finally, as $\boldsymbol{y} \in \mathfrak{x}^k$ for all $k \in \mathbb{N}_0$, $\mathfrak{x}^k \to \boldsymbol{y}$ $(k \to \infty)$. □

The subset property (1.3) allows us to prove

$$\text{IBU}\,(\hat{\mathcal{F}}(\mathfrak{x}), \boldsymbol{f}(\boldsymbol{m})) \subseteq ((\underline{\hat{\mathcal{F}}(\mathfrak{x})})^{-1} - (\overline{\hat{\mathcal{F}}(\mathfrak{x})})^{-1}) \boldsymbol{f}(\boldsymbol{m}) \tag{5.22}$$

for all $\mathscr{x} \subseteq \mathscr{x}^0$ with $\mathscr{y} \in \mathscr{x}$ and a vector $\mathscr{m} \in \mathscr{x}$. We use (5.22) to prove for INBHIN the inequality

$$(5.23) \qquad d(\mathscr{x}^{k+1}) \leqq ((\underline{\hat{\mathscr{F}}(\mathscr{x}^k)})^{-1} - (\overline{\hat{\mathscr{F}}(\mathscr{x}^k)})^{-1})|\mathscr{f}'(\mathscr{x}^k)|d(\mathscr{x}^k).$$

Choosing α so close to 1 that

$$1 > \alpha \geqq \rho((\underline{\mathscr{D}(\mathscr{y})} - \mathscr{L})^{-1}\mathscr{U}),$$

we obtain the following bound for the asymptotic convergence rate of INBHIN:

$$(5.24) \qquad R_1(\text{INBHIN}, \mathscr{y}) \leqq \min\{\rho(((\underline{\hat{\mathscr{F}}(\mathscr{y})})^{-1} - (\overline{\hat{\mathscr{F}}(\mathscr{y})})^{-1})|\mathscr{f}'(\mathscr{y})|), \alpha\} < 1.$$

The above discussion concerns the solution of (2.2). We can apply INBHIN and another method of this kind not presented in this paper to more general problems, for example general rectangles instead of $[0, 1] \times [0, 1]$, different mesh sizes in the x- and y-directions, three-dimensional problems, or under suitable conditions $(a(x)u_x)_x + (b(y)u_y)_y = f(x, y, u(x, y))$. INBHIN profits from the fact that IBU can be applied to matrices with $\mathscr{T} = \alpha\mathscr{I}, \alpha \in \mathbb{R}$. Generalizations of this kind will be treated in a forthcoming paper.

6. Implementation of the methods. We want to compare the following methods numerically: INB (respectively INBHIN), ING (respectively INGHIN), INESVKD (as an example for a Gauss–Seidel type method), the nonlinear block SOR method (NBSOR) and the generalized conjugate gradient method (GCG). The numerical experiments were carried out on the CYBER 175 of the Wissenschaftliches Rechenzentrum Berlin (WRB)—cycle time: 25 ns, storage available for computations: 128 K 60-bit words with a 48-bit mantissa—in PASCAL (compiler version 2.2 of the ETH Zürich).

Due to the lack of an interval arithmetic at the machine level of the CYBER 175 of the WRB, the interval arithmetic had to be simulated by PASCAL procedures at the program level. This caused a considerable increase of computation time for the interval methods. Each interval-arithmetic operation involves a procedure call, a number of extra arithmetic operations, storage operations and branchings to simulate the correct rounding. Numerical experiments for the PASCAL 2.2 compiler of the CYBER 175 showed that one interval-arithmetic operation $(+, -, \times, /)$ requires ten to seventeen times the computation time of the corresponding floating-point operation. This fact, however, should not be a reason to reject immediately the use of interval methods. Machines are already available with a suitable arithmetic and compilers avoiding this situation. At the universities of Karlsruhe and Kaiserslautern in West Germany, for example, PASCAL and FORTRAN compilers have been developed, together with a suitable machine arithmetic, for several computers, that offer the necessary facilities for interval arithmetic [13].

Interval operations are implemented including a correctly rounding arithmetic. As the interval bounds do not have to be addressed and rounded separately, an interval operation is executed even faster than two corresponding floating-point operations. Such an arithmetic can be developed for every microprogrammable machine. A fair comparison with "ordinary" floating-point methods should take account of this fact.

The procedures for the interval operations (see [2, App. B]) are programmed in such a way that the computation time is reduced as much as possible. We do not use a rounding function in order to avoid unnecessary function calls and to be able to round individually in each case. The above mentioned compiler requires the rounding B1 from [2, App. B]. Because of the rounding, machine- and compiler-dependent numbers enter into the programs. It is possible to enter these numbers in octal format on the CYBER 175. We use a procedure GLOB to do this. The numbers are

MINPOS := 0001 4000 0000 0000 0000B

smallest machine number a with $a+a>0$;

FAKTORB := 1717 7777 7777 7777 7777B

largest machine number <1;

FAKTORC := 1720 4000 0000 0000 0001B

smallest machine number >1.

As we often need the number π, we use an octal inclusion $\underline{\pi}<\pi<\bar{\pi}$ with

$\underline{\pi}$:= 1721 6220 7732 5042 0550B, $\bar{\pi}$:= 1721 6220 7732 5042 0551B.

A data type INTERVALL simplifies the computations with intervals: INTERVALL = RECORD U, O: REAL END. The number of point operations in the sometimes complicated procedures for interval-arithmetic operations depends on the interval operands and cannot be predicted. Therefore if we want to compare several interval methods with each other and also include floating-point methods in our discussion, then the only meaningful criterion of comparison that remains is the compution time.

On the CYBER 175 there is a system functions CLOCK to measure the computation time. We do not consider I/O-time. The programming of the discussed methods benefits from the known structure of the systems of equations to be solved. The programs can be written in such a way that only the really necessary operations take place. $f'(x^0)$, $\hat{\mathcal{F}}(x^0)$ and $\mathcal{A}(x^0)$ being interval M-matrices, we can apply special interval procedures requiring that the signs of some interval bounds be known in advance.

Numerical experiments have shown, for example, that the reduction method IBU using these procedures needs only about eight times the computation time of the well-known Buneman algorithm using floating-point arithmetic.

In view of the above-mentioned problems concerning interval arithmetic, it seems reasonable to divide the observed computation times for the interval methods by at least a factor of four in order to obtain a realistic comparison with floating-point methods.

As a convergence criterion for the floating-point methods we use (see [12])

$$\|x^{k+1}-x^k\|_\infty<10^{-6}. \tag{6.1}$$

For interval methods we use

$$\|q(x^{k+1},x^k)\|_\infty=\max\{\|\overline{x^{k+1}}-\overline{x^k}\|_\infty,\|\underline{x^{k+1}}-\underline{x^k}\|_\infty\}<10^{-6} \tag{6.2}$$

to compare them with the point methods. Note that under the condition $x^k\to y$ $(k\to\infty)$ we have $q(x^{k+1},x^k)\to o$ $(k\to\infty)\Leftrightarrow d(x^{k+1})\to o$ $(k\to\infty)$ (see [2, App. A, Lemma 1]).

The second part of § 6 contains details of the methods applied in this paper and of their programming.

GCG. We programmed several versions of the generalized conjugate gradient method following the description in [7, p. 324–6]. The programs GCG I+II do not include convergence safeguards; i.e., no line searches and no restarts take place. This reduces the computation time, although convergence cannot be guaranteed. In order to compare INB and GCG directly, we solve the linear systems occurring in the GCG method by the Buneman algorithm, while INB uses the reduction method IBU. Other methods were possible for GCG as well as for INB. In view of the Poisson equation to be discussed here, they should not substantially change the results of the comparison between INB and GCG.

The programs GCG I and GCG II correspond to INB with IBUM I and IBUM II (see the description of INB). GCG II stores intermediate results from the reduction part of the Gauss algorithm for tridiagonal matrices used in the Buneman algorithm; in GCG I this Gauss algorithm is always entirely carried out. The program GCG II+ RESTART modifies GCG II by periodically restarting the iteration, carrying out one line search per cycle with the Newton method. During each iteration the convergence safeguards given by Lemma 2 and 3 in [7] are verified.

NBSOR. NBSOR is programmed as described in [12]. We apply a special Gauss algorithm for tridiagonal matrices of the form $(-1, 4+d_i, -1)_m$; $d_i \geqq 0$, $i \in \{1, \cdots, m\}$, in the Newton steps (compare the description of INB).

HINESVKD. To carry out auxiliary steps with HINESVKD in the $(k+1)$st main step of INBHIN or INGHIN, we use the diagonal part $\mathcal{D}(x^k)$ of $\mathcal{f}'(x^k)$ for all auxiliary steps relative to x^k and store $\mathcal{D}(x^k)$ in a vector of the dimension $[1 \cdots N] \times$ INTERVALL. For ING we take the version (3.2) of IGA for band matrices. $\mathcal{f}'(x^0)$ being an interval M-matrix, we can consider known signs for the bounds of the matrix coefficients.

INB. The implementation of INB requires more detailed explanations. The $m-1$ numbers $2\cos((2l-1)\pi/2^{r+1})$, $1 \leqq l \leqq 2^r$, $1 \leqq r \leqq k$, are independent of the iteration process. We compute them before the iteration loop by an interval version ICOS of the cosinus and store them in a $[1 \cdots m-1] \times$ INTERVALL vector.

We construct the matrix $\mathcal{A}(x^k)$ by (5.7). The matrix $\mathcal{f}'(x^k)$ for the problem (2.2) is constant except for the diagonal part. Therefore the transformation $\mathcal{f}'(x^k) \to \hat{\mathcal{F}}(x^k)$ concerns only the diagonal part $\mathcal{D}(x^k)$. As we do not need the matrix $\mathcal{D}(x^k)$ again in INB, we do not want to store it, and we proceed as follows to carry out (5.7):

```
(6.3)  $j := 1(1)m$
           compute $A_{jj1}(x^k)$;
           $A_{jj}(x^k) := A_{jj1}(x^k)$
       $l := 2(1)m, j := 1(1)m$
           compute $A_{jjl}(x^k)$;
           $\underline{A_{jj}(x^k)} := \min\{\underline{A_{jj}(x^k)}, \underline{A_{jjl}(x^k)}\}$
           $\overline{A_{jj}(x^k)} := \max\{\overline{A_{jj}(x^k)}, \overline{A_{jjl}(x^k)}\}$.
```

We store the intervals $A_{jj}(x^k)$ in a $[1 \cdots m] \times$ INTERVALL vector. We thus only have to know $m = \sqrt{N}$ coefficients if we apply IBU to the matrix $\hat{\mathcal{F}}(x^k)$. Note that $m = p = q$ in (2.2).

The simple structure of the matrices $\mathcal{G}_j^{(r)}$ appearing in IBU in the problem (2.2) enables us to simplify IGA in (3.2). Let

$$x := \text{IGA}(\mathcal{A}, x)$$

where $\mathscr{A} := (-1, D_i, -1)_m$, $\underline{D_i} \geqq 2$ $(1 \leqq i \leqq m)$, is an interval M-matrix. We replace (3.2) by

```
(6.4) TRIGAM.
      $k := 2(1)m$
          $D_k := D_k - 1/D_{k-1}$
          $X_k := X_k + X_{k-1}/D_{k-1}$
      $X_m := X_m/D_m$
      $k := m-1(-1)1$
          $X_k := (X_k + X_{k+1})/D_k$.
```

In some steps IBU requires that IGA be applied several times with the same matrix but with different right-hand sides. Therefore we compute in a procedure TRIREM an auxiliary $[1 \cdot\cdot m] \times$ INTERVALL vector containing the new D_k's of the reduction part which are independent of the right-hand side. The two vectors with the old and the new D_k's will then be used to carry out in another procedure TGASTF all operations concerning the right-hand sides.

The vectors $\mathscr{y}$, $\mathscr{x}$, $\mathscr{p}_j^{(r)}$, $\mathscr{q}_j^{(r)}$—see(4.5)—do not appear simultaneously in the computations of IBU. We define a $[1 \cdot\cdot m] \times [1 \cdot\cdot m] \times$ INTERVALL vector $\mathscr{w}$ first to transfer $\mathscr{y}$ in IBU and later to leave IBU with $\mathscr{x}$. We also need a $[1 \cdot\cdot (m-1)/2, 1 \cdot\cdot m] \times$ INTERVALL auxiliary vector $\mathscr{z}$. The following table illustrates for $k=2$, hence $m=7, N=49$, the values of $\mathscr{w}$ and $\mathscr{z}$ in the course of the execution of IBU. "–" means void, "×" no change compared to the previous line.

TABLE 1
Memory organization for IBU.

$\mathscr{w}_1$	$\mathscr{w}_2$	$\mathscr{w}_3$	$\mathscr{w}_4$	$\mathscr{w}_5$	$\mathscr{w}_6$	$\mathscr{w}_7$	$\mathscr{z}_1$	$\mathscr{z}_2$	$\mathscr{z}_3$	r	
$\mathscr{y}_1$	$\mathscr{y}_2$	$\mathscr{y}_3$	$\mathscr{y}_4$	$\mathscr{y}_5$	$\mathscr{y}_6$	$\mathscr{y}_7$	–	–	–	0	Begin of IBU
×	$\mathscr{q}_2^1$	×	$\mathscr{q}_4^1$	×	$\mathscr{q}_6^1$	×	$\mathscr{p}_2^1$	$\mathscr{p}_4^1$	$\mathscr{p}_6^1$	1	Reduction part $r=1$
×	×	×	$\mathscr{q}_4^2$	×	×	×	×	$\mathscr{p}_4^2$	×	2	Reduction part $r=2(1)k$
×	×	×	$\mathscr{x}_4$	×	×	×	×	×	×	2	Solution part $r=k(-1)1$
×	$\mathscr{x}_2$	×	×	×	$\mathscr{x}_6$	×	×	×	×	1	
$\mathscr{x}_1$	×	$\mathscr{x}_3$	×	$\mathscr{x}_5$	×	$\mathscr{x}_7$	×	×	×	0	Solution part $r=0$

We define two versions, IBUM I and IBUM II, constructed only for problem (2.2) and differing by the application of TRIGAM, TRIREM and TGASTF. In IBUM I we apply TRIREM once and then TGASTF m times in the reduction part for $r=1$ and the solution part for $r=0$. For $r=2, \cdots, k$ in the reduction part and for $r=1, \cdots, k$ in the solution part we apply TRIGAM $(k-1)m+k+1$ times. The matrices $\mathscr{G}_j^{(r)}$ have to be recomputed for each application of TRIGAM. However, this requires only m interval additions per matrix. In IBUM II we compute once, using TRIREM, all intervals needed for the solution parts of IGA for $r=2, \cdots, k$ in the reduction part and for $r=1, \cdots, k$ in the solution part of IBU. This requires an additional auxiliary $[1 \cdot\cdot (m-3)/2] \times [1 \cdot\cdot m] \times$ INTERVALL vector $\mathscr{ww}$:

$r := 1(1)k-1, j := 1(1)2^r$,

$$\mathscr{ww}_{jj} := \text{TRIREM}\,(\mathscr{G}_j^{(r)}) \quad \text{with } jj := j + \sum_{i=1}^{r-1} 2^i.$$

For $r \in \{1, \cdots, k-1\}$ only TGASTF has to be carried out. For $r = k$ there is—in both parts of IBU—only one right-hand side to be treated. Thus TRIGAM will be applied.

For the methods INGHIN and INBHIN with possible auxiliary steps, we need an additional $[1 \cdot\cdot N] \times$ INTERVALL vector containing the diagonal $\mathscr{D}(x^k)$ of $f'(x^k)$. In executing INBHIN, the intervals $\mathscr{A}_{jjl}(x^k)$ of $\mathscr{D}(x^k)$ appear while we compute $\mathscr{A}(x^k)$. The initial inclusion x^0 of the solution y of (2.2) can be determined using [14, Thm. 13.4.6, part c]:

$$\overline{x}^0 := \mathcal{M}^{-1}|\Phi(o)|, \qquad \underline{x}^0 := -\overline{x}^0. \tag{6.5}$$

From $f_u \geqq 0$—see § 2—we conclude the isotony of f required in [14]. The point vector x^0 is computed by a point version PIBUM I of IBUM I without an interval arithmetic and stored in $\bar{x}$ where x is the $[1 \cdot\cdot N] \times$ INTERVALL vector containing the iterates x^k one after the other:

$$\bar{x} := |\Phi(o)|, \quad \bar{x} := \text{PIBUM I}(\mathcal{M}, \bar{x}), \quad x := [-\bar{x}, \bar{x}].$$

PIBUM I requires an additional $[1 \cdot\cdot (m-1)/2, 1 \cdot\cdot m]$ vector z. Table 2 shows the storage needed by the methods we discuss. We consider only arrays of a minimum size of $m-1$. For INB we consider three different choices for the vectors m^k:

$$\begin{aligned} &\text{version } M: \quad m^k := m(x^k) \\ &\text{version } O: \quad m^k := \overline{x}^k \qquad \text{for all } k \in \mathbb{N}_0. \\ &\text{version } U: \quad m^k := \underline{x}^k \end{aligned}$$

TABLE 2
Storage requirement.

		$m = 63$	$m = 127$
GCG I (also with restarts)	$6.5N + 9.5m$	26,397	106,045
GCG II (also with restarts)	$7N + 8m$	27,405	110,109
NBSOR	$3N + 16m$	12,915	50,419
INESVKD	$2N + 10m$	8,568	33,528
ING	$(4m+7)N + 10m$	1,028,601	8,307,705
INGHIN	$(4m+9)N + 10m$	1,036,539	8,339,963
INB + IBUM I M	$6N + 19m$	25,011	99,187
INB + IBUM I O/U	$5N + 19m$	21,042	83,058
INB + IBUM II M	$7N + 16m$	28,791	114,935
INB + IBUM II O/U	$6N + 16m$	24,822	98,806
INBHIN + IBUM I	$8N + 19m$	32,949	131,445
INBHIN + IBUM II	$9N + 16m$	36,729	147,193

Table 3 gives the number of interval-arithmetic operations per iteration step for interval methods and the number of floating-point operations per iteration step for the GCG method. These numbers do not include convergence tests and auxiliary steps, whose number cannot be predicted (i.e., for the interval methods with HINESVKD, GCG with restarts and NBSOR, the latter always use inner iterations). Neither do they include the problem-dependent values for the evaluation of functional expressions. Despite the fact that on modern computers the computation time is, in most cases, not proportional to the number of arithmetic operations—even if the different kinds

TABLE 3
Total number of arithmetic operations per iteration step (INBHIN, INGHIN *and* GCG *without auxiliary steps; all methods without tests of convergence criteria*).

		$m=63$	$m=127$
INESVKD	$11N-4$	43,655	177,415
HINESVKD	$9N-4$	35,717	172,161
ING (HIN)	$2N^2+(\frac{11}{3}m+\frac{15}{2})N-\frac{13}{6}m-4$	32,452,388	527,920,788
INB (HIN) + IBUM I	$(7k+14)N-(6k-4)m-(5k+11)$	192,807	899,119
INB (HIN) + IBUM II	$(4k+20)N-(7k+6)m-(3k+7)$	156,155	703,555
GCG I with a_1, b_1	$(7k+26)N-6km-(5k+11)$	240,183	1,092,164
GCG II with a_1, b_1	$(4k+32)N-(7k+10)m-(3k+7)$	203,531	896,595

of operations are counted separately—and in spite of the strong variations in computation time of the various procedues for the simulation of interval operations, this table may serve as a first orientation.

7. Numerical results. We have chosen the following examples:

a$_i$) $f(s,t,u)=10^i u^3/(1+s^2+t^2)$, $(s,t)\in I^0$, $u\in\mathbb{R}$, $i=-4(1)4$

$$u(s,t)=\begin{cases}1, & s=0,\ t\in[0,1] \text{ or } t=0,\ s\in[0,1],\\ 2-e^s, & t=1,\ s\in[0,1],\\ 2-e^t, & s=1,\ t\in[0,1].\end{cases}$$

b) $f(s,t,u)=u^3,\quad (s,t)\in I^0,\ u\in\mathbb{R},$

$u(s,t)=0,\quad (s,t)\in\Gamma.$

c) $f(s,t,u)=0,\quad (s,t)\in I^0,\ u\in\mathbb{R},$

$u(s,t)=0,\quad (s,t)\in\Gamma.$

d) $f(s,t,u)=e^u,\quad (s,t)\in I^0,\ u\in\mathbb{R},$

$u(s,t)=s+2t,\quad (s,t)\in\Gamma.$

Example a$_0$) can be found in [2, § 22], Examples b)–d) in [12] and [15].

The boundary functions are continuous and we have $f_u \geqq 0$ on $I\times\mathbb{R}$. Thus (0.1) and (2.2) have unique solutions. From § 2 we conclude $\underline{\Phi'(x^0)} \geqq \mathcal{O}$—see (5.1)—and $\underline{A_{jj}(x^0)} \geqq 0$ for the diagonal coefficients

$$A_{jj}(x^0)=4+h^2[\min\{\underline{f_u(jh, ih, X_{ij}^{(0)})}|i\in\{1,\cdots,m\}\},$$
$$\max\{\overline{f_u(jh, ih, X_{ij}^{(0)})}|i\in\{1,\cdots,m\}\}]$$

of $\hat{\mathcal{F}}(x^0)$. We we can summarize:

1) $f'(x^0)$ is an interval M-matrix, $\hat{\mathcal{F}}(x^0)$ is an interval L-matrix;

2) $\mathcal{M} \leqq \overline{f'(x^0)} \leqq \underline{\hat{\mathcal{F}}(x^0)}$.

It follows by (1.2) that $\hat{\mathcal{F}}(x^0)$ is also an interval M-matrix, an important condition for IBU in INB and INBHIN. The convergence properties of NBSOR have been investigated in [12], those of GCG in [7]. The convergence of INESVKD (and also of HINESVKD) to y follows from [2, § 22, Thm. 1], considering the proof of assertion 2 in Theorem 5.3.

For all interval methods we chose $m^k := m(x^k)$ for all $k\in\mathbb{N}_0$. INB was also computed with $m^k := \overline{x}^k$ for all $k\in\mathbb{N}_0$ (version O) and $m^k := \underline{x}^k$ for all $k\in\mathbb{N}_0$ (version

TABLE 4
Numerical results for the examples a_1, a_2, a_{-3}, b, d
(*upper rows*: *computation time*, *lower rows*: *number of iteration steps*).

Example		a_0		a_1		a_{-3}		b		d	
	m	63	127	63	127	63	127	63	127	63	127
NBSOR		173	326	158	313	176	326	204	342	186	338
		174	1,000	137	1,040	138	1,030	246	1,700	243	1,820
INESVKD*		3,600	12,000	3,100	10,500	3,700	13,000	3,800	13,000	3,400	12,000
		1,000	15,000	880	12,600	1,040	15,000	720	10,000	700	10,000
ING*		4	4	5	5	3	3	4	4	4	4
		400	6,400	520	8,400	300	5,200	400	6,600	400	6,600
GCG I	a_1, b_1	7	6	16	15	3	3	8	7	7	7
		5.0	18.7	11.5	46.8	2.2	9.4	5.6	21.6	5.4	23.3
GCG II	a_1, b_1	7	6	16	15	3	3	8	7	7	7
		4.6	16.8	10.6	41.7	2.0	8.4	5.2	19.3	5.0	21.1
	a_2, b_2	7	7	13	11	4	4	6	6	7	7
		5.2	21.7	9.6	34.2	2.9	12.3	4.3	18.3	5.5	23.2
	a_1, b_2	21	16			9	7	95	30	13	12
		15.4	49.2	>150	>150	6.6	21.4	68.5	91.3	10.1	39.1
GCG II+	a_1, b_1	7	6	11	10	3	3	7	7	8	8
Restart 5		7.4	26.3	11.9	43.7	3.2	13.7	7.3	30.2	8.9	37.4
	a_2, b_2	7	7	12	12	3	3	5	6	8	8
		7.8	31.7	13.5	55.8	3.4	14.4	5.3	27.9	9.4	39.5
	a_1, b_2	7	7	13	12	3	3	7	7	9	9
		7.7	31.4	14.4	55.3	3.4	14.3	7.7	31.6	10.7	45.4
GCG II+	a_1, b_1	7	6	13	14	3	3	7	7	8	8
Restart 10		7.0	24.7	13.0	57.5	3.2	13.7	7.0	28.7	8.1	35.4
	a_2, b_2	8	7	16	15	3	3	5	6	8	9
		8.5	30.5	16.7	65.8	3.4	14.4	5.2	26.7	9.0	42.1
	a_1, b_2	7	8	22	13	3	3	9	9	13	13
		7.4	34.1	22.5	56.6	3.4	14.3	9.3	38.3	14.5	60.6
INB+	M	5	5	8	8	3	3	4	4	7	7
IBUM I		4.5	20.1	7.3	32.9	2.7	11.3	3.2	14.5	6.4	25.5
INB+	M	5	5	8	8	3	3	4	4	7	7
IBUM II		3.9	17.2	6.3	28.0	2.4	10.7	2.7	12.5	5.5	22.6
	O	7	7	16	17	3	3	5	5	12	12
		5.4	24.1	14.3	59.6	2.4	10.6	3.4	15.8	8.4	38.8
	U	8	8	18	19	3	3	4	4	13	14
		6.2	27.5	15.8	66.5	2.4	10.7	2.8	12.3	9.1	44.9

U). $\mathscr{f}'(x^0)$ being an interval M-matrix, IGA can be carried out. Thus INGHIN converges to $\mathscr{y}$ by Theorem 5.3.

$\hat{\mathscr{F}}(x^0)$ is also an interval M-matrix. We further have the inequalities

$$(-1,2,-1)_m \leqq \underline{\mathscr{A}(x)} - 2\left\{\cos\left(\frac{2l-1}{2^{r+1}}\pi\right)\right\}\mathscr{I} \tag{7.1}$$

with $m=2^{k+1}-1$ for all $x \subseteq x^0$, $r \in \{1, \cdots, k\}$, $l \in \{1, \cdots, 2^r\}$. The eigenvalues $\lambda_i = 2(1-\cos(i\pi/(m+1)))$ of the matrix $\mathscr{C} := (-1, 2, -1)_m$ are positive. As from a positive definite symmetric L-matrix $\mathscr{C}$ is an M-matrix. From (1.2) and Theorem 4.1 we conclude that IBU can be applied to $\hat{\mathscr{F}}(x^0)$. The remarks concerning the convergence of INGHIN to y are also valid for INBHIN.

The examples were computed for $N \times N$-systems with $N = m^2$ and $m = 3, 7, 15, 31, 63, 127$ are far as the storage requirements could be fulfilled and the computation time was acceptable.

Due to the choice of simple functions f and g in our examples, we could use instead of (6.7) easily computed approximations for the starting vector x^0 required in the interval methods:

$x^0 := ([-1, 2])_{i=1}^N$ in examples a_1)–a_3),

$x^0 := (([-1, 3])_{i=1}^m, ([-1.5, 3])_{i=m+1}^N)$ in example d).

For examples b) and c) we chose the first vector. GCG and NBSOR were computed with $x^0 = (-1)_{i=1}^N, (2)_{i=1}^N, (50)_{i=1}^N$.

Tables 4 and 5 contain the number of iteration steps and the computation time in seconds needed to fulfill the convergence criterion. For the reasons we explained

TABLE 5
Numerical results for example a_i for several different values of i and $m = 63$.

	i	−4	−2	0	2	4
INB+ IBUM II	M	3 2.7	3 2.7	5 4.5	13 11.7	35 31.0
NBSOR		176 136	176 136	173 134	134 119	92 85.5
GCG II+ Restart 5	a_1, b_1	3 3.3	4 4.2	7 7.4	19 19.6	Overflow
	a_2, b_2	3 3.3	5 5.4	7 7.8	19 20.9	39 43.2
	a_1, b_2	3 3.4	5 5.4	7 7.7	22 24.8	Overflow
GCG II+ Restart 10	a_1, b_1	3 3.3	4 4.2	7 7.0	15 15.0	40 37.2
	a_2, b_2	3 3.3	5 5.4	8 8.5	29 29.8	46 48.0
	a_1, b_2	3 3.4	5 5.3	7 7.4	27 28.2	37 38.3
GCG II	a_1, b_1	3 2.0	4 2.7	7 4.6	40 26.3	112 74.0
	a_2, b_2	4 2.9	4 2.9	7 5.2	20 14.9	42 31.2
	a_1, b_2	7 5.1	11 8.0	21 15.4	>110	>200

in § 6, we divided the computation times for interval methods by four. In the tables we note the results only for $m=63$ and $m=127$, the other values of m being too small for practical problems. For GCG and NBSOR we note only the results for $x^0=(2)_{i=1}^N$. NBSOR does not seem to be very sensitive to changes in the starting vector. For GCG (and also INB) bad starting vectors increase considerably the computation time. There was no noticeable difference between the results for the first two starting vectors for GCG.

In the present examples we never needed auxiliary steps for INB. By cancelling the auxiliary method HINESVKD we saved $2N$ words of memory. This enabled us to solve larger systems. All versions of GCG were computed with the parameters (a_1, b_1), (a_2, b_1), (a_1, b_2), (a_2, b_2)—see [7]. (a_1, b_1) and (a_2, b_1) always needed the same number of iterations and the same computation time. GCG II+RESTART was tested with restarts after every 5 and every 10 iterations.

For the slower version GCG I we only note the results for (a_1, b_1), for INB with IBUM I only the results for the version M. The results for INESVKD and ING are estimated from the values for $m=7, 15, 31$ in view of an excessive storage requirement by ING and unacceptable computation times for both.

Table 5 demonstrates for $m=63$ the effect of increasing nonlinearity on NBSOR and the faster versions of GCG and INB. For example c), which we do not note in the tables, INB and GCG always needed the minimum number of two iterations.

We make the following observations:

1. Despite their quadratic convergence property, ING/INGHIN cannot be used for problems of this kind because of excessive storage and time requirements.

2. NBSOR and INESVKD need only a relatively small number of arithmetic operations per step; the moderate storage requirement is appealing, too. However, their asymptotic convergence speed is quite low. The asymptotic convergence rates can be estimated by

$$R_1(\text{INESVKD}, y) \leqq \cos\frac{\pi}{m+1} \qquad \text{(see 5.18)},$$

$$R_1(\text{NBSOR}, y) \leqq \frac{2-\cos(\pi/(m+1)) - 2\sqrt{1-\cos(\pi/(m+1))}}{2-\cos(\pi/(m+1)) + 2\sqrt{1-\cos(\pi/(m+1))}} \qquad \text{(see [16, § 4.5])}.$$

Both estimates tend to 1 for $m\to\infty$.

With strongly increasing nonlinearity the computation time of NBSOR decreases while that of INB and GCG increases. In examples of the kind discussed here NBSOR was always clearly slower than INB and GCG. NBSOR is not as sensitive as INB or GCG to modifications of the starting vector. Different choices of the starting vector confirm this fact. On the other hand, suitable starting vectors for INB and possibly also for GCG can be computed using [14, Thm. 13.4.6]. A starting vector near the solution does not guarantee a higher convergence speed for NBSOR.

Example a$_1$) with $m=15$ has a solution $y \in ([-0.1, 1])_{i=1}^N$,

$$x_0 := (2)_{i=1}^N \quad \text{computation time 19.8 s},$$

$$x_0 := (50)_{i=1}^N \quad \text{computation time 19.0 s}.$$

3. INB and INBHIN require for $m \geqq 31$ four or five times as many arithmetic operations as INESVKD and two or three times as much memory as INESVKD and NBSOR. However, the numerical results reveal a behavior similar to superlinear or

quadratic convergence. Due to the rules of interval arithmetic the estimate (5.24) is rather crude. It shows, however, that the asymptotic rate of convergence of INB and INBHIN tends to zero for $d(\hat{\mathscr{F}}(y)) \to \mathcal{O}$ and is zero in the linear case for all $m \in \mathbb{N}$ (example c). We add results for the choice $m^k := x^k \; \forall k \in \mathbb{N}_0$ and $m^k := \underline{x}^k \; \forall k \in \mathbb{N}_0$ because in these cases we do not have to store the vector m^k. This choice of m^k leads to numerical results worse than those for the midpoint vector.

On the other hand, this choice offers some advantages which we shall examine in another paper: a better approximation of the asymptotic rate of convergence than (5.24), no need at all for an auxiliary sequence to ensure the convergence to y.

The difference between the versions M and O/U grows with increasing diameter or the diagonal coefficients of $\hat{\mathscr{F}}(\cdot)$. The diameter increases with increasing nonlinearity, for example. With increasing diameters of the coefficients of $\hat{\mathscr{F}}(y)$ INB becomes slower (compare Table 5). For linear problems, INB reduces to a simple application of IBU. The results for example c), however, show the necessity of two steps, the second being necessary for the examination of the convergence criterion. NBSOR behaves like a linear block SOR method for systems with a small nonlinearity.

We computed example d) for several values of m with starting vectors determined by (6.7). In the worst case the computation of the starting vector required about 1% of the total computation time. Note that INB requires only a starting vector which includes the solution to ensure the convergence to the latter. As this vector is very easy to compute, we can even speak of global convergence.

4. The generalized CG method showed a behavior in a certain sense similar to that of INB. The versions GCG I and GCG II without restarts, i.e. without guaranteed convergence, need more floating-point operations per step than INB with IBUM I and IBUM II need interval-arithmetic operations per step. Even with an optimal microprogrammed or hardware interval arithmetic INB should be slower than the generalized CG method without restarts if both need the same number of iterations (examples a_{-4}), a_{-3}), d)).

In examples with a stronger nonlinearity (a_0), a_1), a_2), a_4), b)) the number of iteration steps for INB is sometimes considerably smaller than that for GCG. The difference increases with increasing nonlinearity. Despite the higher complexity of interval arithmetic, the smaller number of steps results in a shorter computation time for INB—provided a sufficiently fast arithmetic is available. The choice between the parameters (a_1, b_1), (a_1, b_2), (a_2, b_1), (a_2, b_2) for GCG (see [7]) is not obvious. (a_1, b_1) and (a_2, b_1), which produced the same results, seem to be the most reliable. The parameters (a_2, b_2) produced sometimes slightly better, but often considerably worse, results than the above ones. The parameters (a_1, b_2) gave the worst results. For strongly nonlinear functions (examples a_3), a_4)) divergence or an overflow occurred several times with (a_1, b_2) and (a_2, b_2); in example a_4) even for (a_1, b_1) and with restarts an overflow took place.

The comparison with GCG I and GCG II is a theoretical one as they do not guarantee convergence. A realistic comparison involves the version with periodical restarts and the convergence safeguards described in [7]. The convergence safeguards clearly increase the computation time per step. This affects, in particular, examples which need only a few iteration steps. For examples which do not necessarily converge without the safeguards the latter naturally have a positive effect on the computation time (example a_2), a_3), a_4)).

The computation time is also determined by the length of the restart cycle. This length has to be guessed from experience since there exists no rule for an optimal cycle length. The number of steps as well as the computation time (modified for INB) for GCG with restarts always remained higher than those for INB with IBUM II (version M).

5. In all the examples that we tested, INB never needed auxiliary steps to ensure convergence. The auxiliary method for INB has the advantage that it does not have to be carried out regularly. Auxiliary steps are executed only if the iteration process seems to stop on an interval vector whose diameter in each component is greater than the prescribed precision, which was never the case in our examples.

The version M of INB never needed a larger number of iteration steps than any of the GCG versions. The versions of GCG with convergence safeguards were always slower than INB+IBUM II (version M). Note that, whether necessary or not, the safeguards have to be carried out periodically, and a parameter control must be carried out in each step.

One drawback of GCG consists in the necessary choice of the optimal length of the restart cycle and the optimal choice of the step length formulas which are represented by the parameters (a_i, b_j), $i, j \in \{1, 2\}$.

6. For the range of applications of INB (which is certainly smaller than that of GCG), this interval method seems to be a good alternative to the well-known floating-point methods. In almost all of our examples INB was superior to NBSOR even with a simulated interval arithmetic. In order to compete with the generalized CG method, an optimal, or at least nearly optimal, arithmetic like the one developed in [13] is recommended.

7. Another advantage of INB results from the fact that the interval bounds are always rounded in the outside direction (seen from the interval): INB theoretically converges to the solution. Computed until the desired precision is obtained, the bounds of the final interval vector enable us to recognize immediately the correct digits.

Remarks. 1) Section 7 contains some characteristic results from [16]. A more detailed comparison and a PASCAL program can be found in [16].

2) Recently other interval-arithmetic algorithms have been developed which are based on Newton-SOR and Newton-ADI methods [8]. For examples of the type presented in this paper they seem to require more computation time than our method and need more storage. They are, however, applicable to more general equations, for example irregular regions, that do not lead to matrices of the special type required by our method.

Abbreviations for programs and algorithms.

GCG I, GCG II, GCG II + Restart	Generalized conjugate gradient method	[12, § 6]
HINESVKD	Modification of INESVKD as an auxiliary method	(5.14)
IBU	Reduction algorithm for interval matrices	(4.5)
IBUM I, IBUM II	Special versions of IBU for the Poisson equation	§ 6
IGA	Interval-arithmetic Gauss algorithm	(3.2)
INB	Interval-arithmetic Newton-like method with IBU	(5.10)
INBHIN	INB with the auxiliary method HINESVKD	(5.14)
ING	Interval-arithmetic Newton method	(5.5)
INGHIN	ING with HINESVKD	(5.14)
NBSOR	Nonlinear block successive overrelaxation method	[12, § 6]

Acknowledgment. I wish to thank Professor Dr. G. Alefeld for his helpful suggestions.

REFERENCES

[1] G. ALEFELD, *Über die Durchführbarkeit des Gaußschen Algorithmus bei Gleichungen mit Intervallen als Koeffizienten*, Computing, Suppl. 1 (1977), pp. 15–19.

[2] G. ALEFELD AND J. HERZBERGER, *Einführung in die Intervallrechnung*, Bibliographisches Institut, Mannheim-Wien-Zürich, 1974.

[3] L. BERS, F. JOHN AND M. SCHECHTER, *Partial Differential Equations*, Wiley Interscience, New York–London–Sydney, 1964.

[4] B. L. BUZBEE, G. H. GOLUB AND C. W. NIELSON, *On direct methods for solving Poisson's equations*, SIAM J. Numer. Anal., 7 (1970), pp. 627–656.

[5] O. BUNEMAN, *A compact noniterative Poisson solver*, Report 294, Institute for Plasma Research, Stanford Univ., Stanford, CA, 1969.

[6] B. L. BUZBEE, *Applications of fast Poisson solvers to the numerical approximation of parabolic problems*, Technical Report LA-4950-T, Los Alamos Scientific Laboratory, Los Alamos, NM, 1972.

[7] P. CONCUS, G. GOLUB AND D. O'LEARY, *Numerical solution of nonlinear elliptic partial differential equations by a generalized conjugate gradient method*, Computing, 19 (1978), pp. 321–339.

[8] H. CORNELIUS, *Untersuchungen zu einem intervallarithmetischen Iterationsverfahren mit Anwendungen auf eine Klasse nichtlinearer Gleichungssysteme*, Doctoral dissertation, Technische Universität Berlin, Berlin, 1981.

[9] K. FAN, *Topological proof for certain theorems on matrices with nonnegative elements*, Monatshefte Math., 62 (1958), pp. 219–237.

[10] W. HACKBUSCH, *On the fast solution of nonlinear elliptic equations*, Numer. Math., 32 (1979), pp. 83–95.

[11] ———, *Survey of convergence proofs for multi-grid iterations*, in Special Topics of Applied Mathematics, J. Frehse, D. Pallaschke and U. Trottenberg, eds., North-Holland, Amsterdam, 1980.

[12] L. A. HAGEMAN AND T. A. PORSCHING, *Aspects of nonlinear successive overrelaxation*, SIAM J. Numer. Anal., 12 (1975), pp. 316–335.

[13] U. KULISCH AND CH. ULLRICH, eds., *Wissenschaftliches Rechnen und Programmiersprachen*, German chapter of the ACM/Berichte 10, Teubner-Verlag, Stuttgart, 1982.

[14] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York and London, 1970.

[15] J. M. ORTEGA AND M. ROCKOFF, *Nonlinear difference equations and Gauss–Seidel-type iterative methods*, SIAM J. Numer. Anal., 3 (1966), pp. 497–513.

[16] H. SCHWANDT, *Schnelle fast global konvergente Verfahren für die Fünf-Punkt-Diskretisierung der Poissongleichung mit Dirichletschen Randbedingungen auf Rechteckgebieten*, Doctoral dissertation, Technische Universität Berlin, Berlin, 1981.

[17] J. STOER AND R. BULIRSCH, *Einführung in die Numerische Mathematik* II, Springer-Verlag, Berlin–Heidelberg–New York, 1976.

[18] W. TÖRNIG, *Monoton konvergente Iterationsverfahren zur Lösung nichtlinearer Differenzen-Randwertprobleme*, Beiträge Numer. Math., 4 (1975), pp. 245–257.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

COMPARISON OF SOME DIRECT METHODS FOR COMPUTING STATIONARY DISTRIBUTIONS OF MARKOV CHAINS*

W. J. HARROD† AND R. J. PLEMMONS†

Abstract. The purpose of this paper is to report on a comparison of an implementation of a simple direct LU factorization method, suggested by Funderlic and Mankin [SIAM J. Sci. Stat. Comput., 2 (1981), pp. 375–383], with other direct methods recommended by Paige, Styan and Wachter [J. Stat. Comput. Simul., 4 (1975), pp. 173–186], for computing stationary distributions of Markov chains. A backward error analysis is developed and conditioning problems are addressed. The method is stable without pivoting, it is numerically efficient and it lends itself to symmetric pivoting to preserve sparsity.

Key words. direct methods, error analysis, LU factorization, M-matrix, Markov chain, probability distribution vector, sparsity schemes, stochastic process

1. Introduction.

This paper is concerned with methods for computing the stationary distribution vector p of an ergodic Markov chain with probability transition matrix Q. Here Q is an $n \times n$ irreducible, row stochastic matrix and p is an n-vector with positive probabilities satisfying

$$Q^T p = p, \tag{1.1}$$

where T denotes the transpose. Since Q is an irreducible nonnegative matrix ($Q \geqq 0$), by the Perron–Frobenius theory (see, e.g., Varga [1962, p. 30] or Berman and Plemmons [1979, p. 27]), p is the unique positive vector satisfying $\sum_{i=1}^{n} p_i = 1$.

The computation of the stationary distributions of a Markov chain is of widespread interest (Kemeny and Snell [1960], Golub and Seneta [1973], Paige, Styan and Wachter [1975], Meyer [1975], W. J. Stewart [1978], G. W. Stewart [1980], Hunter [1982], Kemeny [1981]). More generally, the stationary distribution vector p satisfying (1.1) is important in many areas of the mathematical sciences, including queueing networks (e.g., Kaufman [1983]), input-output economic models (e.g., Berman and Plemmons [1979, Chap. 9] and Duchin and Szlyd [1979]) and in compartmental analysis tracer models (e.g., Sheppard and Householder [1951] and Funderlic and Mankin [1981]). Such computations are also related to the discrete Neumann problem in partial differential equations (e.g., Plemmons [1976]). Techniques for computing p where Q is large and sparse have traditionally been iterative and often involve some variation of the power method for computing eigenvectors (see, e.g., W. J. Stewart [1978]), although more recently methods based upon matrix splitting such as the Gauss–Seidel method have been investigated (see Kaufman [1983] for an interesting study of such methods). Combined direct-iterative methods have recently been considered by Koury, McAllister and Stewart [1984] and by Funderlic and Plemmons [1984]. Our interest here is in a direct method based upon an LU factorization of a certain M-matrix (see § 2 for definitions) by Gaussian elimination without pivoting.

Paige, Styan and Wachter [1975] have considered a variety of direct methods for computing the stationary distribution vector p for an ergodic Markov chain with rate transition matrix Q. They recommended two methods which are based upon the transformation of $I - Q^T$ into a nonsingular matrix by a rank-one modification,

* Received by the editors September 7, 1982. This research was supported in part by the U.S. Army Research Office under contract DAAG29-81-K-0132 and the National Science Foundation under grant MCS-82-19500.

† Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, North Carolina 27650.

followed by the solution of an associated nonsingular system of linear equations. Specifically letting $e^T = (1, \cdots, 1)$, it follows that if x is any n-vector such that $e^T x \neq 0$, that is, if x is not in the range of A, then

$$I - Q^T + x e^T \tag{1.2}$$

is nonsingular and p is then the solution to the nonsingular system of linear equations

$$(I - Q^T + x e^T)p = x. \tag{1.3}$$

We note that this rank-one modification concept has recently been extended to more general problems by Hunter [1982], Kemeny [1981] and Harrod [1982].

Based upon the numerical comparison of a collection of direct methods for computing p available at that time, Paige, Styan and Wachter [1975] recommended the use of the nonsingular system (1.3) with two possible choices of x. We name these methods as follows (here e_j denotes the jth unit vector):

PSW *method* 1. Solve

$$[I - Q^T + (Q^T e_j) e^T]p = Q^T e_j, \tag{1.4}$$

PSW *method* 2. Solve

$$[I - Q^T + e_j e^T]p = e_j. \tag{1.5}$$

Note that in PSW method 1, the vector x in (1.3) is the jth row of the stochastic matrix Q while in PSW method 2, x is the jth unit vector. Paige, Styan and Wachter recommended the solution of (1.4) or (1.5) by Gaussian elimination with row pivoting for numerical stability. Notice that these nonsingular rank-one modifications may produce a more dense matrix than $I - Q^T$, and also that row pivoting is now, in contrast to the Funderlic–Mankin scheme, generally required for the LU factorization of (1.2) since the column diagonal dominance of $I - Q^T$ can be lost (see § 2). We will remark further on these topics in § 4.

Other direct methods, such as replacing the jth row v of $I - Q^T$ by $e^T = (1, \cdots, 1)$ and then solving the resulting nonsingular system

$$[I - Q^T + e_j(e - v)^T]p = e_j,$$

were considered by Paige, Styan and Wachter. They also considered the use of generalized matrix inverses, but they found that PSW methods 1 and 2 performed best, in terms of accuracy and numerical efficiency, of all the methods they examined.

Our purpose in this paper is to provide a simple alternative to the nonsingular rank-one modification methods of Paige, Styan and Wachter. This method, which was suggested first by Funderlic and Mankin [1981] for the solution of compartmental tracer analysis problems, involves the direct LU decomposition of $I - Q^T$ or some symmetric permutation of $I - Q^T$. The algorithm is developed in § 2, and a complete backward error analysis is also given. The scheme essentially corresponds to one step of inverse iteration as described in Wilkinson [1965, p. 619] for a certain eigenvector problem. Numerical comparisons of this method with PSW methods 1 and 2 on a variety of test examples are reported in § 3. These examples involve some actual data from queueing network analysis and the problems vary in the degree of ill-conditioning and sparsity. Some general observations about the results of these comparisons are given in § 4 along with some comments on conditioning. We conclude that this direct LU factorization method for $I - Q^T$ provides at least as much accuracy as the nonsingular PSW methods 1 and 2. Moreover, this scheme requires no pivoting for numerical stability and facilitates the use of symmetric pivoting to preserve sparsity.

2. The partition factorization method. As defined in § 1, Q denotes an irreducible stochastic probability transition matrix of order n associated with an ergodic, finite Markov chain. Let

$$A = I - Q^T. \tag{2.1}$$

In this context, our purpose is to compute the unique stationary probability distribution vector $p = (p_i)$, $p_i > 0$, $\sum_{i=1}^n p_i = 1$, which solves the homogeneous system of linear equations

$$Ap = 0. \tag{2.2}$$

The coefficient matrix A has several useful properties. First, A is an M-matrix. By an *M-matrix* we mean a matrix with all nonpositive off-diagonal entries and with all eigenvalues having nonnegative real parts. Since A is irreducible, it has rank $n-1$. Moreover,

$$e^TA = e^T(I - Q^T) = e^T - e^TQ^T = e^T - e^T = 0,$$

so that A is column diagonally dominant. In particular,

$$a_{ii} = -\sum_{j \neq i} a_{ij}, \qquad 1 \leqq i \leqq n.$$

(For a discussion of the theory and applications of M-matrices in the mathematical sciences see, e.g., Berman and Plemmons [1979, Chaps. 6–10].) It follows that each principal submatrix A_k formed by deleting the kth row and kth column of A is a nonsingular M-matrix.

Although the coefficient matrix A in (2.1) is singular, (2.2) can always be solved in a stable way by Gaussian elimination on A, as described by Funderlic and Mankin [1981] or Funderlic and Plemmons [1981]. Indeed, A has an LU factorization

$$A = LU \tag{2.3}$$

where L is an M-matrix with unit diagonal and U is an M-matrix of rank $n-1$ with $u_{nn} = 0$. The following simple algorithm can thus be used to compute the stationary probability distribution vector p. It essentially corresponds to one step of the inverse power method with e_n as the right-hand side.

DIRECT FACTORIZATION METHOD.

1. Factor $A = LU$ by Gaussian elimination.

Let

$$\tilde{U} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{n-1,n} \\ & & & 1 \end{bmatrix}. \tag{2.4}$$

2. Solve $\tilde{U}y = e_n$ by back substitution.
3. Scale y to $p = (1/\sum_{i=1}^n y_i)y$.

It was shown in Funderlic, Neumann and Plemmons [1982] that the elements of A do not grow during the elimination phase of step 1 of the algorithm. In particular, we see that the growth factor

$$g_A \equiv \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|} = 1, \tag{2.5}$$

where $a_{ij}^{(k)}$ denotes the i, j entry of the unreduced part of A before the kth step of Gaussian elimination. Thus the method is stable. In particular, suppose the algorithm is executed on a machine with *unit roundoff error* μ in floating point arithmetic. Then a simple backward error analysis, using Wilkinson [1965, p. 215] and Stewart [1973, p. 150], shows that the computed solution $\hat{p}$ to $Ap=0$ satisfies exactly a perturbed homogeneous system of linear equations $(A+E)\hat{p}=0$. Bounds on E are given in the following theorem, which corrects Funderlic and Mankin [1981, eq. (8)], who inadvertently did not take into account the back substitution phase of the algorithm. The theorem is given in a slightly more general form than is necessary in the context of this paper.

THEOREM 1. *Let A be an $n\times n$ irreducible, column diagonally dominant singular M-matrix. Suppose the homogeneous systems of linear equations $Ax=0$ is solved for the positive vector x by Gaussian elimination on A without pivoting in floating point arithmetic with machine unit roundoff error μ. Then there exists a matrix $E=(e_{ij})$ such that the computed solution $\hat{x}$ to $Ax=0$ satisfies exactly the homogeneous system*

$$(A+E)\hat{x}=0 \tag{2.6}$$

where for $1\leqq i,\ j\leqq n$

$$|e_{ij}|\leqq \mu i(3.02+1.01n)\max_t a_{tt}. \tag{2.7}$$

Proof. Consider the LU decomposition of A by Gaussian elimination without pivoting. From Wilkinson [1965, p. 215], there is a matrix $F=(f_{ij})$ such that the computed factors $\hat{L}$ and $\hat{U}$ satisfy

$$A+F=\hat{L}\hat{U}$$

where

$$|f_{ij}|\leqq\begin{cases}2.01\mu(i-1)\max\limits_{i,j,k}|a_{ij}^{(k)}| & \text{if } j\geqq i,\\ 2.01\mu j\max\limits_{i,j,k}|a_{ij}^{(k)}| & \text{if } j<i.\end{cases}$$

Thus by combining these inequalities in terms of i and applying (2.5) and the fact that A is column diagonally dominant, we have

$$|f_{ij}|\leqq 2.01\mu i\max_t a_{tt}. \tag{2.8}$$

The algorithm for the LU decomposition of A by Gaussian elimination yields

$$\hat{U}=\begin{bmatrix}\hat{u}_{11} & \cdots & \hat{u}_{1n}\\ & \ddots & \vdots\\ & & \hat{u}_{n-1,n}\\ & & 0\end{bmatrix}.$$

Set

$$\hat{\hat{U}}=\begin{bmatrix}\hat{u}_{11} & \cdots & \hat{u}_{1n}\\ & \ddots & \vdots\\ & & \hat{u}_{n-1,n}\\ & & 1\end{bmatrix}.$$

Next, if $Ax=0$ is solved using $\hat{U}$ in the back substitution step 2, then by Stewart [1973, p. 150] there exists a matrix $G=(g_{ij})$ such that the computed solution $\hat{x}$ to $\hat{U}x=e_n$ satisfies the system

$$(2.9) \qquad (\hat{U}+G)\hat{x}=e_n$$

where, since $|\hat{u}_{ij}|\leqq \max_t a_{tt}$, for all i and j,

$$(2.10) \qquad |g_{ij}|\leqq 1.01\mu(n+1)\max_t a_{tt}.$$

Then multiplying (2.9) on the left by $\hat{L}$ we have

$$(\hat{L}\hat{U}+\hat{L}G)\hat{x}=\hat{L}e_n=e_n,$$

so since

$$\hat{L}\hat{U}=\hat{L}\hat{U}+e_ne_n^T$$

and $\hat{x}_n=1$, it follows that

$$(\hat{L}\hat{U}+\hat{L}G)\hat{x}=0.$$

Thus $\hat{x}$ satisfies the homogeneous system

$$(A+F+\hat{L}G)\hat{x}=0.$$

Let

$$E=F+\hat{L}G.$$

Then for each i and j, we have

$$|e_{ij}|\leqq |f_{ij}|+\sum_{k=1}^{i}|\hat{l}_{ik}||g_{kj}|$$

and, as $|\hat{l}_{ik}|\leqq 1$ for each i and k,

$$|e_{ij}|\leqq |f_{ij}|+\sum_{k=1}^{i}|g_{ij}|.$$

Then applying (2.8) and (2.10) we have

$$|e_{ij}|\leqq 2.01\mu i\max_t a_{tt}+1.01i(n+1)\max_t a_{tt},$$

which yields inequality (2.7). □

Observe that the matrix $I-Q^T$, where Q is an irreducible stochastic matrix, satisfies the hypotheses of the theorem. Moreover, $|a_{ij}|\leqq 1$ for each i and j. This establishes the following corollary, which provides a backward error analysis for the computation of stationary distributions by the direct factorization method.

COROLLARY 1. *Let Q be the rate transition matrix for an ergodic, finite Markov chain. Let $\hat{p}$ be the approximation to the stationary probability distribution vector p, computed by Gaussian elimination on $I-Q^T$ without pivoting, in precision μ floating point arithmetic. Then there exists a matrix E such that $\hat{p}$ satisfies the homogeneous system*

$$(I-Q^T+E)\hat{p}=0,$$

where for $1\leqq i, j\leqq n$,

$$|e_{ij}|\leqq \mu i(3.02+1.01n).$$

We proceed now to describe the general algorithm for computing stationary distributions to be compared in this paper with the PSW methods 1 and 2, which were described earlier. First we recall that it was established by Funderlic and Plemmons [1981] that any M-matrix A whose rows can be scaled to produce a column diagonally dominant matrix has the property that for each permutation matrix P, PAP^T has an LU factorization $PAP^T = LU$, where L is an M-matrix with unit diagonal and where U is an M-matrix. A converse of this result was given by Varga and Cai [1981].

As before, let Q be an $n \times n$, irreducible stochastic matrix and set $A = I - Q^T$. For any $1 \leqq k \leqq n$, let A_k denote the principal submatrix obtained by deleting the kth row and the kth column of A. Accordingly, let P denote the permutation matrix (introduced for notation purposes only) such that PAP^T has the partitioned form

$$PAP^T = \begin{bmatrix} A_k & y_k \\ z_k^T & \alpha_{nn} \end{bmatrix}. \tag{2.11}$$

Here A_k is an $(n-1)\times(n-1)$ diagonally dominant, nonsingular M-matrix and y_k is an $(n-1)$-dimensional vector with nonpositive entries.

The following general scheme for computing p is based upon the partition form (2.11).

Partition factorization method (PF method)

1. Choose $1 \leqq k \leqq n$ and partition A into the form (2.11).
2. Solve $A_k\tilde{x} = -y_k$ for $\tilde{x} = (x_1, \cdots, x_{n-1})^T$ by Gaussian elimination on A_k without pivoting.
3. Set $x_n = 1$ and $p = (1/\sum_{i=1}^n x_i)P^T x$.

Observe that since A_k is column diagonally dominant, pivoting for stability is unnecessary in the LU factorization of A_k by Gaussian elimination in step 2. Indeed the growth factor g_{A_k} is one for each $1 \leqq k \leqq n$. Also, observe that the partition factorization method with $k = n$ is mathematically the same as the direct factorization method applied to A, which was given earlier. To see this, note that if A_k is factored into $A_k = L_k U_k$ by Gaussian elimination, then U_k is the leading $(n-1)\times(n-1)$ principal submatrix of $\tilde{U}$, given by (2.4), while $L_k^{-1}y_k$ is the first $n-1$ entries of the last column of $\tilde{U}$. Thus the matrix $\tilde{U}$ given by (2.4) in the direct factorization method has the partitioned form

$$\tilde{U} = \begin{bmatrix} U_k & L_k^{-1}y_k \\ 0 & 1 \end{bmatrix}.$$

More generally, for $1 \leqq k \leqq n$, it follows that the partition factorization method using the submatrix A_k of A is mathematically the same as the direct factorization method applied to PAP^T, where the permutation matrix P is given by (2.11). *Thus the backward error analysis given in Theorem* 1 *for the direct factorization method also holds for the partition factorization method.*

There are some possible advantages of the flexibility of choosing an arbitrary principal submatrix A_k to factorize in the solution process for p. First, one might choose an A_k having a particular sparsity pattern. For example, if A has a dense row, then it could be excluded from the solution process. Secondly, the condition numbers $\|A_k\|\,\|A_k^{-1}\|$ may differ widely for $1 \leqq k \leqq n$, so one might expect better numerical results if a well-conditioned A_k were chosen. However, as we shall note in the test results of § 3, the choice of A_k in the partition factorization method appears to have little effect on the numerical accuracy obtained, up to the precision of the machine. As a partial explanation for these observations we give the following theorem, which relates the smallest singular values of the $(n-1)\times(n-1)$ principal submatrices A_k to the smallest nonzero singular value of A.

THEOREM 2. *Let $A = I - Q^T$, where Q is an $n \times n$ irreducible stochastic matrix and let A_k denote the principal submatrix obtained from A by deleting the kth row and the kth column for $1 \leqq k \leqq n$. If σ_{n-1} denotes the second smallest singular value of A and $\sigma_{n-1}^{(k)}$ denotes the smallest singular value of A_k then*

$$\sigma_{n-1}^{(k)} \leqq \sigma_{n-1}. \tag{2.12}$$

Proof. Without loss of generality it will be assumed that $k = n$. Then A can be written in the partitioned form

$$A = \begin{bmatrix} A_n & y_n \\ z_n^T & a_{nn} \end{bmatrix}. \tag{2.13}$$

Let $\sigma(A^TA) = \{\sigma_j^2\}_{j=1}^n$ denote the eigenvalues of A^TA where $0 = \sigma_n < \sigma_{n-1} \leqq \cdots \leqq \sigma_1$, and let $\sigma(A_n^TA_n) = \{\alpha_j^2\}_{j=1}^{n-1}$ denote the eigenvalues of $A_n^TA_n$ where $0 < \alpha_{n-1} \leqq \alpha_{n-2} \leqq \cdots \leqq \alpha_1$, and let $\sigma(A_n^TA_n + z_nz_n^T) = \{\gamma_j^2\}_{j=1}^{n-1}$ denote the eigenvalues of $A_n^TA_n + z_nz_n^T$ where $0 \leqq \gamma_{n-1} \leqq \gamma_{n-2} \leqq \cdots \leqq \gamma_1$.

The matrix $A_n^TA_n + z_nz_n^T$ is a principal leading submatrix of the symmetric matrix A^TA. Thus by the interlacing property

$$0 = \sigma_n \leqq \gamma_{n-1} \leqq \sigma_{n-1} \leqq \cdots \leqq \gamma_1 \leqq \sigma_1. \tag{2.14}$$

Also, applying Wilkinson [1965, pp. 97–98], it follows that

$$\gamma_j^2 - \alpha_j^2 = m_j(z_n^Tz_n) \quad \text{for } j = 1, 2, \cdots, n-1$$

where $0 \leqq m_k \leqq 1$ and $\sum_{k=1}^{n-1} m_k = 1$. Hence

$$\alpha_j \leqq \gamma_j \quad \text{for } j = 1, 2, \cdots, n-1.$$

Applying (2.14) it follows that

$$0 \leqq \alpha_j \leqq \gamma_j \leqq \sigma_j \quad \text{for } j = 1, 2, \cdots, n-1.$$

In particular, if $j = n-1$, then

$$0 \leqq \alpha_{n-1} \leqq \sigma_{n-1}.$$

The result follows. □

Note that Theorem 2 shows that if the smallest nonzero singular value of A is small, then such is the case for the smallest singular value of each A_k. This implies that if the total problem of solving $Ap = 0$ is ill-conditioned, then any choice of A_k in the partition factorization method can lead to an ill-conditioned problem, a situation that has been observed in our test problems. These and other observations concerning the test results are discussed in § 4.

3. Numerical comparisons. Numerical comparisons were performed for the partition factorization (PF) method and the Paige, Styan and Wachter (PSW) methods 1 and 2. Note that in computing the stationary distribution vector p for a stochastic matrix of order n, in each of the three algorithms being tested there are n possibly different systems of linear equations that can be used to obtain p.

The computer program for each algorithm was executed on an IBM 3081 computer at the Triangle Universities Computation Center. The software was written in

FORTRAN and compiled using the IBM Extended Optimizing Compiler in single and double precision floating point arithmetic.

The relative error was computed by comparing the single precision result with the double precision approximate solution, found by using the IMSL subroutine EIGRF. The *relative error* reported was computed as $\|p-\hat{p}\|_1/\|p\|_1$, where p is the double precision approximation and will be referred to as the *exact solution*, and $\hat{p}$ is the single precision approximation computed by one of the three algorithms. The vectors p and $\hat{p}$ are both normalized so that $\|p\|_1=1$ and $\|\hat{p}\|_1=1$. Hence the relative error is bounded above by 2. The *residual error* computed was $\|\hat{p}-Q^T\hat{p}\|_1$ and would theoretically be 0 if $\hat{p}=p$.

One of the problems associated with the PSW methods is that the sparse structure of the matrix A may be modified and the resulting updated matrix may no longer be sparse. In order to measure this effect, the numbers of nonzero entries in the matrix A and in the rank-one modified matrix in the PSW methods 1 and 2 are reported when A is sparse.

Various test problems were used to compare the three algorithms being considered. One type of matrix used is a nearly completely decomposable (NCD) matrix. The stochastic matrix

$$Q=Q^*+\varepsilon C$$

where $\varepsilon>0$, Q^* is a block diagonal stochastic matrix and C has row sums zero, is called a *nearly completely decomposable matrix.* If Q^* has m diagonal blocks, then Q^* has 1 as an eigenvalue of multiplicity at least m. Therefore the matrix Q has at least $m-1$ eigenvalues close to 1. The parameter ε is called the *degree of coupling* and is a measure of the decomposability of the matrix Q. Such matrices often arise in the analysis of queueing networks (e.g., Koury, McAllister and Stewart [1984]).

The following test problems were used to compare the PF method and the PSW methods 1 and 2.

Test problem 1. This matrix is of order 10 and the condition numbers for the submatrices A_k of order 9 range from 2.046 E+01 to 2.090 E+17.

Test problem 2. This matrix is of order 8 and is commonly called Courtois's Matrix (see Courtois [1977]). Although the entries in the sixth row do not sum to 1, it has been used to test several algorithms (e.g., Stewart [1980] and Vantilbourgh [1981]). It is nearly completely decomposable, and the degree of coupling is 0.001.

Test problem 3. This matrix is of order 6 and the condition numbers for the submatrices A_k of order 5 range from 1.644 E+01 to 1.425 E+07.

Test problem 4. This is a class of matrices of order 10 which are nearly completely decomposable. This particular class of matrices was used by Paige, Styan and Wachter [1975] for test purposes.

Test problem 5. This matrix of order 84 is banded and sparse. It arose from the queueing network analysis of a job line production model, studied in the Industrial Engineering Department at North Carolina State University.

Special notation. If B is a square matrix, then $N(B)$ denotes the percent of nonzero entries in B. If B is also nonsingular then the term $K_1(B)$ denotes the *condition number* of the matrix and is defined by $K_1(B)=\|B\|_1\|B^{-1}\|_1$.

The term *maximum relative error* means the maximum of all the relative error terms in the one norm, for the n possible different linear systems for a particular problem and algorithm. Similar definitions are applied to the terms *minimum relative error* and *maximum and minimum residual errors.*

Test problem 1.
Matrix Q:

$$\begin{bmatrix} .2 & 0 & 0 & .6 & 0 & 0 & 0 & 0 & 0 & .2 \\ 0 & .1 & 0 & 0 & .6 & 0 & .3 & 0 & 0 & 0 \\ 0 & .1 & 0 & 0 & 0 & 0 & 0 & .8 & 0 & .1 \\ 0 & 0 & .6 & 0 & .3 & 0 & 0 & 0 & 0 & .1 \\ 0 & .5 & 0 & 0 & .5 & 0 & 0 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & .2 & 0 & 0 & 0 & .3 & 0 \\ 0 & 0 & 0 & 0 & .7 & 0 & .2 & 0 & 0 & .1 \\ .1 & 0 & .9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & .1 & 0 & 0 & 0 & .8 & 0 & 0 & 0 & .1 \\ 0 & .4 & 0 & 0 & 0 & .4 & 0 & 0 & 0 & .2 \end{bmatrix}.$$

Exact solution:

$$p = (3.18129230\,\mathrm{E}-16, 3.13420278\,\mathrm{E}-01, 2.93617551\,\mathrm{E}-15, 3.07485879\,\mathrm{E}-16, 5.43805217\,\mathrm{E}-01, 7.88809424\,\mathrm{E}-03, 1.17532604\,\mathrm{E}-01, 2.56247898\,\mathrm{E}-15, 2.36642827\,\mathrm{E}-03, 1.49873790\,\mathrm{E}-02)^T.$$

Second smallest singular value of A: 1.424 E−01.

	PF method	PSW method 1	PSW method 2
maximum relative error	2.525 E−06	2.479 E−06	3.676 E−06

	PF method	PSW method 1	PSW method 2
minimum relative error	1.454 E−07	1.757 E−07	6.856 E−08

	PF method	PSW method 1	PSW method 2
maximum residual error	4.452 E−06	7.882 E−07	7.881 E−07

	PF method	PSW method 1	PSW method 2
minimum residual error	4.555 E−07	1.827 E−07	1.830 E−07

$$N(A) = 33\%, \quad \max_j N(A + (Q^T e_j)e^T) = 51\%, \quad \max_j N(A + e_j e^T) = 40\%,$$

$$\min_j K_1(A_j) = 2.046\,E+01, \quad \max_j K_1(A_j) = 2.090\,\mathrm{E}+17.$$

Test problem 2.
Matrix Q:

$$\begin{bmatrix}
.85 & 0 & .149 & .0009 & 0 & .00005 & 0 & .00005 \\
.1 & .65 & .249 & 0 & .00009 & .00005 & 0 & .00005 \\
.1 & .8 & .0996 & .0003 & 0 & 0 & .0001 & 0 \\
0 & .0004 & 0 & .7 & .2995 & 0 & .0001 & 0 \\
.0005 & 0 & .0004 & .399 & .6 & .0001 & 0 & 0 \\
0 & .00005 & 0 & 0 & .00005 & .6 & .2499 & .15 \\
.00003 & 0 & .00003 & .00004 & 0 & .1 & .8 & .0999 \\
0 & .00005 & 0 & 0 & .00005 & .1999 & .25 & .55
\end{bmatrix}.$$

Exact solution:

$$p = (8.92826528\,\mathrm{E}-02, 9.27576375\,\mathrm{E}-02, 4.04883120\,\mathrm{E}-02,$$
$$1.58533191\,\mathrm{E}-01, 1.18938207\,\mathrm{E}-01, 1.20385481\,\mathrm{E}-01,$$
$$2.77795252\,\mathrm{E}-01, 1.01819266\,\mathrm{E}-01)^T.$$

Second smallest singular value of A: 1.906 E−04.
Degree of coupling: 0.001.

	PF method	PSW method 1	PSW method 2
maximum relative error	1.710 E−04	1.180 E−03	1.097 E−03

	PF method	PSW method 1	PSW method 2
minimum relative error	3.957 E−05	3.905 E−04	1.111 E−04

	PF method	PSW method 1	PSW method 2
maximum residual error	1.149 E−06	1.009 E−06	9.946 E−07

	PF method	PSW method 1	PSW method 2
minimum residual error	1.891 E−07	7.152 E−07	7.860 E−07

$$N(A) = 62\%, \quad \max_j N(A + (Q^T e_j)e^T) = 83\%, \quad \max_j N(A + e_j e^T) = 69\%,$$

$$\min_j K_1(A_j) = 1.021\,\mathrm{E}+04, \quad \max_j K_1(A_j) = 2.041\,\mathrm{E}+04.$$

Test problem 3.
Matrix Q:

$$\begin{bmatrix} 0.999999 & 1.0\,\text{E}-07 & 2.0\,\text{E}-07 & 3.0\,\text{E}-07 & 4.0\,\text{E}-07 \\ 0.4 & 0.3 & 0 & 0 & 0.3 \\ 5.0\,\text{E}-07 & 0 & 0.999999 & 0 & 5.0\,\text{E}-07 \\ 5.0\,\text{E}-07 & 0 & 0 & 0.999999 & 5.0\,\text{E}-07 \\ 2.0\,\text{E}-07 & 3.0\,\text{E}-07 & 1.0\,\text{E}-07 & 4.0\,\text{E}-07 & 0.999999 \end{bmatrix}.$$

Exact solution:

$$p = (3.15217329\,\text{E}-01, 1.95652135\,\text{E}-07, 9.81883865\,\text{E}-02,$$
$$2.35144881\,\text{E}-01, 3.51449206\,\text{E}-01)^T.$$

Second smallest singular value of A: 1.007 E−06.

	PF method	PSW method 1	PSW method 2
maximum relative error	2.265 E−02	3.888 E−02	2.265 E−02

	PF method	PSW method 1	PSW method 2
minimum relative error	1.313 E−02	6.115 E−03	1.313 E−02

	PF method	PSW method 1	PSW method 2
maximum residual error	3.040 E−08	4.172 E−07	3.636 E−07

	PF method	PSW method 1	PSW method 2
minimum residual error	5.900 E−09	3.576 E−07	3.576 E−07

$$\min_j K_1(A_j) = 1.644\,\text{E}+01, \qquad \max_j K_1(A_j) = 1.425\,\text{E}+07.$$

Test problem 4.
Matrix Q: Let

$$\bar{Q}(\varepsilon) = \begin{bmatrix} .1 & .3 & .1 & .2 & .3 & \varepsilon & 0 & 0 & 0 & 0 \\ .2 & .1 & .1 & .2 & .4 & 0 & 0 & 0 & 0 & 0 \\ .1 & .2 & .2 & .4 & .1 & 0 & 0 & 0 & 0 & 0 \\ .4 & .2 & .1 & .2 & .1 & 0 & 0 & 0 & 0 & 0 \\ .6 & .3 & 0 & 0 & .1 & 0 & 0 & 0 & 0 & 0 \\ \varepsilon & 0 & 0 & 0 & 0 & .1 & .2 & .2 & .4 & .1 \\ 0 & 0 & 0 & 0 & 0 & .2 & .2 & .1 & .3 & .2 \\ 0 & 0 & 0 & 0 & 0 & .1 & .5 & 0 & .2 & .2 \\ 0 & 0 & 0 & 0 & 0 & .5 & .2 & .1 & 0 & .2 \\ 0 & 0 & 0 & 0 & 0 & .1 & .2 & .2 & .3 & .2 \end{bmatrix}.$$

If s_j denotes the inverse of the sum of the entries in the jth row of the matrix $\bar{Q}(\varepsilon)$ and $D = \operatorname{diag}(s_1, s_2, \cdots, s_n)$, then $Q(\varepsilon) = D\bar{Q}(\varepsilon)$ is a nearly completely decomposable stochastic matrix of order 10. Let $A(\varepsilon) = I - Q(\varepsilon)$.

Degree of coupling: $\varepsilon/(1+\varepsilon)$.

ε	Second smallest singular value of $A(\varepsilon)$
1.0 E−01	4.211 E−01
1.0 E−03	4.898 E−04
1.0 E−05	4.898 E−06
1.0 E−07	4.899 E−08

	ε	PF method	PSW method 1	PSW method 2
maximum relative error	1.0 E−01	4.417 E−06	6.738 E−06	7.304 E−06
	1.0 E−03	2.320 E−04	5.506 E−04	2.876 E−04
	1.0 E−05	1.586 E−02	2.501 E−02	3.901 E−02
	1.0 E−07	8.294 E−01	*	*

	ε	PF method	PSW method 1	PSW method 2
minimum relative error	1.0 E−01	1.661 E−06	1.177 E−06	2.052 E−06
	1.0 E−03	6.453 E−05	6.164 E−05	7.774 E−05
	1.0 E−05	8.296 E−03	7.814 E−03	7.034 E−03
	1.0 E−07	6.424 E−01	8.167 E−01	8.167 E−01

	ε	PF method	PSW method 1	PSW method 2
maximum residual error	1.0 E−01	8.102 E−06	1.143 E−06	1.512 E−06
	1.0 E−03	7.825 E−06	1.501 E−06	1.262 E−06
	1.0 E−05	7.946 E−06	1.281 E−06	1.352 E−06
	1.0 E−07	6.395 E−06	*	*

	ε	PF method	PSW method 1	PSW method 2
minimum residual error	1.0 E−01	4.395 E−07	6.556 E−07	6.034 E−07
	1.0 E−03	4.145 E−07	5.997 E−07	6.631 E−07
	1.0 E−05	5.772 E−07	5.557 E−07	6.631 E−07
	1.0 E−07	2.236 E−07	5.774 E−07	5.848 E−07

ε	$\min_j K_1(A_j)$	$\max_j K_1(A_j)$
1.0 E−01	7.494 E+01	1.755 E+02
1.0 E−03	6.284 E+03	9.216 E+03
1.0 E−05	6.261 E+05	9.135 E+05
1.0 E−07	6.261 E+07	9.134 E+07.

$$N(A) = 48\%, \quad \max_j N(A + (Q^T e_j)e^T) = 74\%, \quad \max_j N(A + e_j e^T) = 56\%.$$

Test problem 5.
Matrix Q (zero/nonzero structure): See Fig. 1.

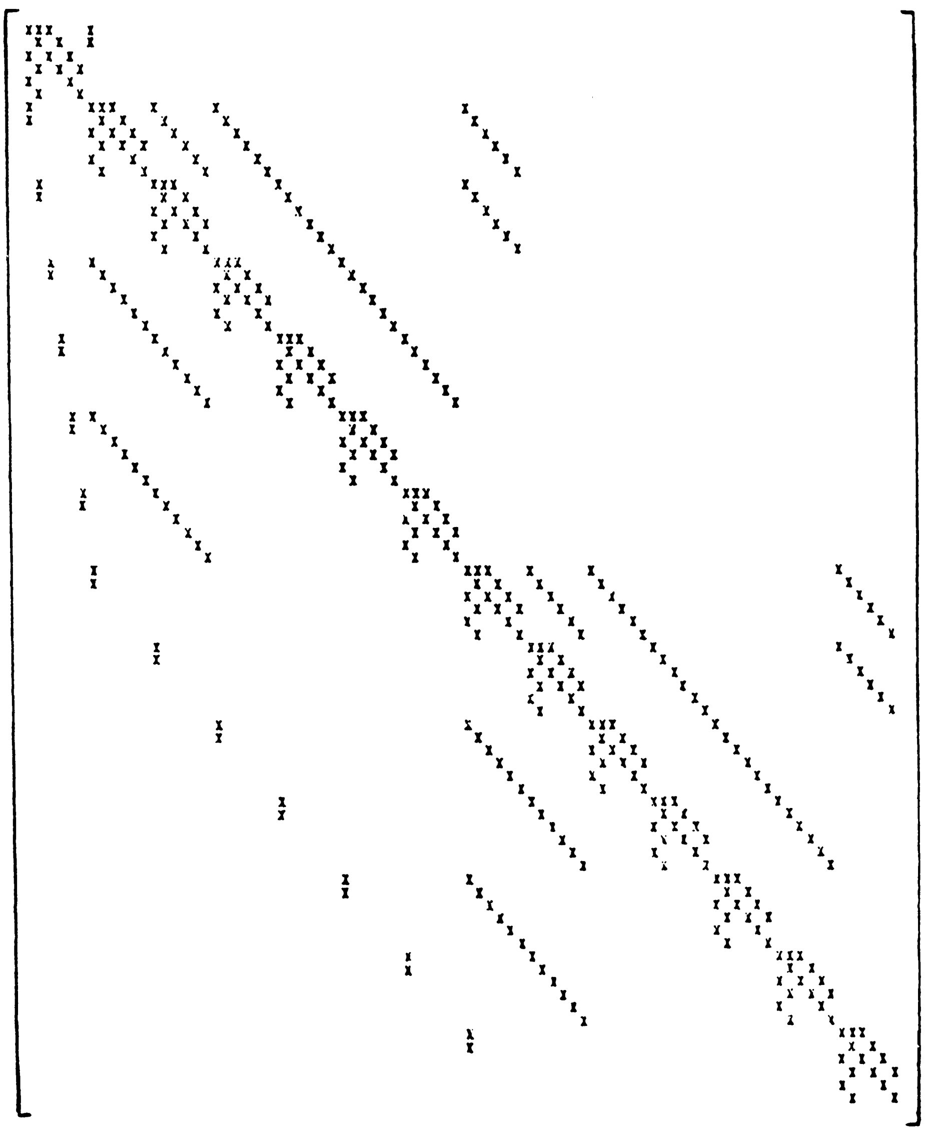


FIG 1.

Zero/nonzero structure of the PSW *matrix* $A+(Q^Te_{43})e^T$ *for test problem* 5: See Fig. 2.

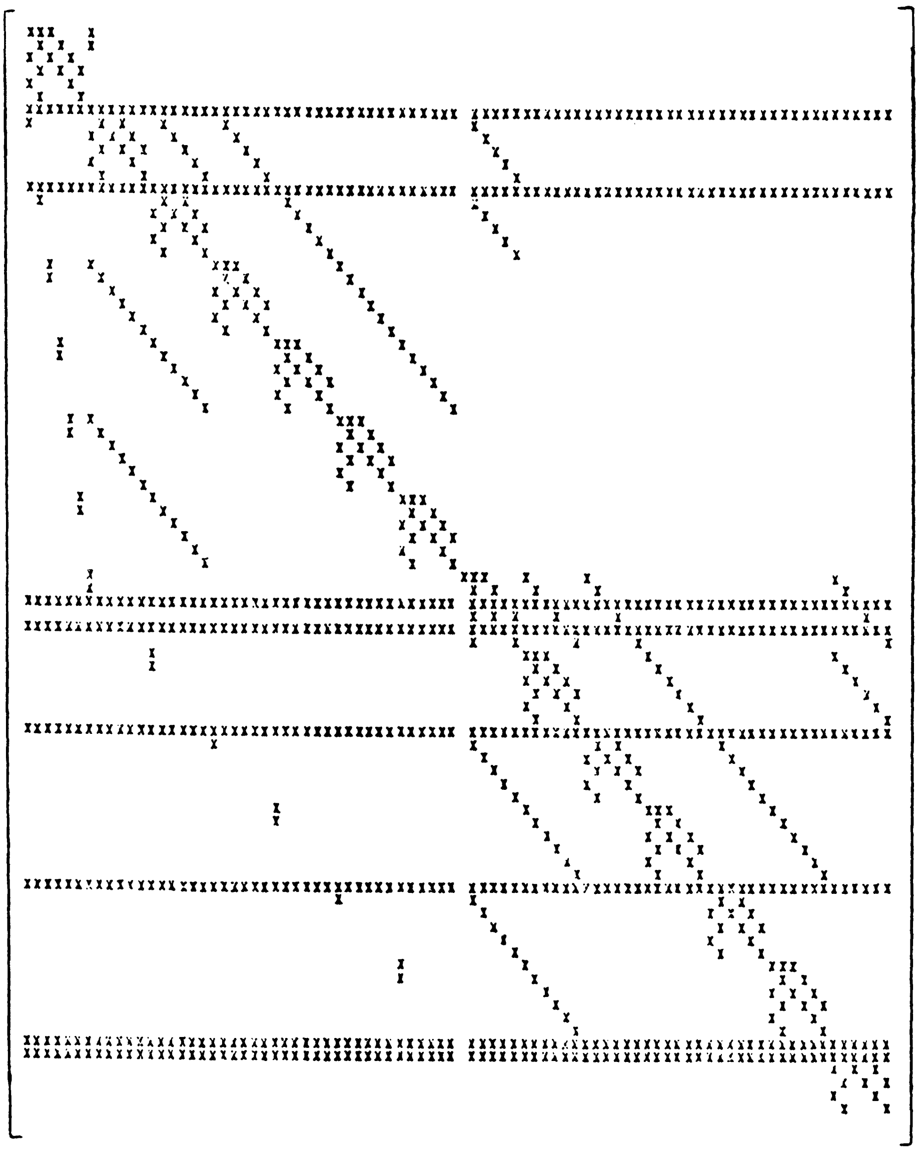


FIG 2.

Second smallest singular value of A: 3.305 E−02.

	PF method	PSW method 1	PSW method 2
maximum relative error	2.293 E−05	3.319 E−05	3.038 E−05

	PF method	PSW method 1	PSW method 2
minimum relative error	7.735 E−06	1.974 E−05	1.952 E−05

	PF method	PSW method 1	PSW method 2
maximum residual error	9.756 E−06	5.388 E−06	8.509 E−06

	PF method	PSW method 1	PSW method 2
minimum residual error	1.449 E−06	2.358 E−06	5.841 E−06

$$N(A)=5\%, \quad \max_j N(A+(Q^Te_j)e^T)=14\%, \quad \max_j (N(A+e_je^T)=6\%,$$

$$\min_j K_1(A_j)=1.125\,\mathrm{E}+02, \quad \max_j K_1(A_j)=1.108\,\mathrm{E}+03.$$

4. Concluding remarks. The paper is concluded with some observations concerning the performances of the nonsingular rank-one modification methods recommended by Paige, Styan and Wachter [1975] (PSW methods 1 and 2) and our partition factorization (PF) method. We comment first about the particular test problems considered in the preceeding section and then give some general observations.

1. In test problem 1 all three algorithms produced 5 or more decimal digits of accuracy in IBM single precision in each situation. (Here the PF method was tested using each of the 10 principal submatrices A_k of A having order 9. The PSW method 1 was tested using each of the 10 rows of Q and the PSW method 2 was tested using each unit vector e_j, $1 \leqq j \leqq 10$.) Some fill-in to $I-Q^T$ occurred as a result of PSW rank-one modifications. Notice that the one-condition numbers of the A_k range up to near 2×10^{17}. However, the PF method still obtains uniformly good results using each A_k. The second smallest singular value of $I-Q^T$ is near .14, and thus the overall problem of computing p is apparently well-conditioned.

2. Test problem 2 is Courtois's matrix which is nearly completely decomposable. Here the three methods obtain moderately accurate results in IBM single precision (between 3 and 6 decimal digits) in each situation.

3. Between only 1 and 2 decimal digits of accuracy are obtained by the three methods applied to test problem 3. However, the residual error is small in each case. Observe that the second smallest singular value of $I-Q^T$ is around 10^{-6}.

4. Test problem 4 represents a class of test situations involving a nearly completely decomposable matrix in which the degree of coupling decreases from 10^{-1} to 10^{-7}, resulting in a comparable decrease in the second smallest singular value of $I-Q^T$. Here the PF method produces the smallest maximum relative error in each case.

5. For the job line production model in test problem 5, all three methods obtained relatively accurate results. However, the PSW methods 1 and 2 destroy the sparse banded structure of $I-Q^T$, as we illustrate.

6. In general for all the problems tested, the residual error was between 10^{-6} and 10^{-9}, even though in many cases the relative error was large. Thus the residual error may not be good measure of the accuracy of the approximation to p here, for ill-conditioned problems.

7. Recall that pivoting generally is necessary in the LU factorizations of the nonsingular rank-one updates of $I - Q^T$ in PSW methods 1 and 2. However, the PF method requires no pivoting for numerical stability. Thus it facilitates the possible application of symmetric pivoting to preserve sparsity (see Funderlic and Plemmons [1984]) without the use of any threshold pivoting parameter.

8. Finally, we observe that the three methods all obtained comparable accuracy on each test problem. In particular, the PF method obtained uniformly good or uniformly poor results on a given test problem (depending upon the magnitude of the second smallest singular value), even though there was often a considerable variation of the condition numbers of the principal submatrices used in the PF method. This is in agreement with Theorem 2, and it leads us to infer that the accuracy obtained by the PF method is relatively independent of the choice of principal submatrices, up to the constraints of the machine precision. Thus the choice of the principal submatrix A_k for the PF method could be based upon other considerations, such as sparsity. For example, if Q has one dense column, say the kth, then the extreme fill-in in computing the LU factorization might be avoided by using A_k in the PF method.

REFERENCES

A. Berman and R. J. Plemmons [1979], *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York.

P. J. Courtois [1977], *Decomposability: Queueing and Computer System Applications*, Academic Press, New York.

S. L. Dodd, D. F. McAllister and W. J. Stewart [1981], *An iterative method for the exact solution of Coxian queueing networks*, in Proc. ACM/Sigmetrics Conference on Measurement and Modeling of Computer Systems, September 1981, Association for Computing Machinery, New York, pp. 97–104.

F. Duchin and D. B. Szlyd [1979], *Application of sparse matrix techniques to inter-regional input-output analysis*, Economics of Planning, 15, pp. 142–167.

R. E. Funderlic and J. B. Mankin [1981], *Solution of homogeneous systems of linear equations arising from compartmental models*, this Journal, 2, pp. 375–383.

R. E. Funderlic and R. J. Plemmons [1981], *LU decomposition of M-matrices by elimination without pivoting*, Linear Algebra Appl., 41, pp. 99–110.

———, [1984], *Incomplete factorization methods for certain M-matrices*, SIAM J. Algebraic Disc. Meth., 5 pp. 33–42.

R. E. Funderlic, M. Neumann and R. J. Plemmons [1982], *LU decompositions of generalized diagonally dominant matrices*, Numer. Math., 40, 57–69.

G. H. Golub and E. Seneta [1973], *Computation of the stationary distribution of an infinite Markov chain*, Bull. Austral. Math. Soc., 8, pp. 333–341.

W. J. Harrod [1982], *Rank modification methods for certain singular systems of linear equations*, Dissertation, Univ. of Tennessee, Knoxville.

J. J. Hunter [1982], *Generalized inverses and their application to applied probability problems*, Linear Algebra Appl., 45, pp. 157–198.

L. Kaufman [1983], *Matrix methods for queueing problems*, this Journal, 4 (1983), pp. 525–552.

J. G. Kemeny [1981], *Generalization of a fundamental matrix*, Linear Algebra Appl., 38, pp. 193–226.

J. G. Kemeny and J. L. Snell [1960], *Finite Markov Chains*, Van Nostrand, Princeton, NJ.

R. Koury, D. F. McAllister and W. J. Stewart [1984], *A numerical comparison of block iterative methods with aggregation for computing stationary probability vectors for nearly completely decomposable Markov chains*, SIAM J. Algebraic Disc. Meth., 5, pp. 164–186.

C. D. Meyer [1975], *The role of the group generalized inverse in the theory of finite Markov chains*, SIAM Rev., 17, pp. 443–464.

C. C. Paige, P. H. Styan and P. G. Wachter [1975], *Computation of the stationary distribution of a Markov chain*, J. Statist. Comput. Simulation, 4, pp. 173–186.

R. J. Plemmons [1976], *Regular splittings and the discrete Neumann problem*, Numer. Math., 25, pp. 153–161.

C. W. SHEPPARD AND A. S. HOUSEHOLDER [1951], *The mathematical basis of the interpretation of tracer experiments in closed steady-state systems*, J. Appl. Phys., 22, pp. 510–520.

G. W. STEWART [1973], *Introduction to Matrix Computations*, Academic Press, New York.

———, [1980], *Computable error bounds for aggregated Markov chains*, Technical Report 901, Computer Science Dept., Univ. of Maryland, College Park, May 1980.

W. J. STEWART [1978], *A comparison of numerical techniques in Markov modelling*, Comm. ACM, 21, pp. 144–151.

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

H. VANTILBORGH [1981], *The error of aggregation in decomposable systems*, Report R453, Mobil Research Laboratory, Brussels, Belgium, 1981.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

GENERATING CORRELATION MATRICES[*]

GEORGE MARSAGLIA[†] AND INGRAM OLKIN[‡]

Abstract. This paper describes a variety of methods for generating random correlation matrices, with emphasis on choice of random variables and distributions so as to provide matrices with given structure, expected values or eigenvalues.

Key words. random correlation matrices, random numbers, Monte Carlo, simulation

1. Introduction. A correlation matrix is a symmetric, positive semi-definite matrix with 1's on the diagonal. Numerous papers have been devoted, in whole or in part, to the problem of generating random correlation matrices, but usually with a particular application in mind. Papers concerned with correlation matrices having a particular structure are those of Tucker, Koopman and Linn (1969), Herzberg (1969), Dempster, Schatzoff and Wermuth (1977) and Ryan (1978), while Bendel and Afifi (1977), Chalmers (1975), Bendel and Mickey (1978), Johnson and Welch (1980) are concerned with eigenvalues of the generated matrices.

There seems to be need for a discussion of general methods that may be used to generate random correlation matrices which meet various requirements such as structural, distribution of elements, expected values or eigenvalues, with emphasis on possible choices of the random elements at each stage to achieve the desired objective. The following three sections describe methods classified in three ways: random correlation matrices with given expected values; in the form TT'; with given eigenvalues.

2. Random correlation matrices with given mean. Let C be a given correlation matrix and let $X=(x_{ij})$ be a random symmetric matrix with zeros on the diagonal and with means $E[x_{ij}]=0$. Then $C+X$ will be a random correlation matrix with expected value C if, and only if, the eigenvalues of $C+X$ are nonnegative. Viewing X as a perturbation of C, we may use well-known results: adding the symmetric matrix X to C cannot change the eigenvalues of C by more than $\|X\|_2$, the 2-norm of X. See, e.g., Stewart (1968). Since X is symmetric, its 2-norm is its spectral radius, which is bounded by both $\|X\|_E=[\sum x_{ij}^2]^{1/2}$ and $\|X\|_1=\|X\|_\infty=\max_i \sum_j |x_{ij}|$. Thus if any one of $\|X\|_2$, $\|X\|_E$ or $\|X\|_1=\|X\|_\infty$ is less than λ, the least eigenvalue of C, then $C+X$ will be positive definite.

This provides a variety of methods for generating random correlation matrices $C+X$ with expected value a given positive definite correlation matrix C with λ the least eigenvalue of C:

2.1. Let $A=(a_{ij})$ be a symmetric matrix with zeros on the diagonal such that $\|A\|_\infty=\max_i \sum_j |a_{ij}|<\lambda$. For $j>i$, generate x_{ij} such that the marginal distribution of each x_{ij} is in the interval $|x_{ij}|<a_{ij}$ and such that $E[x_{ij}]=0$. Then, with $x_{ii}=0$ and $x_{ji}=x_{ij}$ for $i>j$, $R=C+X$ will be a random correlation matrix with expected value C. A simple way to do this is with x_{ij} independent uniform in $|x|<|a_{ij}|$, $i<j$.

2.2. For $i<j$ generate $x_{12}, x_{13}, \cdots, x_{n-1,n}$ with a radially symmetric distribution in (or on) the unit $n(n-1)/2$-sphere. Then $R=C+2^{-1/2}\lambda X$ will be a random correlation matrix with expected value C.

[*] Received by the editors August 11, 1982, and in revised form February 10, 1983. This research was supported by the National Science Foundation.

[†] Computer Science Department,Washington State University, Pullman, Washington 99164.
[‡] Department of Statistics, Stanford University, Stanford, California 94305.

2.3. Using the method described in § 4, generate a random correlation matrix R with eigenvalues in the interval $(1-\lambda, 1+\lambda)$, where λ is the least eigenvalue of C. Then $C+R-I$ is a random correlation matrix with expected value C.

3. Random correlation matrices of the form TT'. In the method of the previous section the matrix C was given, and a random matrix X was then chosen so that $C+X$ was positive definite. An easier way to guarantee definitiveness (but with less control on the distributions) is to form TT' from a random $n\times m$ matrix T, $m \geqq n$. In order that the result be a correlation matrix, the rows of T must have length one, and thus the problem may be put in geometric terms: *TT' is a random correlation matrix if, and only if, the rows of T are random points on the unit m-sphere.*

This leads to a variety of methods. Probably the easiest is one that merely generates a matrix T of independent uniform variates, and then forms TT' after normalizing by dividing each row by its root-mean-square. A faster method requires about half as many variates by starting with T lower triangular ($t_{ij}=0$ if $i<j$). There are two easy ways to make an initial row $x_1, \cdots, x_m$ of T a point on the unit m-sphere by normalizing: root-mean-square,

$$x_i \leftarrow \pm x_i/(\sum x_j^2)^{1/2}$$

and root-absolute-mean,

$$x_i \leftarrow \pm(|x_i|/\sum |x_j|)^{1/2},$$

with the $\pm$ optional. For two initial rows $x_1, \cdots, x_m$ and $y_1, \cdots, y_m$ the two kinds of normalization lead to marginal densities in TT' for a ratio or a sum of ratios:

$$(\sum \pm x_i y_i)/(\sum x_j^2 \sum y_j^2)^{1/2} \tag{3.1}$$

and

$$[\sum \pm [|x_i y_i|/(\sum |x_j| \sum |y_j|)]^{1/2}]. \tag{3.2}$$

If the initial elements of T are uniform or exponential, the distribution of (3.2) is more tractable than that of (3.1). If the initial elements are standard normal variates, then the distribution of (3.1) may be given explicitly; we will do so below. The central limit theorem is more readily applied to the sum in (3.2), but since (3.1) approaches the ratio of normal variates, both cases lead to the elements of TT' being approximately normal for m large. Choosing a random $\pm$ when normalizing the rows will center the densities at the origin.

Monte Carlo experiments to compare methods for generating the initial elements of T and normalizing procedures should be worthwhile. In order to have exact distributions against which Monte Carlo results may be compared, we will give explicit densities for the case that the matrix T is initially chosen with nonzero elements independent standard normal, then its rows projected onto the unit m-sphere by the root-mean-square normalization (3.1). This will include triangular or any other $n\times m$ matrix T. For example, if an initial T has the form:

$$\begin{pmatrix} a & 0 & 0 & b & c & 0 \\ r & s & t & u & 0 & 0 \\ 0 & v & 0 & w & 0 & x \end{pmatrix}$$

with nonzero elements standard normal, and if each row is then made a point on the unit 6-sphere by dividing by its root-mean-square, what are the marginal densities of the elements of TT'?

THEOREM. *If $Z=\sum_1^k X_iY_i/[\sum_1^m X_i^2 \sum_1^n Y_i^2]^{1/2}$, $0<k\leqq m\leqq n$, with the X's and Y's independent standard normal random variables, then Z^2 is distributed as the product of beta variates*:

$$Z^2 \sim \beta_{1/2,(n-1)/2}\beta_{k/2,(m-k)/2}$$

and the density of Z is, for $-1<z<1$:

$$\frac{\Gamma(\frac{1}{2}n)\Gamma(\frac{1}{2}m)}{\Gamma(\frac{1}{2})\Gamma(\frac{1}{2}n-\frac{1}{2})\Gamma(\frac{1}{2}k)\Gamma(\frac{1}{2}m-\frac{1}{2}k)}\int_{z^2}^1 (y-z^2)^{(n-3)/2}y^{(k-m)/2}(1-y)^{(m-k)/2-1}\,dy.$$

(*When $k=m$, Z^2 is the single beta variate $\beta_{1/2,(n-1)/2}$, with corresponding density for Z.*)

Proof. We may view Z as an inner product, $\alpha\beta'$, with α and β independent points on the $(m+n-k)$-sphere. For example, when $k=2$, $m=3$, $n=4$ and

$$\alpha=(X_1,X_2,X_3,0,0)\Big/\left[\sum_1^3 X_i^2\right]^{1/2},\qquad \beta=(Y_1,Y_2,0,Y_3,Y_4)\Big/\left[\sum_i^4 Y_i^2\right]^{1/2},$$

then the random point α is the projection of $(X_1,X_2,X_3,0,0)$ onto the 5-sphere, so that α is independent of $\sum X_i^2$ and, similarly, β is independent of $\sum Y_i^2$. Thus, in general, $Z=\alpha\beta'$ is independent of the product $\sum X_i^2\sum Y_i^2$. The latter product is distributed as the product of independent gamma variates:

$$\sum_1^m X_i^2\sum_1^n Y_i^2\sim 4\gamma_{m/2}\gamma_{n/2}.$$

It follows that, with $V=\sum_1^k X_iY_i$,

$$E[Z^{2r}]E[\gamma_{m/2}^r]E[\gamma_{n/2}^r]=E[Z^{2r}\gamma_{m/2}^r\gamma_{n/2}^r]=E[V^{2r}].$$

Thus $E[Z^{2r}]=E[V^{2r}]/E[\gamma_{m/2}^r\gamma_{n/2}^r]$ and, using the moments of V^2 from the lemma below,

$$E[Z^{2r}]=\frac{\Gamma(\frac{1}{2}k+r)\Gamma(\frac{1}{2}+r)}{\Gamma(\frac{1}{2}k)\Gamma(\frac{1}{2})}\,\frac{\Gamma(\frac{1}{2}n)\Gamma(\frac{1}{2}m)}{\Gamma(\frac{1}{2}n+r)\Gamma(\frac{1}{2}m+r)}.$$

If $\beta_{a,b}$ is a random variable having a beta distribution with parameters a and b, then $E\beta_{a,b}^r=B(a+r,b)/B(a,b)$. Since Z^2 is a bounded random variable, its distribution is determined by its moments, namely, $EZ^{2r}=E\beta_{1/2,(n-1)/2}^r\beta_{k/2,(m-k)/2}^r$. The density of Z^2 is obtained from the density of a product and of a square root. $\square$

LEMMA. *Let $V=\sum_1^k X_iY_i$, with the X's and Y's independent standard normal. Then*

$$E[V^{2r}]=\frac{4\Gamma(\frac{1}{2}k+r)\Gamma(\frac{1}{2}+r)}{[\Gamma(\frac{1}{2}k)\Gamma(\frac{1}{2})]}.$$

Proof. Because of the radial symmetry of the distributions of $(X_1,X_2,\cdots,X_k)$ and $(Y_1,Y_2,\cdots,Y_k)$, we may assume that $(X_1,X_2,\cdots,X_k)$ has the form $(W,0,\cdots,0)$ with W distributed as $(X_1^2+\cdots+X_k^2)^{1/2}$, so that $W^2/2$ is distributed as a gamma variate $\gamma_{k/2}$ with parameter $k/2$, i.e. $W^2\sim 2\gamma_{k/2}$. Then $V^2\sim 4\gamma_{k/2}\gamma_{1/2}$, where $\gamma_{k/2}$ and $\gamma_{1/2}$ are independent. Then

$$E[V^{2r}]=4E[\gamma_{k/2}^r]E[\gamma_{1/2}^r]=4\Gamma(\tfrac{1}{2}k+r)\Gamma(\tfrac{1}{2}+r)/[\Gamma(\tfrac{1}{2}k)\Gamma(\tfrac{1}{2})].\qquad\square$$

Using root-mean-square normalization on a set of k independent normal variates produces a point on the surface of the unit k-sphere and the resulting spherically symmetric distribution seems the most desirable for many applications; it is one of the few methods for which the resulting distributions of elements in the correlation matrix can be given explicitly, as above. If a fast method for generating normal variates is not available, there are other efficient methods for generating uniform points on a k-sphere—see, e.g., Marsaglia (1972), (1980).

4. Generating a correlation matrix with given eigenvalues. Methods for generating a correlation matrix whose eigenvalues are close to a given set may be based on classical perturbation theory: if the ordered eigenvalues of C are $\lambda_1 \leqq \cdots \leqq \lambda_n$ and those of $C+X$ are $\mu_1 \leqq \cdots \leqq \mu_n$, then (Hoffman and Weilandt (1953)):

$$\sum (\lambda_i - \mu_i)^2 \leqq \|X\|_E^2.$$

Thus choosing a random symmetric matrix X with zeros on the diagonal and $\|X\|_E$ small will ensure that $C+X$ is a correlation matrix with eigenvalues close to C. A disadvantage of this method is that $\|X\|_E$ may be so small that all the random matrices $C+X$ will look like C. (If the expected value is supposed to be C, as in § 2, and if C is nearly singular, then all X's for which $C+X$ is definite may include only X's of small norm. But closeness of eigenvalues need not require closeness of the matrices.)

Another approach may be used: choose $D = \operatorname{diag}\{d_1, \cdots, d_n\}$ so that $\sum (\lambda_i - d_i)^2 < \varepsilon$; then choose a random orthogonal matrix P so that PDP' is a correlation matrix. Putting $\varepsilon = 0$ would then handle the case where the eigenvalues must be exactly a given set.

The trace of an $n \times n$ correlation matrix must be n, so that we may put the problem in general form: given an $n \times n$ positive semi-definite matrix A whose trace is n, choose a random orthogonal matrix P so that PAP' is a correlation matrix, i.e., has 1's on the diagonal. It is elementary to prove by induction that there are such P's: Choose any point α on the n-sphere $\alpha\alpha' = 1$ so that it also satisfies $\alpha A \alpha' = 1$, (the Rayleigh quotient guarantees the range of $\alpha A \alpha'$ will include 1 if the trace of A is n) and any orthogonal matrix $P = \binom{\alpha}{B}$ whose first row is α. Then

$$\begin{pmatrix} \alpha \\ B \end{pmatrix} A(\alpha' B') = \begin{pmatrix} 1 & \alpha A B' \\ B A \alpha' & B A B' \end{pmatrix}$$

and BAB' is an $(n-1)\times(n-1)$ positive definite matrix with trace equal to $(n-1)$. This is backward induction so that we need only consider the case $n=2$, for which the solution is explicit.

The most difficult problem in implementing this algorithm is in choosing a random point α from the n-sphere so that $\alpha A \alpha' = 1$. There seems to be no easy, explicit way to do this, but if one has a way, the following scheme may be used to generate all of the rows of the required orthogonal matrix P:

Start with the symmetric idempotent matrix $E = I$.

Choose α_1 from the row space of E, subject to $\alpha_1 \alpha_1' = \alpha_1 A \alpha_1' = 1$, and replace E by $E - \alpha_1' \alpha_1$.

Choose α_2 from the row space of E, subject to $\alpha_2 \alpha_2' = \alpha_2 A \alpha_2' = 1$, and replace E by $E - \alpha_2' \alpha_2$.

. . .

Choose α_n from the row space of E, subject to $\alpha_n \alpha_n' = \alpha_n A \alpha_n' = 1$, and replace E by $E - \alpha_n' \alpha_n$.

The resulting $\alpha_1, \alpha_2, \cdots, \alpha_n$ will be orthonormal with $\alpha_i A \alpha_i' = 1$, so that the matrix P with rows $\alpha_1, \cdots, \alpha_n$ will be orthogonal and PAP' will have unit diagonal elements.

The α's will be orthogonal because α_2 is in the row space of $I-\alpha_1'\alpha_1$, α_3 is in the row space of $I-\alpha_1'\alpha_1-\alpha_2'\alpha_2$, and so on.

This provides an easy-to-follow algorithm, but for one aspect: How do we "choose α from the row space of E, subject to $\alpha\alpha'=\alpha A\alpha'=1$?" Assume $A=D$, a diagonal matrix of the given eigenvalues. Spherical symmetry in the choice of α's will make the result invariant under choice of the initial symmetric matrix A, provided it has the given eigenvalues. The set $\alpha\alpha'=\alpha D\alpha'=1$ is a subset of the surface of the unit n-sphere, a pair of "spherical ellipses"—one the reflection of the other through the origin. A random slice through the sphere may not hit the two "ellipses," but if it does, it provides a nice way, by choosing one of the resulting four points of intersection.

This then suggests a rejection procedure for sampling α from the row space of a matrix E, subject to $\alpha\alpha'=\alpha D\alpha'=1$: Choose two points ξ and η from the row space of E, each in the form ζE, with $\zeta=(z_1,\cdots,z_n)$ having independent normal coordinates. If the plane determined by ξ, η and the origin cuts the set $\alpha\alpha'=\alpha D\alpha'=1$, take one of the four points of intersection and follow the algorithm. If the plane does not cut the set, choose a new plane. The plane determined by ξ and η may be represented as $r\xi+\eta$, rather than the conventional $r_1\xi+r_2\eta$, in view of the fact that we project onto the unit n-sphere. The condition that the plane cut the set $\alpha\alpha'=\alpha D\alpha'=1$ now becomes: $b^2<ac$, where $a=\xi(I-D)\xi'$, $b=\xi(I-D)\eta'$, $c=\eta(I-D)\eta'$.

These details are now incorporated into an explicit algorithm.

ALGORITHM. Given the diagonal matrix $D=\operatorname{diag}(d_1,\cdots,d_n\}$ with $d_i\geqq 0$ and $\sum d_i=n$, this algorithm produces a random orthogonal matrix P such that PDP' is a correlation matrix (1's on the diagonal).

(i) (*Initial state*) Start with the $n\times n$ matrix E and the index k: $E\leftarrow I$, $k\leftarrow 1$.

(ii) Generate a random vector $\xi=(x_1,\cdots,x_n)$ in the row space of E. [Let $\xi=(z_1,\cdots,z_n)E$, with the z's independent normal.] Compute $a=\sum(1-d_i)x_i^2$.

(iii) Generate another random vector $\eta=(y_1,\cdots,y_n)$ in the row space of E.

(iv) Compute $b=\sum(1-d_i)x_iy_i$ and $c=\sum(1-d_i)y_i^2$. If $d^2=b^2-ac\leqq 0$, go to step (iii).

(v) Put $r=(b\pm d)/a$, with the $\pm$ chosen at random. Then the vector $\zeta=r\xi-\eta$, when normalized with a random sign: $\zeta\leftarrow\pm\zeta/(\zeta\zeta')^{1/2}$ is a random choice for the kth row of P. Replace: $E\leftarrow E-\zeta'\zeta$, increment k and go to step (ii), unless $k=n$, in which case $n-I$ rows of P have been generated, and the last row may be any normalized vector in the row space of E.

The above algorithm may be compared to two others: Bendel and Mickey (1978) choose a random orthogonal matrix P, form PDP', then use 2×2 rotations (never reflections) to make the diagonal elements unity, one at a time. Chalmers (1975) provides an algorithm that chooses the rows of P sequentially to produce the required form, with rejection sampling from an n-cube at each stage. Our approach is similar, but with the intent to describe more fully the available choices for each row, providing possibly simpler algorithms or better control on the resulting distributions or expectations. Choosing the available elements from spherically symmetric distributions makes the resulting PAP' invariant under choice of the initial A with given eigenvalues.

The above algorithm is easily programmed. Its numerical stability is similar to that of Gram–Schmidt orthogonalization: quite good. In fact, omitting the rejection parts of the algorithm, steps (iii) and (iv), and putting $\zeta=\xi$ in step (v), produces an orthogonal matrix by the Gram–Schmidt process, which is the initial step in the Bendel–Mickey method. Steps (iii) and (iv) of the algorithm are performed a random number of times; the best possible is n times. We find an average of less than $1.5n$ times, for random initial choices of the eigenvalues.

Interested readers may wish to see if the correlation matrices produced by the algorithm have distributions that suit their needs. The distributions may be compared with those produced by the Bendel–Mickey method, which can be extended in the following way: Start with the given diagonal matrix of eigenvalues, D, and choose a random orthogonal matrix P to get $A = PAP'$. Then A has trace n, but will generally have no 1's on the diagonal. It will have a principal 2×2 submatrix of the form $\binom{ab}{bc}$ with $a<1<c$ or $c<1<a$. There are then four possible 2×2 orthogonal matrices (two of them rotations, two reflections) that reduce $\binom{ab}{bc}$ to $\binom{1r}{rs}$. Choosing one of the four at random leads to a similarity reduction of A that puts a 1 on its diagonal. Applying the procedure n times produces a random correlation matrix, all 1's on the diagonal. Choosing from the four possibilities at each step should extend the distributional properties of the end result, as should randomly choosing the 2×2 submatrices that lead to the reductions.

5. A conjecture. Generating a correlation matrix with given eigenvalues may be considered a special case of a more general matrix problem: given a symmetric matrix A, find an orthogonal P such that PAP' has equal values on its diagonal. It is easy to prove the existence of such P's and algorithms discussed above provide generating procedures. It is natural to ask: can other diagonals also be made to have equal elements? Such a "striped" matrix is called a Toeplitz matrix, elements t_{ij} with $t_{ij} = t_{rs}$ if $i-j=r-s$. For example, the 4×4 symmetric Toeplitz matrices have the form:

$$\begin{pmatrix} a & b & c & d \\ b & a & b & c \\ c & b & a & b \\ d & c & b & a \end{pmatrix}.$$

We conjecture the following, which we have only so far been able to prove for $n \leqq 4$:

CONJECTURE. *Given an $n\times n$ real symmetric matrix A, there is a real orthogonal matrix P such that PAP' is a Toeplitz matrix (t_{ij}), with $t_{ij}=t_{rs}$ for $|i-j|=|r-s|$, or the equivalent: every set of n real numbers is the spectrum of some real symmetric Toeplitz matrix.*

REFERENCES

R. B. BENDEL AND A. A. AFIFI, (1977), *Comparison of stopping rules in forward stepwise regression*, J. Amer. Statist. Assoc., 72, pp. 46–53.

R. B. BENDEL AND M. R. MICKEY, (1978), *Population correlation matrices for sampling experiments*, Commun. Statist-Simul. Comput., B7(2), pp. 163–182.

C. P. CHALMERS, (1975), *Generation of correlation matrices with given eigen-structure*, J. Statist. Comput. Simul., 4, pp. 133–139.

A. P. DEMPSTER, M. SCHATZOFF AND N. WERMUTH, (1977), *A simulation study of alternatives to least squares*, J. Amer. Statist. Assoc., 72, pp. 77–90.

P. A. HERZBERG, (1969), *The parameter of cross-validation*, Psychometrika Monograph Supplement No. 16, 34, part 2.

A. J. HOFFMAN AND H. W. WEILANDT, (1953), *The variation of the spectrum of a normal matrix*, Duke Math. J., 20, pp. 27–39.

D. G. JOHNSON AND W. J. WELCH, (1980), *The generation of pseudo-random correlation matrices*, J. Statist. Comput. Simul., 11, pp. 55–69.

G. MARSAGLIA, (1972), *Choosing a point from the surface of a sphere*, Ann. Math. Statist., 43, pp. 645–646.

———, (1980), *Generating a normal sample with a given mean and variance*, J. Statist. Comput. Simul., 11, pp. 71–73.

T. P. RYAN, (1980), *A new method of generating correlation matrices*, J. Statist. Comput. Simul., 11, pp. 79–85.

G. W. STEWART, (1973), *Introduction to Matrix Computation*. Academic Press, New York.

L. R. TUCKER, R. F. KOOPMAN AND R. F. LINN, (1969), *Evaluation of factor analytic research procedures by means of simulated correlation matrices*, Psychometrika, pp. 34, 421–459.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

DISTRIBUTION OF THE RATIO OF QUADRATIC FORMS IN NORMAL VARIABLES—NUMERICAL METHODS*

R. LUGANNANI† AND S. O. RICE†

Abstract. This paper is concerned mostly with the distribution of the ratio of two quadratic forms in normal variables. The probability density of the ratio is obtained as an integral and closed-form expressions are given for several special cases. However, in the general case numerical methods must be used to evaluate the density. A method of numerical integration is presented and illustrated by an example. The behavior of the density around the points at which it is irregular is examined. The general case of the distribution of the ratio of two composite random variables is discussed briefly.

Key words. quadratic forms in normal variables, ratio of random quadratic forms, numerical evaluation of probability densities

1. Introduction.

Much of this paper is concerned with the calculation of the probability density $p(y)$ of the ratio

$$\frac{z'Az}{z'Bz} \tag{1}$$

of quadratic forms where A and B are symmetric real matrices of order n, z is a column of correlated normal random variables $z_1, z_2, \cdots, z_n$, and z' is its transpose. The probability that (1) exceeds y is

$$Q(y) = \int_y^\infty p(y')\,dy'. \tag{2}$$

Most of our results assume that $z'Bz$ is positive definite. This case is also the one that has been studied the most in the past. Gurland [1], [2], [3], [4] has dealt with the problem of finding the distribution of general ratios. In particular he has shown [2] that when $z'Bz$ is positive definite, $Q(y)$ is equal to the probability that the quadratic form $z'(A-yB)z$ exceeds zero. He then shows that $p(y)$ and $Q(y)$ can be expressed as an infinite series of Laguerre functions. Davies [9] has given an algorithm to calculate the distribution of a quadratic form in normal variables, and has noted its application to the calculation of the distribution of the ratio of two quadratic forms. Recently the distribution of the ratio (1) has become of interest in tracking and detection problems. See Kanter [5] and the references he cites.

Here we propose to calculate $p(y)$ by using numerical integration to evaluate an integral based on the inversion formula for the characteristic function.

2. The ratio of quadratic forms in normal variables—integral for the probability density.

Here and in the remainder of the paper, except for Appendices A and B, it will be assumed that the quadratic form $z'Bz$ is positive definite. When we set

$$z'Az = \Phi_1, \qquad z'Bz = \Phi_2,$$

we see from equation (A.2) in Appendix A that the probability density of the ratio $z'Az/z'Bz$ is given by the integral

$$p(y) = \frac{1}{2\pi}\int_{-\infty}^{\infty} dt \int_{-\infty}^{\infty} dz_1 \int_{-\infty}^{\infty} dz_2 \cdots \int_{-\infty}^{\infty} dz_n\, \Phi_2 \hat{p}(z) \exp\,[it(\Phi_1 - y\Phi_2)]. \tag{3}$$

* Received by the editors June 6, 1981.

† Department of Electrical Engineering and Computer Sciences, University of California, San Diego, La Jolla, California 92093.

We seek a tractable method of calculating $p(y)$ when the probability density $\hat{p}(z)$ of the z's has the normal form

$$\hat{p}(z)=(2\pi)^{-n/2}|V|^{-1/2}\exp[-\tfrac{1}{2}(z-\xi)'V^{-1}(z-\xi)], \tag{4}$$

where ξ is the column matrix of the mean values ξ_j of the z's, V is the covariance matrix, and $|V|$ is its determinant.

From (4) and $z'V^{-1}\xi=\xi'V^{-1}z$, we see that part of the integrand in (3) can be written as

$$\begin{aligned}\hat{p}(z)\exp[it(\Phi_1-y\Phi_2)]=(2\pi)^{-n/2}|V|^{-1/2}\exp[-\tfrac{1}{2}z'\{V^{-1}-2it(A-yB)\}z\\ +\xi'V^{-1}z-\tfrac{1}{2}\xi'V^{-1}\xi].\end{aligned} \tag{5}$$

Substituting this in (3) and setting $u=it$ gives the integral we wish to evaluate:

$$\begin{aligned}p(y)=\frac{1}{2\pi i}\int_{-i\infty}^{i\infty}du\int_{-\infty}^{\infty}dz_1\cdots\int_{-\infty}^{\infty}dz_n(2\pi)^{-n/2}|V|^{-1/2}(z'Bz)\\ \cdot\exp[-\tfrac{1}{2}z'\{V^{-1}-2u(A-yB)\}z+\xi'V^{-1}z-\tfrac{1}{2}\xi'V^{-1}\xi].\end{aligned} \tag{6}$$

3. Integration with respect to $z_1, z_2, \cdots, z_n$. The n-fold integration with respect to the z's in (6) can be performed with the help of a transformation used to reduce a quadratic form to the sum of squares. Let $\hat{A}$ and L be $n\times n$ matrices such that

$$\hat{A}=A-yB,\qquad V=LL', \tag{7}$$

where L is a lower diagonal matrix (Choleski decomposition of V). Let λ_j be the jth eigenvalue, $\lambda_1\geqq\lambda_2\geqq\cdots\geqq\lambda_n$, of $L'\hat{A}L$ and p_j be the corresponding normalized eigencolumn ($L'\hat{A}L$ is symmetric and λ_j is real). Incidentally, λ_j is also an eigenvalue of $V\hat{A}$. Let P be the orthogonal matrix formed by setting the columns p_j side by side:

$$P=[p_1,p_2,\cdots,p_n],\qquad (I\lambda_j-L'\hat{A}L)p_j=0. \tag{8}$$

Then it can be shown (the details are given in Appendix C) that (6) goes into

$$p(y)=\frac{1}{2\pi i}\int_{-i\infty}^{i\infty}G(u)H(u)\,du, \tag{9}$$

where

$$\begin{aligned}&G(u)=\exp[\phi(u)],\\ &\phi(u)=\frac{1}{2}\sum_{j=1}^{n}[\hat{\xi}_j^2(1-2u\lambda_j)^{-1}-\hat{\xi}_j^2-\ln(1-2u\lambda_j)],\\ &H(u)=\sum_{j=1}^{n}\frac{\hat{B}_{jj}}{1-2u\lambda_j}+\sum_{j=1}^{n}\sum_{k=1}^{n}\frac{\hat{B}_{jk}\hat{\xi}_j\hat{\xi}_k}{(1-2u\lambda_j)(1-2u\lambda_k)},\\ &\hat{B}=(LP)'B(LP),\\ &\hat{\xi}=(LP)^{-1}\xi=P'L^{-1}\xi.\end{aligned} \tag{10}$$

The quantities $\hat{\xi}_j$ and $\hat{B}_{jk}$, $j\neq k$, are determined only to within an arbitrary sign because the sign of p_j and the signs of the diagonal elements of L are arbitrary. However $\hat{B}_{jj}$ and $\hat{B}_{jk}\hat{\xi}_j\hat{\xi}_k$ are uniquely determined, assuming no repeated eigenvalues. Although y does not appear explicitly in the integral (9) for $p(y)$, the values of λ_j, $\hat{\xi}_j$, and $\hat{B}_{jk}$ depend on y as well as on A, B, and V.

A similar development starting with equation (A.3) leads to

$$Q(y)=\frac{1}{2\pi i}\int_{-i\infty}^{i\infty}\frac{G(u)du}{u}, \tag{11}$$

where the path for u is indented to the right at $u=0$.

4. Numerical evaluation of the integral for $p(y)$. Since it is difficult to evaluate $p(y)$ and $Q(y)$ analytically except in special cases, we are led to consider the problem of evaluating the integrals (9) and (11) by numerical integration. We shall speak mostly about $p(y)$ but $Q(y)$ can be dealt with in much the same manner.

The symmetry of its integrand about the real u-axis allows us to write the integral (9) for $p(y)$ as

$$p(y)=\text{Real}\,\frac{1}{\pi i}\int_0^{i\infty}G(u)H(u)\,du. \tag{12}$$

In some cases most of the contribution to the value of the integral will come from the region around $u=0$ and any convenient numerical integration method can be used. However, in other cases the integrals converge slowly and the following procedures are useful. It should be noted that $p(y)$ is zero when either $\lambda_1<0$ or $\lambda_n>0$. Also, the performance of the numerical integration may deteriorate when y is near a value that makes the determinant of $A-yB$ equal to zero (Appendix D).

a. *Shifting the path.* The path of integration in (9) can be shifted by an arbitrary amount as long as it does not pass over a singularity of $G(u)$. Let the shifted path cross the real u-axis at $u=u_0$. Then, just as in the special case (12) where $u_0=0$, we can write

$$p(y)=\text{Real}\,\frac{1}{\pi i}\int_{u_0}^{i\infty}G(u)H(u)\,du. \tag{13}$$

A good choice of u_0 is the point on the real axis where $G(u)=\exp[\phi(u)]$ is a minimum, i.e., a saddle point of $\exp[\phi(u)]$. Accordingly we take u_0 to be the appropriate root (which may have to be obtained numerically) of

$$\frac{d\phi(u)}{du}=0, \tag{14}$$

where

$$\frac{d\phi(u)}{du}=\sum_{j=1}^{n}[\xi_j^2(1-2u\lambda_j)^{-2}+(1-2u\lambda_j)^{-1}]\lambda_j. \tag{15}$$

Saddle point theory suggests the change of variable $u=u_0+ibv$, where

$$b=\sqrt{2/\phi''(u_0)} \tag{16}$$

and $\phi''(u)$ is the derivative of (15). This takes (13) into

$$p(y)=\frac{b}{\pi}\int_0^{\infty}\text{Real}\,[G(u)H(u)]\,dv,\qquad u=u_0+ibv. \tag{17}$$

The integral (17) can be evaluated efficiently by the trapezoidal rule when most of the contribution to its value comes from the region around $v=0$. In this case good first trial values of the spacing $h=\Delta v$ are $h=1.0,\,0.5,\,0.25$.

The integral for $Q(y)$ corresponding to (17) is

$$Q(y) = \text{Real}\,\frac{b}{\pi}\int_0^\infty \frac{G(u)\,dv}{u}, \tag{18}$$

where the contribution ($=1$) of the pole at $u=0$ must be added to the right-hand side when $u_0<0$.

b. *A numerical integration formula when $\hat{\xi}'\hat{\xi}$ is not large.* When none of the transformed mean values $\hat{\xi}_j$ is large, the ultimate $O(1/u^{1+n/2})$ decrease of the integrand in (17) sets in early and a transformation suited to this type of decrease can be used (see [7, § V]). Thus the transformed version of (17), namely,

$$\begin{aligned} p(y) &= \frac{b}{\pi}\int_{-\infty}^{\infty} \text{Real}\,[G(u)H(u)]\left(\frac{dv}{dx}\right)dx,\\ u &= u_0 + ibv,\\ v &= \exp[2n^{-1/2}e^x - n^{1/2}e^{-x}],\\ \frac{dv}{dx} &= v[2n^{-1/2}e^x + n^{1/2}e^{-x}], \end{aligned} \tag{19}$$

can be efficiently evaluated by the trapezoidal rule if the $\hat{\xi}_j$'s are not large. The behavior of the integrand in (19) for large x suggests that truncation at $x=\pm x_1$, where

$$x_1 = \ln(Mn^{-1/2}) \tag{20}$$

gives a truncation error of $O[\exp(-M)]$. Actually a slightly larger value of x_1 may be needed to reduce the error to $\exp(-M)$.

The decreasing sequence $h = x_1/10,\ x_1/20,\ x_1/40$ of trapezoidal rule spacings, $h=\Delta x$, was used in the calculations for the example discussed below in § 5.

c. *Numerical integration when $\hat{\xi}'\hat{\xi}$ is large.* In this case the numerical integration is complicated by the fact that the integrand may oscillate rapidly as Imag (u) increases and the final $O(1/u^{1+n/2})$ decrease may not set in until u is beyond the region of significant contribution. If the numerical integration methods described above perform poorly, it may be necessary to study the integrand in order to find a suitable method.

A procedure that we have found useful is to first calculate the path of steepest descent (of $\exp[\phi(u)]$) from the saddle point u_0, i.e., the path along which $\text{Imag}\,[\phi(u)]=0$ and $\text{Real}\,[\phi(u)]$ decreases. This path can be calculated step by step by using the recurrence relation (Appendix of [7])

$$\begin{aligned} u_m &= u_{m-1} + d_{m-1}, \qquad m = 1, 2, 3, \cdots,\\ d_m &= -|\phi'(u_m)|\Delta/\phi'(u_m), \end{aligned} \tag{21}$$

starting with $m=1$ and $d_0 = i\Delta$. Here Δ is a small arbitrary step length, $\phi'(u)$ is the $d\phi(u)/du$ given by (15), and u_0 is the appropriate solution of $\phi'(u)=0$.

At the same time the path is calculated, we can also calculate $\phi(u_m)$ and the approximation

$$\begin{aligned} p(y) &\approx \sum_{m=1}^{\infty} \text{Real}\,[(u_m - u_{m-1})(f_m + f_{m-1})/2],\\ f_m &= \exp[\phi(u_m)]\frac{H(u_m)}{\pi i}. \end{aligned} \tag{22}$$

We suppose that we can choose a constant angle θ, $0<\theta<\pi$, such that the path of steepest descent is roughly described by the straight line

$$u=u_0+vb\,e^{i\theta}, \tag{23}$$

where b is given by (16) and v runs from 0 to ∞. Now we can use a generalization of (19),

$$p(y)=\int_{-\infty}^{\infty}\text{Real}\left[\frac{G(u)H(u)b\,e^{i\theta}}{\pi i}\right]\left(\frac{dv}{dx}\right)dx, \qquad u=u_0+vb\,e^{i\theta}, \tag{24}$$

where v and dv/dx are again defined as functions of x by the last two equations in (19). In evaluating (24) it may be necessary to use values of the trapezoidal rule spacing ($h=\Delta x$) somewhat smaller than those mentioned for (19).

5. Numerical integration example. Here the calculation of $p(y)$ is discussed for the case

$$y=\frac{2z_1z_2}{2z_1^2+z_2^2}, \tag{25}$$

where z_1 and z_2 are normal variables with covariance matrix and mean values

$$V=\begin{bmatrix}1 & .5\\ .5 & 1\end{bmatrix},\qquad \xi=\begin{bmatrix}1.25\\ 1.75\end{bmatrix}. \tag{26}$$

First we outline the steps in the numerical integration method of calculating $p(y)$ described in §§ 3 and 4. In this work we take y to have the typical value $y=0.5$. Then we apply the results of Appendix D to examine the discontinuities of $p(y)$. It turns out that $p(y)$ becomes infinite at $y=\pm 1/\sqrt{2}$. Note that $p(y)$ for (25) can also be calculated by the "direct method" (Appendix B).

In the notation of § 3, the matrices A, B, and $\hat{A}$ associated with (25) are

$$A=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix},\quad B=\begin{bmatrix}2 & 0\\ 0 & 1\end{bmatrix},\quad \hat{A}=\begin{bmatrix}-2y & 1\\ 1 & -y\end{bmatrix}, \tag{27}$$

and one of the choices for L in $V=LL'$ leads to

$$L=\frac{1}{2}\begin{bmatrix}2 & 0\\ 1 & \sqrt{3}\end{bmatrix},\qquad L'\hat{A}L=\frac{1}{4}\begin{bmatrix}4-9y & (2-y)\sqrt{3}\\ (2-y)\sqrt{3} & -3y\end{bmatrix}. \tag{28}$$

We want to use the transformed version (19) of the integral (9) to calculate $p(y)$ for $y=0.5$. When $y=0.5$ the eigenvalues of $L'\hat{A}L$ and the corresponding eigencolumn matrix are

$$\lambda_1=.411,\quad \lambda_2=-.991,\quad P=\begin{bmatrix}.771 & -.637\\ .637 & .771\end{bmatrix}.$$

Also the transformation matrix LP and the transformed matrix $\hat{B}=(LP)'B(LP)$ are

$$LP=\begin{bmatrix}.771 & -.637\\ .937 & .349\end{bmatrix},\qquad \hat{B}=\begin{bmatrix}2.067 & -.655\\ -.655 & .933\end{bmatrix}.$$

Solving the equations $(LP)\hat{\xi}=\xi$ gives $\hat{\xi}_1=1.791$ and $\hat{\xi}_2=0.206$.

At this stage we know the values of the quantities λ_j, $\hat{\xi}_j$, and $\hat{B}_{jk}$ that appear in the integral (9) for $p(y)$ and in equation (14) for the saddle point u_0. The values of u_0 and b are found to be $u_0=0.166$ and $b=0.600$.

Evaluation of the integral (19) by the trapezoidal rule with truncation at $x = \pm x_1$ where $x_1 = 2.65$ (from (20) with $n = 2$ and $M = 20$) gives Table 1. Here $h = \Delta x$ is the spacing and $N = 2x_1/h + 1$ is the number of terms in the trapezoidal sum. From the last line in Table 1 we conclude that

(29) $$p(0.5) = 0.9751646\cdots.$$

When $p(y)$ is calculated in this way for a range of values of y, it is found to vanish if $|y| > 1\sqrt{2}$ and to resemble a skewed letter U in the region $|y| < 1/\sqrt{2}$.

TABLE 1

N	h	$p(0.5)$
21	.265	.9749 8094
41	.133	.9751 6465
81	.066	.9751 6464

Now we consider the discontinuities of $p(y)$. According to (D.1), $p(y)$ or its derivatives higher than some order are discontinuities at the zeros of $\det \hat{A}$, i.e., from (27), at $y = \pm 1/\sqrt{2}$. We shall examine the behavior of $p(y)$ around $y = 1/\sqrt{2}$ by following the steps outlined in Appendix D.

The eigenvalues λ_1 and λ_2 of $V\hat{A}$, and of $L'\hat{A}L$, can be shown to be the roots of the characteristic equation

(30) $$\lambda^2 + (3y-1)\lambda + \tfrac{3}{4}(2y^2-1) = 0,$$

and therefore when $y = 1/\sqrt{2}$, λ_1 is zero and $\lambda_2 = -(3/\sqrt{2}-1)$. When, as in step (i) in Appendix D, we replace the integrand in (9) by the form it assumes when $u \to \pm i\infty$ and λ_1 is slightly greater than 0, we get an approximation for $p(y)$ that holds when y is slightly less than $1/\sqrt{2}$:

(31) $$p(y) \approx \frac{\exp[-\hat{\xi}_1^2/2 - \hat{\xi}_2^2/2]}{2\pi i} \int_{-i\infty}^{i\infty} du \frac{\exp[(\hat{\xi}_1^2/2)/(1-2u\lambda_1)]}{(1-2u\lambda_1)^{1/2}(1-2u\lambda_2)^{1/2}} \cdot \left(\frac{\hat{B}_{11}}{1-2u\lambda_1} + \frac{\hat{B}_{11}\hat{\xi}_1^2}{(1-2u\lambda_1)^2}\right).$$

Here $\hat{B}_{11}$ and the transformation matrix LP connecting ξ and $\hat{\xi}$ are calculated from V and from $\hat{A}$ at $y = 1/\sqrt{2}$. Equation (31) suggests that we consider the integral

(32) $$J_\mu = \frac{1}{2\pi i} \int_{-i\infty}^{i\infty} du \frac{\exp[a/(1-2u\lambda_1)]}{(1-2u\lambda_1)^\mu (1-2u\lambda_2)^{1/2}},$$

where $a = \frac{1}{2}\hat{\xi}^2$ and μ is either $\frac{3}{2}$ or $\frac{5}{2}$. J_μ can be evaluated by using (D.5) and (D.6) or by expanding the exponential function and using (E.1). The confluent hypergeometric series in $J_{3/2}$ and $J_{5/2}$ can be combined to given an error function, and the last term in (30) shows that $-\lambda_1\lambda_2 = 3(1-2y^2)/4$ (cf. (D.4)). The result is the approximation

(33) $$p(y) \approx \frac{\exp[-\hat{\xi}_1^2/2 - \hat{\xi}_2^2/2]}{\sqrt{1-2y^2}} \frac{2\hat{B}_{11}}{\pi\sqrt{3}} \left[1 + \sqrt{\frac{\pi}{2}}\hat{\xi}_1 e^{\hat{\xi}_1^2/2} \operatorname{erf}\left(\frac{\hat{\xi}_1}{\sqrt{2}}\right)\right],$$

which holds when $y \to 1/\sqrt{2}$ from below. The value of $\hat{B}_{11}$ is found to be $3/(3-\sqrt{2})$.

The above derivation of (33) illustrates the general procedure discussed in Appendix D. In the special case of our example, n is equal to 2 and (33) can be obtained more readily from Appendix B.

6. Examples of $p(y)$. Here examples are presented to show some of the forms that $p(y)$ can take. Some, and possibly all, of them are known. The z's are assumed to be independent normal random variables with zero means and unit variances.

a. $y = z_1z_2/z_1^2$. If z_1 and z_2 are regarded as rectangular coordinates, changing to polar coordinates gives $y = \tan\theta$ and

$$p(y) = \pi^{-1}/(1+y^2). \tag{34}$$

b. $y = (2z_1z_2 + 2z_3z_4 + \cdots + 2z_{2k-1}z_{2k})/(z_1^2 + z_2^2 + \cdots + z_{2k}^2)$. With the help of (9) and Appendix E we get

$$p(y) = \frac{1}{2^{k-1}} \frac{\Gamma(k)}{[\Gamma(k/2)]^2}(1-y^2)^{k/2-1}, \qquad |y|<1, \tag{35}$$

and $p(y) = 0$ for $|y|>1$. Special cases for $|y|<1$ are

$$\begin{aligned} k&=1, \quad & p(y) &= \pi^{-1}(1-y^2)^{-1/2},\\ k&=2, & p(y) &= \tfrac{1}{2},\\ k&=3, & p(y) &= 2\pi^{-1}(1-y^2)^{1/2}. \end{aligned}$$

c. $y = (z_1^2 + z_2^2 + \cdots + z_{2k}^2)/(2z_1z_2 + \cdots + 2z_{2k-1}z_{2k})$. This case is just the reciprocal of Case b. We have $p(y) = 0$ for $|y|<1$ and, from (A.4),

$$p(y) = \frac{1}{2^{k-1}} \frac{\Gamma(k)}{[\Gamma(k/2)]^2}|y|^{-k}(y^2-1)^{k/2-1}, \qquad |y|>1. \tag{36}$$

In particular, when $k=2$, $p(y) = 1/(2y^2)$ for $|y|>1$.

d. $y = 2z_1z_2/(z_3^2 + z_4^2 + \cdots + z_n^2)$, $n \geqq 3$. $p(y)$ is an even function of y. If $y>0$,

$$\begin{aligned} p(y) &= \frac{n-2}{4\pi i}\int_{-i\infty}^{i\infty}(1-u^2)^{-1/2}(1+uy)^{-n/2}\,du \\ &= \frac{n-2}{2}\frac{\Gamma(n/2)(1+y)^{-n/2}}{\Gamma(1/2)\Gamma(n/2+1/2)}F\left(\frac{n}{2},\frac{1}{2};\frac{n+1}{2};\frac{1-y}{1+y}\right). \end{aligned} \tag{37}$$

The properties of hypergeometric functions given by [8, formulas 15.1.21 and 15.3.10] can be used to show that when $y \to \infty$,

$$p(y) \to \frac{n-2}{2}\frac{\Gamma(n/2)}{\Gamma^2(n/4+1/2)}(2y)^{-n/2},$$

and when $y \to +0$,

$$p(y) \to \frac{n-2}{\pi}\begin{bmatrix} \dfrac{1}{2}\ln\left(\dfrac{8}{y}\right) - \left(1+\dfrac{1}{3}+\cdots+\dfrac{1}{n-2}\right), n \text{ odd} \\ \dfrac{1}{2}\ln\left(\dfrac{2}{y}\right) - \left(\dfrac{1}{2}+\dfrac{1}{4}+\cdots+\dfrac{1}{n-2}\right), n \text{ even} \end{bmatrix}.$$

In the special case $n=3$

$$p(y) = (K-E)/[\pi(1+y)^{3/2}m],$$

where $m=(1-y)/(1+y)$ and K and E are complete elliptic functions with parameter m [8, § 17.3].

e. $y=(z_n^2+z_{n-1}^2+\cdots+z_{m+1}^2)/(z_m^2+\cdots+z_1^2)$, $1\leqq m\leqq n-1$. Here $y\geqq 0$ and

$$p(y)=\frac{\Gamma(n/2)}{\Gamma(m/2)\Gamma(n/2-m/2)}y^{(n-m)/2-1}(1+y)^{-n/2}. \tag{38}$$

f. $y=(7z_1^2+4z_2^2+2z_3^2+z_4^2)/(z_1^2+z_2^2+z_3^2+z_4^2)$. Equation (D.1) indicates that $p(y)$ has irregularities at $y=1$, 2, 4, 7. Numerical integration shows that $p(y)$ is continuous and is zero outside of $1\leqq y\leqq 7$. Between 2 and 4, $p(y)$ is constant and equal to $.2727998\cdots$ (a surprise), and $dp(y)/dy$ becomes very large near the ends of the intervals $1<y<2$ and $4<y<7$. The procedure described in Appendix D shows that near $y=1$, $dp(y)/dy\to\pi^{-1}(y-1)^{-1/2}/(6\cdot 3\cdot 1)^{1/2}$. Similar behavior occurs at 2, 4, and 7.

g. *$n=3$ or 4, $\xi=0$, and $z'Bz$ positive definite.* In these cases the integral (9) for $p(y)$ is an elliptic integral [8, § 17.1]. However, we have not studied this aspect of $p(y)$.

Appendix A. Some known or readily derived results regarding the distribution of the ratio of two arbitrary functions. Here the probability density $\hat{p}(z_1,z_2,\cdots,z_n)$ is not necessarily normal, and the real functions $\Phi_1(z_1,z_2,\cdots,z_n)$ and $\Phi_2(z_1,z_2,\cdots,z_n)$ need not be quadratic forms. These functions of the z's will be written as $\hat{p}(z)$, Φ_1, and Φ_2, and are assumed to be such that the results hold. The following expressions for the density $p(y)$ and the probability $Q(y)$ that Φ_1/Φ_2 exceeds y can be derived by starting with

$$p(y)=\frac{1}{2\pi}\int_{-\infty}^{\infty}dv\int_R dz\,\hat{p}(z)\exp\left[-ivy+iv\frac{\Phi_1}{\Phi_2}\right] \tag{A.1}$$

and making changes of variable of the type $v=t\Phi_2$, $dv=\Phi_2\,dt$. In (A.1), $dz=dz_1\,dz_2\cdots dz_n$ and R is the n-dimensional range associated with $\hat{p}(z)$.

a. When $\Phi_2\geqq 0$ throughout R (see Cramér [6, Prob. 6, p. 317]),

$$p(y)=\frac{1}{2\pi}\int_{-\infty}^{\infty}dt\int_R dz\;\Phi_2\hat{p}(z)\exp[it(\Phi_1-y\Phi_2)]=T(y) \tag{A.2}$$

and

$$Q(y)=\frac{1}{2\pi i}\int_{-\infty}^{\infty}dt\,t^{-1}\int_R dz\,\hat{p}(z)\exp[it(\Phi_1-y\Phi_2)]=S(y), \tag{A.3}$$

where the path of integration for t in (A.3) is indented downwards at $t=0$.

b. When $\Phi_1\geqq 0$ throughout R we can set $y_r=1/y=\Phi_2/\Phi_1$ and calculate its density $p_r(y_r)$ from (A.2). Then

$$p(y)=p_r\left(\frac{1}{y}\right)\Big/y^2. \tag{A.4}$$

We also have the relations

$$\begin{aligned} &p(y)=T(y),\quad y>0, && p(y)=-T(y),\quad y<0,\\ &Q(y)=S(y)-P_-,\quad y>0, && 1-Q(y)=S(y)-P_+,\quad y<0\end{aligned} \tag{A.5}$$

where the probabilities

$$P_\pm=\int_{R_\pm}\hat{p}(z)\,dz$$

are independent of y, and R_+ is the part of R in which $\Phi_2 > 0$ and R_- is the part in which $\Phi_2 < 0$.

c. When both Φ_1 and Φ_2 change sign in R we can obtain

$$p(y) = \frac{1}{\pi} \int_0^\infty dt \int_R dz\, |\Phi_2| \hat{p}(z) \cos(t\Phi_1 - t\Phi_2 y) \tag{A.6}$$

by inverting the order of integration in (A.1), making the change of variable $v = |\Phi_2|t$, $dv = |\Phi_2|\,dt$, and taking the real part. The presence of $|\Phi_2|$ in (A.6) complicates the integration with respect to $z_1, z_2, \cdots, z_n$. It can be removed at the cost of introducting another integration by substituting

$$|\Phi_2| = \frac{2\Phi_2}{\pi} \int_0^\infty \sin(t\Phi_2 x) \frac{dx}{x}$$

and writing the product of the sine and cosine as the difference of two sines. This gives

$$p(y) = \frac{1}{\pi^2} \int_0^\infty dt \int_0^\infty \frac{dx}{x} \operatorname{Imag}[f(t, y-x) - f(t, y+x)], \tag{A.7}$$

where

$$f(t, w) = \int_R dz\, \Phi_2 \hat{p}(z) \exp[it(\Phi_1 - w\Phi_2)].$$

We also have the similar result

$$Q(y) = \frac{1}{2} - \frac{1}{\pi^2} \int_0^\infty \frac{dt}{t} \int_0^\infty \frac{dx}{x} \operatorname{Real}[F(t, y-x) - F(t, y+x)], \tag{A.8}$$

where

$$F(t, w) = \int_R dz\, \hat{p}(z) \exp[it(\Phi_1 - w\Phi_2)].$$

Appendix B. Probability density of the ratio of two unrestricted quadratic forms in normal variables z_1 and z_2. Let z_1 and z_2 be normally distributed with unit variances, correlation coefficient ρ, and means ξ_1, ξ_2. An expression for $p(y)$, where

$$y = (a_{11}z_1^2 + 2a_{12}z_1z_2 + a_{22}z_2^2)/(b_{11}z_1^2 + 2b_{12}z_1z_2 + b_{22}z_2^2), \tag{B.1}$$

can be obtained directly by integrating the joint density $\hat{p}(z_1, z_2)$ over the infinitesimal region (in the z_1, z_2 plane) in which the right-hand side of (B.1) lies between y and $y + dy$. Here the denominator, $z'Bz$, is *not* required to be definite.

It is found that

$$p(y) = \hat{p}(\xi_1, \xi_2) \sum_{j=1}^{2} C_j \left[1 + F\left(\frac{b_j}{a_j^{1/2}}\right)\right], \tag{B.2}$$

where

$$C_j = \frac{1}{a_j} \left| \frac{d}{dy} x_j \right|, \qquad a_j = \frac{1 - 2\rho x_j + x_j^2}{2(1-\rho^2)},$$

$$b_j = \frac{(\rho x_j - 1)\xi_1 + (\rho - x_j)\xi_2}{2(1-\rho^2)},$$

$$F(v) = \pi^{1/2} v \exp(v^2) \operatorname{erf}(v),$$

in which x_j, $j = 1, 2$, are functions of y given by the roots of

$$(a_{22} - yb_{22})x^2 + 2(a_{12} - yb_{12})x + (a_{11} - yb_{11}) = 0. \tag{B.3}$$

When the roots are complex, $p(y)$ is zero.

Remarks. The term within square brackets in (B.2) appears because it is the value of

$$a \int_0^\infty t e^{-at^2}(e^{2bt} + e^{-2bt})\, dt.$$

The combination of functions comprising $F(v)$ appears in an expression given by Kanter [5] for $p(y)$ when $y = z_1 z_2 / z_2^2$ (in our notation). Kanter presents curves of $p(y)$ for several values of ρ and a wide range of ξ_1 and ξ_2.

Appendix C. Details of the integration sketched in § 3. The integral (6) for $p(y)$ can be reduced to the integral (9) in § 3 with the help of the transformations

$$z = LP\hat{z}, \qquad \xi = LP\hat{\xi}, \tag{C.1}$$

where L and P are the $n \times n$ matrices in (7) and (8). By using

$$P' = P^{-1}, \qquad (L'\hat{A}L)P = P\Lambda, \tag{C.2}$$

where $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n)$, it can be shown that

$$\begin{aligned} z'V^{-1}z &= \hat{z}'\hat{z}, & \xi'V^{-1}z &= \hat{\xi}'\hat{z}, \\ z'\hat{A}z &= \hat{z}'\Lambda\hat{z}, & \xi'V^{-1}\xi &= \hat{\xi}'\hat{\xi}. \end{aligned} \tag{C.3}$$

Furthermore,

$$\frac{\partial(z_1, z_2, \cdots, z_n)}{\partial(\hat{z}_1, \hat{z}_2, \cdots, \hat{z}_n)} = |LP| = |L|, \qquad |V| = |L|^2,$$

and the transformations $z = LP\hat{z}$ carries the quadratic form $z'Bz$ into

$$\hat{z}'\hat{B}\hat{z} = \sum_{j,k=1}^{n} \hat{B}_{jk}\hat{z}_j\hat{z}_k,$$

where

$$\hat{B} = (LP)'B(LP). \tag{C.4}$$

The relations just given enable us to write the integral (6) as

$$\begin{aligned} p(y) = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} du \int_{-\infty}^{\infty} d\hat{z}_1 \cdots \int_{-\infty}^{\infty} d\hat{z}_n\, (2\pi)^{-n/2}(\hat{z}'\hat{B}\hat{z}) \\ \cdot \exp\left[\sum_{j=1}^{n}\left\{-\frac{1}{2}(1 - 2u\lambda_j)\hat{z}_j^2 + \hat{\xi}_j\hat{z}_j - \frac{1}{2}\hat{\xi}_j^2\right\}\right]. \end{aligned} \tag{C.5}$$

The integrations with respect to the $\hat{z}_j$'s can now be performed by using

$$\int_{-\infty}^{\infty} x^m \exp\left[-\frac{1}{2}\alpha x^2 + \beta x\right] dx = g_m\left(\frac{2\pi}{\alpha}\right)^{1/2} \exp\left[\frac{1}{2}\beta^2\alpha^{-1}\right], \tag{C.6}$$

where $g_0 = 1$, $g_1 = \beta\alpha^{-1}$, and $g_2 = \beta^2\alpha^{-2} + \alpha^{-1}$. The result is the desired integral (9):

$$p(y) = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} G(u)H(u)\, du. \tag{C.7}$$

Appendix D. Discontinuities of $p(y)$. Here we present some heuristic reasoning regarding the discontinuities of $p(y)$. First we show that if $y=b$ is a root of the determinantal equation $\det \hat{A}=0$, i.e., of

$$\text{(D.1)} \qquad |A-yB|=0,$$

then either $p(y)$ or its derivatives higher than some order are discontinuous at $y=b$.

When $y=b$ is a root of (D.1) the characteristic equation

$$\text{(D.2)} \qquad |\lambda I-L'\hat{A}L|=0$$

shows that at least one of the eigenvalues, say λ_k, is zero. Then the factor $(1-2u\lambda_k)$ in the integral (9) for $p(y)$ is unity and the integrand does not decrease at the usual $O(1/u^{1+n/2})$ rate as $u\to i\infty$. Indeed, the integral or some of its derivatives with respect to y will not converge, thereby indicating a discontinuity at $y=b$. Note that this argument is based on (9) which assumes normal z's and a positive definite $z'Bz$.

A closer examination of the integrand of (9) when λ_k is small and $u\to i\infty$ suggests that if $p(y)$ becomes infinite at $y=b$ it can do so in only the following ways (aside from the spike $p(y)=\delta(y-b)$).

$$\text{(D.3)} \qquad \begin{aligned} &p(y)\to C\ln|y-b|,\\ &p(y)\to C(b-y)^{-1/2} \text{ when } y<b, \text{ and } p(y) \text{ bounded when } y>b,\\ &p(y)\to C(y-b)^{-1/2} \text{ when } y>b, \text{ and } p(y) \text{ bounded when } y<b,\end{aligned}$$

where C is a constant. The $\ln|y-b|$ or $(b-y)^{-1/2}$ behavior occurs when the integrand in (9) is $O(1/u)$ or $O(1/u^{1/2})$, respectively, as $u\to\pm i\infty$ at $\lambda_k=0$. Jumps of finite size can also occur in $p(y)$ when the integrand is $O(1/u)$.

There appears to be no easy way to calculate the constant C in (D.3). A rather lengthy method is to:

(i) replace the integrand in (9) by the form it assumes when $u\to\pm i\infty$ and λ_k is close to 0,
(ii) evaluate the resulting integral by using Appendix E,
(iii) find the limiting form of the value of the integral as $\lambda_k\to 0$; and
(iv) express λ_k in terms of $y-b$ with the help of the characteristic equation (D.2).

Step (ii) is possible because we are assuming that $p(b)$ is infinite and that when $\lambda_k=0$ the integrand is $O(1/u)$ or $O(1/u^{1/2})$ as $u\to\pm i\infty$. If in step (iii) the argument of a hypergeometric function tends to unity as $\lambda_k\to 0$, one of the linear transformations given in [8, § 15.3] can be used to obtain the limiting form. A relation of particular use in step (iv) is

$$\text{(D.4)} \qquad \lambda_1\lambda_2\cdots\lambda_n=(\det V)(\det\hat{A}).$$

As an example of steps (ii) and (iii), suppose that step (i) gives integrals of the form

$$\text{(D.5)} \qquad J=\frac{1}{2\pi i}\int_{-i\infty}^{i\infty}\frac{\exp[a/(1-2u\lambda_k)]\,du}{(1-2u\lambda_k)^{\mu}(1-2u\lambda_s)^{\sigma}(1-2u\lambda_t)^{\tau}},$$

where $\lambda_s>\lambda_k>0>\lambda_t$, $\mu=\frac{3}{2},2,\frac{5}{2},3,\cdots$, and σ and τ are equal to one of $0,\frac{1}{2},1$ subject to $\frac{1}{2}\leqq\sigma+\tau\leqq 1$. When τ is 0 or 1, closing the path of integration on the left shows that $J=O(1)$ as $\lambda_k\to 0$. When $\tau=\frac{1}{2}$, the power series for $\exp(x)$, equation (E.2), and the transformations [8, §§ 15.3.3 and 15.3.10] show that as $\lambda_k\to 0$,

$$\text{(D.6)} \qquad J=\frac{\Gamma(\mu-1/2)}{2\Gamma(\mu)\sqrt{-\pi\lambda_k\lambda_t}}M\left(\mu-\frac{1}{2};\mu;a\right)+O(1), \qquad \tau=\frac{1}{2}, \quad \sigma=0,$$

(D.7) $$J \approx -\frac{1}{2\pi\sqrt{-\lambda_s\lambda_t}}\left[\left(\gamma + \ln\left\{\lambda_k \frac{\lambda_s^{-1} - \lambda_t^{-1}}{4}\right\}\right) e^a + \sum_{n=0}^{\infty} \frac{a^n \psi(\mu+n)}{n!}\right], \qquad \tau = \sigma = \frac{1}{2}.$$

Here $M(\cdot)$ is a confluent hypergeometric function that can be expressed in terms of the error function if μ is half an odd integer, or in terms of modified Bessel functions if μ is an integer [8, §§ 13.4 and 13.6]. In (D.7) $\gamma = 0.5772 \cdots$ is Euler's constant and $\psi(z)$ is the derivative of $\ln \Gamma(z)$. When a becomes large, the summation in the second line of (D.7) is approximately equal to $(\exp a) \ln (\mu + a)$.

The values of the covariance matrix V and the values of the means ξ_i affect the strengths, but not the positions, of the discontinuities. The shape of $p(y)$ is also affected. In particular, if some of the ξ_i's are large and $\hat{y} = \xi' A \xi / \xi' B \xi$ is not close to a discontinuity, $p(y)$ often tends to resemble a normal density with mean $\hat{y}$.

Appendix E. Useful contour integrals. Here we state three integrals that occur in the study of the discontinuities of $p(z)$ and when $L'\hat{A}L$ has two or three (possibly multiple) eigenvalues. The first is a variation of Pochhammer's contour integral for the beta function:

(E.1) $$\frac{1}{2\pi i}\int_L \frac{du}{(u-a)^\alpha (b-u)^\beta} = \frac{\Gamma(\alpha+\beta-1)}{(b-a)^{\alpha+\beta-1}\Gamma(\alpha)\Gamma(\beta)},$$

where Real $(\alpha + \beta) > 1$ and the path of integration L runs from $u = -i\infty$ to $u = i\infty$. The points $u = a$, $u = b$ lie to the left and to the right, respectively, of L. When L does not pass between a and b the integral is zero.

The second integral is

(E.2) $$\begin{aligned}&\frac{1}{2\pi i}\int_L \frac{du}{(u-a)^\alpha (b-u)^\beta (c-u)^\gamma} \\ &\qquad = \frac{\Gamma(\alpha+\beta+\gamma-1)(c-a)^{-\gamma}}{\Gamma(\alpha)\Gamma(\beta+\gamma)(b-a)^{\alpha+\beta-1}} F\left(\gamma, 1-\alpha; \beta+\gamma; \frac{c-b}{c-a}\right),\end{aligned}$$

where Real $(\alpha + \beta - \gamma) > 1$, $F(\cdot)$ is a hypergeometric function, and the path L runs from $-i\infty$ to $+i\infty$ with a on the left and b and c on the right. The arguments of the various factors are determined by continuation from the special case in which all of the arguments are zero when $u = 0$ and a, b, c are real with $a < 0 < b < c$. Equation (E.2) can be obtained by starting with this special case and taking L to be the line Real $u = (a+b)/2$. Writing $c - u$ as $\hat{\alpha}(u-a) + \hat{\beta}(b-u)$, expanding $(c-u)^{-\gamma}$ in powers of $\hat{\alpha}(u-a)/\hat{\beta}(b-u)$ and integrating termwise with the help of (E.1) leads to (E.2).

The third integral is much like the second:

(E.3) $$\begin{aligned}&\frac{1}{2\pi i}\int_L \frac{du}{(a-u)^\alpha (u-b)^\beta (u-c)^\gamma} \\ &\qquad = \frac{\Gamma(\alpha+\beta+\gamma-1)(a-c)^{-\gamma}}{\Gamma(\alpha)\Gamma(\beta+\gamma)(a-b)^{\alpha+\beta-1}} F\left(\gamma, 1-\alpha; \beta+\gamma; \frac{b-c}{a-c}\right).\end{aligned}$$

Now b and c lie to the left of L and a to the right. The arguments are determined by continuation from the special case $c < b < 0 < a$.

REFERENCES

[1] J. GURLAND, *Inversion formula for distribution of ratios*, Ann. Math. Stat., 19 (1948), pp. 228–237.

[2] ———, *Distribution of quadratic forms and ratios of quadratic forms*, Ann. Math. Stat., 24 (1953), pp. 416–427.

[3] J. GURLAND, *Distribution of definite and of indefinite quadratic forms*, Ann. Math. Stat., 26 (1955), pp. 122–127.
[4] ———, *Quadratic forms in normally distributed random variables*, Sankhyā, 17 (1956), pp. 37–50.
[5] I. KANTER, *The ratios of functions of random variables*, IEEE Trans. Aerosp. Electron. Syst., AES-13 (1977), pp. 624–630.
[6] H. CRAMÉR, *Mathematical Methods of Statistics*, Princeton Univ. Press, Princeton, NJ, 1966.
[7] S. O. RICE, *Efficient evaluation of integrals of analytic functions by the trapezoidal rule*, 52 (1973), pp. 707–722.
[8] M. ABRAMOWITZ AND I. A. STEGUN, *Handbook of Mathematical Functions*, Natl. Bur. Stand., Appl. Math. Series No. 55, Washington, DC, 1964.
[9] R. B. DAVIES, *The distribution of a linear combination of Chi-squared random variables*, Algorithm AS155, Appl. Stat. 29, 3 (1980), pp. 323–333.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 2, June 1984

ONE-LEG FORMULAS FOR STIFF ORDINARY DIFFERENTIAL EQUATIONS*

DANIEL S. WATANABE† AND QASIM M. SHEIKH†

Abstract. New one-leg formulas for stiff ordinary differential equations are presented. They are generalizations of the backward differentiation formulas and have regions of instability which potentially are infinite and fill most of the right half-plane. This property makes the formulas more effective than the backward differentiation formulas in detecting unstable problems. Numerical results illustrating their efficiency and stability properties are presented.

Key words. stiff ordinary differential equations, one-leg formulas, region of absolute stability, unstable problems

1. Introduction. There are several effective codes available for solving stiff ordinary differential equations. However, Lindberg [5] observed that these codes will produce reasonable looking solutions which are not close to the desired solution for certain problems. This difficulty can arise in several ways including a stepsize control strategy that selects a stepsize so large that an active region of the solution is missed entirely, or a formula that is stable for an unstable problem and completely ignores increasing components of the solution. Unfortunately, the backward differentiation formulas which are the most widely used stiff formulas suffer from this disadvantage. For example, the backward Euler formula

$$y_n - y_{n-1} = hf(x_n, y_n),$$

when applied to the scalar test problem

$$y' = \lambda y, \qquad y(0) = 1,$$

produces a decreasing numerical solution $\{y_n\}$ if $|h\lambda - 1| > 1$. However, the exact solution $y(x) = e^{\lambda x}$ of the test problem is decreasing only if $\operatorname{Re}(\lambda) < 0$. Lindberg gives test problems for which the eigenvalues change from large negative values to large positive values. Linear multistep formulas which are stable in large portions of the right half-plane fail to detect this change in the eigenvalues unless the stepsize h is extremely small.

Our goal was to develop formulas which are more effective than the backward differentiation formulas in detecting unstable problems. The formulas, however, should be as efficient as the backward differentiation formulas and have essentially the same form to facilitate adapting current production codes to the new formulas. One-leg formulas were first introduced by Dahlquist [1]. This paper presents new one-leg generalizations of the backward differentiation formulas whose regions of instability potentially are infinite and fill most of the right half-plane. The formulas appear to possess the desired properties. In the following sections, we first derive the formulas and describe their regions of absolute stability. We then describe the implementation of the formulas in a Nordsieck formulation essentially identical to that of the backward differentiation formulas. Hence it is relatively easy to adapt Nordsieck formulation codes like DIFSUB to the new formulas. Finally we present numerical results illustrating the efficiency of the formulas and their effectiveness in detecting unstable problems.

* Received by the editors March 30, 1982, and in revised form December 29, 1982. This work was supported in part by the U.S. Department of Energy under contract DE-AC02-76ER02383.

† Department of Computer Science, University of Illinois, Urbana, Illinois 61801.

2. One-leg formulas. Given the initial-value problem

$$y' = f(x, y), \qquad y(0) = y_0, \tag{1}$$

we wish to generate a sequence $\{y_n\}$ which approximates the sequence of exact values $\{y(x_n)\}$ on the equispaced mesh $\{x_n\}$ with fixed stepsize h. Given the k past values $y_{n-1}, y_{n-2}, \cdots, y_{n-k}$, we seek a polynomial $p_k(x)$ of degree k which interpolates these values and the future value y_n and choose y_n so that $p_k(x)$ satisfies the differential equation exactly at the collocation point $x_n + \theta_k h$. Hence

$$p_k(x_{n-i}) = y_{n-i}, \qquad i = 0, 1, \cdots, k, \tag{2}$$

$$p_k'(x_n + \theta_k h) = f(x_n + \theta_k h, p_k(x_n + \theta_k h)). \tag{3}$$

If we write

$$p_k(x) = \sum_{i=0}^{k} y_{n-i} w_i(\theta),$$

where $\theta = (x - x_n)/h$ and $w_i(\theta)$ is the Lagrange weight function

$$w_i(\theta) \equiv \prod_{\substack{j=0 \\ j \neq i}}^{k} (\theta + j)/(j - i),$$

we can write the collocation condition (3) as the k-step one-leg formula

$$\sum_{j=0}^{k} \alpha_j y_{n-j} = hf\left(x_n + \theta_k h, \sum_{j=0}^{k} \beta_j y_{n-j}\right), \tag{4}$$

where the coefficients α_j and β_j are given by

$$\alpha_j \equiv w_j'(\theta_k), \tag{5a}$$

$$\beta_j \equiv w_j(\theta_k). \tag{5b}$$

Note that the choice $\theta_k = 0$ yields the k-step backward differentiation formula. Using the generating polynomials

$$\rho(\xi) = \sum_{j=0}^{k} \alpha_j \xi^{k-j} \quad \text{and} \quad \sigma(\xi) = \sum_{j=0}^{k} \beta_j \xi^{k-j},$$

and the shift operator E defined by $y(x + h) = Ey(x)$, we can write the one-leg formula in the form

$$\rho(E) y_{n-k} - hf(\sigma(E) x_{n-k}, \sigma(E) y_{n-k}) = 0. \tag{6}$$

Dahlquist [2] has analyzed the local discretization error of one-leg formulas with smoothly varying stepsize. Since the analysis is complicated, we shall restrict ourselves to a brief summary of the results relevant to our subclass of formulas. The error consists of two components: the *differentiation* error associated with the operator $\rho(E)$, and the derivative of the *interpolation* error associated with the operator $h\sigma(E)$. Let $h_n = x_{n+1} - x_n$. If $(h_n - h_{n-1})/h_n^2$ is bounded, then the differentiation error is dominant for our subclass of one-leg formulas even if the Jacobian matrix f_y is large in norm. The leading term of the local discretization error has the form $C(\theta) h^{k+1} y^{(k+1)}$, where

$$C(\theta) = (-1)^{k+1} \sum_{j=0}^{k} \alpha_j(\theta)(\theta + j)^{k+1}/(k+1)!.$$

Hence the k-step formula is of order k.

If we apply the one-leg formula (4) to the model equation $y' = \lambda y$, we obtain a difference equation whose characteristic equation is $\rho(\xi) - \mu\sigma(\xi) = 0$, where $\mu \equiv \lambda h$. The locus in the μ-plane for which a root of this equation has magnitude 1 is obtained by plotting $\mu = \rho(e^{i\phi})/\sigma(e^{i\phi})$ for $0 \leqq \phi \leqq 2\pi$. The region of absolute stability is the exterior of this curve. It is desirable to maximize the size of this locus in the right half-plane, particularly near the imaginary axis, in order to increase the probability of detecting instability in the initial-value problem.

We can vary the size of the stability locus by varying the collocation point. The rightmost point of the curve corresponds to $\phi = \pi$, and the locus increases in size as this point increases in magnitude. If we choose the collocation point so that $\sigma(e^{i\phi}) = 0$, then this point is at infinity. However, the region of absolute stability should not include the point at infinity because a formula stable at infinity is unsuited to very stiff problems. Hence we arbitrarily choose the collocation point so that the rightmost point is approximately 2×10^4. There is such a point in each of the k intervals (x_{n-k}, x_{n-k+1}), $(x_{n-k+1}, x_{n-k+2}), \cdots$, and (x_{n-1}, x_n), and we choose the point in the last interval because it is closest to the current mesh point x_n. For $k = 1$ this choice yields a formula whose error constant is almost zero so that it is virtually of order 2. In order to fix the order at 1 to prevent our code from "chattering" between the 1-step and 2-step formulas, we arbitrarily choose $\theta_1 \equiv -0.48$ so that the rightmost point is 50. Figure 1 presents the region of stability for the one-leg formula for $k = 2$. The regions for $k = 3(1)6$ are identical at this scale.

Although the region of absolute instability is large, the amplification factor is essentially 1 in most of the region. A fairer comparison of the one-leg and backward

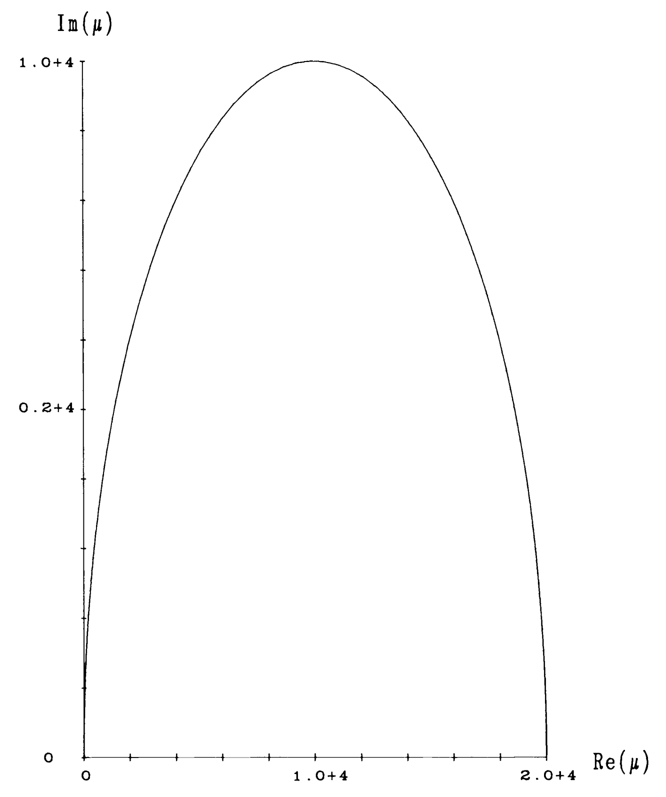


FIG. 1. *Region of absolute stability for 2-step one-leg formula.*

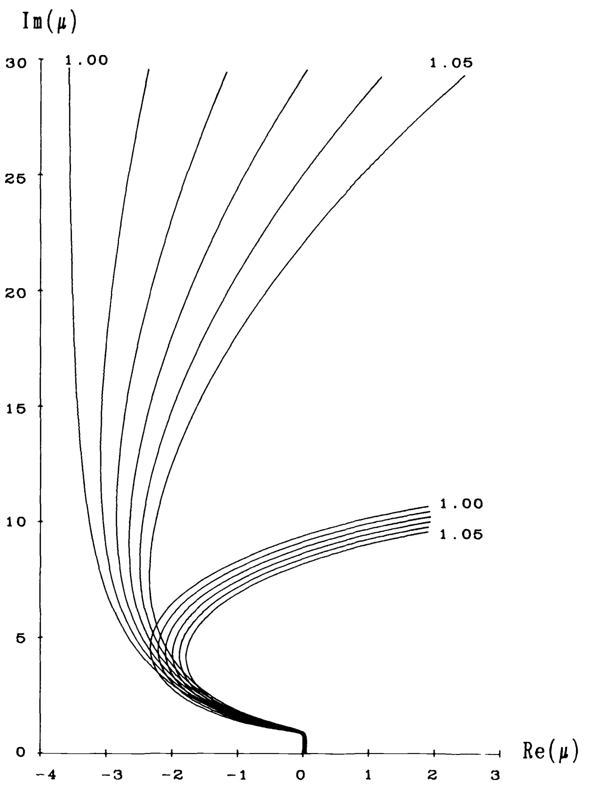


FIG. 2. *Loci for* 5*-step one-leg and backward differentiation formulas for amplification factor* $\zeta = 1.00(0.01)1.05$.

differentiation formulas is obtained by comparing the loci given by $\mu = \rho(\zeta e^{i\phi})/\sigma(\zeta e^{i\phi})$ for $0 \leqq \phi \leqq 2\pi$ and amplification factor $\zeta > 1$. Figure 2 presents the loci for the 5-step one-leg and backward differentiation formulas for $\zeta = 1.00(0.01)1.05$. The loci for a given value of ζ for the k-step one-leg formulas corresponding to the k possible collocation points tend to increase in size as $|\theta + k/2|$ increases. Hence our choice of collocation point has the added advantage of yielding larger loci.

3. Implementation. Let $\mathbf{y}_n \equiv [y_n, y_{n-1}, \cdots, y_{n-k}]^T$ be the vector of saved information for the k-step formula. An initial approximation $\mathbf{y}_{n,(0)}$ to $\mathbf{y}_n$ is predicted by extrapolating the polynomial defined by $\mathbf{y}_{n-1}$ to x_n. This approximation is corrected using $\mathbf{y}_n \equiv \mathbf{y}_{n,(0)} + \omega\mathbf{l}$, where $\mathbf{l} \equiv [1, 0, \cdots, 0]^T$ and ω is chosen to satisfy the one-leg formula(4) written in the form $F(\mathbf{y}_{n,(0)} + \omega\mathbf{l}) = 0$, where

$$F(\mathbf{y}) = hf(x_n + \theta_k h, p_k(x_n + \theta_k h)) - hp_k'(x_n + \theta_k h)$$
$$= hf\left(x_n + \theta_k h, \sum_{j=0}^{k} \beta_j y_{n-j}\right) - \sum_{j=0}^{k} \alpha_j y_{n-j}.$$

The Nordsieck representation of the saved information $\mathbf{a}_n \equiv [y_n, hy_n^{(1)}, \cdots, h^k y_n^{(k)}/k!]^T$ is convenient to use in practice because it facilitates changing the stepsize. We can transform the preceding formulation using the matrix T for which $\mathbf{a}_n = T\mathbf{y}_n$. The predictor assumes the form $\mathbf{a}_{n,(0)} = P\mathbf{a}_{n-1}$, where P is the Pascal triangle matrix with $p_{ij} = \binom{j}{i}$. The vector $\mathbf{l}$ is transformed into the first column of T. However, it is convenient to define the scaled vector $\mathbf{l}_N = T\mathbf{l}/c(\theta_k)$, where the scaling factor c

will be chosen to simplify the equations. The corrector assumes the form $\mathbf{a}_n = \mathbf{a}_{n,(0)} + \omega \mathbf{l}_N$, where ω is chosen to satisfy $F(\mathbf{a}_{n,(0)} + \omega \mathbf{l}_N) = 0$, where

$$F(\mathbf{a}) \equiv hf\left(\sum_{j=0}^{k} (\theta_k)^j (\mathbf{a})_j\right) - \sum_{j=0}^{k} j(\theta_k)^{j-1}(\mathbf{a})_j, \tag{7}$$

and $(\mathbf{a})_i$ is the ith component of the vector $\mathbf{a}$. This nonlinear equation is solved by the Newton iteration

$$\omega_{(m+1)} = \omega_{(m)} - \left[\left(\frac{\partial F}{\partial \mathbf{a}}\right) \cdot \mathbf{l}_N\right]^{-1} F(\mathbf{a}_{n,(0)} + \omega_{(m)} \mathbf{l}_N).$$

By setting $\mathbf{a}_{n,(m)} \equiv \mathbf{a}_{n,(0)} + \omega_{(m)} \mathbf{l}_N$, we can rewrite this equation as

$$\mathbf{a}_{n,(m+1)} = \mathbf{a}_{n,(m)} - \left[\left(\frac{\partial F}{\partial \mathbf{a}}\right) \cdot \mathbf{l}_N\right]^{-1} F(\mathbf{a}_{n,(m)}) \mathbf{l}_N. \tag{8}$$

The Jacobian for this iteration is

$$\left(\frac{\partial F}{\partial \mathbf{a}}\right) \cdot \mathbf{l}_N = h \sum_{j=0}^{k} (\theta_k)^j (\mathbf{l}_N)_j \frac{\partial f}{\partial y} - \sum_{j=1}^{k} j(\theta_k)^{j-1}(\mathbf{l}_N)_j. \tag{9}$$

By convention $\mathbf{l}_N = T\mathbf{l}$ and $(\mathbf{l}_N)_1 = 1$ for the backward differentiation formula, and the second term on the right reduces to 1 because $\theta_k = 0$. Hence it is convenient to choose the scaling factor

$$c(\theta_k) \equiv \sum_{j=1}^{k} j(\theta_k)^{j-1}(T\mathbf{l})_j \tag{10}$$

so that this term is identically 1. This scaled formulation is essentially identical to the Nordsieck formulation of the backward differentiation formulas, the principal difference being the two extra summations required to evaluate $F(\mathbf{a})$. Hence it is relatively easy to adapt a code like DIFSUB to the one-leg formulas.

Table 1 lists θ and $c(\theta)$ for the k-step formulas for $k = 1(1)6$. The coefficients $\mathbf{l}_N$ are easily computed using the well-known coefficients $T\mathbf{l}$ of the backward differentiation formulas [4].

TABLE 1
Parameters θ and $c(\theta)$ for k-step formulas.

k	θ	$c(\theta)$
1	−0.4800000000000000+0	1.0000000000000000
2	−0.2928432170456855+0	0.8047711886362097
3	−0.1770743416328997+0	0.8153794425272162
4	−0.1075602822332437+0	0.8562575824253686
5	−0.6490390333680813−1	0.8972668593024493
6	−0.3869806056420695−1	0.9305954894447051

For a system of equations, the Newton iteration (8) requires the solution of the linear system of equations $J\mathbf{x} = \mathbf{b}$, where the Jacobian matrix J must be updated whenever the stepsize h is changed. This updating can be performed efficiently for systems of moderate size using a scheme given by Enright [3]. We write the Newton iteration (8) as

$$\mathbf{a}_{n,(m+1)} = \mathbf{a}_{n,(m)} - \frac{1}{hq}\left[\frac{\partial \mathbf{f}}{\partial \mathbf{y}} - \left(\frac{1}{hq}\right) I\right]^{-1} F(\mathbf{a}_{n,(m)}) \mathbf{l}_N,$$

and set $J = \partial\mathbf{f}/\partial\mathbf{y} - (1/hq)I$. Instead of using the typical LU factorization, we use the factorization $J = LHL^{-1}$, where L is a lower triangular matrix and H is an upper Hessenberg matrix. If the stepsize is changed from h to $\hat{h}$, then

$$\frac{\partial\mathbf{f}}{\partial\mathbf{y}} - \left(\frac{1}{\hat{h}q}\right)I = L\left(H + \left(\frac{1}{hq} - \frac{1}{\hat{h}q}\right)I\right)L^{-1}.$$

Hence only the diagonal elements of H need be changed to accommodate the stepsize change.

4. Numerical examples. We tested the one-leg formulas on a variety of problems, and the following examples are typical. Krogh [4] has proposed the following nonlinear stiff test problem. The nonlinear differential equations

$$z_i' = -\beta_i z_i + z_i^2,$$

for $i = 1, \cdots, 4$, have the solutions

$$z_i = \beta_i/(1 + c_i e^{\beta_i x}).$$

If the initial value $z_i(0) = -1$, then $c_i = -1 - \beta_i$. If we set $\mathbf{y} = U\mathbf{z}$, where $\mathbf{z} = (z_i)^T$ and U is a unitary matrix, then the differential equation for $\mathbf{y}$ is

$$\mathbf{y}' = -B\mathbf{y} + U\mathbf{w},$$

where $B = U\,\mathrm{diag}\,(\beta_i)U^*$, and $\mathbf{w} = (z_i^2)^T$. The eigenvalues λ_i of the Jacobian matrix are $2z_i - \beta_i$. If $\mathbf{y}(0) = \mathbf{z}(0)$, then $\lambda_i \to -|\beta_i|$ as $x \to \infty$. Here as in [4], we set $\beta_1 = 1000$, $\beta_2 = 800$, $\beta_3 = -10$, $\beta_4 = 0.001$, and $U = (\frac{1}{2} - \delta_{ij})$, where δ_{ij} is the Kronecker delta.

We solved this problem on an IBM 4341 using a version of DIFSUB [6] and a code DIFSOL, based on this version, which implements the Nordsieck formulation of the one-leg formulas with Enright's technique for updating the Jacobian. The Jacobian was evaluated using numerical differentiation. Table 2 presents the total number of

TABLE 2
Comparison of DIFSUB *and* DIFSOL *for Krogh's problem.*

code	ε	evaluations	Jacobians	time	error
DIFSUB	1.0-3	301	25	3.0	2.7-3
DIFSOL	1.0-3	280	18	3.1	3.6-3
DIFSUB	1.0-6	719	33	5.0	5.1-6
DIFSOL	1.0-6	609	29	5.8	5.6-6

derivative evaluations, Jacobian evaluations, and time in seconds required to reach the first mesh point after $x = 1000$, and the maximum absolute error in the components of $\mathbf{y}$ observed up to that point for the prescribed local error tolerance $\varepsilon = 10^{-3}$ and 10^{-6}. The results indicate that the one-leg formulas are as efficient as the classical backward differentiation formulas. The additional work required to evaluate $F(\mathbf{a})$ is not very significant for this system and should be negligible for larger systems because it is proportional to the number of equations while the work required for the linear algebraic operations is proportional to the cube of the number of equations.

Lindberg [5] describes two test problems in which the eigenvalues change from large negative values to large positive values. We consider only the first for brevity because the results are similar for both problems. The first problem is

$$\begin{aligned} y_1' &= 10^4 y_1 y_3 + 10^4 y_2 y_4, & y_1(0) &= 1,\\ y_2' &= -10^4 y_1 y_4 + 10^4 y_2 y_3, & y_2(0) &= 1,\\ y_3' &= 1 - y_3, & y_3(0) &= -1,\\ y_4' &= -0.5 y_3 - y_4 + 0.5, & y_4(0) &= 0. \end{aligned}$$

The exact solution of this problem can be characterized as follows. Clearly $y_3(x) = 1 - 2\,e^{-x}$ and $y_4 = x\,e^{-x}$. If we set $\mathbf{y} = [y_1, y_2]^T$, then $\mathbf{y}' = A(x)\mathbf{y}$ and $\mathbf{y}(0) = [1, 1]^T$, where

$$A(x) = 10^4 \begin{bmatrix} 1-2\,e^{-x} & x\,e^{-x} \\ -x\,e^{-x} & 1-2\,e^{-x} \end{bmatrix}.$$

The eigenvalues of $A(x)$ are

$$\lambda_{1,2}(x) = 10^4(1 - 2\,e^{-x} \pm ix\,e^{-x}).$$

Initially the eigenvalues are -10^4 and approach 10^4 as $x \to \infty$. We solved the problem using DIFSUB with local error tolerance $\varepsilon = 10^{-p}$ for $p = 3(1)12$. We also solved the problem using DIFSOL with $\varepsilon = 10^{-p}$ for $p = 3(1)5$. The analytic Jacobian was used in both cases. DIFSUB failed to detect the presence of large positive eigenvalues and produced decaying solutions. DIFSOL also failed to detect the positive eigenvalues for $\varepsilon = 10^{-3}$ and 10^{-4}, but it did detect them and produce a qualitatively correct solution for $\varepsilon = 10^{-5}$. Table 3 presents the 2-norm of $\mathbf{y}$ and the approximations to $\mathbf{y}$ computed by DIFSUB and DIFSOL, and the associated local stepsizes for $\varepsilon = 10^{-5}$.

TABLE 3
Comparison of DIFSUB *and* DIFSOL *for Lindberg's problem.*

	exact	DIFSUB		DIFSOL	
x	$\lVert\mathbf{y}\rVert_2$	h	$\lVert\mathbf{y}\rVert_2$	h	$\lVert\mathbf{y}\rVert_2$
0.00	1.4+0	—	1.4+0	—	1.4+0
0.33	8.4−1009	3.7−2	1.1−8	4.9−2	8.5−7
0.79	1.7−1313	8.5−2	3.5−9	1.7−2	1.4−5
0.88	6.5−1262	8.5−2	1.1−7	6.7−3	1.5−5
0.90	2.1−1246	—	—	1.6−3	2.8−5
0.91	4.2−1238	—	—	4.1−5	1.1+7
0.93	1.8−1220	—	—	4.1−5	2.4+20
0.95	1.8−1201	—	—	4.1−5	2.3+39
0.97	3.9−1181	8.5−2	2.2−9	4.1−5	5.5+59
0.98	1.8−1170	—	—	4.1−5	2.4+70
3.91	1.8+8469	2.1−1	2.5−9	—	—

The solution computed by DIFSOL exhibits the correct rate of growth for $x > 0.93$, but it is too large and begins to grow prematurely because DIFSOL does not track the solution below the error tolerance and the region of instability for the k-step one-leg formula contains a small portion of the left half-plane. Thus in a system where the eigenvalues have large imaginary components and small negative real components, the solution would grow rather than decay as it should. Furthermore the detection of positive eigenvalues is sensitive to the error tolerance and the details of the Newton iteration. For example, we have observed that decreasing the error tolerance, evaluating the Jacobian by numerical differentiation, or increasing the number of Newton iterations can delay the growth of the computed solution. The stability properties of the one-leg formulas are thus not ideal, but these formulas are more likely to warn the user of possible difficulties when the eigenvalues approach the right half-plane.

Acknowledgment. We wish to thank Prof. Germund Dahlquist for his interest in this work and his instructive comments and suggestions.

REFERENCES

[1] G. Dahlquist, *On stability and error analysis for stiff nonlinear problems*, Report TRITA-NA-7508, Royal Institute of Technology, Stockholm, 1975.

[2] ———, *On the local and global errors of one-leg methods*, Report TRITA-NA-8110, Royal Institute of Technology, Stockholm, 1981.

[3] W. H. Enright, *Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations*, ACM Trans. Math. Software, 4 (1978), pp. 127–136.

[4] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[5] B. Lindberg, *On a dangerous property of methods for stiff differential equations*, BIT, 14 (1974), pp. 430–436.

[6] R. D. Skeel and A. K.-Y. Kong, *Blended linear multistep methods*, Report UIUCDCS-R-76-800, Dept. Computer Science, Univ. Illinois at Urbana-Champaign, 1976.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

NUMERICAL METHODS FOR LARGE SPARSE LINEAR LEAST SQUARES PROBLEMS*

MICHAEL T. HEATH†

Abstract. Large sparse least squares problems arise in many applications, including geodetic network adjustments and finite element structural analysis. Although geodesists and engineers have been solving such problems for years, it is only relatively recently that numerical analysts have turned attention to them. In this paper we present a survey of numerical methods for large sparse linear least squares problems, focusing mainly on developments since the last comprehensive surveys of the subject published in 1976. We consider direct methods based on elimination and on orthogonalization, as well as various iterative methods. The ramifications of rank deficiency, constraints, and updating are also discussed.

Key words. sparse least squares, normal equations, elimination, orthogonalization, Givens rotations, iterative methods

1. Introduction.

The method of least squares often means different things to different people. For statisticians, least squares is a method for estimating unknown parameters. For engineers and experimental scientists, it is a method for curve fitting or data smoothing. For numerical analysts, least squares has come to mean solving nonsquare systems of equations, that is, systems whose equations and unknowns differ in number. Of course, such a system does not necessarily have a solution, and so instead we settle for minimizing some norm of the residual. Although other norms are sometimes used, the (possibly weighted) Euclidean norm is by far the most common, hence the name "least squares." If such a minimum residual solution is not unique, then an additional requirement is imposed, such as that the norm of the solution itself also be minimal.

From both practical and computational points of view, linear problems are an extremely important subclass. This is true not only because many systems are inherently linear, but also because many algorithms for solving nonlinear problems require the solution of a succession of linear subproblems. For this reason, a great deal of effort over an extended period of time has gone into the development of a number of effective algorithms for solving linear least squares problems. In many cases the seeds of these algorithms are contained in computing practices which predate electronic computers and were not necessarily originally intended for least squares problems. These techniques were later sharpened and refined, particularly with regard to their numerical properties, for use on digital computers before finally receiving a systematic modern implementation for solving least squares problems. The present survey is concerned with a further phase of development, the extension of these algorithms to handle large sparse least squares problems.

* Received by the editors December 10, 1982. This research was sponsored jointly by the National Geodetic Survey of the National Ocean Survey, NOAA, U. S. Department of Commerce, under Interagency Agreement No. 40-1108-80, and by the Applied Mathematical Sciences Research Program, Office of Energy Research, U. S. Department of Energy under contract W-7405-eng-26 with the Union Carbide Corporation. A version of this paper was presented at the Sparse Matrix Symposium, Fairfield Glade, Tennessee, October 24-27, 1982. This paper was typeset at Oak Ridge National Laboratory using an Autologic APS-μ5 phototypesetter. Final copy was produced on May 16, 1983.

† Mathematics and Statistics Research Department, Computer Sciences Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830.

To fix notation, the problem we wish to consider is

$$\min_x \|b-Ax\|_2, \tag{1.1}$$

where A is an $m \times n$ matrix and b and x are vectors of dimension m and n, respectively. Standard assumptions are that $m \geqslant n$ and $rank(A)=n$, in which case the solution to (1.1) is unique. Problems in which $m<n$ or $rank(A)<\min\{m,n\}$ do occur, however, in many important contexts. For such underdetermined or rank deficient problems, (1.1) does not have a unique solution, so that the minimum norm solution, or some other particular solution, is usually what is required. Other important generalizations of (1.1) include the imposition of constraints and updating the solution when new data are added. For a comprehensive discussion of algorithms for least squares problems see [44]. For statistical interpretation see [58].

This survey is primarily concerned with methods which are effective for solving (1.1) when the matrix A is large and sparse, which means that A contains relatively few nonzero elements. Areas in which large sparse least squares and related problems arise include geodesy, photogrammetry, image enhancement, structural analysis, spline data smoothing, and mathematical programming.

In order to make the solution of very large sparse least squares problems computationally feasible with respect to execution time and storage requirements, algorithms for such problems should store and operate on only the nonzero entries of the matrix and should try to minimize the creation of new nonzeros as computations proceed. Another important consideration is the manner in which the data are accessed, since auxiliary storage is often required for large problems. Conventionally, the rows of the matrix A correspond to observations (successive measurements or replications), whereas the columns of A correspond to the unknown variables or parameters of the problem. Thus it is more convenient to generate, store, and access the data matrix by rows rather than by columns.

In addition to sparsity preservation and data management, questions of numerical stability must also be taken into account when evaluating algorithms for sparse least squares problems. Indeed, the main purpose of much of the research on least squares algorithms in the past twenty years has been to improve on the numerical shortcomings of more traditional methods. As in other areas of sparse matrix computations, here, too, the tradeoffs between sparsity and stability are of vital importance.

2. Normal equations. If $rank(A)=n$, then the solution to (1.1) is given by the solution to the system of normal equations

$$A^TAx=A^Tb. \tag{2.1}$$

If $rank(A)=m$, the minimum norm solution to (1.1) is given by $x=A^Ty$, where y is the solution to the linear system

$$AA^Ty=b. \tag{2.2}$$

In either case, the matrix of the linear system is symmetric and positive definite so that the solution can be obtained by Cholesky factorization. For system (2.1) the latter factorization takes the form $A^TA=R^TR$, with R upper triangular, so that (2.1) can be solved by forward and backward substitution. Most of the following discussion will be given in terms of system (2.1) for the overdetermined case; an analogous discussion is applicable to system (2.2) for the underdetermined case. (See [13] for a survey of basic methods for underdetermined problems.)

For dense problems, algorithms based on this approach are quite efficient, requiring only about half as many arithmetic operations as competing methods when $m \gg n$, although this advantage diminishes for more nearly square problems. The normal equations method is also very attractive for sparse problems, since excellent software packages are available for solving large sparse symmetric positive definite linear systems (eg., YSMP [20], SPARSPAK [27] and MA27 [19]). These software packages symmetrically reorder the equations and unknowns so that the Cholesky factor suffers relatively little fill (creation of new nonzeros). A number of reordering heuristics are known which can dramatically reduce the fill resulting from factorization. These include the minimum degree algorithm, various dissection schemes, and various bandwidth or profile minimization schemes.

Let P and Q be permutation matrices of order n and m, respectively. Then

$$(QAP)^T QAP = P^T A^T Q^T QAP = P^T A^T AP.$$

We conclude that the row ordering of A has no bearing on the normal equations, but reordering the columns of A corresponds to a symmetric row and column permutation of $A^T A$. In particular, the rows of A can be processed sequentially from an auxiliary file in arbitrary order. In the software packages mentioned in the previous paragraph, the symmetric reordering is determined by analyzing the structure of the graph corresponding to the symmetric matrix. For the symmetric matrix of the system of normal equations this graph is easily determined: the nonzeros of each row of A generate a clique in the graph of $A^T A$. (See [27] for relevant graph-theoretic concepts and terminology.) It is important to emphasize that, since pivoting is not required for numerical stability in the Cholesky algorithm, the reordering phase is entirely symbolic and takes place before any floating point computation. Furthermore, by anticipating all fill in advance, dynamic storage allocation is unnecessary, so that an efficient static data structure can be used.

Thus, an algorithm implementing (2.1) for sparse least squares problems goes like this (an analogous algorithm implements (2.2) for the underdetermined case):

ALGORITHM 1. Normal Equations.

1. Determine the structure (not the numerical values) of $A^T A$.
2. Find a permutation matrix P such that $P^T A^T AP$ has a sparse upper triangular Cholesky factor R.
3. Factor $P^T A^T AP$ symbolically, generating a row-oriented data structure for R.
4. Compute $A^T A$ and $A^T b$ numerically, processing the rows of A one by one from an external file.
5. Factor $P^T A^T AP = R^T R$ numerically.
6. Solve $R^T z = P^T A^T b$.
7. Solve $Ry = z$.
8. $x = Py$.

The use of the normal equations is as old as the method of least squares itself. The normal equations are easy to derive and understand, and, as we have seen, they adapt nicely to deal with sparse problems. Indeed, for many people "normal equations" and "least squares" are virtually synonymous, and algorithms based on the normal equations are still by far the most commonly used for solving least squares problems (see, e.g., [36]). The only trouble in this apparent paradise is that use of the normal equations may lead to numerical difficulties which are relatively harmless in some cases, but can be disastrous in others.

The potential numerical problems associated with the normal equations spring from two sources. One is the potential loss of information in explicitly computing the cross-product matrix A^TA and vector A^Tb. The other source of difficulty is the fact that the condition number of the matrix A^TA is the square of that of A, so that an accurate solution of (2.1) may be difficult or impossible to compute if A itself is already poorly conditioned (see, e.g., [60], [48], [32]). If A is not of full column rank, then A^TA is singular and the Cholesky factorization algorithm breaks down. Near rank degeneracy causes similar numerical problems in finite precision arithmetic. The upshot of all this is that forming and solving the normal equations requires relatively high working precision in order to guarantee suitable accuracy, which would often entail an unacceptable increase in storage requirements for very large problems. The principal motivation for the other methods we will discuss is to alleviate these numerical difficulties.

Another problem which the reader may have noticed is that the sparsity of A does not guarantee that A^TA is comparably sparse. Indeed, if an otherwise sparse A has even one dense row, then, barring numerical cancellation, A^TA is completely full. Clearly, such a situation is disastrous for the normal equations algorithm. The day can be saved, however, by initially omitting any rows of A which would cause excessive fill in A^TA (assuming full rank is still maintained) and later updating the solution to incorporate the effect of the omitted rows. Such updating procedures are common in least squares applications when new observations are added to a previously solved problem. The possibility of excessive fill in forming A^TA is sometimes given as justification for some alternative method. Barring accidental cancellation, however, all direct methods suffer the same or a similar fate when confronted with a matrix A having one or more dense rows. On the other hand, the updating procedures necessary to cope with such situations can be more stably implemented using other techniques, such as orthogonal transformations.

3. Elimination methods.

For simplicity of presentation, in this section we assume $rank(A)=n$. Carrying out Gaussian elimination with row and column interchanges on the $m \times n$ matrix A leads to a factorization of the form

$$P_1AP_2=LU, \tag{3.1}$$

where P_1 and P_2 are permutation matrices of order m and n, respectively, L is an $m \times n$ unit lower trapezoidal matrix, and U is an $n \times n$ upper triangular matrix. Utilizing this factorization, together with the orthogonal invariance of the Euclidean norm and the change of variable

$$UP_2^Tx=y, \tag{3.2}$$

problem (1.1) becomes

$$\min_y \|P_1b-Ly\|_2. \tag{3.3}$$

One way to solve problem (3.3) is by means of the system of normal equations

$$L^TLy=L^TP_1b. \tag{3.4}$$

The solution x to (1.1) is then recovered by solving the triangular system (3.2). It appears that little has been gained by this approach compared to solving system (2.1) directly. As observed by Peters and Wilkinson [52], however, by using an appropriate pivoting strategy in the elimination (i.e., choice of P_1 and P_2), it is possible to control the conditioning of L, isolating any ill conditioning of A in U. Thus, it should be safe numerically to form and solve system (3.4). Other authors have suggested the use of orthogonalization techniques to solve (3.3), but there is no real advantage over applying such methods directly to (1.1) unless the problem is only slightly overdetermined (see [12] and [53]).

This elimination scheme of Peters and Wilkinson has been extended and adapted for sparse problems by Björck and Duff [8]. When A is sparse, the permutations P_1 and P_2 must be chosen to preserve sparsity in L and U as well as to enhance the conditioning of L and the numerical stability of the factorization. A threshold pivoting strategy is used by Björck and Duff as a compromise between considerations of sparsity and stability. Note that here, too, one or more dense rows in A could cause unacceptable fill in L^TL, and so an updating scheme should be used to account for such rows. Usually the nonzero pattern of L is very much like that of A, and so solving system (3.4) requires about the same work and storage as system (2.1). Thus, the improved numerical behavior of the Peters-Wilkinson scheme is bought at the possibly high price of computing the factorization (3.1). One saving grace is that since U is not involved in solving (3.3), U may be written out on an external file and then recalled for the back substitution (3.2), thereby conserving main storage.

Björck and Duff introduce a modification of the Peters-Wilkinson scheme in which the solution x is split into two parts, one of which does not involve the normal equations (3.4). Using the partitioning

$$L=\begin{bmatrix} L_1 \\ L_2 \end{bmatrix}, \qquad P_1 b=\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{3.5}$$

where L_1 is a matrix of order n and b_1 is a vector of dimension n, let the n-vector c be defined by

$$L_1 c = b_1.$$

Note that this is equivalent to taking c to be the first n components of the transformed right-hand side vector, if the latter is processed simultaneously with A during the elimination. Let the $(m-n)$-vector d be defined by

$$d=b_2-L_2c.$$

We now observe that

$$P_1(b-Ax)=\begin{bmatrix} 0 \\ d \end{bmatrix}-Lz,$$

where

$$z=UP_2^Tx-c.$$

Thus, if z is the solution of the problem

$$\min_z \left\| \begin{bmatrix} 0 \\ d \end{bmatrix}-Lz \right\|_2,$$

and x_1 and x_2 are given by the triangular systems

$$UP_2^Tx_1=c, \qquad UP_2^Tx_2=z,$$

then $x=x_1+x_2$ is the solution of (1.1). There are two principal advantages to splitting the solution into two parts in this manner. First, since $\|b-Ax_1\|_2=\|d\|_2$, if $\|d\|_2$ is sufficiently small (i.e., the original problem (1.1) is nearly consistent), then x_1 is already an adequate solution, so that x_2, and hence z, need not be computed at all. Second, any ill conditioning in L affects only the computation of the "correction" term x_2.

The steps of the Peters-Wilkinson method as modified by Björck and Duff are summarized in the following algorithm.

ALGORITHM 2. Elimination.

```
1. Compute the factorization (3.1), choosing $P_1$ and $P_2$ to produce sparse $U$ and
   $L$ and well conditioned $L$.
2. Solve $L_1c=b_1$, with $L_1$ and $b_1$ as in (3.5).
3. Solve $UP_2^Tx_1=c$.
4. $d=b_2-L_2c$.
5. If $\|d\|_2<\epsilon$, set $x=x_1$ and stop.
6. Solve $L^TLz=L^T\begin{bmatrix}0\\d\end{bmatrix}$ using Algorithm 1.
7. Solve $UP_2^Tx_2=z$.
8. $x=x_1+x_2$.
```

In their paper [8], Duff and Björck show how to extend this algorithm to handle rank deficient problems, weighted problems, constraints, and updating. Delves and Barrodale [15] have published a related elimination algorithm in which first a square subsystem of (1.1) is solved by the usual square LU factorization, then the remaining rows of A are treated as updates. Such an approach is advantageous for problems which are only slightly overdetermined. Their algorithm does not appear to have been implemented as yet specifically for sparse problems.

Another family of elimination methods is based on the fact that the solution x to (1.1) and the residual $r=b-Ax$ must satisfy the augmented $(m+n)\times(m+n)$ linear system

$$\begin{bmatrix} I & A \\ A^T & O \end{bmatrix}\begin{bmatrix} r \\ x \end{bmatrix}=\begin{bmatrix} b \\ 0 \end{bmatrix}. \tag{3.6}$$

This system has been used by Björck [2] in studying iterative refinement for least squares solutions, and its use for sparse problems has been advocated by Hachtel, who calls this the "sparse tableau" approach [37]. The idea here is simply to use a standard sparse solver for square linear systems to solve (3.6). Note that block elimination applied to (3.6) without pivoting yields the usual normal equations for x. The hope is that the sparse square system solver will use the additional freedom in choosing pivots to find a considerably more sparse factorization. Although system (3.6) is symmetric, it is indefinite, so that the pivoting must take account of numerical stability as well as sparsity. While a sparse symmetric indefinite factorization is certainly possible [19], there are actually certain advantages to ignoring the symmetry of (3.6) and using a nonsymmetric system solver [18]. On the other hand, ignoring symmetry incurs the heavy penalty of having to store two copies of A.

4. Orthogonalization methods.

4.1. Basic methods. We have seen that the chief difficulty with the normal equations is numerical: the information loss and ill conditioning associated with the explicit formation of the cross-product matrix. The elimination method of Peters and Wilkinson tries to lessen these numerical effects. Another alternative is to avoid explicit formation of the cross-product matrix altogether by computing its triangular Cholesky factor R directly from A, and this can be done by means of orthogonal factorization. An orthogonal matrix Q of order m is computed which reduces $[A,\ b]$ to the form

$$QA=\begin{bmatrix} R \\ O \end{bmatrix}, \qquad Qb=\begin{bmatrix} c \\ d \end{bmatrix}, \tag{4.1}$$

where R is an upper triangular matrix of order n and c is a vector of dimension n. (We consider first the $m \geqslant n$ case; the underdetermined case will be taken up later in this section.) To see that R is indeed the Cholesky factor of the cross-product matrix (assuming Q is chosen so that R has positive diagonal entries), we need merely note that

$$A^TA=A^TQ^TQA=[R^T \quad O]\begin{bmatrix} R \\ O \end{bmatrix}=R^TR.$$

Since the Euclidean norm is invariant under orthogonal transformation, the solution to (1.1) may be obtained by solving the triangular system $Rx=c$.

There are three principal methods for computing the factorization (4.1): Gram-Schmidt orthogonalization ([55], [1]), Householder reflections ([41], [32], [9]), and Givens rotations ([31], [21-23], [38]). Both Gram-Schmidt and Householder reduce A to triangular form by annihilating all the subdiagonal elements in an entire column at each step. Though effective for dense problems, this column-oriented, "sledge hammer" approach has serious drawbacks for large sparse problems. The trouble is that at each step each column in the remaining unreduced portion of the matrix which has a nonzero inner product with the column being reduced takes on the sparsity pattern of their union. Although this newly created fill scattered throughout the unreduced matrix will eventually be annihilated by the orthogonalization process, in the mean time it must be stored, greatly increasing storage requirements beyond that required for R (see Fig. 1). Givens rotations are a much more appropriate tool in this context because of their ability to introduce zeros more selectively and in a more flexible order [21]. In particular, R can be built up gradually as the rows of A are processed one by one in their natural order (or any other desired order), intermediate fill is confined to R and the working row, and the unreduced rows of A can remain untouched on external storage until their turn for actual reduction (see Fig. 2).

A problem which plagues all orthogonalization methods is that even if A and R are sparse, it is unlikely that the orthogonal matrix Q will be particularly sparse. There has been some study of maintaining sparsity in Q ([61],[11],[16]), but the outlook in general is quite unpromising, especially since sparsity must be maintained simultaneously in R. Instead, most practical procedures simply discard the orthogonal transformations which make up Q as they are used in processing the matrix and right hand side vector. For a simple problem with a single right hand side no real harm is done, since Q is not actually needed to compute the least squares solution once R and c have been obtained. But for more complicated problems, such as those having mul-

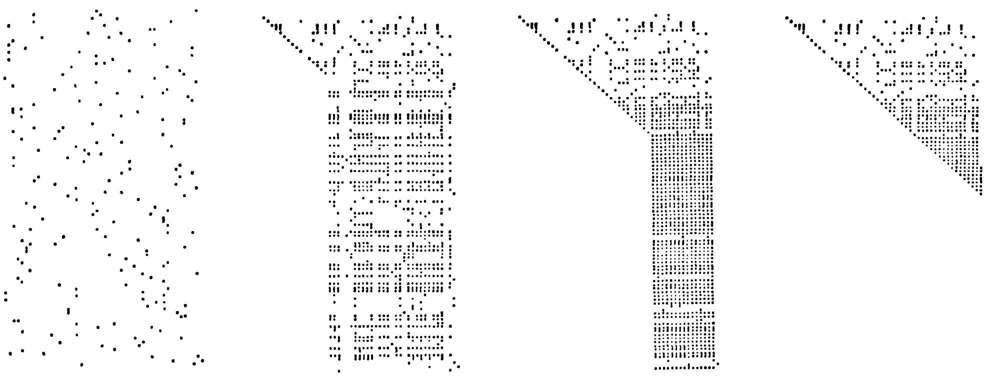


FIG. 1. *Reducing a sparse matrix by Householder reflections.*

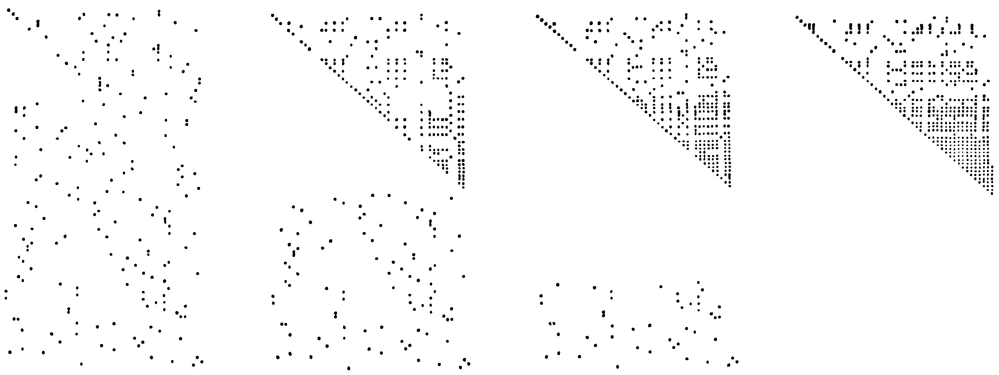


FIG. 2. *Reducing a sparse matrix by Givens rotations.*

$$A = \begin{bmatrix} x & & & & \\ & x & & & \\ & & x & & \\ & & & x & \\ & & & & x \\ x & & & & \\ x & & & & \\ x & & & & \\ x & x & x & x & x \end{bmatrix} \begin{array}{l} \left.\begin{array}{c} \\ \\ \\ \\ \\ \end{array}\right\} n \\ \left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right\} k \end{array}, \quad PA = \begin{bmatrix} x & x & x & x & x \\ x & & & & \\ x & & & & \\ x & & & & \\ x & & & & \\ & x & & & \\ & & x & & \\ & & & x & \\ & & & & x \end{bmatrix}$$

FIG. 3. *Cost for reducing A: $O(n^2)$, PA: $O(kn^2)$.*

tiple right-hand sides which are not known in advance, or certain applications which require explicit computation of an orthogonal basis, getting along without Q can be a headache. One alternative is to write the orthogonal transformations on auxiliary storage as they are generated. Each Givens rotation can be represented by a single floating point number [59], thereby economizing storage. In other cases the need for Q can be circumvented, although not always with equivalent numerical stability. For example, one way to handle multiple right hand sides is to solve the system

$$R^T R x = A^T b \tag{4.2}$$

using the R already computed, so that only the original matrix A, which is available on an external file, is needed to transform subsequent right hand sides. Although system (4.2) shares some of the numerical shortcomings of the normal equations method, at least R is accurately computed, and the accuracy of the solution can be improved by a few iterations of iterative refinement [4].

Having decided to use Givens rotations, there remains the problem of choosing a good row and column ordering for A. There has been a good deal of work ([16],[23],[62]) on choosing these orderings dynamically: the ordering is determined according to some local minimization-of-fill criterion as numerical computations proceed, and storage is inserted into the data structure as needed to accommodate any fill generated. Such algorithms are very similar in spirit to square, nonsymmetric linear system solvers in that access to the whole unreduced matrix is required at each step for possible pivot selection. An important difference, however, is that the stability of orthogonal transformations allows the selection to be based solely on sparsity considerations (assuming the problem is not too disparately weighted: see further comments on row ordering below).

A different approach is taken in [24], which is patterned more after symmetric positive definite linear system solvers. Recall our earlier remark in discussing the normal equations that the row ordering for A has no bearing on the structure of $A^T A$, but that the column ordering for A determines the structure of $A^T A$ and hence that of R, its Cholesky factor. Thus the structure of $A^T A$ can be analyzed symbolically in advance of any numerical computation in order to find a good column order for A which will limit fill in R, and also to set up a data structure which will accommodate any fill in R which does occur. Such a static data structure can be very efficient, requiring none of the garbage collection and other overhead associated with dynamic data structures.

An algorithm based on these considerations is as follows:

ALGORITHM 3. Triangularization by Givens Rotations.

1. Determine the structure (not the numerical values) of $A^T A$.
2. Find a permutation matrix P such that $P^T A^T A P$ has a sparse upper triangular Cholesky factor R.
3. Factor $P^T A^T A P$ symbolically, generating a row-oriented data structure for R.
4. Compute R and c numerically, processing the rows of $[AP,\ b]$ one by one using Givens rotations.
5. Solve $Ry = c$.
6. $x = Py$.

Steps 1 through 3 of Algorithm 3 are the same as those of Algorithm 1 and can be carried out very efficiently using standard sparse matrix software designed for symmetric positive definite linear systems. In particular, assuming the Givens rotations are either discarded or written on secondary storage, Algorithm 3 exploits sparsity to the same degree and requires the same amount of primary storage as Algorithm 1, but is more stable numerically.

The details of step 4 of Algorithm 3 are of some interest. Let a^T be a given row of AP to be processed next. Let j be the subscript of the first nonzero component of a^T. It is shown in [24] that there is space in row j of the data structure of R to accommodate a^T. If row j of the data structure is still vacant, as will likely be the case early in the row by row processing, then a^T may simply be placed into row j of the data structure. If, on the other hand, row j of the data structure is already occupied by previously stored numerical values, then row j may be used to annihilate the first nonzero of a^T with a Givens rotation. It is further shown in [24] that the resulting "shorter" row can also be accommodated in the data structure, even though some fill may have occurred as a result of the transformation. Thus, the process may be repeated until either an unoccupied row is found in which to place the working row or all its nonzeros have been annihilated.

Although the order in which the rows of A are processed in step 4 does not affect the structure of R, it does affect the amount of fill created in the working row and hence the total cost of computing R. Fig. 3 gives an extreme example of the difference row ordering can make with respect to numerical factorization cost. Heuristic row ordering rules have been suggested which can substantially reduce computational costs for certain classes of problems, but the general relationship between row and column orderings is not yet well understood (see [24] and [28]). Another factor which may dictate a particular row ordering is that any heavily weighted rows should be processed first for optimum numerical stability (see [44], pp. 103-106).

Specialized algorithms which adapt orthogonalization techniques to problems having banded or similar structure are given in [14], [44], and [54].

4.2. Extensions and generalizations. Algorithm 3 lends itself to a number of useful extensions and generalizations in order to handle more difficult or complicated problems. For example, the problem may be so large that R will not fit in main memory, and hence auxiliary storage must be used. Indeed, for extremely large problems such as the geodetic readjustment of the North American Datum [42], storage requirements may even exceed the virtual address space of the largest computers, so that auxiliary space cannot be managed implicitly by a paging algorithm. In [26] an algorithm is given in which such large problems are partitioned by incomplete nested dissection into a sequence of smaller subproblems, each of which is processed by the basic Givens algorithm, eventually producing the solution to the original problem.

It is clear that Algorithm 3 suffers the same catastrophic fill as the other methods we have discussed when confronted with a matrix A having one or more dense rows. It is also sometimes desired that some of the equations in a linear system be satisfied exactly while the remaining equations are satisfied only in the least squares sense. For example, it may be required that the sum of all the variables be equal to 1 or some other prescribed constant. Extensions to Algorithm 3 which enable it to incorporate such constraints and/or updating are derived in [39]. Further generalization to allow arbitrary rank along with constraints and updating is given in [7].

Another implicit assumption in Algorithm 3 is that $rank(A)=n$. The usual generalization of the orthogonal factorization (4.1) for dense rank deficient problems is to use column interchanges during the orthogonalization process to obtain a factorization of the form

$$QAP=\begin{bmatrix} R & S \\ O & T \end{bmatrix}, \tag{4.3}$$

where R is an upper triangular matrix of order $k=rank(A)$, P is a permutation matrix which performs the column interchanges, and the elements of T are negligible in magnitude. In Algorithm 3, however, the column ordering for A represented by P is fixed in advance to preserve sparsity, and cannot be altered during the numerical phase. It is shown in [39] that a factorization of the form (4.3) can nevertheless be obtained without explicit column pivoting, leading in turn to a basic or minimum-norm solution for underdetermined and/or rank deficient sparse linear least squares problems.

An alternate approach for underdetermined problems is to apply Algorithm 3 to A^T rather than A, yielding an orthogonal factorization of the form

$$A=[R^T \;\; O]Q. \tag{4.4}$$

This enables us to replace system (2.2) by the system

$$R^TRy=b, \tag{4.5}$$

which can be solved by forward and back substitution. Such an approach fits in well with the philosophy of discarding Q, but it would appear to have the same condition squaring effect as that suffered by (2.2) and (4.2). In practice, however, the use of (4.5) usually yields results which are comparable to the theoretically more accurate algorithm which involves Q explicitly [56]. This surprising behavior has been explained by Paige [49], who shows that as long as A is not too ill conditioned, the error resulting from (4.5) depends essentially on the condition number of A rather than the condition number squared. In using (4.4) and (4.5) for the sparse case via Algorithm 3, we must now avoid dense columns rather than dense rows. Appropriate updating procedures are given in [29].

5. Iterative methods. For some large sparse least squares problems iterative methods are a useful alternative to the direct methods we have discussed thus far. Unlike direct methods, which compute an approximation to the exact solution in a finite number of steps, iterative methods successively improve an initial approximate solution until the approximation is acceptably close to the exact solution. One advantage of this approach is that one need not spend time computing an unnecessarily accurate solution if the data do not warrant it. Direct methods, on the other hand, generally have fixed accuracy and do not produce meaningful intermediate results. Iterative methods are also especially appropriate for problems in which the entries of the matrix are easily generated on demand. In such cases the matrix need not be stored at all, but instead can be defined by its action on vectors.

In principle any iterative method for symmetric positive definite (or semidefinite) linear systems can simply be applied to the system of normal equations (2.1). Explicit formation of the cross-product matrix A^TA can be avoided by keeping the normal equations in factored form

$$A^T(b-Ax)=0. \tag{5.1}$$

In this way the matrix A is used only to compute matrix-vector products of the form Ax and A^Ty. Although this formulation avoids some of the numerical difficulties and fill which can result from explicitly forming A^TA, the convergence rate of iterative methods based on (5.1) still depends on the spectrum of A^TA, and can therefore be very slow. Because of this the main emphasis in research on iterative methods for least squares problems has been to accelerate convergence by means of various splittings and preconditioners.

A splitting takes the form $A=M-N$, leading to a sequence of least squares problems of the form

$$Mx_{k+1} \simeq Nx_k + b.$$

Preconditioning is a change of variable $z=Cx$, where C is a nonsingular matrix of order n, so that

$$b-Ax=b-AC^{-1}z.$$

In either case the strategy is to choose M or C so that M or AC^{-1} gives a more favorable spectrum than A, thereby speeding convergence. Since these two approaches are essentially equivalent [10], we will concentrate on preconditioning methods.

In implementing a preconditioner for least squares problems matrix-vector products of the form Ax and A^Ty become $AC^{-1}z$ and $C^{-T}A^Ty$, respectively. Of course the matrix product AC^{-1} is never explicitly computed, but instead is treated as the product of two successive operators. Thus each iteration will require solution of linear systems of the form

$$Cx=z \quad \text{and} \quad C^Tx=y. \tag{5.2}$$

For this reason C is usually chosen to be diagonal or triangular so that systems (5.2) can be solved easily. Several different preconditioners have been used effectively for least squares problems:

1. Diagonal scaling. $C=diag(d_i)$, where the d_i are norms of the columns of A.

2. SSOR preconditioning [5]. $C=I+\omega L^T$, where A has been scaled so that $A^TA=L+I+L^T$ with L strictly lower triangular, and ω is a scalar relaxation parameter.

3. Incomplete Cholesky factorization ([45],[46]). $C=R'$, where R' is an approximation to the Cholesky factor R but is more sparse. Note that the true Cholesky factor R would be the ideal choice for C since then AC^{-1} is orthogonal.

4. *LU* preconditioning [57]. $C=U$, where U is the upper triangular factor from a factorization of the form (3.1).

5. Gauss-Jordan preconditioning [43]. $C=A_1^{-1}$, where A_1 is a square, nonsingular submatrix of A of order n.

Notice that this last preconditioner gives an algorithm that is essentially the same as the direct method of Delves and Barrodale [15] mentioned in section 3, except that the updating is done iteratively rather than in closed form. This is just one example of the many ways in which preconditioning blurs the distinction between direct and iterative methods.

As a concrete example of an iterative method for least squares problems we now consider the method of conjugate gradients. There are many variants of conjugate gradients which are theoretically equivalent but differ in their numerical behavior. One of the most effective for reasonably well conditioned least squares problems is this early version due to Hestenes and Stiefel [40]:

ALGORITHM 4. Conjugate Gradients.

```
1. $x_0=0$
2. $r_0=b$
3. $s_0=A^T r_0$
4. $\gamma_0=\|s_0\|_2^2$
5. $p_1=s_0$
6. For $k=1, 2, \ldots$ repeat the following:
    a. $q_k=Ap_k$
    b. $\alpha_k=\gamma_{k-1}/\|q_k\|_2^2$
    c. $x_k=x_{k-1}+\alpha_k p_k$
    d. $r_k=r_{k-1}-\alpha_k q_k$
    e. $s_k=A^T r_k$
    f. $\gamma_k=\|s_k\|_2^2$
    g. $\beta_k=\gamma_k/\gamma_{k-1}$
    h. $p_{k+1}=s_k+\beta_k p_k$.
```

Although this algorithm theoretically produces the exact solution to (1.1) in at most n iterations, with the rounding errors of finite precision arithmetic it may require far more or far fewer than n iterations to yield a satisfactory solution. If a preconditioner is used to accelerate convergence, then the matrix-vector products Ap and $A^T r$ are replaced by $AC^{-1}p$ and $C^{-T}A^T r$ implemented by solving systems of the form (5.2). In using a preconditioner there is a tradeoff between the reduction in the number of iterations required and the increase in work per iteration.

A variant of conjugate gradients which is effective for more ill conditioned least squares problems has been developed by Paige and Saunders in [50], which should be consulted for algorithmic details. Their algorithm is based on the bidiagonalization procedure of Golub and Kahan [33] in the same way that the conjugate gradient algorithm is related to Lanczos tridiagonalization. The bidiagonalization approach has also been adapted to obtain regularized solutions of ill-posed problems ([6],[47]).

These bidiagonalization algorithms can be thought of as a natural extension of the singular value decomposition method to solve sparse problems. The singular value decomposition, which has the form

$$A=U\Sigma V^T, \tag{5.3}$$

where U and V are orthogonal matrices of order m and n, respectively, and Σ is an $m \times n$ nonnegative diagonal matrix, is in many ways the most satisfactory numerical method for solving least squares problems ([33], [35], [44]). Unfortunately the full decomposition (5.3) is not computationally useful for large sparse problems because the orthogonal matrices U and V are generally too dense. Through Lanczos-style bidiagonalization, however, a few dominant triples (σ,u,v) of singular values and vectors can be generated even for very large matrices [34].

6. Concluding remarks. In this paper we have surveyed the principal numerical methods available for solving large sparse linear least squares problems. We have concentrated on developments since the surveys [3], [18], and [30] published in 1976. Several of these methods have been compared in numerical experiments using a variety of test problems ([18],[25],[50],[57]). Although much more testing is needed on a broader spectrum of test examples, some preliminary conclusions are possible:

1. For well conditioned problems the traditional method of normal equations implemented using modern sparse matrix techniques is very effective and hard to beat, especially on problems for which $m \gg n$ and which are quite sparse.
2. For more difficult problems which require the greater numerical stability of orthogonal factorization, the method of Givens rotations can be implemented so as to use essentially the same storage as the normal equations method (see Algorithm 3).
3. For problems which are only slightly overdetermined, or are overdetermined but consistent, a method based on elimination is likely to be best.
4. Several effective techniques are available for problems which are well suited to iterative solution. If conditioning is a problem, the method of Paige and Saunders [50] is especially to be recommended. A preconditioner can help speed convergence but must be chosen carefully.

Most of the methods we have discussed have been implemented in computer software which is publicly available, or soon will be:

1. Several symmetric positive definite linear system solvers (e.g., SPARSPAK [27], YSMP[20], MA27 [19]) are available which could form the basis of an efficient implementation of the normal equations method. Since it handles indefinite problems as well, MA27 could also be used to solve the augmented system (3.6).
2. The implementation of the Peters-Wilkinson elimination method by Björck and Duff [8] is to be included in the Harwell Subroutine Library. Their code uses a modified version of the Harwell subroutine MA28 [17] to compute the *LU* factorization (3.1).
3. Software implementing the algorithms of George and Heath [24], [39] is to be included in a new, expanded version of SPARSPAK, giving it the capability of solving nonsymmetric and nonsquare problems. The new modules rely heavily on the existing SPARSPAK for the symbolic parts of the computation.
4. The work of Zlatev [62] is implemented in the software package LLSS01 [63]. To conserve storage, this code allows for an incomplete triangular factorization (determined by a drop tolerance), followed by iterative refinement.
5. The iterative algorithm of Paige and Saunders has been implemented as subroutine LSQR and is available from ACM TOMS [51]. This code solves damped least squares problems as well as ordinary least squares and nonsymmetric equations.

There is considerable room for further research on algorithms for sparse least squares problems. Better row and column orderings are needed, as well as a better understanding of the relationship between them. The tradeoffs between static and dynamic data structures need further study. In elimination methods, the structural relationships between A and L and between U and R need to be better understood. Also worthy of exploration are row elimination schemes using stabilized elementary eliminators, as opposed to the use of a general threshold (partial or complete) pivoting approach. More sophisticated criteria are needed for withholding rows which lead to excessive fill in the Cholesky factor. The numerical behavior of updating schemes needs to be further scrutinized and improved. Existing algorithms should be extended to allow more generality with regard to constraints, updating and rank. An important problem for iterative methods is automating the choice of an effective preconditioner to speed convergence. With all of these methods relatively little attention has been given to handling problems which are too large to fit in main storage or to the use of advanced computer architectures such as pipelined, array, or parallel processors.

The answers to these and many other outstanding questions will undoubtedly lead to more effective and efficient methods for solving large sparse least squares problems. In addition, advances in sparse least squares computations will have an effect on other areas of sparse matrix computations, such as the application of sparse orthogonal factorizations to problems in optimization, control, and eigenvalue and singular value computations.

REFERENCES

[1] Å. Björck, *Solving linear least squares problems by Gram-Schmidt orthogonalization*, BIT, 7 (1967), pp. 1-21.

[2] ———, *Iterative refinement of linear least squares solutions*, BIT, 7 (1967), pp. 257-278 and 8 (1968), pp. 8-30.

[3] ———, *Methods for sparse least squares problems*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 177-199.

[4] ———, *Comment on the iterative refinement of least squares solutions*, J. Amer. Stat. Assoc., 73 (1978), pp. 161-166.

[5] ———, SSOR *preconditioning methods for sparse least squares problems*, Proc. Comput. Sci. Stat. Symposium on the Interface, 12 (1979), pp. 21-25.

[6] ———, *A bidiagonalization algorithm for solving ill-posed systems of linear equations*, Report LiTH-MAT-R-80-33, Dept. of Mathematics, Linkoping University, Linkoping, Sweden, October 1980.

[7] ———, *A general updating algorithm for constrained linear least squares problems*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 394-402.

[8] Å. Björck and I. S. Duff, *A direct method for the solution of sparse linear least squares problems*, Linear Algebra Appl., 34 (1980), pp. 43-67.

[9] P. Businger and G. H. Golub, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269-276.

[10] Y. T. Chen, *Iterative methods for linear least squares problems*, Research Report CS-75-04, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, February 1975.

[11] Y. T. Chen and R. P. Tewarson, *On the fill-in when sparse vectors are orthonormalized*, Computing, 9 (1972), pp. 53-56.

[12] A. K. Cline, *An elimination method for the solution of linear least squares problems*, SIAM J. Numer. Anal., 10 (1973), pp. 283-289.

[13] R. E. Cline and R. J. Plemmons, *ℓ_2-solutions to underdetermined linear systems*, SIAM Review, 18 (1976), pp. 92-106.

[14] M. G. Cox, *The least squares solution of overdetermined linear equations having band or augmented band structure*, IMA J. Numer. Anal., 1 (1981), pp. 3-22.

[15] L. M. Delves and I. Barrodale, *A fast direct method for the least squares solution of slightly overdetermined sets of linear equations*, J. Inst. Math. Appl., 24 (1979), pp. 149-156.

[16] I. S. Duff, *Pivot selection and row ordering in Givens reduction on sparse matrices*, Computing, 13 (1974), pp. 239-248.

[17] ———, MA28 - *A set of Fortran subroutines for sparse unsymmetric linear equations*, Report R.8730, AERE, Harwell, England, July 1977.

[18] I. S. DUFF AND J. K. REID, *A comparison of some methods for the solution of sparse overdetermined systems of linear equations*, J. Inst. Math. Appl., 17 (1976), pp. 267-280.

[19] ———, MA27 - *A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Report R.10533, AERE, Harwell, England, 1982.

[20] S. C. EISENSTAT, M. H. SCHULTZ AND A. H. SHERMAN, *Algorithms and data structures for sparse symmetric Gaussian elimination*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 225-237.

[21] W. M. GENTLEMAN, *Least squares computations by Givens transformations without square roots*, J. Inst. Math. Appl., 12 (1973), pp. 329-336.

[22] ———, *Error analysis of* QR *decompositions by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189-197.

[23] ———, *Row elimination for solving sparse linear systems and least squares problems*, Proc. 6th Dundee Conf. Numer. Anal., Springer-Verlag, 1976, pp. 122-133.

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

[25] A. GEORGE, M. T. HEATH AND E. NG, *A comparison of some methods for solving sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 177-187.

[26] A. GEORGE, M. T. HEATH AND R. J. PLEMMONS, *Solution of large-scale sparse least squares problems using auxiliary storage*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 416-429.

[27] A. GEORGE AND J. W.-H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[28] A. GEORGE AND E. NG, *On row and column orderings for sparse least squares problems*, SIAM J. Numer. Anal., 20 (1983), pp. 326-344.

[29] A. GEORGE, M. T. HEATH AND E. NG, *Solution of sparse underdetermined systems of linear equations*, SIAM J. Sci. Stat. Comput., to appear.

[30] P. E. GILL AND W. MURRAY, *The orthogonal factorization of a large sparse matrix*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 201-212.

[31] W. GIVENS, *Computation of plane unitary rotations transforming a general matrix to triangular form*, J. Soc. Ind. Appl. Math., 6 (1958), pp. 26-50.

[32] G. H. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206-216.

[33] G. H. GOLUB AND W. KAHAN, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM J. Numer. Anal., 2 (1965), pp. 205-224.

[34] G. H. GOLUB, F. T. LUK AND M. L. OVERTON, *A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix*, ACM Trans. Math. Software, 7 (1981), pp. 149-169.

[35] G. H. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403-420.

[36] J. H. GOODNIGHT, *A tutorial on the* SWEEP *operator*, Amer. Statistician, 33 (1979), pp. 149-158.

[37] G. D. HACHTEL, *Extended application of the sparse tableau approach - finite elements and least squares*, Technical Report, Computer Science Dept., UCLA, 1974.

[38] S. HAMMARLING, *A note on modifications to the Givens plane rotation*, J. Inst. Math. Appl., 13 (1974), pp. 215-218.

[39] M. T. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 223-237.

[40] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stds., B49 (1952), pp. 409-436.

[41] A. S. HOUSEHOLDER, *Unitary triangularization of a nonsymmetric matrix*, J. ACM, 5 (1958), pp. 339-342.

[42] G. B. KOLATA, *Geodesy: dealing with an enormous computer task*, Science, 200 (1978), pp. 421-422.

[43] P. LÄUCHLI, *Jordan-Elimination und Ausgleichung nach kleinsten Quadraten*, Numer. Math., 3 (1961), pp. 226-240.

[44] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[45] T. A. MANTEUFFEL, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473-497.

[46] N. MUNKSGAARD, *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Software, 6 (1980), pp. 206-219.

[47] D. P. O'Leary and J. A. Simmons, *A bidiagonalization-regularization procedure for large scale discretizations of ill-posed problems*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 474-489.

[48] E. E. Osborne, *On least squares solutions of linear equations*, J. ACM, 8 (1961), pp. 628-636.

[49] C. C. Paige, *An error analysis of a method for solving matrix equations*, Math. Comp., 27 (1973), pp. 355-359.

[50] C. C. Paige and M. A. Saunders, LSQR: *An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43-71.

[51] ———, *Algorithm* 583. LSQR: *Sparse linear equations and least squares*, ACM Trans. Math. Software, 8 (1982), pp. 195-209.

[52] G. Peters and J. H. Wilkinson, *The least squares problem and pseudo-inverses*, Comput. J., 13 (1970), pp. 309-316.

[53] R. J. Plemmons, *Linear least squares by elimination and* MGS, J. ACM, 21 (1974), pp. 581-585.

[54] J. K. Reid, *A note on the least squares solution of a band system of linear equations by Householder reductions*, Comput. J., 10 (1967), pp. 188-189.

[55] J. R. Rice, *Experiments on Gram-Schmidt orthogonalization*, Math. Comp., 20 (1966), pp. 325-328.

[56] M. A. Saunders, *Large-scale linear programming using the Cholesky factorization*, Report No. CS 252, Computer Science Dept., Stanford University, Stanford, CA, January 1972.

[57] ———, *Sparse least squares by conjugate gradients: a comparison of preconditioning methods*, Proc. Comput. Sci. Stat. Symposium on the Interface, 12 (1979), pp. 15-20.

[58] G. A. F. Seber, *Linear Regression Analysis*, John Wiley and Sons, New York, 1977.

[59] G. W. Stewart, *The economical storage of plane rotations*, Numer. Math., 25 (1976), pp. 137-138.

[60] O. Taussky, *Note on the condition of matrices*, Math. Comp., 4 (1950), pp. 111-112.

[61] R. P. Tewarson, *On the orthonormalization of sparse vectors*, Computing, 3 (1968), pp. 268-279.

[62] Z. Zlatev, *Comparison of two pivotal strategies in sparse plane rotations*, Comp. Math. Appl., 8 (1982), pp. 119-135.

[63] Z. Zlatev and H. B. Nielsen, LLSS01 - *A Fortran subroutine for solving least squares problems (user's guide)*, Report No. 79-07, Inst. Numer. Anal., Tech. Univ. Denmark, Lyngby, Denmark, 1979.

SIAM J. Sci. Stat. Comput.
Vol. 5, No. 3, September 1984

SPARSE ORTHOGONAL SCHEMES FOR STRUCTURAL OPTIMIZATION USING THE FORCE METHOD*

M. T. HEATH,† R. J. PLEMMONS‡ AND R. C. WARD†

Abstract. Historically there are two principal methods of matrix structural analysis, the displacement (or stiffness) method and the force (or flexibility) method. In recent times the force method has been used relatively little because the displacement method has been deemed easier to implement on digital computers, especially for large sparse systems. The force method has theoretical advantages, however, for multiple redesign problems or nonlinear elastic analysis because it allows the solution of modified problems without restarting the computation from the beginning. In this paper we give an implementation of the force method which is numerically stable and preserves sparsity. Although it is motivated by earlier elimination schemes, in our approach each of the two main phases of the force method is carried out using orthogonal factorizaton techniques recently developed for linear least squares problems.

Key words. structural optimization, force method, orthogonal factorization, Givens rotations, turnback-QR method, sparse least squares, constrained sum-of-squares method

1. Introduction. Given the external loads on a structure, the object of structural analysis is to determine the resulting internal forces, stresses, and displacements. The solution to this problem is provided by a variational principle (minimization of energy) subject to the linear elastic relationships among the nodes and elements of the finite element model of the structure. Either the forces or the displacements may be taken as the primary quantities to be computed, and the other can then be determined as a by-product. These two approaches give rise to the force (or flexibility) method and the displacement (or stiffness) method, respectively. In recent times the displacement method has predominated, largely because it is easier to implement on digital computers, especially for large sparse systems, and makes use of well established techniques of numerical linear algebra. The displacement method can be inefficient, however, for structural optimization problems in which a sequence of related structural analysis problems must be solved (e.g., problems having a fixed layout but differing material properties). The force method is then preferable because it utilizes a portion of earlier computations in order to solve such modified problems without starting the computations over from the beginning. Unfortunately, most implementations of the force method have suffered from excessive fill or numerical difficulties, or both. In this paper we give an implementation of the force method which is numerically stable and preserves sparsity for large-scale problems.

Before stating our problem precisely, we need to develop some notation. The notational conventions used by structural engineers and by numerical analysts are quite different. As a compromise, we will use the same letters to denote various quan-

* Received by the editors June 3, 1983. This research was sponsored jointly by the Applied Mathematical Sciences Research Program, Office of Energy Research, U. S. Department of Energy under contract W-7405-eng-26 with the Union Carbide Corporation and by the U. S. Army Research Office under contract no. DAAG 29-81-k-0132. A version of this paper was presented at the Sparse Matrix Symposium, Fairfield Glade, Tennessee, October 24-27, 1982. This paper was typeset at Oak Ridge National Laboratory using an Autologic APS-μ5 phototypesetter. Final copy was produced on November 4, 1983.

† Mathematics and Statistics Research Department, Computer Sciences Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830.

‡ Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, North Carolina 27650.

tities as are commonly used by structural engineers (see, e.g., [20]), but we will retain the usual convention in numerical analysis that lower case letters represent vectors, while upper case letters represent matrices. Our notation is summarized in Table 1.

TABLE 1

equilibrium matrix	: E
element flexibility matrix	: D
element stiffness matrix	: D^{-1}
system stiffness matrix	: $K=ED^{-1}E^T$
self-stress matrix	: B
system flexibility matrix	: $F=B^TDB$
nodal load vector	: p
system force vector	: f
nodal displacement vector	: r
system displacement vector	: v
redundant force vector	: x

The known, defining quantities for a structural analysis problem are the equilibrium matrix E, the element flexibility matrix D (or equivalently the element stiffness matrix D^{-1}), and the nodal load vector p. The unknowns to be determined are the system force vector f, the system displacement vector v, and the nodal displacement vector r. The remaining variables in Table 1, such as the redundant force vector x, are derived, intermediate quantities. We assume that E is an $m \times n$ matrix of rank m and that D is a symmetric positive definite matrix of order n. Here D is block diagonal with the diagonal blocks having orders ranging from one to six, depending on the finite element model of the structure. An example, taken from [13], of a two-dimensional frame and its equilibrium matrix are shown in Figs. 1 and 2.

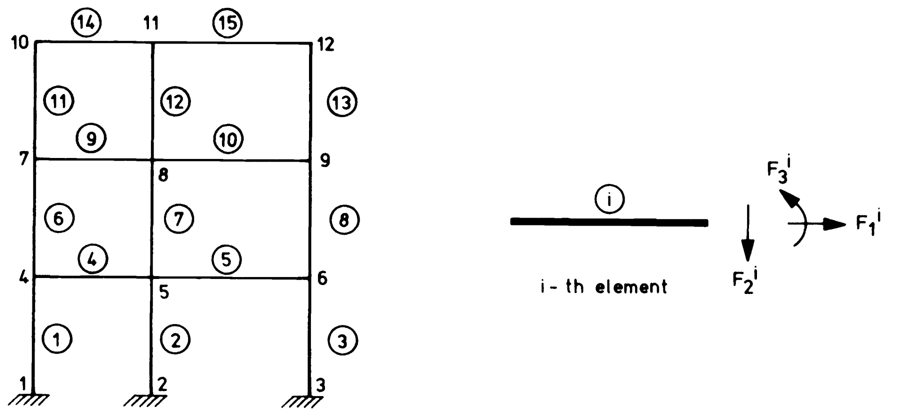


FIG. 1. *Two-dimensional frame with element and node numbering.*

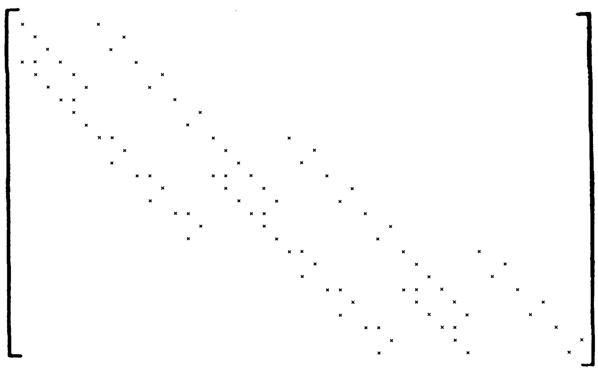


FIG. 2. *Equilibrium matrix E for two-dimensional frame.*

The linear elastic relationships for the system are

1. **Equilibrium equation:**

$$Ef=p$$

2. **Compatibility equation:**

$$E^Tr=v$$

3. **Material equation:**

$$Df=v.$$

The equilibrium and compatibility equations represent constraints on the forces and displacements, respectively, and the material equation represents the material characteristics.

The minimum principle of complementary potential energy can be stated as the quadratic programming problem

$$\min_f \tfrac{1}{2} f^T Df \quad \text{subject to} \quad Ef=p. \tag{1.1}$$

A similar minimum principle can be stated in terms of displacements rather than forces, but, as we shall see, both the force method and the displacement method can be derived from (1.1). First-order necessary conditions for a solution to the quadratic programming problem (1.1) are given by the $(m+n) \times (m+n)$ system of linear equations

$$\begin{bmatrix} D & E^T \\ E & O \end{bmatrix} \begin{bmatrix} f \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ p \end{bmatrix}, \tag{1.2}$$

where λ is an is an m-vector of Lagrange multipliers (see, e.g., [16], p. 224). A number of numerical methods are at our disposal for solving system (1.2); the force and displacement methods correspond to two of these. The most obvious approach is simply to solve the entire system directly, applying any convenient algorithm for square linear systems. System (1.2) is symmetric, but indefinite, so that some form of pivoting for numerical stability will be required if a direct elimination method is used. One standard approach to such problems, the use of block 2×2 pivots [3], has recently been implemented for large sparse systems [6].

Although not at all unreasonable, the direct solution of (1.2) is somewhat unsatisfying because one feels that the imposition of constraints ought to reduce rather than increase the dimensionality of a problem. If instead we carry out block elimination on (1.2) we obtain the smaller $m \times m$ system

$$ED^{-1}E^T\lambda = -p, \tag{1.3}$$

which can be solved for λ. We then solve for the system force vector f using the $n \times n$ block diagonal system

$$Df = -E^T\lambda. \tag{1.4}$$

The symmetric matrix $K = ED^{-1}E^T$ of system (1.3) is called the *stiffness matrix* and this approach is called the *stiffness* or *displacement method* (note that λ is simply the negative of the nodal displacement vector r). Both systems (1.3) and (1.4) are symmetric and positive definite and therefore each could be solved by applying the Cholesky factorization algorithm, for which well-developed software is available to handle large sparse problems. As with the normal-equations method for least squares problems, however, explicit formation of the cross-product matrix K worsens the condition of the problem and can lead to serious loss of accuracy.

As in the least squares case, a numerically superior alternative is provided by orthogonal transformations. First, suppose we have a factorization of the form

$$D^{-1} = GG^T, \tag{1.5}$$

where G is a square matrix of order n. Such a factorization, involving the diagonal blocks of D, is usually directly available at the element level, or could be computed if necessary by the Cholesky algorithm. Now compute the orthogonal factorization

$$EG = [L \quad O]Q, \tag{1.6}$$

where L is a nonsingular, lower triangular matrix of order m and Q is an orthogonal matrix of order n. Using the factorization (1.6), system (1.3) becomes

$$LL^T\lambda = -p, \tag{1.7}$$

which can be solved by forward and backward substitution. In effect, the Cholesky factor of K has been obtained without explicitly forming K. This approach is known as the *natural factor method* because of the use of (1.5) (see [1]).

Suppose we have solved a structural analysis problem for given E, D, and p. With either the standard stiffness method or the natural factor variant of it, if the element flexibility matrix D is now changed, then the entire computation must be repeated from the beginning. Such a situation is not uncommon in multiple redesign problems, structural optimization problems, and nonlinear elastic analysis, in which the geometry of the system is fixed but the material properties vary over a sequence of problems. In such cases a more efficient approach is provided by the force method (see, e.g., [20] and [13]).

The force method is motivated by the observation that problem (1.1) is asking for a weighted, minimum-norm solution to the equilibrium equation. Given any particular solution s to such an underdetermined system, any other solution can be expressed as a sum of s and a solution to the corresponding homogeneous system $Ef = 0$. Thus we may write the system force vector f as

$$f = s + Bx, \tag{1.8}$$

where the $n \times (n-m)$ matrix B, called the *self-stress matrix*, is chosen so that its columns form a basis for the null space of E. Once s and B have been determined, we then need to determine the proper linear combination of the columns of B, represented by the vector x, so that f solves problem (1.1). If we substitute the representation for f given by (1.8) into the upper row of system (1.2) we get the system

$$DBx = -Ds - E^T\lambda. \tag{1.9}$$

If we now premultiply system (1.9) by B^T and note that $B^TE^T = O$, we obtain the system

$$B^TDBx = -B^TDs, \tag{1.10}$$

from which we can now compute x, and thence f. The vector x is called the *redundant force vector* and the symmetric matrix $F = B^TDB$ of system (1.10) is called the *system flexibility matrix*. The displacements, if needed, can be computed from the overdetermined but consistent system

$$E^Tr = v = Df.$$

The force method is usually carried out in two phases:

Phase 1. Compute a particular solution s of the equilibrium equation, together with a self-stress matrix B such that $EB = O$.

Phase 2. Compute the redundant force vector x from system (1.10) and the system force vector f from (1.8).

Since phase 1 depends only on E and p, it need be executed only once in order to solve a sequence of problems which differ only in D, with just phase 2 being repeated for each new value of D. In order for the force method to be viable in practice, however, we need numerically stable algorithms for both phases which preserve sparsity. This is the subject of the next two sections.

2. The force method: phase 1.

2.1 General methods. There are two basic approaches to phase 1 of the force method: elimination and orthogonalization. Both approaches can be implemented in several different ways, leading to a number of distinct methods for performing phase 1. (See [4] for a survey of methods for solving general underdetermined systems.) Each method requires a preliminary factorization of the equilibrium matrix E, then s and B are computed using the resulting factors. In order to preserve numerical stability, sparsity, or both, it may be necessary to perform row and column interchanges on the equilibrium matrix E prior to, or as part of, the factorization process. Such row and column permutations are equivalent to relabeling the equations and unknowns in the equilibrium equation, and may be expressed by permutation matrices P_1 and P_2 of order m and n, respectively. Thus, in effect, the factorization is actually carried out on the reordered matrix P_1EP_2.

We now list the principal methods for phase 1, then discuss their relative merits. In some of the factorizations we will need identity matrices of various sizes, and so for clarity a subscript will be used to indicate the order of each identity matrix.

1. **Gauss-Jordan elimination.**

$$P_1EP_2=M[I_m \;\; J],$$

$$s=P_2\begin{bmatrix} I_m \\ O \end{bmatrix}M^{-1}P_1p,$$

$$B=P_2\begin{bmatrix} -J \\ I_{n-m} \end{bmatrix},$$

where M is $m\times m$ and J is $m\times(n-m)$.

2. **Row elimination.**

$$P_1EP_2=L[U_1 \;\; U_2], \tag{2.1}$$

$$J=U_1^{-1}U_2, \tag{2.2}$$

$$s=P_2\begin{bmatrix} I_m \\ O \end{bmatrix}U_1^{-1}L^{-1}P_1p, \tag{2.3}$$

$$B=P_2\begin{bmatrix} -J \\ I_{n-m} \end{bmatrix}, \tag{2.4}$$

where L and U_1 are lower and upper triangular matrices, respectively, of order m, and U_2 is $m\times(n-m)$.

3. **Column elimination.**

$$P_1EP_2=[L \;\; O]U,$$

$$[N_1 \;\; N_2]=U^{-1},$$

$$s=P_2N_1L^{-1}P_1p,$$

$$B=P_2N_2,$$

where L is a lower triangular matrix of order m, U is an upper triangular matrix of order n, and N_1 and N_2 are $n\times m$ and $n\times(n-m)$ matrices, respectively.

4. ***QR* factorization.**

$$P_1EP_2=Q[R_1 \;\; R_2], \tag{2.5}$$

$$J=R_1^{-1}R_2, \tag{2.6}$$

$$s=P_2\begin{bmatrix} I_m \\ O \end{bmatrix}R_1^{-1}Q^TP_1p, \tag{2.7}$$

$$B=P_2\begin{bmatrix} -J \\ I_{n-m} \end{bmatrix}, \tag{2.8}$$

where R_1 is an upper triangular matrix of order m, R_2 is $m\times(n-m)$, and Q is an orthogonal matrix of order m.

5. ***LQ* factorization.**

$$P_1EP_2=[L \;\; O]Q,$$
$$[Q_1 \;\; Q_2]=Q^T,$$
$$s=P_2Q_1L^{-1}P_1p,$$
$$B=P_2Q_2,$$

where L is a lower triangular matrix of order m, Q is an orthogonal matrix of order n, and Q_1 and Q_2 are $n\times m$ and $n\times(n-m)$ matrices, respectively.

6. **Singular value decomposition.**

$$P_1EP_2=V[\Sigma \;\; O]Q,$$
$$[Q_1 \;\; Q_2]=Q^T,$$
$$s=P_2Q_1\Sigma^{-1}V^TP_1p,$$
$$B=P_2Q_2,$$

where Σ is a nonnegative diagonal matrix of order m, V and Q are orthogonal matrices of order m and n, respectively, and Q_1 and Q_2 are $n\times m$ and $n\times(n-m)$ matrices, respectively.

For convenience in describing the various methods we have reused some symbols (e.g., J, L, Q) which do not necessarily have the same meaning for different methods; all symbols should be taken as locally defined for a particular method only. For each method it is easily verified by direct substitution that $Es=p$ and $EB=O$. Of course, the indicated matrix inverses are not computed explicitly, but are implicitly handled by simultaneously transforming the right hand side during the factorization process and by backsubstitution in triangular systems.

The main points of comparison among the methods are computational efficiency, storage requirements, and numerical properties. Method 1, Gauss-Jordan elimination, is the method traditionally used by structural engineers, but it has severe limitations when dealing with large sparse problems. A very large number of eliminations is generally required in order to produce the identity matrix I_m, and they can cause a great deal of intermediate fill, which will itself eventually be eliminated, but in the interim must be stored. In addition, the matrix J is likely to be rather dense and therefore B will require a great deal of storage unless $n-m$ is quite small. Finally, Gauss-Jordan elimination can encounter numerical difficulties, even when pivoting is used (see [12], pp. 50-51).

Method 2, row elimination, is superior to method 1 numerically, requires less work, and causes less intermediate fill. Even though the matrix J is not produced directly but must be computed by backsubstitution in the triangular system $U_1J=U_2$, this is more than offset by the lower cost and fill of the initial factorization. Moreover, there exists good software for computing LU factorizations of sparse matrices (e.g., [5], [21], [23]). In addition to trying to limit fill, it is also desirable to keep U_1 and L as well conditioned as possible, and so a compromise pivoting strategy should be used. Unfortunately, it is difficult to control the fill in J, which may be quite dense except in special situations which we will discuss later. In fact, for the same permutation matrices P_1 and P_2, methods 1 and 2 produce the same matrix J.

Method 3, column elimination, is comparable in work and fill to Gauss-Jordan elimination and also tends to yield a rather dense matrix B. The self-stress matrix B which results from applying method 2 to the equilibrium matrix of Fig. 2 is shown in Fig. 3.

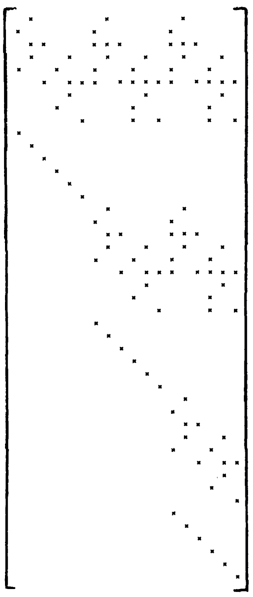


FIG. 3. *Self-stress matrix B.*

Methods using orthogonalization are numerically superior to their counterparts which use elimination, but orthogonalization methods generally require more work and cause more fill than elimination-based methods. Moreover, even if the triangular factor is relatively sparse, it is unlikely that comparable sparsity can be maintained in the orthogonal factor. Methods 4 and 5 are obvious analogues of methods 2 and 3, respectively, but despite their numerical superiority they have been little used in conjunction with the force method because of the lack of effective implementations for sparse problems. Fortunately, an effective sparse orthogonal factorization scheme has recently been developed in the context of linear least squares problems [8]. In that algorithm the orthogonal matrix Q is computed as a product of a sequence of Givens rotations, which are discarded as they are used in order to conserve storage. Such an approach is inappropriate for method 5, which makes explicit use of the orthogonal factor Q in defining s and B, but is ideally suited to implementing method 4, which makes no use of Q after the initial factorization (the vector $Q^T P_1 p$ which is needed to compute s as in (2.7) can be computed during the factorization process by applying the rotations to the permuted right hand side).

In computing an orthogonal factorization of the form (2.5), column pivoting is normally used in order to guarantee that R_1 is nonsingular and as well conditioned as possible. In the algorithm of [8], however, the column ordering P_2 is fixed in advance in order to maintain sparsity in the triangular factor, and therefore column pivoting for numerical reasons is not permissible. It is shown in [11] how a factorization of the form (2.5) can still be computed without explicit column pivoting. In effect, when all processing of the rows of E by Givens rotations is complete, in the sparse triangular data structure of order n there are $n-m$ null rows. The columns of R_1 and R_2 are those corresponding to nonzero and zero diagonal entries, respectively. (See [11] for details, especially regarding numerical tolerances required.) The net result is that this algorithm provides a sparse implementation of method 4 which is analogous to a sparse LU implementation of method 2. Unfortunately, it also shares with method 2 the tendency to produce a rather dense matrix B. We will address this difficulty for both methods shortly.

Method 6, the singular value decomposition, is theoretically equivalent (but numerically superior) to the eigenforce method [20], in which the eigenvalue-eigenvector decomposition of the $n \times n$ matrix $E^T E$ is computed. Although the singular value decomposition is the most reliable method of all numerically, it is prohibitively expensive in both work and storage for large problems and is therefore not a viable alternative for large-scale structural analysis.

From the above discussion it is evident that the most promising methods for phase 1 are methods 2 and 4, with the former likely to be the most efficient, while the latter is more reliable numerically. Since we regard numerical reliability as of paramount importance, we will emphasize method 4, QR factorization, throughout the remainder of this paper, although we will rely on recent developments of method 2 [13] for motivation.

Before we leave the general discussion of methods, it is perhaps worthwhile to point out that the natural factor approach to the displacement method can also be exploited in the force method. In particular, if we have a factorization of the form (1.5), then we can apply any of the orthogonalization methods to EG instead of E. By treating the problem explicitly as a weighted least squares problem in this manner, the system force vector f can be computed directly without the need for solving system (1.10) in phase 2. In doing so, however, we would be giving up the principal advantage of the force method, since B would then depend on D and would have to be recomputed whenever D changed.

2.2 Band schemes. For a general sparse equilibrium matrix E it is difficult to say anything about the structure of the resulting self-stress matrix B other than that it might be quite full. If E has a banded structure, however, Topcu [22] and Kaneko, Lawo and Thierauf [13] have shown how method 2 can be adapted to compute a B which also is banded. By modifying and extending their algorithm, we will show how method 4 can also be used to produce a banded self-stress matrix B. Since E and B are not square matrices, perhaps we should first point out that we do not mean banded with respect to the usual main diagonal, but with respect to a line from the upper left corner to the lower right corner of the matrix. Also, for our purposes the distinction between band and profile (also known as variable band, envelope, or skyline) implementations is unimportant, and so we will use the shorter term "band" to mean both possibilities.

For most structural analysis problems there is a fairly obvious natural ordering of the nodes and elements of the underlying physical structure in which the equilibrium matrix E is banded [20]. In any case we give a simple recipe for finding such an ordering for any sparse E:

1. Apply a bandwidth minimization algorithm (e.g., Reverse Cuthill-McKee [10]) to the nonzero structure of the symmetric matrix E^TE. This gives a column permutation P_2.

2. Sort the rows of EP_2 into increasing order with respect to the subscript of the first (i.e., leftmost) nonzero entry in each row, breaking ties arbitrarily. This gives a row permutation P_1.

The resulting matrix P_1EP_2 will have the desired band structure. (Note that, due to the combinatorial difficulty of bandwidth minimization and the consequent heuristic nature of the algorithms, the resulting band structure is not necessarily optimal, but is usually nearly optimal. Also, the matrix E^TE is not formed numerically, but its nonzero structure is analyzed symbolically.) We will henceforth assume that the equilibrium matrix E is given with such an ordering. Some additional reordering of rows and columns may be required by the algorithms given below.

We now consider what happens when methods 2 and 4 are applied to a banded equilibrium matrix E. In order to produce a triangular factorization of the form (2.1) or (2.5) we annihilate the subdiagonal elements of each successive column of E using either Gaussian elimination or Givens rotations. In either case the computations are restricted in each column to the block of rows having nonzeros in that column, and, due to the band structure, this block should be relatively small. In order to ensure that the multipliers are less than 1 in Gaussian elimination, row interchanges may be required, but only within blocks. A convenient way in which to implement this procedure is to set up a banded triangular data structure based on symbolic factorization of the (reordered) symmetric matrix E^TE (see step 1 of the reordering algorithm given above), then process the rows of E one by one, reading them from an external file. Alternatively, if E is stored in main memory, then the factorization can be done in place, but additional storage must be available to allow for possible growth in the band due to rotations or row interchanges.

With either elimination or rotations, as we proceed down the main diagonal of E we may encounter a column which is already zero on and below the main diagonal. Indeed, if the bandwidth of E is sufficiently small, this situation must occur before we reach the bottom of E, since the first k columns of E cannot be linearly independent if the (k,k) element of E lies to the left of the band. If column k is such a column, then in order to ensure that our final triangular matrix U_1 or R_1 is nonsingular, we move column k to the $(m+1)$ st position and move columns $k+1$ through $m+1$ to the left by one position before continuing with the annihilation procedure. Of course it is not necessary actually to move the columns in storage, but merely to record the permutation implicitly by means of relabeling. We should also note that in floating point arithmetic one would not expect to find exact zeros but would instead test against some small tolerance. In this regard, the use of orthogonal transformations should prove superior to elimination at detecting such linear dependencies. When the subdiagonal annihilation process is complete and the bottom of E is reached, the resulting factorization will have the form schematically depicted in Fig. 4. Thus, the matrix U_2 or R_2 inherits a lower band structure from E.

The structure we have just described would be of little benefit were it not for the fact that it carries over to the resulting self-stress matrix B. Specifically, the matrix J formed according to (2.2) or (2.6) has the same lower band structure as U_2 or R_2, and when we form the matrix B according to (2.4) or (2.8) the permutation matrix P_2

(which incorporates the column permutations described in the previous paragraph) simply places the rows of the identity matrix I_{n-m} at the "breakpoints" in the structure of J, thereby preserving a lower band structure. These concepts are illustrated pictorially in Fig. 5.

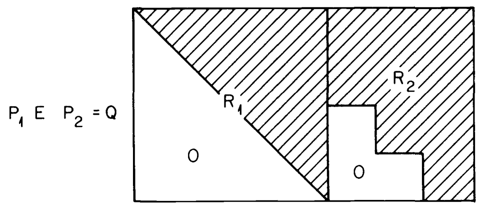


FIG. 4. *Orthogonal factorization of E.*

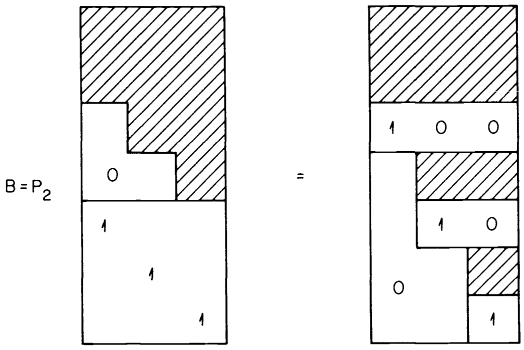


FIG. 5. *Self-stress matrix B.*

Lower band structure alone is not enough to make the storage of B practical. Therefore we now outline a "turnback" procedure, based on that originally developed in [22] and [13], which determines a self-stress matrix having an upper band structure as well. For each column j of B, let

$$k_j = \min\{k : b_{ij} = 0 \text{ for all } i > k\}.$$

(The k_j are just a convenient way of describing the lower band structure of B; the k_j indicate where the rows of I_{n-m} are located within B.) Since $EB = O$, each column of B reveals a linear dependency among the columns of E; indeed, for each j the first k_j columns of E are linearly dependent. The idea of the turnback procedure is, for each column of B, to find a smaller subset of the columns of E which is still linearly dependent. In order to find such a subset we begin with column k_j of E and work backward through the columns of E until a dependent set is found. Linear dependence can be determined numerically by either LU or QR factorization, implemented as in methods 2 and 4 above. The turnback procedure was originally proposed using LU factorization, but an orthogonalization approach provides a more robust indicator of linear dependence, so we state our version of the algorithm in that form.

We now consider the first step of the turnback procedure. Let

$$t_1=\max\{1,k_1-m-1\}.$$

Let E_1 be the matrix whose columns are columns k_1, k_1-1, ... , t_1 of E, in that order. If we carry out the orthogonalization process on E_1 analogous to (2.5) but stop when the first zero diagonal element is encountered, the result is a factorization of the form

$$E_1=Q_1\begin{bmatrix} T_1 & c_1 & V_1 \\ O & 0 & W_1 \end{bmatrix}, \tag{2.9}$$

where T_1 is a nonsingular, upper triangular matrix of order s_1, say. (The subscripts in (2.9) are merely meant to indicate that this is the first step of the turnback procedure, and have no relation to any subscripts used earlier.) Thus we now know that columns k_1-s_1-1, ... , k_1 of E are linearly dependent, and we can determine the coefficients of that dependency by solving the triangular linear system $T_1y_1=c_1$. We then have

$$E_1\begin{bmatrix} y_1 \\ -1 \\ 0 \end{bmatrix}=0.$$

Permuting this vector back into the column ordering of E and augmenting it with zeros corresponding to those columns of E which are not in E_1, we obtain a vector z_1 which belongs to the null space of E. In Fig. 6 this new null-space vector z_1 is compared schematically with the first column of the matrix B computed by (2.8). The vector z_1 forms the first column of a new, banded null-space basis matrix Z.

The remaining steps 2, ..., $n-m$ of the turnback procedure are similar to step 1, but with one important difference. At step j we start with column k_j of E and work backward, but we want to avoid the possibility of finding again any linear dependency which has already been found on a previous step. Thus at step j we omit from the matrix E_j the dependent column (i.e., the column where the zero diagonal occurred)

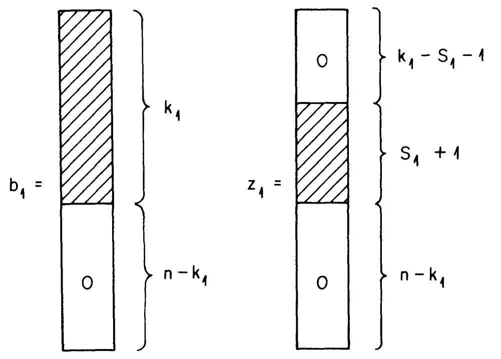


FIG. 6. *Columns of B and Z compared.*

from each previous step. This strategy may result in some "spikes" in the upper band structure of Z, but is necessary to ensure that Z has full column rank. It is important to note that we do not need the matrix B in order to generate Z, but only the indices k_j, which are directly available from the factorization (2.5). Thus we need allocate only enough storage for the banded basis matrix Z, not the much more full matrix B. Of course a great deal of computation is required to generate a banded basis matrix in this manner, but the philosophy of the force method is that this investment is worthwhile if the matrix is to be used repeatedly in phase 2 to solve a long sequence of related problems. The banded self-stress matrix resulting from applying the turnback-QR process to the equilibrium matrix of Fig. 2 is shown in Fig. 7.

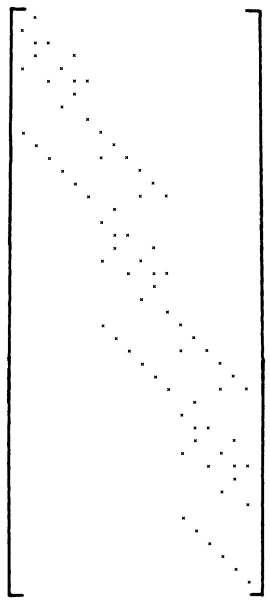


FIG. 7. *Banded self-stress matrix Z.*

The turnback procedure described above is modeled after that of [22] and [13], although there are some important differences (these include the use of orthogonal transformations in finding the linear dependencies as well as in the original factorization, the way the coefficients of the dependencies are determined, and the omitting of dependent columns to ensure full rank). A more radically different approach is to compute a banded null-space basis matrix by applying a turnback procedure directly to the matrix $[R_1, R_2]$ of (2.5) (or the matrix $[U_1, U_2]$ of (2.1) if elimination is being used), rather than the original matrix E. This is certainly possible, since the nonsingularity of Q implies that E and $[R_1, R_2]$ have the same null-space.

A turnback procedure can be implemented on the matrix $[R_1, R_2]P_2^T$ (i.e., in the original column order of E) by simply rotating each column having a zero diagonal into the preceding columns, in order from right to left. As soon as such a column is annihilated, a linear dependency has been found. What we are in fact doing is computing the complete orthogonal factorization (see [15], p. 13)

$$P_1EP_2=Q[R \quad O]V, \tag{2.10}$$

where R is a nonsingular upper triangular matrix of order m and Q and V are orthogonal matrices of order m and n, respectively. If we now partition V as

$$V^T = [V_1 \;\; V_2],$$

where V_2 is $n\times(n-m)$, then P_2V_2 is a basis for the null space of E. If the columns of R_2 in (2.5) are annihilated in the indicated order (left to right and bottom to top) in arriving at (2.10), then P_2V_2 will be banded. These concepts are illustrated in Fig. 8.

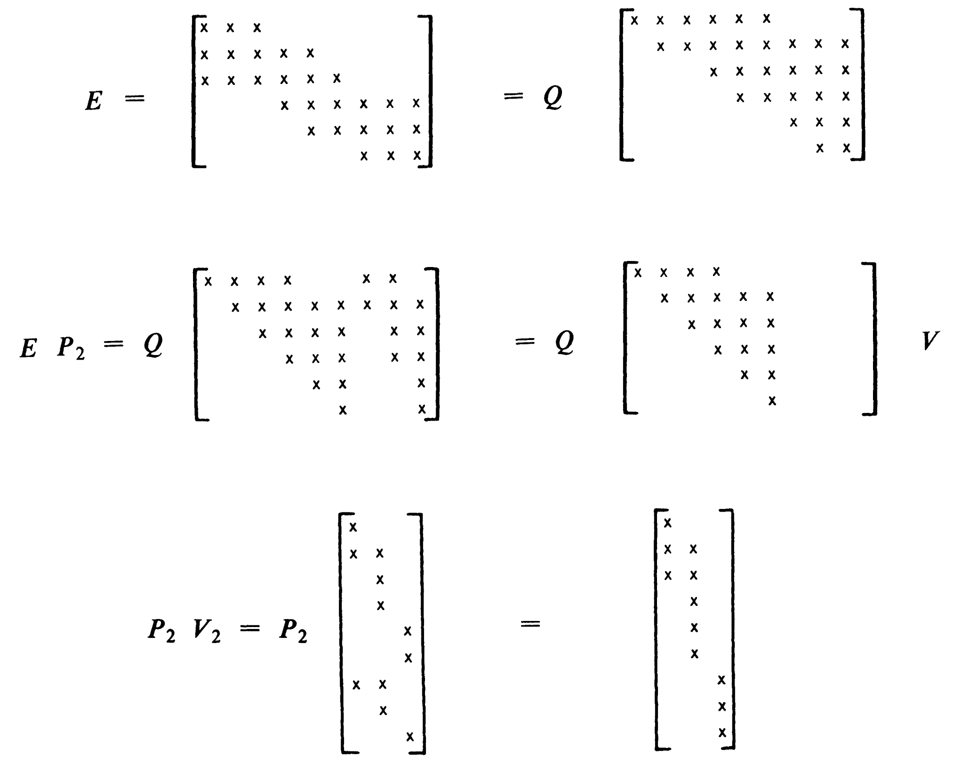


FIG. 8. *Complete orthogonal factorization and resulting self-stress matrix. (Nonzero structure depends on values of numerical entries.)*

This procedure applied to $[R_1, R_2]$ should require about the same amount of work as the turnback procedure applied to E, since the bandwidths of the two matrices are about the same. Determining linear dependence in terms of the computed quantities in $[R_1, R_2]$ might be less robust than working with the original data in E, but this effect is lessened by using orthogonal transformations at all stages.

3. The force method: phase 2.

3.1 General methods. Given a particular solution s of the equilibrium equation and a self-stress matrix B whose columns form a basis for the null space of E, the main task of phase 2 of the force method is the computation of the redundant force vector x which satisfies system (1.10). The system force vector f is then given by (1.8).

System (1.10) is simply the normal equations for the weighted least squares problem

$$\min_x \|G^{-1}(Bx+s)\|_2, \tag{3.1}$$

where G is as in (1.5). Several methods for solving problems of the form of (3.1) are described in [15]. The traditional method of normal equations consists of the direct application of Cholesky's method to the symmetric positive definite matrix $F=B^TDB$. With the reordering necessary to limit fill for large sparse problems, this gives an algorithm of the form

Normal equations

$$P_2FP_2^T=R^TR,$$

$$y=-R^{-T}P_2B^TDs,$$

$$x=P_2^TR^{-1}y,$$

where R is an upper triangular matrix of order $n-m$ and P_2 is a permutation matrix of order $n-m$. (Again, the indicated inverses imply backsubstitution processes rather than explicit inversion.) Excellent computer software is available for efficiently carrying out this approach for large problems (e.g., [6], [7], [10]).

Unfortunately, explicitly forming the cross product matrix $F=B^TDB$ can lead to significant loss of information and worsening of the conditioning of the problem. A better approach in this regard is to apply orthogonal transformations to the matrix $G^{-1}B$, leading to an algorithm of the form

Orthogonal factorization

$$P_1G^{-1}BP_2^T=Q\begin{bmatrix}R\\O\end{bmatrix},$$

$$\begin{bmatrix}c\\d\end{bmatrix}=-Q^TP_1G^{-1}s,$$

$$x=P_2^TR^{-1}c,$$

where R is an upper triangular matrix of order $n-m$, P_1 and P_2 are permutation matrices of order n and $n-m$, respectively, Q is an orthogonal matrix of order n, and c and d are vectors of length $n-m$ and m, respectively. An implementation of this approach which is effective for large sparse problems is given in [8]. Comparisons of the normal equations, orthogonal factorization, and other methods on a variety of finite element and other types of problems are given in [8] and [9].

3.2 Paige's method. Although orthogonal factorization is numerically superior to the normal equations, poor results may be obtained with either method when the element flexibility matrix D is ill conditioned. In a series of papers ([17], [18], [19], [14]) Paige has developed a scheme which can considerably reduce this difficulty. Following Paige, if we define the weighted residual vector

$$u=G^{-1}(Bx+s),$$

then problem (3.1) can be written in the equivalent form

$$\min_{x,u} \; u^T u \quad \text{subject to} \quad Gu = Bx + s. \tag{3.2}$$

Because of its special form, problem (3.2) is sometimes referred to as the linearly constrained sum-of-squares (*LCSS*) problem. In addition to leading to a better numerical method, (3.2) also has important theoretical advantages over (3.1) in that it requires no restrictive assumptions regarding the ranks of the matrices involved. In particular, it is possible to compute a G satisfying (1.5) and suitable for use in (3.2) even if the element stiffness matrix is only semidefinite (see [15], p. 124). For this reason, problem (3.2) has also been referred to as generalized least squares. We will not need the full generality of *LCSS*, since we assume that B has been determined so that it has full column rank.

We first present a formulation of Paige's *LCSS* scheme which is easy to understand, then present a second formulation which is usually more efficient. In order to avoid excessively complicated notation in describing the algorithms, we will omit the permutation matrices which may be necessary in order to preserve sparsity, ensure numerical stability, or estimate rank.

First formulation [18]. We begin by performing the orthogonal transformation

$$Q^T B = \begin{bmatrix} R \\ O \end{bmatrix}, \quad Q^T G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix}, \quad Q^T s = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}, \tag{3.3}$$

where Q is an orthogonal matrix of order n, R is a nonsingular upper triangular matrix of order $n-m$, and the remaining matrices are partitioned conformally. Applying this orthogonal transformation to (3.2), the constraint equation breaks into the two equations

$$G_1 u = Rx + s_1, \tag{3.4a}$$

$$G_2 u = s_2. \tag{3.4b}$$

Since (3.4a) can be solved for x given any value of u, problem (3.2) therefore becomes

$$\min_{u} \; u^T u \quad \text{subject to} \quad G_2 u = s_2. \tag{3.5}$$

Problem (3.5) is simply that of computing the minimum norm solution to an underdetermined linear system and can therefore be solved by any of the methods discussed previously for phase 1. Using method 5, for example, we compute an orthogonal factorization

$$G_2 = [L \;\; O]V, \tag{3.6}$$
$$[V_1 \;\; V_2] = V^T,$$

where L is a lower triangular matrix of order m, V is an orthogonal matrix of order n, and V_1 and V_2 are $n \times m$ and $n \times (n-m)$ matrices, respectively. Assuming L is nonsingular we now obtain

$$u = V_1 L^{-1} s_2,$$

then x is recovered from the triangular system (3.4a). Singularity of R or L due to rank deficiency of B or G can also be handled by this scheme (see [18] for details).

Second formulation [14]. The first formulation does not take advantage of any special structure the matrix G may have (G will be triangular if it is computed by Cholesky factorization, and in our case G has block diagonal structure as well); indeed, that structure is in general destroyed by the first orthogonal transformation Q. Paige has given a second formulation of *LCSS* in which the two orthogonal transformations Q and V are computed simultaneously in a manner which retains the triangular structure of G throughout the computations. The approach taken is a familiar "zero chasing" technique using Givens rotations. For implementation details, see [19].

The result is a factorization of the form

$$Q^T[s,\ B,\ GV]=\begin{bmatrix} y_1 & o & L_1 & O \\ y_2 & R^T & L_{21} & L_2 \end{bmatrix}, \tag{3.7}$$

where R^T, L_1, and L_2 are lower triangular matrices of order $n-m$, m, and $n-m$, respectively, and Q and V are orthogonal matrices of order n and m, respectively. (Note that the matrices in (3.7) are not necessarily identical to the corresponding matrices in (3.3) and (3.6).) Applying transformation (3.7) and using the change of variable

$$V^T u=\begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

problem (3.2) becomes

$$\min_{u_1,u_2}\ u_1^T u_1+u_2^T u_2 \quad \text{subject to} \quad \begin{bmatrix} L_1 & O \\ L_{21} & L_2 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}=\begin{bmatrix} O \\ R^T \end{bmatrix}x+\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}.$$

Thus u_1 is completely determined by the equation

$$L_1 u_1=y_1,$$

and the functional is minimized by taking $u_2=o$. Finally, the solution x may now be determined from the system

$$R^T x=L_{21}u_1-y_2.$$

Again we have not used the full generality of Paige's method, which allows rank degeneracy in B and G, leading to trapezoidal rather than triangular matrices in the factorization (3.7).

In both formulations of *LCSS* the computations can be arranged so that it is unnecessary to save the orthogonal matrices Q and V. This is an important factor in keeping storage requirements to a minimum. For solving a sequence of problems with fixed B but varying G, in theory it is necessary to compute the orthogonal transformation Q only once. This approach is in keeping with the philosophy of the force method, but its computational advantage may be nullified by the necessity of storing Q for subsequent use.

Our interest is in problems for which B is banded and G is block diagonal and lower triangular. By a careful implementation of the Givens rotations which carry out the orthogonal factorization (3.7), these properties can be used to advantage and at least partially preserved. For example, consider the matrix shown in Fig. 2, which leads to a stacked banded form (see Fig. 7)

$$B=\begin{bmatrix}B_1\\B_2\\B_3\end{bmatrix},$$

where each B_i is banded, with more rows than columns. If G is partitioned conformally, the constraint equation of (3.2) becomes

$$\begin{bmatrix}G_1 & O & O\\O & G_2 & O\\O & O & G_3\end{bmatrix}u=\begin{bmatrix}B_1\\B_2\\B_3\end{bmatrix}x+s. \tag{3.8}$$

In carrying out the factorization (3.7), after the first two blocks of B have been reduced to lower triangular form, only the block immediately below G_1 in G will have filled in. As observed in [19], p. 171, we may at this point drop the columns of G corresponding to G_1 and continue the computation with a problem having the same form as (3.8) but with one less block. This technique generalizes to problems having any number of blocks, and such structure is common in the analysis of two- or three-dimensional frames. Dropping columns in this manner is of critical importance for large problems, since otherwise a full lower triangular matrix L could be generated. Another important observation in determining storage requirements is that R^T has the same band structure as the lower triangle of B^TB.

4. Concluding remarks. The force method is an attractive method for solving sequences of related structural analysis problems. We have suggested an implementation of the force method which uses turnback-*QR* for phase 1 and Paige's *LCSS* for phase 2. Such an approach has two principal advantages: band structure is exploited in order to solve very large problems, and orthogonal transformations are used to maintain numerical stability and avoid forming normal equations. We therefore expect the force method to become a more computationally viable alternative to the displacement method for structural optimization.

The algorithms we have described are currently being implemented in computer programs and tested on a collection of practical structural analysis problems kindly provided by I. Kaneko of the University of Wisconsin and M. Lawo of the University of Essen. Our test problems include two- and three-dimensional frames, and plane stress, plate bending, and turbine problems. The results of these numerical tests and comparisons will be reported in detail elsewhere [2].

REFERENCES

[1] J. H. Argyris and O. E. Brönlund, *The natural factor formulation of the stiffness matrix displacement method*, Comput. Meth. Appl. Mech. Engrg., 5 (1975), pp. 97-119.

[2] M. W. Berry, M. T. Heath, R. J. Plemmons and R. C. Ward, *Numerical comparisons of some elimination and orthogonal schemes for computing a banded basis for the null space*, in preparation.

[3] J. R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp., 31 (1977), pp. 163-179.

[4] R. E. Cline and R. J. Plemmons, *l_2-solutions to underdetermined linear systems*, SIAM Rev., 18 (1976), pp. 92-106.

[5] I. S. Duff, MA28 - *A set of Fortran subroutines for sparse unsymmetric linear equations*, Report R.8730, AERE, Harwell, England, July 1977.

[6] I. S. Duff and J. K. Reid, MA27 - *A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Report R.10533, AERE, Harwell, England, 1982.

[7] S. C. Eisenstat, M. C. Gursky, M. H. Schultz and A. H. Sherman, *Yale sparse matrix package I: the symmetric codes*, Int. J. Numer. Meth. Engrg., 18 (1982), pp. 1145-1151.

[8] A. George and M. T. Heath, *Solution of sparse linear least squares problems using Givens rotations*, Linear Algebra Appl., 34 (1980), pp. 69-83.

[9] A. George, M. T. Heath and E. Ng, *A comparison of some methods for solving sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., 4 (1983), pp. 177-187.

[10] A. George and J. W.-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[11] M. T. Heath, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 223-237.

[12] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, John Wiley and Sons, New York, 1966.

[13] I. Kaneko, M. Lawo and G. Thierauf, *On computational procedures for the force method*, Int. J. Numer. Meth. Engrg., 18 (1982), pp. 1469-1495.

[14] S. Kourouklis and C. C. Paige, *A constrained least squares approach to the general Gauss-Markov linear model*, J. Amer. Stat. Assoc., 76 (1981), pp. 620-625.

[15] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[16] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, MA, 1973.

[17] C. C. Paige, *Numerically stable computations for general univariate linear models*, Comm. Stat., B7 (1978), pp. 437-453.

[18] C. C. Paige, *Computer solution and perturbation analysis of generalized linear least squares problems*, Math. Comp., 33 (1979), pp. 171-183.

[19] C. C. Paige, *Fast numerically stable computations for generalized linear least squares problems*, SIAM J. Numer. Anal., 16 (1979), pp. 165-171.

[20] J. Robinson, *Integrated Theory of Finite Element Methods*, John Wiley and Sons, New York, 1973.

[21] A. H. Sherman, NSPIV - *A Fortran subroutine for sparse Gaussian elimination with partial pivoting*, ACM Trans. Math. Software, 4 (1978), pp. 391-398.

[22] A. Topcu, *A contribution to the systematic analysis of finite element structures using the force method* (in German), Doctoral Thesis, University of Essen, Germany, 1979.

[23] Z. Zlatev, J. Wasniewski, and K. Schaumburg, Y12M - *Solution of large and sparse systems of linear algebraic equations*, Springer-Verlag, New York, 1981.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

NONLINEARLY PRECONDITIONED KRYLOV SUBSPACE METHODS FOR DISCRETE NEWTON ALGORITHMS*

TONY F. CHAN† AND KENNETH R. JACKSON‡

Abstract. We propose an algorithm for implementing Newton's method for a general nonlinear system $f(x)=0$ where the linear systems that arise at each step of Newton's method are solved by a preconditioned Krylov subspace iterative method. The algorithm requires only function evaluations and does not require the evaluation or storage of the Jacobian matrix. Matrix-vector products involving the Jacobian matrix are approximated by directional differences. We develop a framework for constructing preconditionings for this inner iterative method which do not reference the Jacobian matrix explicitly. We derive a nonlinear SSOR type preconditioning which numerical experiments show to be as effective as the linear SSOR preconditioning that uses the Jacobian explicitly.

Key words. Krylov subspace methods, conjugate gradient methods, preconditioning, nonlinear systems, discrete Newton algorithms, directional differencing

1. Introduction.

One of the most common methods for solving an n by n nonlinear systems of the form

$$f(x)=0 \tag{1}$$

is Newton's method:

Start with an initial guess x_0.

Repeat until convergence:

(2) Solve $J(x_i)\delta x=-f(x_i)$,

(3) Set $x_{i+1}=x_i+\delta x$,

where $J(x)$ is the Jacobian matrix $f_x(x)$.

For many large problems, most of the work is done in solving the linear system (2). This is usually accomplished by evaluating and LU-factoring the Jacobian J and backsolving for δx. However, when J is large and sparse, it is natural to consider iterative methods, which usually require much less storage. Moreover, truncated forms of Newton's method [4], [22], in which the linear system (2) is only solved approximately by an iterative method, can be implemented easily.

Among the most popular and successful iterative methods for large sparse systems are methods based on the Krylov subspace, for example, the Chebyshev method [14], [18] and the conjugate gradient method [14], [15]. These methods all have the property that only matrix-vector products involving the coefficient matrix are needed. In the context of applications to Newton's method, the matrix-vector product Jv can be approximated by the directional difference $(f(x+dv)-f(x))/d$, where d is the scalar difference interval. This has the advantage of requiring only function evaluations and avoiding explicit evaluation and storage of the Jacobian matrix. Thus, it is well-suited for large sparse problems, especially if the Jacobian is difficult to evaluate or store.

* Received by the editors January 27, 1983, and in revised form July 15, 1983. This work was supported in part by Air Force grant AFOSR-81-0193, by Department of Energy grant DE-ACO2-81ER10996, and by National Sciences and Engineering Research Council of Canada grant U0133.

† Computer Science Department, Yale University, New Haven, Connecticut 06520.

‡ Computer Science Department, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

Often, it is desirable to precondition the Krylov subspace methods as the rate of convergence is often unacceptably slow otherwise, especially for ill-conditioned problems. However, since most preconditioning techniques require J explicitly [2], [6], [7], [13], [16], it is not obvious how to apply them in the context of directional differencing, as J is not explicitly available. We address this problem in this paper.

After a review of discrete Newton algorithms and Krylov subspace methods in § 2, we develop in § 3 a framework for constructing nonlinear preconditionings that do not require J explicitly. We then give in § 4 a specific preconditioning algorithm that is based on nonlinear SSOR sweeps. The efficiency of this nonlinear preconditioned Krylov subspace Newton method is discussed in § 5. Numerical experiments in § 6 show that this nonlinear preconditioning is as effective as the linear SSOR preconditioning that requires J explicitly. Some conclusions are given in § 7.

2. Discrete Newton algorithms and Krylov subspace methods.

Newton's method is among the best methods for solving general nonlinear systems. Part of the reason for its success is its quadratic rate of local convergence. Also, it is often more robust than competing methods, for example quasi-Newton methods, especially for optimization problems [12], [22].

These advantages of Newton's method are offset by the requirement that, at each step, the user compute the Jacobian matrix explicitly and solve the linear system (2). For problems for which the Jacobian is difficult to calculate (for example in optimization the Jacobian corresponds to the Hessian matrix which consists of *second* derivatives of the cost function), methods have been proposed which do not require the user to compute the Jacobian explicitly. Among these are Quasi-Newton methods [5], Nonlinear Conjugate Gradient methods [8] and Discrete Newton methods [12], [22], [23].

Discrete Newton methods are perhaps the most natural of the three classes of methods, in that they retain the form of Newton's method but approximate the Jacobian matrix by finite differences of the function values. These differences are often taken along the unit coordinate directions:

$$\frac{\partial f}{\partial x_i} \approx (f(x+de_i)-f(x))/d, \tag{4}$$

where d is an appropriately chosen difference interval.[1] It is easy to see that it takes n function evaluations to obtain an approximation for J. However, for large and sparse problems, other directions may be more efficient, in terms of both work and storage. At least two approaches using finite differences are possible.

The first approach is to order the columns of J into groups according to the sparsity pattern of J so that the nonzero elements in each group can be evaluated with only *one* function evaluation by finite differencing along a carefully chosen direction. For matrices with highly structured sparsity patterns, this can result in a significant reduction in the number of function evaluations needed to obtain an approximation for J. For example, a tridiagonal Jacobian can be approximated by three function evaluations, independent of the dimension of the matrix. Curtis, Powell and Reid [17] are credited as being the first to make such an observation and they proposed a heuristic algorithm for obtaining a good (but generally not optimal) column ordering for matrices with arbitrary sparsity patterns. Recently, there has been a lot of interest in developing more efficient finite-differencing schemes by solving related graph coloring problems

[1] For reliable and automatic techniques for choosing the finite difference interval d, we refer the reader to [11], [12].

[3], [19], [26], [27]. We call a Newton method employing this approach to approximate the Jacobian the standard discrete Newton algorithm.

The second approach is rather different in that it does not attempt to compute an explicit approximation to the Jacobian matrix at all. Methods in this class use a Krylov subspace iterative method (e.g. the conjugate gradient method) to solve approximately the linear system (2) at each step of Newton's method. In the application of these iterative methods, one does not need to access the Jacobian matrix explicitly. Instead, all that is required is the ability to form matrix-vector products of the form Jv for a given vector v. The central theme of the methods in this class is to approximate matrix-vector products of this form by a directional difference:

$$J(x)v \approx (f(x+dv)-f(x))/d. \tag{5}$$

Note that the direction v is usually unknown a priori and depends on the iterates in the Krylov subspace method. Only function evaluations are needed. This method is especially attractive if only a few matrix-vector products are needed at each Newton step, as is often the case in truncated Newton algorithms. For general dense matrices that arise in optimization problems, O'Leary [23] has shown that this method takes fewer operations than the standard discrete Newton method when $n > 39$. Garg and Tapia [9] and Nash [22] have also used this technique in optimization problems. Gear and Saad [10] have employed this idea successfully in solving stiff ordinary differential equations. We shall call this the Directional Differencing Discrete Newton (DDDN) Method.

Traditionally, Krylov subspace methods were developed for matrices with special properties, e.g. symmetric positive definiteness. However, for a general function f, the Jacobian J does not necessarily possess these properties. Even for optimization problems, although J is often symmetric, it need not be positive definite. Recently, the application of Krylov subspace methods to general linear systems has attracted a lot of research interest and significant progress has been achieved. For a survey of recent research in this area the reader is referred to [7], [16].

In this paper, we are concerned with preconditioning the Krylov subspace methods in order to accelerate their convergence. Most preconditionings are based upon a splitting of the form $J = M - N$ and can be implemented by supplying a routine for computing the matrix-vector product $M^{-1}v$ for a given vector v. A good preconditioning is one for which M is a good approximation to J and $M^{-1}v$ is inexpensive to compute.

The matrix M is usually defined in terms of the elements of J. This is true, for example, of the most commonly used incomplete LU-factorizations [13], [20] and the SSOR-type preconditionings [1], [2]. This creates a problem in the context of applications to Newton's method with directional differencing since the Jacobian matrix J is not available explicitly. Computing the elements of J explicitly in order to compute and store the preconditioning M would take away the advantage of the directional differencing. It is thus desirable to be able to apply a preconditioning algorithm that requires function evaluations only and does not reference the Jacobian explicitly. In the next section, we derive a class of nonlinear preconditionings with this property.

3. Nonlinear preconditioning.

In the context of a Krylov subspace method, a preconditioning M is needed only to compute matrix-vector products of the form

$$w = M^{-1}v. \tag{6}$$

That is, we want to be able to solve the linear system

$$Mw = v. \tag{7}$$

The optimal preconditioning, at least as far as convergence is concerned, is the choice $M = J$. However, this leads to solving the linear system

$$Jw = v \tag{8}$$

which is as difficult as the original problem (2). On the other hand, it shows that, for any preconditioning M, *$M^{-1}v$ can be viewed as an algorithm for obtaining an approximate solution of* (8). With this in mind, we can approximate the term Jw in (8) by a directional difference and solve

$$F(w) \equiv (f(x+dw) - f(x))/d - v = 0 \tag{9}$$

for w. At first glance, this appears to be as difficult as the original nonlinear problem (1). The crucial observation, however, is that *any approximate method for solving* (9) *will constitute a preconditioning M for J, where M is generally a nonlinear operator.*

We note that (9) does not involve J explicitly. Also, there are many algorithms for solving nonlinear systems that do not involve the Jacobian J explicitly. Therefore, we have a framework for constructing many possible preconditionings that require function evaluations only.

One may argue that nothing is gained by this approach, because any method used to solve (9) approximately can also be applied to the original equation $f(x) = 0$ as well. There is a difference, however. Independent of the method used to solve (9), the outer method is still Newton's method, and thus for example local quadratic convergence is achievable [4]. If the same method applied to (9) is applied to (1) directly, that method will determine the convergence of the outer loop. The resulting method may converge much less quickly and, in particular, quadratic convergence may be lost. See for example Mittelmann [21] who considered an algorithm for nonlinear finite element problems in which nonlinear Gauss–Seidel iterates are accelerated by the conjugate gradient method.

Whether Newton's method is the appropriate choice for the outer algorithm probably depends on the particular problem, but there is some evidence [22] that it is among the best choice for general nonlinear problems. In any case, the general framework in this section allows *any* algorithm for solving (9) to be used as a preconditioning for the Krylov subspace method used to solve the linear systems in Newton's method. If the method used for the nonlinear preconditioning does not require explicit reference to the Jacobian, then the overall directional differencing discrete Newton algorithm can be implemented using only function evaluations. In the next section, we derive one such nonlinear preconditioning.

4. A nonlinear SSOR preconditioning.

A class of methods for solving a nonlinear system $F(w) = 0$ that does not use the Jacobian matrix explicitly is the class of

```
Nonlinear Relaxation Methods [24]:
  Start with some initial guess for $w$ and a relaxation factor $\omega$.
  Repeat until convergence:
    Do $i = 1, \cdots, n, n, \cdots, 1$
      (0) Save $(w_i)_{\text{old}} \Leftarrow w_i$.
      (1) Solve for $w_i$ in $F_i(w_1, \cdots, w_i, \cdots, w_n) = 0$
          assuming that all other components of $w$ are fixed at
          their current values.
    (2) Set $w_i \Leftarrow (1-\omega)(w_i)_{\text{old}} + \omega w_i$.
  End do
End Repeat
```

The above algorithm can be applied directly to (9), provided we specify the initial guess for w, the relaxation factor ω, and the number of times the outer loop is to be repeated. By looking at the linear SSOR preconditioning for J, we shall show that the initial guess $w=0$ and one iteration of the outer loop are appropriate choices.

Let J be written as

$$J = D - L - U$$

where D, L and U are the diagonal, the strictly lower triangular, and strictly upper triangular parts of J, respectively. The linear SSOR preconditioning for J can be written as [1], [2]

$$M = \omega(2-\omega)(D-\omega L)D^{-1}(D-\omega U).$$

Thus, $M^{-1}v$ can be computed by two backsolves with triangular matrices. However, there is another well-known interpretation for $M^{-1}v$. Consider the following two-stage iteration:

$$(D-\omega L)w_1 = ((1-\omega)D+\omega U)w_0 + \omega v, \tag{10}$$

$$(D-\omega U)w_2 = ((1-\omega)D+\omega L)w_1 + \omega v. \tag{11}$$

It is easily verified that, if $w_0=0$, then $w_2 = M^{-1}v$. On the other hand, it is also well-known that this two-stage iteration is equivalent to the nonlinear SSOR algorithm applied to the linear system $F(w) \equiv Jw - v$ with one iteration of the outer loop [24]. For nonlinear systems, $F(w)$ in (9) is an approximation to $Jw - v$. Thus, it follows that, for the nonlinear SSOR preconditioning, $w_0 = 0$ is an appropriate initial guess and the outer loop should be iterated once. It also follows that, *if f is linear, the nonlinear* SSOR *preconditioning reduces to the linear* SSOR *preconditioning for J.*

We note that the problem to be solved in Step (1) of the nonlinear SSOR preconditioning algorithm is a scalar one. Any one-dimensional nonlinear equation solver can be used. Moreover, the problem in Step (1) does not have to be solved exactly. For example, if one step of Newton's method is used [24], then we have the following algorithm:

```
Algorithm NSSORP: (One Step Nonlinear SSOR–Newton Preconditioning)
  Given $\omega$ and $v$. Returns preconditioned $v$ in $w$.
  $w \Leftarrow 0$.
  For $i = 1, \cdots, n, n, \cdots, 1$
    Compute $(DJ)_i \Leftarrow$ the $i$th diagonal elements of $J(x+dw)$.
    Set $w_i \Leftarrow w_i - \omega F_i(w)/(DJ)_i$.
```

The diagonal elements $(DJ)_i$ of the Jacobian J can either be supplied by the user or be approximated by finite differences. We note that, for efficiency, Algorithm NSSORP requires the function f to be supplied in component form.

5. Efficiency. In this section, we compare the efficiency of the following four discrete Newton algorithms:

(a) Form J using a sparse Jacobian algorithm, use a Krylov subspace method to solve (2).

(b) Form J using a sparse Jacobian algorithm, use a linear SSOR preconditioned Krylov subspace method to solve (2).

(c) Use a Krylov subspace method to solve (2), with directional differencing for Jv.

(d) Use a preconditioned Krylov subspace method to solve (2), with directional differencing for Jv and preconditioned with Algorithm NSSORP.

Note that all four methods do not require the explicit Jacobian. Method (d) is the method that we proposed in § 4. Method (c) is the same except the Krylov subspace method is not preconditioned. Methods (a) and (b) are the corresponding methods except the Jacobian J is approximated by a sparse Jacobian evaluation algorithm. These are the discrete Newton methods that one might consider using if the Jacobian is not available explicitly.

Concerning the storage requirements, all four methods need storage for the Krylov subspace method, which is usually $O(n)$. Methods (a) and (b), however, require additional storage for the Jacobian J. Thus, methods (c) and (d) might be preferred for problems where the storage of J presents difficulties, e.g. J is too large for the machine, or J is dense with no exploitable structures for storage, or J has a nontrivial sparsity structure which is not convenient to handle.

Next, we compare work. We define one function evaluation of f to be one evaluation of each component of f. First we consider the case that function evaluations are expensive compared to matrix-vector operations. All four methods need one function evaluation to evaluate the right-hand side of the linear system (2) in Newton's method. In addition, method (c) requires one function evaluation per step of the Krylov subspace method (to approximate Jv). Method (d) requires three function evaluations per step (one to approximate Jv and two for the nonlinear preconditioning). The total number of function evaluations of course depends on the number of iterations taken by the Krylov subspace method to achieve the desired accuracy. Let us denote by I_c and I_d the number of iterations taken by method (c) and (d) respectively. For methods (a) and (b), the number of function evaluations needed to evaluate J depends on the sparsity structure of J and the sparse evaluation algorithm. Let us denote this number by s. The total number of function evaluations per Newton step are summarized in Table 5.1. Methods (a) and (b) can easily be modified to reduce the number of function evaluations required at each step by employing the chord variant of Newton's method which uses the same Jacobian approximation for several steps. We note, however, that methods (c) and (d) can be modified in a similar way to reduce the number of function evaluations required at each step by reusing the projection generated from previous iterations [25]. Here, though, for simplicity, we assume these savings are not exploited.

TABLE 5.1
Total number of function evaluations per Newton step.

Method	a	b	c	d
func. eval.	$s+1$	$s+1$	I_c+1	$3I_d+1$

s: number of function evaluations needed to evaluate J.

I_c: number of iterations taken by Krylov subspace method for method (c).

I_d: number of iterations taken by Krylov subspace method for method (d).

We can make the following observations from the table. If the number of Krylov subspace iterations needed is small compared to s, then it is more efficient to use directional differencing (methods (c) and (d)). Such situations arise, for example, in the early stages of a truncated Newton algorithm [4] or if J has an advantageous distribution of eigenvalues (e.g. stiff ODEs with only a few large eigenvalues [10]). On the other hand, if the number of iterations can be reduced by a factor of more than about three by the nonlinear SSOR preconditioning, then it pays to use method (d) instead of method (c). Whether the preconditioning can achieve this depends on the eigenvalue distribution of J and on the accuracy desired. We note that for model partial differential equations, the linear SSOR preconditioning can be shown to reduce the number of iterations by a factor of $O(h^{-1/2})$, where h is the mesh size [2]. Thus, for this class of problems, the benefits of preconditioning are more pronounced for larger problems.

Next, we consider the case that function evaluations are inexpensive compared to matrix-vector operations. For example, if one function evaluation costs approximately the same as forming the matrix-vector product Jv (which is often the case for difference methods for PDEs with simple coefficients), then the costs of methods (c) and (d) would be about the same as that of methods (a) and (b), respectively.

6. Numerical experiments. We have performed some numerical experiments with the following model nonlinear partial differential equation

$$-u_{xx}+2b(e^u)_x+ce^u=R(x), \qquad 0<x<1,$$

with homogeneous Dirichlet boundary conditions. This problem is discretized on a uniform grid with n intervals using standard centered finite differencing. The function $R(x)$ is constructed so that the discrete solution has each component equal to 1. The resulting n by n nonlinear algebraic system is solved by the methods discussed in earlier sections. The coefficients b and c allow us to control the asymmetry and the diagonal dominance of the Jacobian matrix respectively. We note that in practice it would not be cost effective to solve a one-dimensional problem like this one by an iterative method. However, the distribution of the eigenvalues of J for this problem is similar to the distribution of eigenvalues for problems in higher dimensions. Thus, this problem is quite suitable for testing purposes.

For the Krylov subspace method, we used the Direct Incomplete Orthogonalization Method (DIOM) of Saad [28]. This method is related to Arnoldi's method for computing eigenvalues of general nonsymmetric matrices [29]. At the ith step, an orthogonal basis for the ith Krylov subspace is generated and the new solution is constructed so that the corresponding residual is orthogonal to this subspace. Saad has improved the basic method so that the same code can be adapted to symmetric positive definite, symmetric indefinite and general nonsymmetric problems. For the preconditioning, we fix $\omega=1$ (symmetric Gauss–Seidel).

All computations were carried out on a DEC 20, with a machine precision of about 10^{-8} (27 bit mantissa). For the finite difference interval used in the directional differencing, we use $d=10^{-4}$. For the stopping criteria of the inner loop, we use a simple form of truncated Newton strategy:

$$\|J(x_i)\delta x+f(x_i)\|/\|f(x_i)\|<10^{-i-1}.$$

For the convergence of the outer Newton iteration, we use

$$\|f(x_i)\|<10^{-4} \quad \text{and} \quad \|\delta x\|<10^{-4}+10^{-3}\|x_i\|.$$

All norms used are infinity norms.

The main objective of the experiments is to compare the performance of directional differencing methods to that of the corresponding methods using the explicit Jacobian.

For this purpose, we also tested the following two methods:

(e) Exact Jacobian used in Krylov subspace method.

(f) Exact Jacobian used in Krylov subspace method,
linear SSOR preconditioning based on the exact Jacobian.

Another objective is to show the effectiveness of the nonlinear preconditioning. Experiments were carried out with methods (c), (d), (e) and (f) for various values of n, b and c. The results are summarized in Table 6.1. In the table, the columns labelled $\|f\|$ and $\|\text{error}\|$ denote the values of $\|f(x)\|$ and the error in the solution x at termination of the Newton iteration.

TABLE 6.1
Numerical results for model problem.

					No. of DIOM iters. at each step of Newton's method							
Test	n	b	c	Method	1	2	3	4	5	6	$\|f\|$	$\|\text{error}\|$
1	20	1	1	c	20	45	61				.3 (−5)	.4 (−5)
2				e	20	45	64				.2 (−6)	.1 (−5)
3				d	8	10	10				.3 (−5)	.3 (−5)
4				f	8	10	10				.2 (−6)	.1 (−5)
5	20	1	10	c	20	25	30	40			.1 (−5)	.3 (−5)
6				e	20	25	34	35			.3 (−6)	.2 (−5)
7				d	7	7	8	9			.3 (−5)	.3 (−5)
8				f	7	7	8	9			.2 (−6)	.1 (−5)
9	20	10	1	c	20	30	50	55			.5 (−6)	.2 (−6)
10				d	7	5	7	6	7	9	.2 (−5)	1.0 (−5)
11	40	1	1	c	40	*120	108	*120			.6 (−4)	.4 (−3)
12				d	15	24	26				.6 (−5)	.5 (−4)
13	60	0	1	c	30	71	74				.8 (−5)	.9 (−4)
14				d	14	28	31				.3 (−5)	.6 (−4)
15	60	1	1	c	60	110	*140	*140	*140	*140	.5 (−4)	.2 (−3)
16				d	22	55	78				.5 (−5)	.3 (−4)

Notation: $a(b)$ means $a \times 10^b$.
$*I$ means that the desired accuracy was not achieved after I iterations.

Tests 1–4 and 5–8 show that the directional differencing algorithms behave almost exactly like the corresponding algorithm with the explicit Jacobian. They show that the nonlinear preconditioning (Algorithm NSSORP) is as effective as the linear SSOR preconditioning based on the explicit Jacobian.

Tests 9–10 show the effectiveness of the preconditioning for a more nonsymmetric problem (larger b). The reduction of the number of DIOM iterations caused by the preconditioning is more pronounced than for more symmetric problems (e.g. Tests 5 and 7 and Tests 13–14), especially near convergence of the Newton iteration where higher accuracy is needed in the inner loop.

Tests 11–12, 15–16 are similar to Tests 1 and 3 (b and c are the same), except that the size n of the system is larger. They show that the unpreconditioned algorithms converge rather slowly and take a lot of DIOM iterations to achieve the desired accuracy. On the other hand, the preconditioned algorithms converged rather fast and took much fewer iterations. These results confirm that, at least for PDE type problems, the benefit of preconditioning is more pronounced for larger problems.

We have also performed tests on some other non-PDE type problems (e.g. the PEN1 Problem tested in [22], [23]). The trends are similar to the ones reported here, namely, that the directional differencing algorithms behave almost exactly like the corresponding algorithms with the exact Jacobian.

7. Conclusion. In this paper, we propose an algorithm for implementing Newton's method that is particularly attractive for certain large sparse nonlinear systems. A preconditioned Krylov subspace method is used to solve the linear systems that arise at each step of Newton's method. However, the Jacobian matrix is neither stored nor referenced explicitly. The user supplies the nonlinear function only. The main contribution of the paper is a framework in which various nonlinear preconditionings can be derived which require function evaluations only. We derived and tested a nonlinear SSOR preconditioning.

On the limited numerical experiments that we have performed, it is as effective as the linear SSOR preconditioning that uses the exact Jacobian explicitly. These tests also indicate that the nonlinearly preconditioned discrete Newton algorithm has the potential of being as efficient and as robust as the standard discrete Newton algorithms.

REFERENCES

[1] O. AXELSSON, *A generalized* SSOR *method*, BIT, 13 (1972), pp. 443–467.

[2] R. CHANDRA, *Conjugate gradient methods for partial differential equations*, PhD thesis, Dept. Computer Science, Yale Univ., New Haven, CT, 1978.

[3] T. F. COLEMAN AND J. J. MORÉ, *Estimation of sparse Jacobian matrices and graph coloring problems*, Technical Report ANL-81-39, Argonne National Laboratory, Argonne, IL, 1981; also SIAM J. Numer. Anal., 20 (1983), pp. 187–209.

[4] R. S. DEMBO, S. EISENSTAT AND T. STEIHAUG, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[5] J. E. DENNIS AND J. J. MORÉ, *Quasi-Newton methods, motivation and theory*, SIAM Rev., 19 (1977), pp. 46–89.

[6] T. DUPONT, R. P. KENDALL AND H. H. RACHFORD, *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM J. Numer. Anal., 6 (1968), pp. 753–782.

[7] HOWARD C. ELMAN, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, PhD thesis, Yale Univ., New Haven, CT, 1982, Techreport #229.

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

[9] N. K. GARG AND R. A. TAPIA, QDN: *A variable storage algorithm for unconstrained optimization*, Technical Report, Dept. Mathematical Sciences, Rice Univ., Houston, TX, 1980.

[10] W. C. GEAR AND Y. SAAD, *Iterative solution of linear equations in* ODE *codes*, Technical Report UIUCDCS-R-81-1054, Dept. Computer Science, Univ. Illinois, Urbana, 1981; this Journal, 4 (1983), pp. 583–601.

[11] P. E. GILL, W. MURRAY, M. A. SAUNDERS AND M. H. WRIGHT, *A procedure for computing forward-difference intervals for numerical optimization*, Technical Report SOL 81-25, Systems Optimization Lab., Dept. Operations Research, Stanford University, Stanford, CA, 1981.

[12] ———, *Practical Optimization*, Academic Press, New York, 1981.

[13] IVAR GUSTAFSSON, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[14] LOUIS A. HAGEMAN AND DAVID M. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[15] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradient for solving linear systems*, J. Res. NBS, 49 (1952), pp. 409–436.

[16] KANG CHANG JEA, *Generalized conjugate gradient acceleration of iterative methods*, PhD thesis, Univ. Texas, Austin, 1982.

[17] A. R. CURTIS, M. J. D. POWELL AND J. K. REID, *On the estimation of sparse Jacobian matrices*, J. Inst. Maths. Applics., 13 (1974), pp. 117–119.

[18] T. A. MANTEUFFEL, *The Tchebychev iteration for nonsymmetric linear systems*, Numer. Math., 28 (1977), pp. 307–327.

[19] S. T. McCormick, *Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem*, Technical Report SOL 81-22, Dept. Operations Research, Stanford Univ., Stanford, CA, 1981; Math. Progr., 26 (1983), pp. 153–171.

[20] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems in which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[21] Hans D. Mittelmann, *On the efficient solution of nonlinear finite equations* I, Numer. Math., 35 (1980), pp. 277–291.

[22] Stephen G. Nash, *Truncated Newton method*, PhD thesis, Stanford Univ., Stanford, CA, 1982.

[23] Dianne P. O'Leary, *A discrete Newton algorithm for minimizing a function of many variables*, Math. Progr., 23 (1982), pp. 20–33.

[24] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[25] B. N. Parlett, *A new look at the Lanczos algorithm for solving symmetric systems of linear equations*, Lin. Alg. and Appl., 29 (1980), pp. 323–346.

[26] M. J. D. Powell and Ph. L. Toint, *On the estimation of sparse Hessian matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 1060–1074.

[27] John D. Ramsdell, *Structural analysis of large sparse systems of nonlinear equations with applications to fire modeling*, PhD thesis, Harvard University, Cambridge, MA, 1982.

[28] Y. Saad, *Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems*, Technical Report 214, Yale Univ., New Haven, CT, 1982; this Journal, 5 (1984), pp. 203–228.

[29] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, London, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

INCOMPLETE FACTORIZATION METHODS FOR FULLY IMPLICIT SIMULATION OF ENHANCED OIL RECOVERY*

G. A. BEHIE† AND P. A. FORSYTH, JR.†

Abstract. Several incomplete factorization methods (ILU) for strongly nonsymmetric block-banded systems are developed. These systems result from the coupled partial differential equations which occur in the simulation of enhanced oil recovery. Several approximations using natural, diagonal (D2) and alternating diagonal (D4) orderings are used with ORTHOMIN acceleration. These methods can all be used in conjunction with the COMBINATIVE method for multi-phase problems. Use of the modified first order factorization is also investigated. Test results are given for single- and multi-phase problems. Timings for both scalar and vector mode on the CRAY are presented.

Key words. incomplete factorization, preconditioning, nonsymmetric block-banded systems, reservoir simulation

1. Introduction.

Simulation of thermal methods for enhanced oil recovery requires the solution of coupled sets of highly nonlinear partial differential equations. Usually it is necessary to solve 3 to 10 coupled equations per finite difference cell. These equations represent the conservation of the various oil, gas and water components, and energy. The equations are usually discretized using a fully implicit time step scheme with nearest neighbor coupling in space. The resulting set of nonlinear equations is solved using Newton iteration [10], [9], [15]. Since the solution of the Jacobian matrix may represent 90% or more of the total computing cost for a large thermal (steam or in situ combustion) simulation problem, it is clear that effective iterative methods are required. Our methods are designed principally for fully implicit systems. However, these techniques are also applicable to the linear systems arising from an IMPES approximation [3]. The IMPES method uses explicit saturations with only the pressure taken implicitly. This gives rise to only one equation per finite difference cell.

The Jacobian matrix arising from thermal problems is block-banded and strongly nonsymmetric. The block-bands represent the nearest neighbor connections arising from finite difference approximations, with five bands in two dimensions, and seven in three dimensions. The Jacobian also tends to be nondiagonally dominant. Since oil reservoirs have complicated geologies, the equation coefficients (and hence the elements of the Jacobian) are strongly anisotropic with large jump discontinuities of several orders of magnitude. Methods such as LSOR [31] either diverge or converge intolerably slowly.

In order to minimize user intervention and costly numerical experimentation, solution methods for reservoir simulators should not be crucially dependent on iteration parameters. Incomplete factorization methods (ILU) with conjugate gradient acceleration [24], [22], [35] for symmetric problems or ORTHOMIN [32], [4] for nonsymmetric problems have few, if any, iteration parameters. Recently, these methods have been widely used for IMPES [35], fully implicit black oil [4] and fully implicit thermal [4], [29] simulation. The objective of this paper is to generalize these techniques to block-banded nonsymmetric systems, and compare the numerical performance of ILU

* Received by the editors October 19, 1982, and in revised form May 31, 1983. This research was supported by the Associate Members of CMG and by the Alberta/Canada Energy Resources Research Fund administered by the Department of Energy and Natural Resources of the Province of Alberta.

† Computer Modelling Group, 3512–33 Street N.W., Calgary, Alberta, Canada T2L 2A6.

methods on some test problems. The test problems consist of model nonsymmetric single-phase examples, as well as some problems generated from thermal simulations.

As noted by Watts [35], ordering can significantly affect the performance of ILU methods. This is essentially because the problems are highly anisotropic. Since the grid is usually aligned with the principal axes of the permeability tensor, it is natural to assume that some form of diagonal ordering will reduce the directional bias. (For engineering purposes it is commonly assumed that the permeability axis has a constant direction.) Consequently, we will investigate various degrees of factorization for natural, diagonal (D2) and alternating diagonal (D4) orderings. These orderings are described in detail in [26]. The natural orderings are an extension of the methods described in [4], while the D2 methods are a generalization of the work described in [35] to the nonsymmetric block-banded case. Tan and Letkeman [29] suggested D4 ordering in the context of an incomplete factorization. The method described here uses a slightly different approach than the one described in reference [29], and requires less work per iteration. All these techniques may be used in conjunction with the COMBINATIVE method [4]. We will also investigate the effect of using the modified factorization (MILU) [16], [17], [20] to account for the error terms in the incomplete factorization.

2. ILU methods. Given a sparse banded matrix $\mathbf{A}$, an incomplete factorization $\mathbf{LDU}$ of $\mathbf{A}$ is defined to be:

$$\mathbf{LDU}=\mathbf{A}+\mathbf{E} \tag{1}$$

where $\mathbf{E}$ is the error matrix, $\mathbf{L}$, $\mathbf{D}$ and $\mathbf{U}$ are lower triangular, diagonal and upper triangular matrices respectively. In order to minimize the work per iteration, $\mathbf{L}+\mathbf{D}+\mathbf{U}$ should have a sparse banded structure close to the structure of $\mathbf{A}$. However convergence will be more rapid if $\mathbf{E}$ is made as small as possible. This is generally achieved at the expense of extra bands in $\mathbf{L}$ and $\mathbf{U}$. An incomplete factorization can be viewed as carrying out a few steps of Gaussian elimination on $\mathbf{A}$. If the bands of the original matrix are labelled first degree, then higher degree bands are formed by fill-in resulting from elimination. The degree of a fill band is equal to the degree of the band being eliminated plus the degree of the band inducing it. Our use of degree is equivalent to Watts' concept of order [35]. We use the word degree to avoid confusion with the "first order" factorization of Gustafsson [16], [17]. The concept of degree is explained in greater detail in [35] and [5].

Having decided on a particular degree of factorization, the elements of $\mathbf{L}$, $\mathbf{D}$ and $\mathbf{U}$ can be determined by requiring that $\mathbf{LDU}$ be as close as possible to $\mathbf{A}$. The simplest way to achieve this is as follows: if bands of $\mathbf{LDU}$ coincide with bands of $\mathbf{L}+\mathbf{D}+\mathbf{U}$ then the corresponding elements of $\mathbf{LDU}$ are set equal to the elements of $\mathbf{A}$. There are additional bands in $\mathbf{LDU}$ outside the structure of $\mathbf{L}+\mathbf{D}+\mathbf{U}$ which are nonzero and can be represented by the error matrix $\mathbf{E}$ (1). These error terms will be discussed in a later section. Note that there is another strategy for adding extra bands which amounts to adding bands to $\mathbf{L}+\mathbf{D}+\mathbf{U}$ corresponding to the structure of $\mathbf{A}+\mathbf{E}$ at each stage [16]. However, we will use the "degree" strategy defined above in the following. It will also be assumed that the matrix $\mathbf{A}$ is derived from the usual five-point molecule in two dimensions, or the seven-point molecule in three dimensions. These assumptions can be relaxed if necessary [16], [25].

The structure of each of the incomplete factorizations to be used in this paper is given in Figs. 2, 4, and 6. These figures show the nonzero elements in $\mathbf{L}+\mathbf{D}+\mathbf{U}$ as they appear on the finite-difference grid. Consider first a natural ordering of the grid

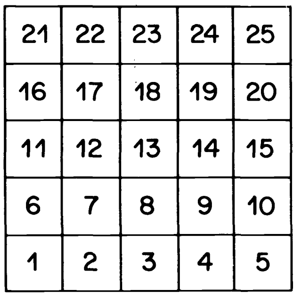


FIG. 1. *Natural ordering in two dimensions.*

points. Natural ordering for a 5×5 two-dimensional grid is shown in Fig. 1. The computational molecule used in naturally ordered factorizations is illustrated in Fig. 2. The point under consideration (corresponding to the diagonal) is labelled D. The points corresponding to bands in $\mathbf{L}$ are labelled by L with a subscript giving the degree of the band, similarly with the bands in $\mathbf{U}$. The plane of each portion of the molecule is also labelled. Restricting attention to the two-dimensional molecule (plane K), it is easily seen that the original five-point operator of nearest neighbor connections is given by the diagonal point (D), and all the points labelled L_1, U_1. This molecule corresponds to a five-banded matrix. The second degree factorization includes all the above elements plus the elements labelled L_2, U_2. These add connections to the $(i-1, j+1)$ and $(i+1, j-1)$ points, and correspond to extra bands in the ILU decomposition adjacent to the outermost bands of the original matrix. Similarly, the third degree factorization consists of all the above elements plus the elements labelled L_3, U_3. These elements correspond to extra bands adjacent to the bands represented by L_2, U_2. In the following, second and third degree naturally ordered factorizations will be investigated. Note that a second degree factorization corresponds to the AB factorization described in [4]. The third degree factorization is the next more accurate factorization according to the strategy outlined above.

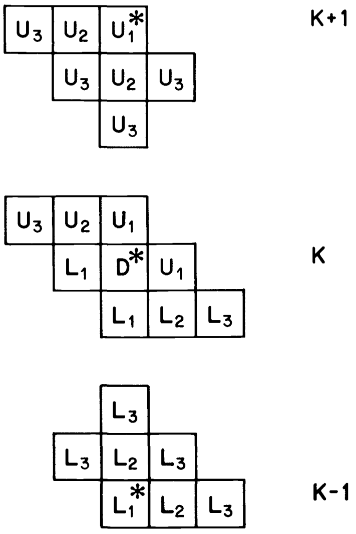


FIG. 2. *Computational molecule for* ILU *natural ordering. The asterisk indicates points directly above or below the diagonal point.*

11	16	20	23	25
7	12	17	21	24
4	8	13	18	22
2	5	9	14	19
1	3	6	10	15

FIG. 3. D2 *ordering in two dimensions.*

Diagonal or D2 ordering of a two-dimensional grid is shown in Fig. 3. The extension to three dimensions is described in [35], [26]. For D2 ordering, we will consider third and fourth degree factorizations in three dimensions, and second through fifth degree factorizations in two dimensions. These factorizations are also described by Watts [35]. We have made the obvious generalization to multi-phase nonsymmetric systems. The molecule for the D2 factorizations is illustrated in Fig. 4.

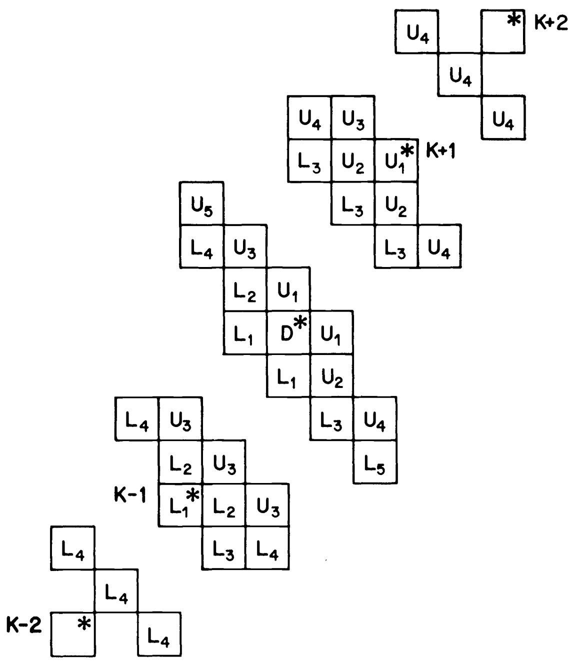
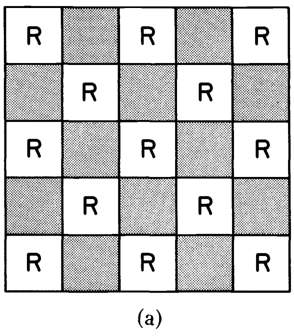


FIG. 4. *Computational molecule for* ILU *with* D2 *ordering. The asterisk indicates points directly above or below the diagonal point.*

For alternating diagonal or D4 ordering, we proceed in a slightly different manner from that in [29]. Firstly, the unknowns are red-black ordered so that $\mathbf{A}x = b$ can be written [36] (see Fig. 5a):

$$\begin{bmatrix} \mathbf{D}_R & \mathbf{A}_R \\ \mathbf{A}_B & \mathbf{D}_B \end{bmatrix} \begin{bmatrix} x_R \\ x_B \end{bmatrix} = \begin{bmatrix} b_R \\ b_B \end{bmatrix} \tag{2}$$

where $\mathbf{D}_R$, $\mathbf{D}_B$ are block diagonal matrices, and $\mathbf{A}_R$ and $\mathbf{A}_B$ are block-banded. For



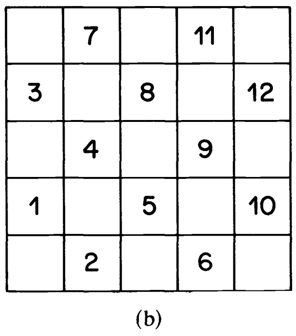


FIG. 5. D4 *ordering in two dimensions.* (a) *red-black,* (b) *diagonal ordering of black points.*

convenience we unitize $\mathbf{D}_R$ so that (2) is equivalent to

$$\begin{bmatrix} \mathbf{I} & \mathbf{A}'_R \\ \mathbf{A}_B & \mathbf{D}_B \end{bmatrix} \begin{bmatrix} x_R \\ x_B \end{bmatrix} = \begin{bmatrix} b'_R \\ b_B \end{bmatrix}. \tag{3}$$

This system is equivalent to:

$$\begin{bmatrix} \mathbf{I} & \mathbf{A}'_R \\ \mathbf{O} & \mathbf{D}_B - \mathbf{A}_B\mathbf{A}'_R \end{bmatrix} \begin{bmatrix} x_R \\ x_B \end{bmatrix} = \begin{bmatrix} b'_R \\ b_B - \mathbf{A}_B b'_R \end{bmatrix}. \tag{4}$$

If we let

$$\begin{aligned} \mathbf{R} &= \mathbf{D}_B - \mathbf{A}_B\mathbf{A}'_R, \\ C &= b_B - \mathbf{A}_B b'_R, \end{aligned} \tag{5}$$

then the system:

$$\mathbf{R}x_B = C \tag{6}$$

is completely decoupled. Of course, the above reduction process can only be used with a nearest neighbor discretization on the original grid. The red points have been eliminated and we need only concern ourselves with the solution of (6). The reduced system $\mathbf{R}$ is then diagonally ordered (see Fig. 5b). A more detailed description of D4 ordering is given in [26]. The bands of $\mathbf{R}$ are defined to be first degree. The idea of using an incomplete factorization on red-black ordered systems was suggested by Axelsson and Gustafsson [2]. In the following we will consider first through third degree factorizations of $\mathbf{R}$ in two dimensions, and a first degree factorization of $\mathbf{R}$ in three dimensions. These molecules are illustrated in Fig. 6. The spaces in the molecule correspond to the red points which have been eliminated. Note that in reference (29) the original system is D4 ordered, and the entire matrix $\mathbf{A}$ is factored. This effectively doubles the number of unknowns compared to the technique described above. Note that the three-dimensional first degree molecule (Fig. 6) corresponds to the three-dimensional molecule of "induced terms" in reference [29]. However, the two-dimensional molecule of induced terms in reference [29] does not not correspond to either a second or third degree factorization of $\mathbf{R}$, since two of the third degree bands are missing.

The intermediate step of red-black ordering has some advantages in developing a fast code. Although the original elements of $\mathbf{R}$ are needed in order to compute its incomplete factorization, they are overwritten by the elements of the factorization. The elements of $\mathbf{R}$ need not be stored explicitly if $\mathbf{D}_B$, $\mathbf{A}_B$ and $\mathbf{A}'_R$ are stored. $\mathbf{D}_B$ and

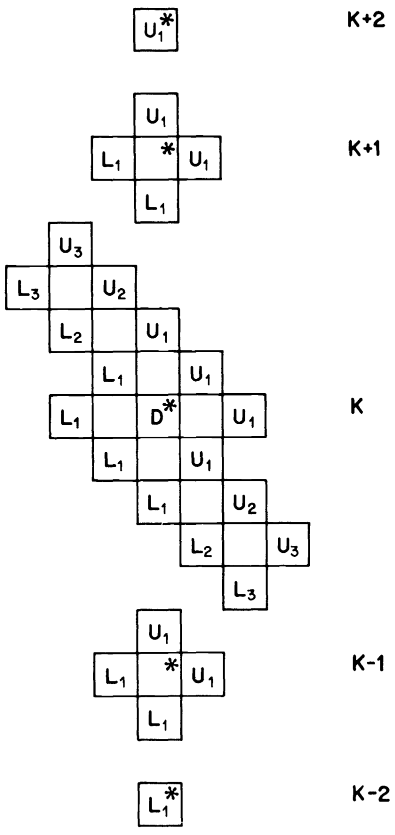


FIG. 6. *Computational molecule for* ILU *with* D4 *ordering. The asterisk indicates points directly above or below the diagonal point.*

$\mathbf{A}_B$ are simply the black portion of the original system $\mathbf{A}$, while $\mathbf{A}'_R$ can be stored in $\mathbf{A}_R$. Acceleration methods such as ORTHOMIN [32], [4] require a matrix-vector multiply at each iteration of the form:

$$y = \mathbf{R}v. \tag{7}$$

In three dimensions the first degree factorization of $\mathbf{R}$ has 19 bands so that a straightforward evaluation of equation (7) would require 19 N_B multiply-adds, where N_B is the number of black points. However, if (7) is evaluated by

$$y = \mathbf{R}v = \mathbf{D}_B v - (\mathbf{A}_B(\mathbf{A}'_R v)), \tag{8}$$

then this costs only 13 N_B multiply-adds. In two dimensions both methods require the same amount of work to evaluate (7). However, the red-black ordering is useful in producing vectorizable code. This will be discussed later.

Recall from (2) that

$$\mathbf{LDU} = \mathbf{A} + \mathbf{E}$$

where the block-bands of the error matrix fall outside the structure of $\mathbf{L}+\mathbf{D}+\mathbf{U}$. The error matrix $\mathbf{E}$ can be taken into account in an approximate way by using the first-order modified factorization (MILU) of Gustafsson [16], [17]. In the following we will use both the ILU and MILU factorizations, except that the arbitrary parameter used by

Gustafsson [16], [17] is not included here. However, this method can occasionally produce undesirable results [13].

3. ORTHOMIN acceleration. The various ILU factorizations discussed above form the basis of an iterative solution method of the system of linear equations:

$$\mathbf{A}x = b. \tag{9}$$

An acceleration procedure is generally added to the ILU factorization. Some acceleration procedures which have been used for nonsymmetric systems are Chebyshev acceleration [23], ORTHODIR AND ORTHORES [37], ORTHOMIN [4], [32] and Lanczos acceleration [18]. The acceleration method used in this paper is ORTHOMIN. It provides a computationally simple, robust acceleration method. It does not require estimation of eigenvalues as does Chebyshev acceleration for example. Some comparisons between ORTHOMIN and the ORTHODIR and ORTHORES algorithms have been made in references [13], [37].

The acceleration is added to the system in (9) which has been preconditioned by the ILU factorization in the following way:

$$[\mathbf{A}(\mathbf{LDU})^{-1}](\mathbf{LDU})x = b. \tag{10}$$

This gives rise to the computational algorithm (for $k = 0, 1, \cdots$):

$$v^{(k)} = (\mathbf{LDU})^{-1} r^{(k)}, \tag{11}$$

$$a_i^{(k)} = \begin{cases} -\dfrac{(\mathbf{A}v^{(k)}, \mathbf{A}q^{(i)})}{(\mathbf{A}q^{(i)}, \mathbf{A}q^{(i)})} & \text{for all } i \text{ in } (m \leqq i \leqq k-1), \\ 0 & \text{otherwise}, \end{cases} \tag{12}$$

$$q^{(k)} = v^{(k)} + \sum_{\substack{i=m \\ m \leqq k-1}}^{k-1} a_i^{(k)} q^{(i)}, \tag{13}$$

$$Aq^{(k)} = \mathbf{A}v^{(k)} + \sum_{\substack{i=m \\ m \leqq k-1}}^{k-1} a_i^{(k)} \mathbf{A}q^{(i)}, \tag{14}$$

$$\omega^{(k)} = \frac{(\mathbf{A}q^{(k)}, r^{(k)})}{(\mathbf{A}q^{(k)}, \mathbf{A}q^{(k)})}, \tag{15}$$

$$x^{(k+1)} = x^{(k)} + \omega^{(k)} q^{(k)}, \tag{16}$$

$$r^{(k+1)} = r^{(k)} - \omega^{(k)} \mathbf{A}q^{(k)}, \tag{17}$$

where

$$m = \text{int}\,(k/(\text{NORTH}+1)) * (\text{NORTH}+1)$$

and int (x) is the largest integer less than or equal to x. Note that this is referred to in [12] as the restarted version of the ORTHOMIN algorithm. The algorithm is restarted every NORTH+1 iterations. Note that the search direction $q^{(k)}$ at each iteration level is constructed to be $\mathbf{A}$ orthogonal to the previous $(k-m)$ search directions. The ORTHOMIN procedure is discussed in more detail in [4]. This reference also discusses guidelines for the choice of NORTH and gives work counts. For low degree ILU factorizations an efficient way to implement ORTHOMIN acceleration is given in reference [11]. These methods were not used in this study since they are not efficient for higher degree factorizations.

4. COMBINATIVE method. Another method of producing an approximate factorization of the sparse block-banded matrix **A** has been discussed in [4]. It works on the assumption that the matrix elements corresponding to certain derivatives (for instance pressure) are more strongly coupled than the others. These equations are decoupled from the others in the following way. Consider a block-row of the Jacobian matrix representing the equations at the (i, j) node for a 2D system with three coupled equations per node. This is illustrated in Fig. 7. It is assumed that we are solving for oil pressure, P_0, water saturation, S_w, and oil saturation, S_0, in each block. The elements marked "D" represent the derivatives with respect to pressure and are strongly coupled to each other. The elements marked "G," in the diagonal block, are eliminated exactly by doing row operations. This of course, changes other elements in the row. The elements marked "ε" are now assumed to be zero, giving a decoupled system for the "D" elements. This system is solved using D4-ordered Gaussian elimination to obtain an estimate of P_0 in each grid block. The estimate is used to remove the remaining derivatives with respect to P_0 to the right-hand side. S_w and S_0 are now approximated using the elements of the diagonal block matrix, which is in lower triangular form. This last step was not done in [4]. It has been included in the present paper since it provides a more accurate approximation of all the variables at the expense of very little extra work.

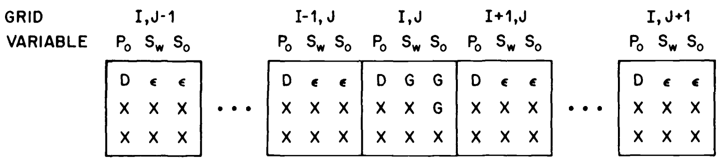


FIG. 7. *Block row of Jacobian matrix for a two-dimensional grid with three coupled equations per node.*

This second type of approximate factorization used in conjunction with an ILU factorization of all the variables and ORTHOMIN acceleration forms the basis of the COMBINATIVE method. The method is characterized by fast convergence. The residual is often reduced by several orders of magnitude at each iteration. This usually results in a smaller residual for a given convergence tolerance than an ILU method alone. The application of the COMBINATIVE method need not be solely to reservoir simulation problems. It could equally well be applied to other types of fluid mechanics problems where pressure is a dominant variable.

The algorithm proceeds in the following way. We call the first approximate factorization (produced by the method described above) $\mathbf{L}_1\mathbf{U}_1$. We call the second approximate factorization (produced by an ILU factorization) $\mathbf{L}_2\mathbf{D}_2\mathbf{U}_2$. Then

(18) $$v^* = (\mathbf{L}_1\mathbf{U}_1)^{-1} r^{(k)}$$

(19) $$r^* = r^{(k)} - \mathbf{A} v^*,$$

(20) $$v^{(k)} = (\mathbf{L}_2\mathbf{D}_2\mathbf{U}_2)^{-1} r^*$$

is followed by ORTHOMIN acceleration as in (12) to (17). Note that the COMBINATIVE method only applies to block-banded systems resulting from fully implicit simulation of multi-phase systems (more than one equation per finite difference cell). ILU factorizations, in contrast, can be used also on the systems of equations arising in single-phase simulation (one equation per finite difference cell). Since the COMBINATIVE method uses Gaussian elimination to solve for the pressure, this method is restricted to problems having a fairly small half-bandwidth. Assuming $nx > ny > nz$ (number of grid blocks in the x, y and z direction respectively), we have found $ny * nz < 50$ on scalar machines or $ny * nz < 100$ on vector machines to be the practical upper limit for the COMBINATIVE method.

5. Programming considerations for vector machines. The methods described above were coded in standard FORTRAN; no machine specific subroutine calls were used. Consequently, this code should be completely portable. The matrix-vector multiply used in ORTHOMIN acceleration can be performed efficiently on a vector machine by multiplying each band individually [21] for naturally ordered factorizations. In the case of D2 ordering, the matrix has curved bands, which are not conducive to efficient matrix-vector multiplication. The unknowns are reordered, and the matrix-vector multiply is carried out using natural ordering. For D4 ordering, the unknowns are reordered into red-black ordering. However, for arbitrary nx, ny (number of grid blocks in the x and y direction respectively) the bands of $\mathbf{R}$ (7) may be curved even for red-black ordering. The bands of $\mathbf{R}$ are straight only if nx is odd in two dimensions, and nx, ny odd in three dimensions. In our code, null blocks are added to the system to ensure this condition. The small amount of extra work required is more than offset by the large gain in speed of vectorized code. On a scalar machine, this extra work can be skipped around. Ensuring that nx, ny are odd also allows the reduction operation (3)–(6) to be fully vectorized. Note that in this case the red blocks are the odd numbered naturally ordered blocks, and the black blocks are the even numbered naturally ordered blocks. The remaining portions of the ORTHOMIN algorithm are trivially vectorizable. All these vector operations are of length equal to the number of unknowns. This section of the code is typically 6–8 times faster in vector mode over scalar mode on the CRAY.

The forward and back substitution phase of the ILU iteration is highly recursive, but some of the operations in each step can be vectorized. Assuming the factors are suitably stored, then all the unknowns in a block can be computed using dot-products. However, it is necessary to gather all the required unknowns into a single vector, and this is not a very efficient operation. The length of these dot-products is $\mathrm{neq} * \mathrm{nbd}$, where neq is the number of unknowns per block, and nbd is the number of block-bands in $\mathbf{L}$ or $\mathbf{U}$. The number of block-bands is typically between 3 and 17 depending on dimension and type of factorization. In spite of the short vectors and the gather operation (which is performed in Fortran), we obtain approximately a 30%–40% reduction in time for the vector over the scalar compilation of the forward and back substitution. Note that for a first degree naturally ordered factorization, the forward and back substitution can be partially vectorized by ordering along diagonals. In reference [30], a 70%–80% reduction in time was reported for the vector mode on the CRAY. However, it is not clear that the same savings will be obtained for block-banded systems. It has also been observed that a first degree naturally ordered factorization sometimes gives very poor results for thermal problems [4].

It might be possible to reduce the recursion in the forward and backward substitution by writing $\mathbf{L}^{-1} = (\mathbf{D}'(\mathbf{I}+\mathbf{L}'))^{-1} \simeq (\mathbf{I}-\mathbf{L}'+\cdots)(\mathbf{D}')^{-1}$ [34]. However, since the $\mathbf{L}^{-1}$ is only approximate, this would affect the convergence of the ILU methods. Our code is also used on scalar machines, so that any change in the algorithm which is slower on a scalar machine is clearly undesirable.

The initial incomplete factorization consists of highly recursive, small $(\mathrm{neq} \times \mathrm{neq})$ matrix–matrix multiplies, and produces slower vector code than scalar code. Consequently, these subroutines are always compiled in scalar mode.

The D4 elimination which is used either by itself or as part of the COMBINATIVE routine can be effectively vectorized, since the inner loop consists of a scalar-vector multiply of length equal to the bandwidth [7].

6. Test problems. There is a lack of standard test problems for strongly nonsymmetric systems. The usual test problems (Stone's problems [28], Kershaw's problem [22]) are symmetric. For symmetric problems of course, the ICCG/MICCG [16], [17], [22], [24], [35] or multi-grid techniques [1], [6] are the most efficient.

Problem 1. Some recent tests have been carried out using the convection diffusion equation [12]:

$$-(P_{xx}+P_{yy})+\beta P_x=0. \tag{21}$$

If (21) is discretized on the unit square with a uniform mesh of size h ($x=ih$, $y=jh$) then (21) becomes

$$4P_{i,j}-P_{i,j-1}-(1+\beta h/2)P_{i-1,j}-(1-\beta h/2)P_{i+1,j}-P_{i,j+1}=0, \tag{22}$$

where central differencing is used on the convective term. The boundary conditions are

$$\begin{aligned} P(x,0)&=0, \qquad P(0,y)=1,\\ P(x,1)&=1, \qquad P_x(1,y)=0. \end{aligned} \tag{23}$$

The derivative boundary condition at $x=1=nh$ is approximated by:

$$(3+\beta h/2)P_{n,j}-P_{n,j-1}-(1+\beta h/2)P_{n-1,j}-P_{n,j+1}=0. \tag{24}$$

Note that for $\beta h/2>1$, (22) loses diagonal dominance and the off-diagonals change sign. This problem was solved on both 33×33 and 65×65 meshes, with $\beta=1000$.

Problem 2. In order to examine the effect of large differences in permeability, we consider the problem

$$\frac{\partial}{\partial x}\left(KX\frac{\partial P}{\partial x}\right)+\frac{\partial}{\partial y}\left(KY\frac{\partial P}{\partial y}\right)-\beta\left(\frac{\partial P}{\partial x}-\frac{\partial P}{\partial y}\right)=-q \tag{25}$$

which is discretized on the unit square with a uniform mesh of size h in the following way:

$$\begin{aligned} &\frac{KX_{i+1/2,j}}{h^2}(P_{i+1,j}-P_{i,j})-\frac{KX_{i-1/2,j}}{h^2}(P_{i,j}-P_{i-1,j})+\frac{KY_{i,j+1/2}}{h^2}(P_{i,j+1}-P_{i,j})\\ &-\frac{KY_{i,j-1/2}}{h^2}(P_{i,j}-P_{i,j-1})-\frac{\beta_{i,j}}{h^2}(P_{i,j}-P_{i-1,j})+\frac{\beta_{i,j}}{h^2}(P_{i,j+1}-P_{i,j})=-\frac{q_{i,j}}{h^2}. \end{aligned} \tag{26}$$

Upstream differencing is used on the convective terms, and $\beta=\beta_{i,j}/h$ so that equation (26) is independent of h^2. The region used is shown in Fig. 8 with:

$$\begin{aligned} &KX=KY=1, \qquad (x,y)\in A,\\ &KX=KY=1000, \qquad (x,y)\in B,\\ &q_1=1, \qquad q_2=.5, \qquad q_3=.6,\\ &q_4=-1.83, \quad q_5=-.27, \quad \beta_{i,j}=10 \end{aligned}$$

where $KX_{i+1/2,j}$, $KY_{i,j+1/2}$ (26) are defined harmonically [3]. The Dirichlet condition $P_{i,j}=1$ is imposed on the boundary of $A\cup B$. Both 33×33 and 65×65 problems were considered.

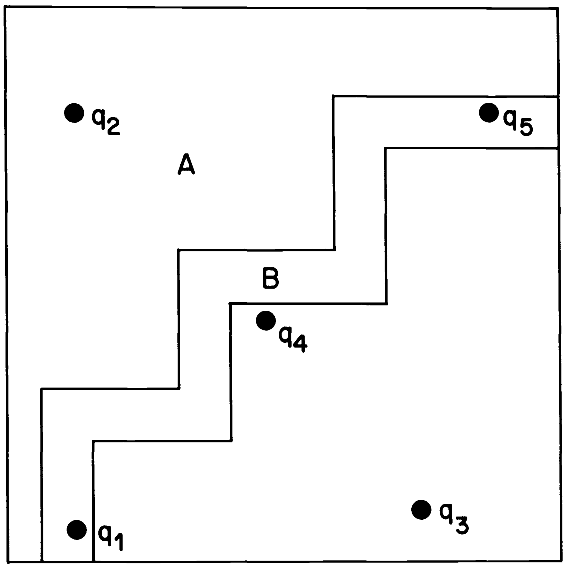


FIG. 8. *Areal geometry for test problem* 2.

Problem 3. A few results will be given for the problem used by Tan and Letkeman [29]. Although this problem is symmetric, it does demonstrate the effect of the modified factorization (MILU), at least for this special case. This problem is Case 1 of reference [29]. This is essentially Poisson's equation on a 10×10 grid with off-diagonals equal to 100 if not zero, and diagonal equal to the negative of the sum of the off-diagonals plus 10^{-3}. A single constant rate injection well of strength $q=10$ is placed in the $i=1$, $j=1$ cell. Neuman boundary conditions are used.

Problem 4. The thermal simulator ISCOM [15] was used to generate a Jacobian matrix from a two-dimensional 31×31 steam injection problem. The matrix was generated from the first Newton iteration of the first time step. With one dead oil

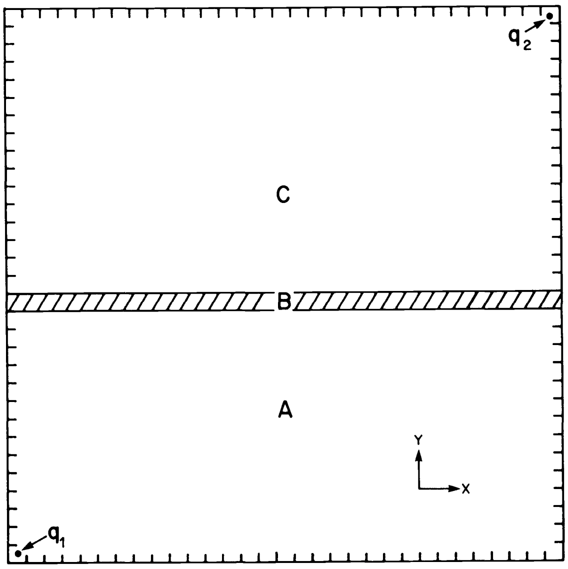


FIG. 9. *Areal geometry for test problem* 4.

component, there were three unknowns per grid block. The uniform 31×31 grid is depicted in Fig. 9. Each block has dimensions $40\times40\times40$ ft, porosity .3 except in region B which has porosity .03. The permeabilities (in darcies) are:

$$\begin{aligned} &KX=.01, && KY=8.0, && (x,y)\in A,\\ &KX=8.0, && KY=.01, && (x,y)\in C,\\ &KX=0, && KY=4\times10^{-4}, && (x,y)\in B. \end{aligned}$$

The oil and water compressibilities are taken to be $2\times10^{-6}\,\text{psi}^{-1}$, with no rock compressibilities. These small compressibilities make this problem quite difficult. The initial pressure everywhere is 130 psi. There is an injection well at q_1, injecting 75% quality steam ($T=650$°F), constant rate 800 bbl/day water equivalent. At q_2 there is a constant pressure production well with bottom hole pressure 14.7 psi. The initial time step was .2 days, while gravity is in the negative y direction.

Problem 5. ISCOM was used to generate a Jacobian matrix from a three-dimensional $11\times11\times8$ steam problem. As before, the number of unknowns per grid block was three, and the matrix was generated from the first iteration, first time step. The x–y cross-section is depicted in Fig. 10, with gravity in the negative y direction.

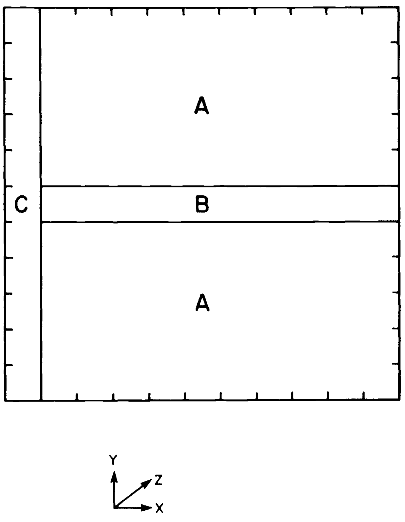


FIG. 10. *Cross-section in x–y plane of the three-dimensional test problem* 5.

Each block has dimensions $40\times40\times40$ ft, porosity .3 except in region B which has porosity .03. The permeabilities (in darcies) are:

$$\begin{aligned} &KX=.01, && KY=.8, && KZ=.01, && (x,y,z)\in A, 1\leqq k\leqq 4,\\ &KX=.8, && KY=.01, && KZ=.8, && (x,y,z)\in A, 5\leqq k\leqq 8,\\ &KX=0, && KY=4\times10^{-4}, && KZ=0, && (x,y,z)\in B, 1\leqq k\leqq 8,\\ &KX=40, && KY=40, && KZ=40, && (x,y,z)\in C, 1\leqq k\leqq 8. \end{aligned}$$

The oil and water compressibilities were 3×10^{-6} psi^{-1}, with no rock compressibilities. The initial pressure was everywhere 130 psi. There were four injection wells at $i=1$, $k=1$, $j=1, 2, 3, 4$, each injecting 75% quality steam ($T=650$°F), 200 bbl/day water equivalent. There was a single constant pressure production well at $i=11, j=11, k=8$, with bottom hole pressure 14.7 psi. The initial time step was .2 days.

7. Results. All the test problems used the convergence test

$$\frac{\|r\|_2}{\|r_0\|_2} < 10^{-6}, \tag{27}$$

where r_0 is the initial residual, r the final residual, and $\|\cdot\|_2$ denotes the l_2 norm. The initial solution was zero everywhere. For test problems 1 to 3, the amount of work required for convergence (27) is given in Tables 1 to 3. The work is given in terms of a work unit (wu), where

$$\text{wu} = \text{number of operations}/N \qquad \text{(multiplications \& divisions)}. \tag{28}$$

This is simply the amount of work/unknown. The work counts include the set-up cost (factorization work), forward and back solve, and ORTHOMIN acceleration. For the D4 ordering, the cost of reduction (2)–(6) and the recovery of the red points (3) is also included.

Generally large values of NORTH (§ 3) reduce the number of iterations required for convergence. However, the cost per iteration increases as NORTH increases. For relatively easy single phase (one equation per cell) problems, we have found a value of NORTH < 5 to be optimal. On the other hand, for practical multi-phase reservoir simulation problems we generally use a value of NORTH in the range 10–15.

Problem 1. The results for problem 1 are given in Table 1. For this problem, NORTH (12) was set equal to 2. Note that the diagonal orderings give a significantly smaller work count than the natural ordering. For some of the methods the modified factorization (MILU) is slightly better than the unmodified factorization, but the difference is not large. Table 1 also gives results for the 65×65 problem for the D4

TABLE 1
Problem 1*
33×33

		ILU	MILU
Natural	2nd degree	240	222
	3rd degree	223	201
D2	2nd degree	137	95
	3rd degree	155	133
	4th degree	100	104
	5th degree	115	120
D4	1st degree	76	77.5
	2nd degree	78.5	81
	3rd degree	96.5	81.5

65×65

		ILU	MILU
D4	1st degree	76	77.5
	2nd degree	78.5	81
	3rd degree	96.5	81.5

* One work unit = N operations.

orderings. Observe that there is no increase in the work/unknown. This is probably because this problem is quite easy, and the D4 method is rapidly convergent.

Elman [12] has solved problem 1 using a first degree naturally ordered incomplete factorization. The emphasis was in investigating the effects of various acceleration methods, such as ORTHOMIN, generalized conjugate residual, and Chebyshev acceleration. The effect of varying the number of previous vectors used in the algorithm was also investigated. Both modified and unmodified factorizations were used on 32×32 and 64×64 grids. Our results should be viewed as comparing different preconditionings with the same acceleration method. We have observed that this problem is sensitive to the choice of acceleration parameters.

Problem 2. The results for problem 2 are given in Table 2. The value of NORTH for these runs was 5. Note that the work counts are generally much higher for this problem than for problem 1, indicating that this problem is more difficult. Except for the higher degree diagonally-ordered factorizations, the MILU methods give better results than the ILU methods. However, for the D2 and D4-ordered factorizations the decrease in work count with increasing degree is more dramatic for the ILU methods than for the MILU methods. A similar trend was noted previously for symmetric problems [5], [6]. As a result the best work count for the 33×33 problem is obtained with the 3rd degree D4 ILU method.

TABLE 2
Problem 2*
33×33

		ILU	MILU
Natural	2nd degree	894	321
	3rd degree	640	300
D2	2nd degree	938	389
	3rd degree	581	353
	4th degree	436	332
	5th degree	295	331
D4	1st degree	409	245
	2nd degree	236	219
	3rd degree	158	214

65×65

		ILU	MILU
D4	1st degree	995	350
	2nd degree	542	302.5
	3rd degree	420	292

* One work unit = N operations.

The results for D4 ordering applied to the 65×65 problem are also given in Table 2. Note the large increase in work/unknown for the ILU methods compared to the MILU methods. This is in spite of the fact that the large differences in permeability make some of the assumptions of the MILU method doubtful [25]. Again, a similar effect has been observed for symmetric problems [5], [6].

The best results for this problem are generally achieved with the higher degree orderings, in particular D4 orderings. This is in contrast to problem 1, where the most effective method was the 1st degree D4 ordering. This merely reflects the fact that higher degree factorizations do not pay off for an easy problem.

Problem 3. Table 3 gives the results for the third degree D4 ordering (NORTH = 5) for problem 3. Note the large difference between the ILU and MILU methods, even for this small 10×10 grid. This difference is even more pronounced for the other orderings. As noted by Tan and Letkeman [29], it is necessary to take the error terms into account for this problem. The results in Table 3 may be compared with those given in reference [29], if we assume that one SIP iteration costs 22 N operations [24]. From [29, Fig. 4], the maximum normalized residual at 7.3 SIP iterations is roughly 2×10^{-7}. This compares with 2×10^{-8} given in Table 3. This improvement is essentially because the D4 method used here operates on the reduced system (6), rather than the full system used in [29]. This effect should be more pronounced for larger problems.

TABLE 3
Problem 3*. *Max normalized residual at convergence* $= 2 \times 10^{-8}$.

		ILU	MILU
D4	3rd degree	272	161.5

* One work unit = N operations.

Problems 4 *and* 5. Tables 4 to 8 give the results in terms of CRAY CPU time for test problems 4 and 5. Recall that these problems are generated from a steam simulation, with 3 equations per grid block. Note that since the factorization work requires matrix-matrix multiplies, the set-up cost will be proportional to neq^3 (neq = number of coupled equations/cell), while the forward and back solve is proportional to neq^2. This effect is of course absent from single-phase problems. The value of NORTH for these runs was NORTH = 10. For comparative purposes we also give the time for Gaussian elimination with D4 ordering [26]. For a three-dimensional problem, D4 ordering requires one-sixth of the work required for natural ordering.

Table 4 gives the results for the two-dimensional problem 4, run in scalar mode on the CRAY. Note the large differences in time for the ILU method, depending on the ordering used. The diagonal orderings seem to be generally better than the natural orderings for this problem which has large permeability contrasts. This confirms the conjectures in references [29], [35]. The best result for the ILU method is with 2nd

TABLE 4
Problem 4. CRAY CPU sec. (OFF = V). D4 Gauss = 4.98.

		ILU	ILU and COMBINATIVE	MILU	MILU and COMBINATIVE
Natural	2nd degree	3.58	1.73	2.75	1.78
	3rd degree	3.90	1.95	2.75	1.83
D2	2nd degree	4.13	1.78	2.48	1.86
	3rd degree	1.94	1.82	1.74	1.94
	4th degree	1.88	2.06	1.86	2.20
	5th degree	1.95	2.04	1.90	2.18
D4	1st degree	2.46	1.45	1.64	1.36
	2nd degree	1.38	1.56	1.36	1.64
	3rd degree	1.39	1.79	1.50	1.72

degree D4 ordering. Next, observe that the ILU+COMBINATIVE method is very effective for the low degree orderings. This indicates that the COMBINATIVE method can resolve problems with large permeability contrasts, even for natural orderings. This is to be expected since the pressure dependence, which is probably most affected by permeability contrasts, is being treated in a fully coupled manner with the COMBINATIVE method. Note also that the COMBINATIVE method reduces the sensitivity to the type of ordering.

The modified factorizations give generally better results than the unmodified. Again we note that for the diagonally-ordered (D2 and D4) factorizations the improvement of the MILU and MILU+COMBINATIVE methods is most pronounced for the lower degree factorizations. The best overall method is a tie between the 2nd degree D4 MILU and the 1st degree D4 MILU+COMBINATIVE. Note that the time for D4 elimination [26] is approximately 3.6 times slower than the best iterative time.

Table 5 gives the results for test problem 4 in vector mode on the CRAY. It is clear that the best results are obtained with the COMBINATIVE methods, indicating that these methods are more vectorizable than the pure ILU/MILU methods. The fastest method overall is the 1st degree D4 MILU+COMBINATIVE. The vector time is 40% less than the scalar time for this method. Note that although the D4 elimination is three times faster in vector mode over scalar, it is still much slower than the best iterative time.

TABLE 5
Problem 4. CRAY CPU sec. (ON = V). D4 Gauss = 1.65.

		ILU	ILU and COMBINATIVE	MILU	MILU and COMBINATIVE
Natural	2nd degree	2.33	1.19	1.87	1.23
	3rd degree	2.50	1.36	1.88	1.33
D2	2nd degree	2.70	1.23	1.73	1.31
	3rd degree	1.43	1.32	1.37	1.44
	4th degree	1.46	1.53	1.51	1.67
	5th degree	1.55	1.59	1.59	1.73
D4	1st degree	1.40	0.82	1.01	0.81
	2nd degree	.95	0.98	.99	1.07
	3rd degree	1.05	1.19	1.16	1.22

Table 6 shows the results for the three-dimensional test problem 5, in scalar mode. The best ILU method by far is the first degree D4. This is because the set-up cost for this method is comparable with the set-up cost for second degree natural, while the

TABLE 6
Problem 5. CRAY CPU sec. (OFF = V). D4 Gauss = 24.5.

		ILU	ILU and COMBINATIVE	MILU	MILU and COMBINATIVE
Natural	2nd degree	9.18	3.07	7.59	3.48
	3rd degree	8.42	4.09	8.43	4.73
D2	3rd degree	7.57	4.21	9.03	4.97
	4th degree	7.69	5.71	10.32	6.72
D4	1st degree	4.12	3.05	5.48	3.58

rate of convergence is much higher. Again, the best results are obtained with the COMBINATIVE methods. The most effective method appears to be first degree D4 ILU+COMBINATIVE, which is just slightly better than second degree natural ILU+COMBINATIVE. With one exception, the modified methods are worse than the unmodified methods. The best iterative method is more than eight times faster than D4 elimination.

Table 7 gives the results for the unmodified methods in vector mode. Observe that the biggest gain in speed (greater than two times) occurs for the D4 methods. This is essentially because part of the set-up work, the reduction process of (2) through (6), can be vectorized. The first degree D4 ILU+COMBINATIVE is now significantly faster than the second degree natural ILU+COMBINATIVE. This method is approximately 3.6 times faster than vectorized D4 elimination.

TABLE 7
Problem 5. CRAY CPU sec. (ON = *V*). D4 Gauss = 5.1.

		ILU	ILU and COMBINATIVE
Natural	2nd degree	5.11	1.76
	3rd degree	4.80	2.64
D2	3rd degree	4.49	2.75
	4th degree	5.10	4.11
D4	1st degree	2.11	1.41

In order to verify that the D4 elimination [26] is being effectively vectorized, a timing was obtained for elimination of an entire row at maximum bandwidth for problem 5. This section of code consists of two loops; the outer loop runs along the bandwidth, while the inner loop eliminates each variable in turn. The inner loop is a scalar-vector multiply, with 3-vector memory references and two floating point operations. Consequently, the loop is memory bound, and can execute at a maximum theoretical rate of 54 megaflops [7], [19]. The observed rate was approximately 40 megaflops, indicating that this code is being reasonably vectorized. Of course, on average the bandwidth for D4 elimination is half the maximum, so this probably accounts for the observed speed-up of only five for the vector D4 elimination.

Intensive assembly language coding can increase the vector performance of Gaussian elimination considerably [8] but assembly language coding will also increase the speed of the iterative methods. Moreover, since a significant portion of the COMBINATIVE method involves Gaussian elimination, any increase in Gaussian elimination speed will increase the speed of COMBINATIVE methods as well.

The ORTHOMIN algorithm as described in § 3 can never diverge, but can become "stuck." In other words the residual does not decrease. This can happen if the symmetric part of the Jacobian matrix is indefinite [13], [37] or if the incomplete factorization becomes ill-conditioned [33]. In practice, the only known property of the Jacobian matrix is that it becomes more diagonally dominant as the time step is reduced [3]. Consequently, a simple solution to poor convergence is to repeat the time step with a smaller time increment. In several hundred simulations with black oil problems [3], we have seen the ORTHOMIN algorithm become stuck only once. For thermal problems (steam, *in situ* combustion) this lack of convergence occurs more frequently. However, we have rerun some of these thermal problems using Gaussian elimination to solve the Jacobian matrix. In almost all cases where the ORTHOMIN algorithm fails, the Newton iteration diverges indicating once again that the time step is too large.

8. Conclusions. For single-phase problems, the high degree D4 ILU methods give the best results. The timings from practical multi-phase simulation problems indicate that methods based on D4 ILU are the preferred methods, on both scalar and vector machines. In particular, D4 ILU+COMBINATIVE can be very effective for highly anisotropic problems satisfying the bandwidth restriction described in § 4.

The effectiveness of the modified factorizations seems to be highly problem dependent. It is difficult to determine when MILU methods would be useful for practical problems.

The largest gain in speed for a vectorized iterative method was a factor of two, versus a factor of five for vectorized D4 elimination. Nevertheless, the iterative methods are still several times faster than D4 elimination.

Acknowledgment. The authors are indebted to Dr. Anthony Au for many useful discussions.

REFERENCES

[1] R. E. Alcouffe, A. Brandt, J. E. Dendy Jr. and J. W. Painter, *The multi-grid method for the diffusion equation with strongly discontinuous coefficients*, this Journal, 2 (1981), pp. 430–454.
[2] O. Axelsson and I. Gustafsson, *On the use of preconditioned conjugate gradient methods for red-black ordered five-point difference schemes*, J. Comp. Phys., 35 (1980), pp. 284–289.
[3] K. Aziz and A. Settari, *Petroleum Reservoir Simulation*, Applied Science Press, London, 1979.
[4] A. Behie and P. K. W. Vinsome, *Block iterative methods for fully implicit reservoir simulation*, Soc. Pet. Eng. J., 22 (1982), pp. 659–668.
[5] A. Behie and P. A. Forsyth, *Comparison of fast iterative methods for symmetric systems*, IMA J. Numer. Anal., 3 (1983), pp. 41–63.
[6] ———, *Multi-grid solution of the pressure equation in reservoir simulation*, paper SPE 10492, Sixth SPE Symposium on Reservoir Simulation, New Orleans, 1982.
[7] D. A. Calahan, *Performance of linear algebra codes on the Cray*-1, paper SPE 7671, Fifth SPE Symposium on Reservoir Simulation, Denver, 1979.
[8] ———, *Vectorized direct solvers for* 2-D *grids*, paper SPE 10522, Sixth SPE Symposium on Reservoir Simulation, New Orleans, 1982.
[9] K. Coats, *In situ combustion model*, paper SPE 8394, SPE 54th Annual Fall Meeting, Las Vegas, 1979.
[10] R. B. Crookston, W. E. Culham and W. H. Chen, *A numerical simulation model for thermal recovery processes*, Soc. Pet. Eng. J., 19 (1979), pp. 37–58.
[11] S. Eisenstat, *Efficient implementation of a class of preconditioned conjugate gradient methods*, this Journal, 2 (1981), pp. 1–4.
[12] H. C. Elman, *Preconditioned conjugate-gradient methods for nonsymmetric systems of linear equations* in Advances in Computer Methods for Partial Differential Equations—IV, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, Rutgers Univ, New Brunswick, NJ, 1981, pp. 409–417.
[13] ———, *Iterative methods for large, sparse nonsymmetric, systems of linear equations*, Ph.D. thesis, Yale Univ. Rep., Yale Univ., New Haven, CT, 129, 1981.
[14] D. G. Feingold and R. S. Varga, *Block diagonally dominant matrices and generalizations of the Gershgorin circle theorem*, Pacific J. Math., 12 (1962), pp. 1241–1250.
[15] J. W. Grabowski, P. K. W. Vinsome, R. C. Lin, A. Behie and B. Rubin, *A fully implicit general purpose finite-difference model for in situ combustion and steam*, paper SPE 8396, SPE 54th Annual Fall Meeting, Las Vegas, 1979.
[16] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.
[17] ———, *On first order factorization methods for the solution of problems with discontinuous material coefficients*, Technical Report, Computer Sciences 77.13 R, Chalmers University of Technology, Goteborg, Sweden, 1977.
[18] L. A. Hageman and D. M. Young, *Applied Iterative Methods*, Academic Press, New York, 1981.
[19] R. W. Hockney and C. R. Jesshope, *Parallel Computers*, Adam Hilger, Bristol, 1981.
[20] J. M. Hyman, *High order incomplete factorizations of sparse matrices*, Los Alamos Report LA-UR-80-3701, Los Alamos National Lab, 1980.

[21] J. I. KARUSH, N. K. MADSEN AND G. H. RODRIGUE, *Matrix multiplication by diagonals on vector/parallel processors*, Lawrence Livermore Report UCID-16899, 1975.

[22] D. S. KERSHAW, *The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations*, J. Comp. Phys., 26 (1978), pp. 43–65.

[23] T. A. MANTEUFFEL, *The Tchebychev iteration for nonsymmetric linear systems*, Numer. Math., 28 (1977), pp. 307–327.

[24] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems in which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[25] ———, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comp. Phys., 44 (1981), pp. 134–155.

[26] H. S. PRICE AND K. COATS, *Direct methods in reservoir simulation*, Trans. SPE of AIME, 257 (1974), pp. 295–308.

[27] P. E. SAYLOR, *Second order strongly implicit symmetric factorization methods for the solution of elliptic difference equations*, SIAM J. Numer. Anal., 11 (1974), pp. 894–908.

[28] H. L. STONE, *Iterative solution of implicit approximations of multidimensional partial differential equations*, SIAM J. Numer. Anal., 5 (1968), pp. 530–558.

[29] T. B. S. TAN AND J. P. LETKEMAN, *Application of* D4 *ordering and minimization in an effective partial matrix inverse iterative method*, paper SPE 10493, Sixth SPE Symposium on Reservoir Simulation, New Orleans, 1982.

[30] B. F. TOWLER AND J. E. KILLOUGH, *Comparison of preconditioners for the conjugate gradient method in reservoir simulation*, paper SPE 10490, Sixth SPE Symposium on Reservoir Simulation, New Orleans, 1982.

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

[32] P. K. W. VINSOME, ORTHOMIN, *An iterative method for solving sparse sets of simultaneous linear equations*, paper SPE 5729, Fourth SPE Symposium on Numerical Simulation of Reservoir Performance, Los Angeles, 1976.

[33] H. A. VAN DER VORST, *Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from* PDE *problems*, J. Comp. Phys., 44 (1981), pp. 1–19.

[34] ———, *A vectorizable variant of some* ICCG *methods*, this Journal, 3 (1982), pp. 350–356.

[35] J. W. WATTS, *A conjugate gradient-truncated direct method for the iterative solution of the reservoir simulation pressure equation*, Soc. Pet. Eng. J., 21 (1981), pp. 345–353.

[36] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

[37] D. M. YOUNG AND K. C. JEA, *Generalized conjugate gradient acceleration of nonsymmetric iterative methods*, Linear Algebra and Appl., 34 (1980), pp. 159–194.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

SPARSE MATRIX METHODS IN OPTIMIZATION*

PHILIP E. GILL†, WALTER MURRAY†, MICHAEL A. SAUNDERS†
AND MARGARET H. WRIGHT†

Abstract. Optimization algorithms typically require the solution of many systems of linear equations $B_k y_k = b_k$. When large numbers of variables or constraints are present, these linear systems could account for much of the total computation time.

Both direct and iterative equation solvers are needed in practice. Unfortunately, most of the off-the-shelf solvers are designed for single systems, whereas optimization problems give rise to hundreds or thousands of systems. To avoid refactorization, or to speed the convergence of an iterative method, it is essential to note that B_k is related to B_{k-1}.

We review various sparse matrices that arise in optimization, and discuss compromises that are currently being made in dealing with them. Since significant advances continue to be made with single-system solvers, we give special attention to methods that allow such solvers to be used repeatedly on a sequence of modified systems (e.g., the product-form update; use of the Schur complement). The speed of factorizing a matrix then becomes relatively less important than the efficiency of subsequent solves with very many right-hand sides.

At the same time, we hope that future improvements to linear-equation software will be oriented more specifically to the case of related matrices B_k.

Key words. large-scale nonlinear optimization, sparse matrices, sparse linear and nonlinear constraints, linear and quadratic programming, updating matrix factorizations

1. Introduction.

1.1. Background.

The major application of sparse matrix techniques in optimization up to the present has been in the implementation of the simplex method for *linear programming* (LP) (see, e.g., Dantzig (1963)). In fact, commercial codes for large LP problems seem to have predated codes for sparse linear equations (even though solving a sparse LP problem requires solving many sparse linear systems). In the commercial world today, more sparse matrix computation is probably expended on linear programs than on any other type of problem, and linear programs involving thousands of unknowns can be solved routinely. Because of the great success of the simplex algorithm and the wide availability of LP codes, many large-scale optimization problems tend to be formulated as purely linear programs. However, we shall see that this limitation is often unnecessary.

Before considering particular methods, we emphasize that methods for large-scale optimization have a special character attributable in large part to the critical importance of linear algebraic procedures. Since dense linear algebraic techniques tend to become unreasonably expensive as the problem dimension increases, it is usually necessary to compromise what seems to be an "ideal" strategy. (In fact, an approach that would not even be considered for small problems may turn out to be the best choice for some large problems.) Furthermore, the relative cost of the steps of many optimization methods changes when the problem becomes large. For example, the performance of

* Received by the editors December 22, 1982, and in final form August 22, 1983. This research was supported by the U.S. Department of Energy under contract DE-AM03-76SF000326, PA no. DE-AT03-76ER72018, the National Science Foundation under grants MCS-7926009 and ECS-8012974; the Office of Naval Research under contract N00014-75-C-0267; and the U.S. Army Research Office under contract DAAG29-81-K-0156.

† Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305.

unconstrained optimization algorithms is often measured by the number of evaluations of the objective function required for convergence. Although simplistic, this is a reasonable gauge of effectiveness for most problems of low dimension because the number of arithmetic operations per iteration tends to be small, and the amount of work required for storage manipulation is negligible. However, as the size of the problem grows, the "housekeeping" (cost of arithmetic and data structures) becomes comparable to, and may even dominate, the cost of function evaluations.

Most optimization methods are iterative; we shall consider algorithms in which the $(k+1)$th iterate is defined as

$$x_{k+1} = x_k + \alpha_k p_k, \tag{1.1}$$

where α_k is a nonnegative scalar, and the n-vector p_k is called the search direction. One of the primary applications of sparse matrix techniques in optimization is in solving one or more systems of linear equations to obtain p_k.

It is usual for thousands of iterations to be required to solve a single large optimization problem, and hence it might appear that the computation time required would be enormous, even with the best available sparse matrix techniques. Fortunately, the linear systems that define p_{k+1} are usually closely related to those that define p_k (and the degree of closeness can be controlled to some extent by the choice of algorithm). In addition, the sequence $\{x_k\}$ will often converge to the solution with only mild conditions on $\{p_k\}$. Consequently, there is a certain flexibility in the definition of p_k. The design of algorithms for large-scale optimization problems involves striking a balance between the effort expended at each iteration to compute p_k and the number of iterations required for convergence.

1.2. Summary. The three main subdivisions of optimization are discussed in turn (unconstrained, linearly constrained, and nonlinearly constrained). A common denominator is the need to solve many systems of linear equations, and the need to *update* various factorizations in order to deal with sequences of related equations. We indicate situations where off-the-shelf software can be applied. Symmetric positive-definite solvers are mainly useful for unconstrained problems, while unsymmetric solvers are essential for dealing with linear constraints. There is an inevitable emphasis on the latter because most large optimization problems currently being solved involve sparse linear constraints.

The principal updating problem is that of replacing one column of a square matrix. However, there exists only one generally available package for updating sparse factors *in situ*. We therefore focus on methods that allow an off-the-shelf solver to be used repeatedly on the same matrix with different right-hand sides. Such methods facilitate more general updates to sparse matrices. In one instance, a sparse indefinite solver is needed.

The final section on nonlinear constraints covers methods that solve a sequence of simpler subproblems, to which the preceding comments apply.

2. Unconstrained optimization.

2.1. Methods for dense problems. The unconstrained optimization problem involves the minimization of a scalar-valued objective function, i.e.

$$\underset{x\in\Re^n}{\text{minimize}}\ F(x).$$

We assume that F is smooth; let $g(x)$ and $H(x)$ denote the gradient vector and Hessian matrix of F.

Many techniques are available for solving unconstrained problems in which n is small (for recent surveys, see, e.g., Brodlie (1977), Fletcher (1980), Gill, Murray and Wright (1981)). The most popular methods compute the search direction as the solution of a system of linear equations of the form

$$H_k p_k = -g_k, \tag{2.1}$$

where g_k is the gradient of F at x_k, and H_k is a suitable symmetric matrix that is most often intended to represent (in some sense) $H(x_k)$. If H_k is positive definite, the solution of (2.1) is the step to the minimum of the local quadratic approximation to F at x_k:

$$\underset{p \in \Re^n}{\text{minimize}}\ g_k^T p + \tfrac{1}{2} p^T H_k p. \tag{2.2}$$

The major distinctions among algorithms involve the definition of H_k.

When H_k is the exact Hessian at x_k or a finite-difference approximation, the algorithm based on solving (2.1) for p_k is called a *Newton-type* method. Newton-type methods tend to be powerful and robust when properly implemented, and exhibit quadratic convergence under mild conditions. However, certain difficulties arise when H_k is indefinite, since the quadratic function (2.2) is unbounded below and the solution of (2.1) may be undefined. Numerous strategies have been suggested for this case, and often involve defining p_k as the solution of a linear system with a positive-definite matrix that is closely related to the Hessian. These techniques include the modified Cholesky factorization of Gill and Murray (1974) and various trust-region strategies (see, e.g., Moré and Sorensen (1982)).

When an exact or finite-difference Hessian is unavailable or too expensive, a popular alternative is to use a *quasi-Newton method* (see Dennis and Moré (1977) for a survey). In a quasi-Newton method, the matrix H_k is an approximation to the Hessian that is *updated* by a low-rank change at each iteration, based on information about the change in the gradient. The hope is that the approximation will improve as the iterations proceed. Quasi-Newton methods typically display a superlinear rate of convergence in practice, and are often more efficient (in terms of computation time) than Newton-type methods.

When n becomes very large, two related difficulties can occur with methods that solve (2.1) directly: excessive computation time and insufficient storage for the $n \times n$ matrix H_k. Fortunately, the Hessian matrices of many large unconstrained problems are quite sparse, and density tends to decrease as n increases. Large problems can thus be solved efficiently using techniques that exploit sparsity in H_k to save work and/or storage, or that do not require storage of H_k.

2.2. Newton-type methods. When the Hessian is sparse and can be computed analytically, a Newton-type method can be implemented by applying standard sparse procedures to solve $H_k p_k = -g_k$. In particular, when H_k is positive definite, any efficient technique for computing a sparse Cholesky factorization may be applied in this context (for a survey of available software, see Duff (1982)). Although many linear systems may need to be solved before the method converges, all of them have the same sparsity pattern, and hence the structure needs to be analyzed only once.

Indefiniteness in a sparse Hessian may be treated using the procedures mentioned for the dense case. The modified Cholesky factorization (Gill and Murray (1974)) has been adapted in a straightforward fashion to treat sparsity (see Thapa (1980)). One advantage of the modified Cholesky approach is that indefiniteness can be detected and corrected while constructing the factorization of the positive-definite matrix to be used in computing p_k; hence, only one sparse factorization needs to be computed at each iteration. With trust-region methods, p_k may be obtained using off-the-shelf software for a sparse Cholesky factorization; however, these methods typically require more than one factorization per iteration.

When the gradient is available, but the exact Hessian is not, a finite-difference approximation to the Hessian may be used as H_k. In the general case, this requires n gradient evaluations. However, if the sparsity pattern of the Hessian is known a priori it is possible to choose special vectors that allow a finite-difference approximation to $H(x)$ to be computed with many fewer than n evaluations of the gradient.

For example, suppose that $H(x)$ is tridiagonal:

$$H(x)=\begin{pmatrix} \times & \times & & & & & & \\ \times & \times & \times & & & & & \\ & \times & \times & \times & & & & \\ & & \times & \times & \times & & & \\ & & & & \cdot\;\cdot\;\cdot & & & \\ & & & & \times & \times & \times & \\ & & & & & \times & \times & \times \\ & & & & & & \times & \times \end{pmatrix}.$$

Consider the vectors

$$y_i=\frac{1}{h\|z_i\|_2}(g(x_k+hz_i)-g(x_k)),\qquad i=1,2,$$

where $z_1=(1,0,1,0,\cdots)^T$, $z_2=(0,1,0,1,\cdots)^T$, and h is an appropriate finite-difference interval. Let $y_{1,i}$ denote the ith component of y_1, and similarly for y_2. The vectors y_1 and y_2 are approximations to the sums of odd and even columns of H_k, respectively. Therefore,

$$y_{1,1}\approx\frac{\partial^2F}{\partial^2x_1},\quad y_{2,1}\approx\frac{\partial^2F}{\partial x_1\partial x_2},\quad y_{1,2}\approx\frac{\partial^2F}{\partial x_1\partial x_2}+\frac{\partial^2F}{\partial x_2\partial x_3},\quad \text{and so on.}$$

Thus, for example,

$$y_{1,2}-y_{2,1}\approx\frac{\partial^2F}{\partial x_2\partial x_3}.$$

In this fashion, all the elements of H_k can be approximated with only two evaluations of the gradient, *regardless of the value of n.*

The idea of analyzing the sparsity pattern of the Hessian in order to determine suitable finite-difference vectors has been the subject of much recent interest. An algorithm for finding finite-difference vectors for a general sparse (*unsymmetric*) matrix is given by Curtis, Powell and Reid (1974), and is based on grouping together columns in which there are no overlapping elements. In the unsymmetric case, the problem of finding a minimum set of vectors can be viewed as a graph coloring problem in the directed graph that represents the sparsity pattern. A proof that finding the minimum set is NP-hard is given in Coleman and Moré (1983), along with practical algorithms (see also Coleman and Moré (1982a)).

A similar relationship with graph coloring can be developed for the case of a symmetric matrix. For example, the requirement of symmetry for a sparse matrix means that the associated column-interaction graph will be *undirected.* The problem of finding a minimum set of finite-difference vectors for a symmetric matrix is NP-complete (a proof for a particular symmetric problem is given in McCormick (1983); see also Coleman and Moré (1982b)). Nonetheless, effective algorithms have been developed based on graph-theoretic heuristics. The algorithms are based on principles similar to those for the unsymmetric case, but are considerably complicated by exploiting symmetry.

A finite-difference Newton-type method for sparse problems thus begins with a procedure that analyzes the sparsity pattern in order to determine suitable finite-difference vectors. Algorithms for finding these vectors have been given by Powell and Toint (1979) and Coleman and Moré (1982b). Once a sparse finite-difference Hessian approximation has been computed, a sparse factorization can be computed as with the exact Hessian.

2.3. Sparse quasi-Newton methods. Because of the great success of quasi-Newton methods on dense problems, it is natural to consider how such methods might be extended to take advantage of sparsity in the Hessian. This extension was suggested first for the case of sparse nonlinear equations by Schubert (1970), and was analyzed by Marwil (1978). Discussions of sparse quasi-Newton methods for optimization and nonlinear equations are given in Toint (1977), Dennis and Schnabel (1979), Toint (1979), Shanno (1980), Steihaug (1980), Thapa (1980), Powell (1981), Dennis and Marwil (1982) and Sorensen (1982). In the remainder of this section we give a brief description of sparse quasi-Newton methods applied to unconstrained optimization.

In quasi-Newton methods for dense problems, the Hessian approximation H_k is *updated* at each iteration by the relationship

$$H_{k+1} = H_k + U_k.$$

The update matrices U_k associated with many dense quasi-Newton methods are of rank two, and can be shown to be the minimum-norm symmetric change in H_k, subject to satisfying the quasi-Newton condition

$$H_{k+1} s_k = y_k, \tag{2.3}$$

where $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$ (see, e.g., Dennis and Moré (1977)). By suitable choice of the steplength α_k in (1.1), the property of hereditary positive-definiteness can also be maintained (i.e., H_{k+1} is positive definite if H_k is). However, the update matrices U_k do not retain the sparsity pattern of the Hessian.

The initial approach to developing sparse quasi-Newton updates was to impose the additional constraint of retaining sparsity on the norm-minimization problem (Powell (1976); Toint (1977)). Let $\mathcal{N}$ be defined as the set of indices $\{(i, j) | H_{ij}(x) = 0\}$, so that $\mathcal{N}$ represents the specified sparsity pattern of the Hessian, and assume that H_k has the same sparsity pattern. A sparse update matrix U_k is then the solution of

$$\begin{aligned} &\underset{U}{\text{minimize}}\ \|U\| \\ &\text{subject to } (H_k + U)s_k = y_k, \\ &\qquad U = U^T, \\ &\qquad U_{ij} = 0 \quad \text{for } (i, j) \in \mathcal{N}. \end{aligned} \tag{2.4}$$

Let $\sigma^{(j)}$ denote the vector s_k with the sparsity pattern of the jth column of H_k imposed. When the norm in (2.4) is the Frobenius norm, the solution is given by

$$U_k = \sum_{j=1}^{n} \lambda_j (e_j \sigma^{(j)T} + \sigma^{(j)} e_j^T), \tag{2.5}$$

where e_j is the jth unit vector and λ is the vector of Lagrange multipliers associated with the subproblem (2.4). The vector λ is the solution of the linear system

$$Q\lambda = y_k - H_k s_k, \tag{2.6}$$

where

$$Q = \sum_{j=1}^{n} (\sigma_j^{(j)} \sigma^{(j)} + \|\sigma^{(j)}\|_2^2 e_j) e_j^T.$$

The matrix Q is symmetric and has the same sparsity pattern as H_k; Q is positive-definite if and only if $\|\sigma^{(j)}\| > 0$ for all j. (The sparse analogue of *any* quasi-Newton formula may be obtained using a similar analysis; see Shanno (1980) and Thapa (1980).)

Thus far, sparse quasi-Newton methods have not enjoyed the great success of their dense counterparts. First, there are certain complications that result from the requirement of sparsity. In particular, note that the update matrix U_k (2.5) is of rank n, rather than of rank two; this means that the new approximate Hessian cannot be obtained by a simple update of the previous approximation. Second, an additional sparse linear system (2.6) must be solved in order to compute the update. Finally, it is not possible in general to achieve the property of hereditary positive-definiteness in the matrices $\{H_k\}$ if the quasi-Newton condition is satisfied (see Toint (1979) and Sorensen (1982)); in fact, positive-definiteness may not be retained even if H_k is taken as the exact (positive definite) Hessian and the initial x_k is very close to the solution (see Thapa (1980)).

In addition to these theoretical difficulties, computational results have tended to indicate that currently available sparse quasi-Newton methods are less effective than alternative methods (in terms of the number of function evaluations required for convergence). However, hope remains that their efficiency may be improved—for example, by relaxing the quasi-Newton condition (2.3), or by finding only an approximate solution of (2.6) (Steihaug (1982)). For a discussion of some possible new approaches, see Sorensen (1982).

2.4. Conjugate-gradient methods. The term *conjugate-gradient* refers to a class of optimization algorithms that generate directions of search without storing a matrix. They are essential in circumstances when methods based on matrix factorization are not viable because the relevant matrix is too large or too dense. We emphasize that there are *two* types of conjugate-gradient methods—linear and nonlinear.

The *linear* conjugate-gradient method was originally derived as an iterative procedure for solving positive-definite symmetric systems of linear equations (Hestenes and Stiefel (1952)). It has been studied and analyzed by many authors (see, e.g., Reid (1971)). When applied to the positive-definite symmetric linear system

$$Hx = -c, \tag{2.7}$$

it computes a sequence of iterates using the relation (1.1). The vector p_k is defined by

$$p_k = -(Hx_k + c) + \beta_{k-1} p_{k-1}, \tag{2.8}$$

and the step length α_k is given by an explicit formula. The matrix H need not be stored explicitly, since it appears only in matrix-vector products.

With exact arithmetic, the linear conjugate-gradient algorithm will compute the solution of (2.7) in at most m ($m \leqq n$) iterations, where m is the number of distinct eigenvalues of H. Therefore, the number of iterations required should be significantly reduced if the original system can be replaced by an equivalent system in which the matrix has clustered eigenvalues. The idea of *preconditioning* is to construct a transformation to have this effect on H. One of the earliest references to preconditioning for linear equations is Axelsson (1974). See Concus, Golub and O'Leary (1976) for details of various preconditioning methods derived from a slightly different viewpoint.

The *nonlinear* conjugate-gradient method is used to minimize a nonlinear function without storage of any matrices, and was first proposed by Fletcher and Reeves (1964). In the Fletcher–Reeves algorithm, p_k is defined as in the linear case by (2.8), where the term $Hx_k + c$ is replaced by g_k, the gradient at x_k. For a nonlinear function, α_k in (1.1) must be computed by an iterative step-length procedure. When the initial vector p_0 is taken as the negative gradient and α_k is the step to the minimum of F along p_k, it can be shown that each p_k is a direction of descent for F.

Many variations and generalizations of the nonlinear conjugate-gradient method have been proposed. The most notable features of these methods are: β_k is computed using different definitions; p_k is defined as a linear combination of *several* previous search directions; p_0 is not always chosen as the negative gradient; and α_k is computed with a relaxed linear search (i.e., α_k is not necessarily a close approximation to the step to the minimum of F along p_k). Furthermore, the idea of preconditioning may be extended to nonlinear problems by allowing a preconditioning matrix that varies from iteration to iteration.

It is well known that rounding errors may cause even the linear conjugate-gradient method to converge very slowly. The nonlinear conjugate-gradient method displays a range of performance that has not yet been adequately explained. On problems in which the Hessian at the solution has clustered eigenvalues, a nonlinear conjugate-gradient method will sometimes converge more quickly than a quasi-Newton method, whereas on other problems the method will break down, i.e. generate search directions that lead to essentially no progress. For recent surveys of conjugate-gradient methods, see Gill and Murray (1979), Fletcher (1980) and Hestenes (1980).

2.5. The truncated linear conjugate-gradient method. Much recent interest has been focussed on an approach to unconstrained optimization in which the equations (2.1) that define the search direction are "solved" (approximately) by performing a limited number of iterations of the *linear* conjugate-gradient method.

Consider the case in which the exact Hessian is used in (2.1). Dembo, Eisenstat and Steihaug (1982) note that the local convergence properties of Newton's method depend on p_k being an accurate solution of (2.1) *only near the solution of the unconstrained problem.* They present a criterion that defines the level of accuracy in p_k necessary to achieve quadratic convergence as the solution is approached, and suggest systematically "truncating" the sequence of linear conjugate-gradient iterates when solving the linear system (2.1) (hence their name of "truncated Newton method"). (See also Dembo and Steihaug (1980) and Steihaug (1980).)

This idea has subsequently been applied in a variety of situations—for example, in computing a search direction from (2.1) when H_k is a sparse quasi-Newton approximation (Steihaug (1982)). We therefore prefer the more specific name of *truncated conjugate-gradient methods.* These methods are useful in computing search directions when it is impractical to store H_k, but it is feasible to compute a *relatively small* number of matrix-vector products involving H_k. For example, this would occur if H_k were the product of several sparse matrices whose product is dense (see § 3.3.1). Truncated conjugate-gradient methods have also been used when the matrix-vector product $H_k v$ is *approximated* (say, by a finite-difference along v); in this case, the computation of p_k requires a number of gradient evaluations equal to the number of linear conjugate-gradient iterations (see, e.g., O'Leary (1982)). In order for these methods to be effective, it must be possible to compute a good solution of (2.1) in a small number of linear conjugate-gradient iterations, and hence the use of preconditioning is important.

With a truncated conjugate-gradient method, complications arise when the matrix H_k is not positive definite, since the linear conjugate-gradient method is likely to break down. Various strategies are possible to ensure that p_k is still a well-defined descent direction even in the indefinite case. For example, the conjugate-gradient iterates may be computed using the Lanczos process (Paige and Saunders (1975)); a Cholesky factorization of the resulting tridiagonal matrix leads to an algorithm that is equivalent to the usual iteration in the positive-definite case. If the tridiagonal matrix is indefinite, a related positive-definite matrix can be obtained using a modified Cholesky factorization. Furthermore, preconditioning can be included, in which case the linear conjugate-gradient iterates begin with the negative gradient transformed by the preconditioning matrix. If the preconditioning matrix is a good approximation to the Hessian, the iterates should converge rapidly. Procedures of this type are described in O'Leary (1982) and Nash (1982).

Further flexibility remains as to how the result of a truncated conjugate-gradient procedure may be used within a method for unconstrained optimization. Rather than simply being used as a search direction, for example, p_k may be combined with previous search directions in a *nonlinear* conjugate-gradient method (see Nash (1982)).

3. Linearly constrained optimization.

3.1. Introduction. The linearly constrained problem will be formulated as

$$\text{LCP} \qquad \begin{aligned} &\underset{x\in\Re^n}{\text{minimize}}\ F(x) \\ &\text{subject to } \mathscr{A}x = b, \\ &\qquad l \leqq x \leqq u, \end{aligned}$$

where the $m \times n$ matrix $\mathscr{A}$ is assumed to be large and sparse. For simplicity, we assume that the rows of $\mathscr{A}$ are linearly independent (if not, some of them may be removed without altering the solution).

The most popular methods for linearly constrained optimization are *active-set* methods, in which a subset of constraints (the *working set*) is used to define the search direction. The working set at x_k usually includes constraints that are satisfied exactly at x_k; the search direction is then computed so that movement along p_k will continue to satisfy the constraints in the working set.

With problem LCP, the working set will include the *general* constraints $\mathscr{A}x = b$ and some of the bounds. When a bound is in the working set, the corresponding variable is *fixed* during that iteration. Thus, the working set induces a partition of x into *fixed* and *free* variables.

We shall not be concerned with details of how the working set is altered, but merely emphasize that the fixed variables at a given iteration are effectively removed from the problem; the corresponding components of the search direction will be zero, and thus the columns of $\mathscr{A}$ corresponding to fixed variables may be ignored. Let A_k denote the submatrix of $\mathscr{A}$ corresponding to the free variables at iteration k; each change in the working set corresponds to a change in the *columns* of A_k. Let n_V denote the number of free variables, and the vector p_k denote the search direction with respect to the free variables only.

By analogy with (2.2) in the unconstrained case, we may choose p_k as the step to the minimum of a quadratic approximation to F, subject to the requirement of remaining on the constraints in the working set. This gives p_k as the solution of the following quadratic program:

$$\begin{aligned} &\underset{p}{\text{minimize}}\ g_k^T p + \tfrac{1}{2} p^T H_k p \\ &\text{subject to } A_k p = 0, \end{aligned} \tag{3.1}$$

where g_k denotes the gradient and H_k the Hessian (or Hessian approximation) at x_k with respect to the free variables.

The solution p_k and Lagrange multiplier λ_k of the problem (3.1) satisfy the $n_V + m$ equations

$$\begin{pmatrix} H_k & A_k^T \\ A_k & \end{pmatrix} \begin{pmatrix} p_k \\ -\lambda_k \end{pmatrix} = \begin{pmatrix} -g_k \\ 0 \end{pmatrix}, \tag{3.2}$$

which will be called the *augmented system.*

One convenient way to represent p_k involves a matrix whose columns form a basis for the null space of A_k. Such a matrix, which will be denoted by Z_k, has $n_V - m$ linearly independent columns and satisfies $A_k Z_k = 0$. The solution of (3.1) may then be computed by solving the *null-space equations*

$$Z_k^T H_k Z_k p_Z = -Z_k^T g_k \tag{3.3}$$

and setting

$$p_k = Z_k p_Z. \tag{3.4}$$

Equations (3.3) and (3.4) define a *null-space* representation of p_k (so named because it explicitly involves Z_k). The vector $Z_k^T g_k$ and the matrix $Z_k^T H_k Z_k$ are called the *projected gradient* and *projected Hessian.*

3.2. Representation of the null space. The issues that arise in representing Z_k when A_k is sparse illustrate the need to compromise strategies that are standard for dense problems. In the rest of this section, we shall drop the subscript k associated with the iteration.

In dense problems, it is customary to use an explicit LQ or some other orthonormal factorization of A in order to define Z. If $AQ = (L\ \ 0)$, where the orthonormal matrix Q is partitioned as $(Y\ Z)$ and L is lower triangular, then $AZ = 0$. In this case, Z has the "ideal" property that its columns are orthonormal, so that formation of the projected Hessian and gradient does not exacerbate the condition of (3.3) and (3.4). Unfortunately, for large problems computation of such a factorization is normally too expensive. (Some current research is concerned with efficient methods for obtaining sparse orthogonal factorizations; see George and Heath (1981). However, the need to update the factors is an even more serious difficulty; see Heath (1982) and George and Ng (1982).)

If an orthogonal factorization is unacceptable, a good alternative is to reduce A to triangular form using Gaussian elimination (i.e., elementary transformations combined with row and column interchanges). This would give an LU factorization in the form

$$P_1AP_2\begin{pmatrix} U & W \\ & I \end{pmatrix} = (L \quad 0), \tag{3.5}$$

where P_1 and P_2 are permutation matrices, U is unit upper triangular, and L is lower triangular. The matrices P_1 and P_2 would be chosen to make U well-conditioned and $\|W\|$ reasonably small. The required matrix

$$Z = P_2\begin{pmatrix} W \\ I \end{pmatrix} \tag{3.6}$$

would no longer have orthonormal columns, but should be quite well conditioned, even if A is poorly conditioned.

Unfortunately, it is not known how to update the factorization (3.5) efficiently in the sparse case when columns of A are altered. However, (3.5) indicates the existence of a square, nonsingular submatrix drawn from the rows and columns of A. We shall assume for simplicity that this matrix comprises the left-most columns of A, i.e.

$$A = (B \quad S), \tag{3.7}$$

where B is nonsingular. (In practice, the columns of B may occur anywhere in A.) It follows from (3.7) and (3.5) (with P_1 and P_2 taken as identity matrices) that $BW + S = 0$, so that $W = -B^{-1}S$. Thus, Z has the form

$$Z = \begin{pmatrix} -B^{-1}S \\ I \end{pmatrix}. \tag{3.8}$$

As long as B in (3.7) is nonsingular, the matrix Z (3.8) will provide a basis for the null space of A. In the absence of the ideal factorization (3.5), the aim must be to choose a B that is as well conditioned as conveniently possible, since this will tend to limit the size of $\|W\|$ and hence the condition of Z.

The partition of the columns of A given by (3.7) induces a partition of the free variables, which will be indicated by the subscripts "B" and "S". The m variables x_B are called the *basic* variables. The remaining s free variables ($s = n_V - m$) are called the *superbasic* variables. For historical reasons, the fixed variables are sometimes called the *nonbasic* variables.

An advantage of the form (3.8) for sparse problems is that operations with Z and Z^T may be performed using a factorization of the matrix B; the matrix Z itself need not be stored. For example, the vector Z^Tg required in (3.3) may be written as

$$Z^Tg = -S^TB^{-T}g_B + g_S. \tag{3.9}$$

(The vector on the right-hand side of (3.9) is called the *reduced gradient*; note that it is simply the projected gradient with a particular form of Z.) Thus, Z^Tg may be obtained by solving $B^Tv = g_B$, and then forming $g_S - S^Tv$. Similarly, to form $p = Zp_Z$, we have

$$p = \begin{pmatrix} -B^{-1}S \\ I \end{pmatrix} p_Z = \begin{pmatrix} -B^{-1}Sp_Z \\ p_Z \end{pmatrix},$$

which gives the system

$$Bp_B = -Sp_Z.$$

With the *reduced-gradient form of* Z (3.8), the problems of representing a null space and computing the associated projections reduce to the familiar operations of factorizing and solving with an appropriate square B.

3.3. Solving for the search direction. At each iteration of an active-set method for LCP, the search direction p with respect to the free variables solves the subproblem (3.1). We have seen that there are mathematically equivalent representations of p; the way in which p is *computed* for sparse problems depends on several considerations, which will be discussed below.

3.3.1. Solving the null-space equations. For sparse problems, it will generally not be possible to solve (3.3) by explicitly forming and then factorizing Z^THZ. Even if H and B are sparse, the projected Hessian will generally be dense. Thus, if a factorization of the projected Hessian is to be stored, the number of superbasic variables at each iteration must be sufficiently small (i.e., the number of fixed variables must be sufficiently large). Fortunately, for many large-scale problems there is an *a priori* upper bound on the number of free variables. For example, if only q of the variables appear nonlinearly in the objective function, the dimension of the projected Hessian matrix at the solution cannot exceed q.

Furthermore, even if the dimension of Z^THZ is small, *forming* the projected Hessian may involve a substantial amount of work; when Z is defined by (3.8), computation of Z^THZ requires the solution of $2s$ systems of size $m \times m$. For this reason, a Newton-type method in which the *projected* Hessian is recomputed at each iteration is not generally practical. By contrast, *quasi-Newton methods* can be adapted very effectively to sparse problems in which the dimension of the projected Hessian remains small, by updating a dense Cholesky factorization of a quasi-Newton approximation to the *projected Hessian*; this is the method used in the MINOS code of Murtagh and Saunders (1977), (1980).

When the projected Hessian cannot be formed or factorized, the null-space equations may be solved using an iterative method that does not require storage of the matrix, such as a truncated conjugate-gradient method (see § 2.5). In order for this approach to be reasonable, the computation of matrix-vector products involving Z and H must be relatively cheap (e.g., when H is sparse); in addition, a good approximation to the solution of (3.3) must be obtained in a small number of iterations. Even when the Hessian is not available, a truncated conjugate-gradient method may be applied to (3.3) by using a finite-difference of the gradient to approximate the vector HZv; an evaluation of the gradient is thus necessary for every iteration of the truncated conjugate-gradient method. Note that this is one of the few methods in which H is not required to be sparse.

Each of the above methods for solving the null-space equations can be adapted to allow for changes in the working set (§ 3.5).

3.3.2. Solving the range-space equations. The null-space equations provide one means of solving for p in the augmented system (3.2), by eliminating λ_k. When H is positive definite, a complementary approach is to solve for λ first, via the *range-space equations*

$$AH^{-1}A^T\lambda = AH^{-1}g, \qquad Hp = A^T\lambda - g.$$

This method would be appropriate if H were sparse, and if A had relatively few rows. The application of a range-space approach to quadratic programming is discussed by Gill et al. (1982).

3.3.3. Solving the augmented system. An alternative method for obtaining p involves treating the augmented system directly. (Variations of this idea have been proposed by numerous authors; see, e.g., Bartels, Golub and Saunders (1970)). The most obvious way to solve (3.2) is to apply a method for symmetric indefinite systems, such as the Harwell code MA27 (Duff and Reid (1982)). In order for the solution of (3.2) to be meaningful, the matrix Z^THZ must be positive definite. Verifying positive-definiteness in this situation is a nontrivial task, since of course the matrix Z^THZ is not computed explicitly. However, the result may sometimes be known a priori—for example, when H itself is positive-definite.

Both H and A change dimension when the working set is altered. Updating procedures for this case are discussed in § 3.6.2.

3.4. Factorizing and solving a square system. The linear systems involving B and B^T are typically solved today using a sparse LU factorization of B. Surveys of techniques for computing such a factorization are given in Duff (1982) and Duff and Reid (1983). The *analyze phase* of a factorization consists of an analysis of the sparsity pattern alone (independent of the values of the elements), and leads to a permutation of the matrix in order to reduce fill-in during the factorization. The *factor phase* consists of computation with the actual numerical elements of the matrix.

We shall mention a few features of certain factorization methods that have particular relevance to optimization (see Duff and Reid (1983) for more details). Since active-set algorithms include a sequence of matrices that undergo column changes, the factorization methods were typically developed to be used in conjunction with an update procedure.

The P^4 algorithm of Hellerman and Rarick (1971), (1972) performs the analyze phase separately from the factor phase, and produces the well-known "bump and spike" structure, in which B is permuted to block lower-triangular form with relatively few "spikes" (columns containing nonzeros above the diagonal). This procedure is very effective if B is nearly triangular. Also, the factor phase is able to use external storage, since it processes B one column at a time. Column interchanges are used to stabilize the factorization. (Row interchanges would destroy the sparsity pattern.) If an interchange is needed at the ith stage, it is necessary to solve a system of the form $L_{i-1}^T y = e_i$ and to compute the quantities y^Ta_j for all remaining eligible spike columns a_j. This involves significant work and also degrades the sparsity of the factors. Thus, a rather loose pivot tolerance must be used to avoid many column interchanges (e.g., $|\mu| \leqq 10^4$, where μ is the largest subdiagonal element in any column of L divided by the corresponding diagonal).

The *Markowitz* algorithm (Markowitz (1957)), on the other hand, performs the analyze and factor phases simultaneously, and hence must run in main memory. It computes dynamic "merit counts" in order to determine the row and column permutations to preserve sparsity and yet retain numerical stability. The Markowitz procedure can achieve a good sparse factorization even with a rather strict pivot tolerance (e.g., $|\mu| \leqq 10$).

In order to indicate how these factor routines perform on matrices that arise in optimization, we give results on five test problems. In the first three problems, the matrix B has "staircase" structure (see, e.g., Fourer (1982)); constraints of this form often arise in the modeling of dynamic systems, in which a set of activities is replicated over several time periods. The fourth and fifth problems arise from the optimal power flow (OPF) problem (see e.g., Stott, Alsac and Marinho (1980)). In this case, B is the Jacobian of the network equations of the power system, and has a *symmetric* sparsity

TABLE 1
Summary of problem characteristics.

	Stair 1	Stair 2	Stair 3	OPF 1	OPF 2
B rows	357	745	1,170	1,200	3,400
B nonzeros	3,500	3,600	7,100	9,000	29,000
P^4 blocks	1	5	13	1	—
P^4 spikes	66	101	157	715	—

pattern (which is not at all triangular!) Table 1 shows some of the relevant features of the problems described, including the results of factorization with the P^4 algorithm.

The number of nonzeros in the initial LU factorization of B is shown in the first two rows of Table 2. The P^4 algorithm is as implemented in the MINOS code of Murtagh and Saunders (1977), (1980); the Markowitz procedure is the Harwell code LA05 (Reid (1976), (1982)). Note that the large number of spikes in the first OPF problem is bound to cause difficulties for the P^4 algorithm.

TABLE 2
Number of nonzeros in initial LU factorization and after k updates.

	Stair 1	Stair 2	Stair 3	OPF 1	OPF 2
L_0U_0 with P^4 (MINOS)	9,400	16,200	32,000	30,400	—
L_0U_0 with Markowitz (LA05)	5,400	4,700	13,500	13,800	75,000
k	50	50	50	30	40
L_kU_k with LA05	7,800	6,000	17,100	15,300	83,000

3.5. Column updates. For problems of the form LCP, each change in the working set involves changing the status of a variable from fixed to free (or vice versa). When a previously fixed variable becomes free, a column of $\mathcal{A}$ is added to A; this poses no particular difficulty, since the new column can simply be appended to S. When a free variable is to become fixed, a column of A must be deleted, and complications arise if the column is in B. Since the number of columns in B must remain constant (in order for B to be nonsingular), it is necessary to *replace* a column of B with one of the columns of S.

Assume that we are given an initial B_0, which thereafter undergoes a sequence of column replacements, each corresponding to one of the free variables becoming fixed on a bound. Let l_k denote the index of the column to be replaced at the kth step, a_k denote the l_kth column of B, v_k denote the new column, and e_{l_k} denote the l_kth column of the identity matrix. After each replacement, we have

$$B_k = B_{k-1} + (v_k - a_k)e_{l_k}^T. \tag{3.10}$$

We shall consider several ways in which systems of equations involving B_k can be solved following a sequence of such changes.

3.5.1. The product-form update. The standard updating technique used in all early sparse LP codes was the product-form (PF) update (e.g., Dantzig and Orchard-Hays (1954)). It follows from the definition of B_k that

$$B_k = B_{k-1} T_k,$$

where

$$B_{k-1} y_k = v_k \quad \text{and} \quad T_k = I + (y_k - e_{l_k}) e_{l_k}^T. \tag{3.11}$$

Note that T_k is a permuted triangular matrix (with only one nontrivial column); equivalently, T_k is a rank-one modification of the identity matrix. The matrix T_k can be represented by storing the index l_k and the vector y_k.

After k such updates we have

$$B_k = B_0 T_1 T_2 \cdots T_k. \tag{3.12}$$

Given a procedure to solve systems of equations involving B_0, (3.12) indicates that solving $B_k v = b$ is equivalent to solving the $k+1$ linear systems

$$B_0 v_0 = b, \quad T_1 v_1 = v_0, \quad \cdots, \quad T_k v_k = v_{k-1}, \tag{3.13}$$

where the systems involving T_j are easy to solve. As k increases, the solution process becomes progressively more protracted, and the storage required to store the updates is strictly increasing. Therefore it becomes worthwhile to compute a factorization of B_k from scratch. Most current systems use an initial triangular factorization $B_0 = L_0 U_0$ (see § 3.4), and recompute the factorization after k updates (typically $k \leqq 50$).

The PF update has two important advantages for sparse problems. First, the vectors $\{y_j\}$ may be stored in a single sequential file, so that implementation is straightforward. Second, any advance in the methods for linear equations is immediately applicable to the factorization of B_0, since the update does not alter the initial factorization. Thus, B_0 may be represented by a "black box" procedure for solving equations (involving both B_0 and B_0^T).

Unfortunately, the PF update has two significant deficiencies. It is numerically unreliable if $|e_{l_k}^T y_k|$ is too small (since T_k is then ill-conditioned), and the growth of data defining the updates is significantly greater than for alternative schemes.

3.5.2. The Bartels–Golub update. The instability of the PF update was first made prominent by Bartels and Golub (1969), who showed as an alternative that an LU factorization can be updated in a stable manner (see also Bartels, Golub and Saunders (1970); Bartels (1971)). Given an initial factorization $B_0 = L_0 U_0$, the updates to L are represented in product form, but the sparse triangular matrix U is stored (and updated) *explicitly*. Thus, instead of the form (3.12) we have

$$B_k = L_0 T_1 T_2 \cdots T_q U_k \equiv L_k U_k, \tag{3.14}$$

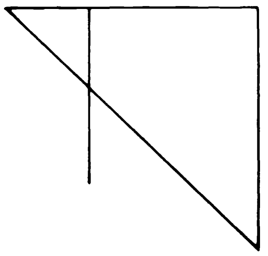
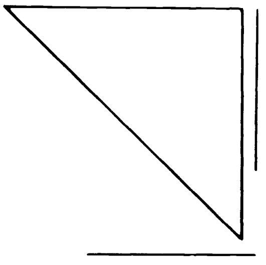
where each T_j represents an update whose construction will be discussed below.

At the kth step, replacing the l_kth column of B_{k-1} gives

$$B_k = L_{k-1} \tilde{U},$$

where $\tilde{U}$ is identical to U_{k-1} except for its l_kth column. Since U_{k-1} is stored as a sparse matrix, it is desirable to restore $\tilde{U}$ to upper-triangular form U_k without causing substantial fill-in. To this end, let P denote a cyclic permutation that moves the l_kth row and column of $\tilde{U}$ to the end, and shifts the intervening rows and columns forward.

We then have

$\tilde{U} =$  , $P^T\tilde{U}P =$  .

The nonzeros in the bottom row of $P^T\tilde{U}P$ may be eliminated by adding multiples of the other rows. However, it follows from the usual error analysis of Gaussian elimination (e.g., Wilkinson (1965)) that this procedure will not be numerically stable unless the size of the multiple is bounded in some way. Hence, we must allow the last row to be *interchanged* with some other row. Formally, the row operations are stabilized elementary transformations (Wilkinson (1965)), which are constructed from 2×2 matrices of the form

$$M=\begin{pmatrix}1 & \\ \mu & 1\end{pmatrix} \quad \text{or} \quad \tilde{M}=\begin{pmatrix} & 1\\ 1 & \mu\end{pmatrix}. \tag{3.15}$$

(Note that the transformation $\tilde{M}$ includes a row interchange.) Each such transformation is represented by the scalar μ, and is unnecessary if the element to be eliminated is already zero. Numerical stability is achieved by choosing between M and $\tilde{M}$ so that the multiplier μ is bounded in size by some moderate number (e.g., $|\mu|\leqq 1$, 10 or 100). The matrices $\{T_j\}$ in (3.14) are constructed from sequences of matrices of the form (3.15).

Unfortunately, elimination of the nonzeros is "easier said than done" in the sparse case. Any transformation of type $\tilde{M}$ amounts to a form of fill-in, since the location of nonzeros in the interchanged rows is unlikely to be the same. A complex data structure is therefore needed to update U_k without losing efficiency during subsequent solves. (Holding individual nonzeros in a linked list, for example, would not be acceptable in a virtual-memory environment.)

The implementation of the BG update by Saunders (1976) capitalizes on the "bump and spike" structure revealed by the P^4 procedure (see § 3.4). Each triangular factor is of the form

$$U_k=\begin{pmatrix} I & E_k\\ & F_k\end{pmatrix},$$

and fill-in can occur only within F_k. If U_0 contains s spikes, the dimension of F_k will be at most $s+k$. Storing F_k as a dense matrix allows the BG update to be implemented with maximum stability ($|\mu|\leqq 1$ in (3.15)), and the approach is efficient as long as s is not unduly large (say, $s\leqq 100$). This implementation has been used for several years in the nonlinear programming system MINOS (Murtagh and Saunders (1977), (1980)). During that period, the number of spikes in U_0 has proved to be favorably small for many sparse optimization models. However, two important applications are now known to give unacceptably large numbers of spikes: time-period models (for which B has a staircase structure) and optimal power-flow problems (for which B has a symmetric sparsity pattern). Some statistics for these problems are given in Table 1 (§ 3.4).

Another implementation of the BG update has been developed by Reid (1976), (1982) as the Fortran package LA05 in the Harwell Subroutine Library. It strikes a compromise between dense and linked-list storage by using a whole row or column of U_k as the "unit" of storage. Thus, the nonzeros in any one row of U_k are held in contiguous locations of memory, as are the corresponding column indices, and an ordered list points to the beginning of each row. To facilitate searching, a similar data structure is used to hold just the sparsity pattern of each column (i.e., the row indices are stored, but not the nonzeros themselves; see Gustavson (1972)). This storage scheme is also suitable for computing an initial LU factorization using the Markowitz criterion and threshold pivoting—a combination that has been eminently successful in practice, particularly on the structures mentioned above. Table 2 (§ 3.4) shows the sparsity of various initial factorizations $B_0 = L_0U_0$ computed by subroutine LA05A, and the moderate rate of growth of nonzeros following k calls to the BG update subroutine LA05C.

Given the row-wise storage scheme for the nonzeros of U_0, it was natural in LA05A for the stability test to be applied *row-wise.* (Thus, each diagonal of U_0 must not be too small compared to other nonzeros in the same row.) This standard threshold pivoting rule is appropriate for single systems, but unfortunately is at odds with the aim of the BG update. The effect is to control the condition of U_0, with no control on the size of the multipliers μ defining L_0.

A preferable alternative is to apply the threshold pivoting test *column-wise*, in order to control the condition of L_0. The resulting L_0, and hence all subsequent factors L_k, will then be a product of stabilized transformations T_j. It follows that *the factors of B_k are likely to be well conditioned if B_k is well conditioned, even if B_0 is not.*

In order to apply the column-wise stability test efficiently, the data structure for computing U_0 needs to be transposed. This and other improvements will be incorporated in a new version of LA05 (Reid, private communication).

At the Systems Optimization Laboratory we have recently implemented some analogous routines as part of a package LUSOL, which will maintain the factorization $L_kB_k = U_k$ following various kinds of updates. The matrices B_k may be singular or rectangular, and the updates possible are column replacement, row replacement, rank-one modification, and addition or deletion of a row or a column. The condition of L_k is controlled throughout for the reasons indicated above. We expect such a package to find many applications within optimization and elsewhere. One example will be to maintain a sparse factorization of the Schur-complement matrix C_k (see §§ 3.5.4–3.6.2), often called the *working basis* in algorithms for solving mathematical programs that have special structure. GUB rows and imbedded networks are examples of such structure; see Brown and Wright (1981) for an excellent overview.

3.5.3. The Forrest–Tomlin update. The update of Forrest and Tomlin (1972) was developed as a means of improving upon the sparsity of the PF update while retaining the ability to use external storage where necessary. In fact the FT update is a restricted form of the BG update, in which no row interchanges are allowed when eliminating the bottom row of $P^T\tilde{U}P$. This single difference removes the fill-in difficulty (but at the expense of losing guaranteed numerical stability).

Algebraically, a new column w_k is added to U_{k-1}, the l_kth column and row are deleted, and the transformations M are combined into a single "row" transformation $R_k = I + e_{l_k}(r_k - e_{l_k})^T$. It can be shown that the required vectors satisfy

$$L_{k-1}w_k = v_k, \quad \text{and} \quad U_{k-1}^T r_k = e_{l_k}, \tag{3.16}$$

and the new diagonal of U_k is $r_k^T w_k$. Most importantly, the multipliers μ are closely related to the elements of r_k, and these can be tested *a posteriori* to determine whether the update is acceptable (see also Tomlin (1975)). In practice a rather undemanding test such as $|\mu| \leqq 10^6$ must be used to avoid rejecting the update too frequently. The FT update is now used within several commercial mathematical programming systems.

3.5.4. Use of the Schur complement. The work of Bisschop and Meeraus (1977), (1980) has recently provided a new perspective on the problem of updating within active-set methods. Suppose that for each update a vector v_j replaces the l_jth column of B_0. A key observation is that the system $B_k x = b$ is equivalent to the system

$$\begin{pmatrix} B_0 & V_k \\ I_k & \end{pmatrix}\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}, \tag{3.17}$$

where

$$V_k = (v_1\ v_2 \cdots v_k), \qquad I_k = (e_{l_1}\ e_{l_2} \cdots e_{l_k})^T.$$

Note that the rectangular matrix I_k is composed of k rows of the identity matrix corresponding to indices of columns that have been replaced. Since the equations $I_k y = 0$ set k elements of y to zero, the remaining elements of y and z together give the required solution x. Similarly, the system $B_k^T y = d$ is equivalent to

$$\begin{pmatrix} B_0^T & I_k^T \\ V_k^T & \end{pmatrix}\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \tag{3.18}$$

if d_1 and d_2 are constructed from d appropriately (with the aid of k arbitrary elements, such as zero).

The matrix in (3.17) may be factorized in several different ways. In the next two sections we consider the simplest factorization

$$\begin{pmatrix} B_0 & V_k \\ I_k & \end{pmatrix} = \begin{pmatrix} B_0 & \\ I_k & C_k \end{pmatrix}\begin{pmatrix} I & Y_k \\ & I \end{pmatrix} \tag{3.19}$$

where

$$B_0 Y_k = V_k, \qquad C_k = -I_k Y_k. \tag{3.20}$$

The $k \times k$ matrix C_k is the *Schur complement* for the partitioned matrix on the left-hand side of (3.19). It corresponds to a matrix of the ubiquitous form $D - WB^{-1}V$ (e.g., see Cottle (1974)).

3.5.5. A stabilized product-form update. From (3.17) and (3.19) we see that the vectors y and z needed to construct the solution of $B_k x = b$ may be obtained from the equations

$$B_0 w = b, \tag{3.21a}$$

$$C_k z = -I_k w, \tag{3.21b}$$

$$y = w - Y_k z. \tag{3.21c}$$

Similarly, the solution of $B_k^T y = d$ is obtained from the two linear systems

$$C_k^T z = d_2 - Y_k^T d_1, \tag{3.22a}$$

$$B_0^T y = d_1 - I_k^T z. \tag{3.22b}$$

Assuming that Y_k is available, the essential operations in (3.21) and (3.22) are a solve with B_0 and a solve with C_k. If k is small enough (say, $k \leqq 100$), C_k may be treated as a dense matrix. It is then straightforward to use an orthogonal factorization $Q_kC_k = R_k(Q_k^TQ_k = I$, R_k upper triangular) or an analogous factorization $L_kC_k = U_k$ based on Gaussian elimination (L_k square, U_k upper triangular). These factorizations can be maintained in a stable manner as C_k is updated to reflect changes to B_k. (The updates involve adding and deleting rows and columns of C_k; see Gill et al. (1974).) The stability of the procedures (3.21) and (3.22) then depends essentially on the condition of B_0. In other words, if B_0 is well conditioned, we have a stable method for solving $B_kx = b$ for many subsequent k.

The method retains several advantages of the PF update. The vectors to be stored (columns of Y_k) satisfy $B_0y_k = v_k$, which is analogous to (3.11). These vectors should have sparsity similar to those in the PF update, and they can be stored sequentially (in compact form on an external file, if necessary). A further advantage is that whenever a column of C_k is deleted, the corresponding vector y_k may be skipped in subsequent uses of (3.21c). This gain would tend to offset the work involved in maintaining the factors of C_k. Because of the parallels, the method described here amounts to a practical mechanism for stabilizing an implementation based on the PF update.

3.5.6. The Schur-complement update. One of the aims of Bisschop and Meeraus (1977), (1980) was to give an update procedure whose storage requirements were independent of the dimension of B_0. This is achievable because the matrix Y_k is not essential for solving (3.17) and (3.18), given V_k and a "black box" for B_0. For example, (3.21c) may be replaced by

$$B_0y = b - V_kz, \tag{3.23}$$

and hence storage for Y_k can be saved at the expense of an additional solve with B_0. Similarly, (3.22a) is equivalent to

$$B_0^Tw = d_1, \qquad C_k^Tz = d_2 - V_k^Tw,$$

again involving a second solve with B_0. Note that the original data V_k will usually be more sparse than Y_k, so that the additional expense may not be substantial.

The storage required for a dense orthogonal factorization of C_k ($\frac{3}{2}k^2$) is small for moderate values of k. As with the PF update, any advance in solving linear equations is immediately applicable to the equations involving B_0.

The method is particularly attractive when B_0 has special structure. For example, certain linear programs have the following form:

$$\begin{aligned}&\text{minimize } c^Tx\\ &\text{subject to } (B_0 \quad N)x = b,\\ &\qquad l \leqq x \leqq u,\end{aligned}$$

where B_0 is a square block-diagonal matrix:

$$B_0 = \text{block-diag}\,(D_0\ D_1 \cdots D_N).$$

Assuming that the square matrices D_j are well conditioned, B_0 provides a natural starting basis for the simplex method.

With the Schur-complement (SC) update, an iteration of the simplex method on such a problem requires four solves with B_0, and hence four solves with each matrix D_j. In certain applications, the matrices D_j are closely related to D_0 (e.g., in time-dependent problems), in which case a further application of the Schur-complement technique would be appropriate. A simplex iteration then involves only solves with D_0.

This is a situation in which one factorization is followed by hundreds or even thousands of solves (involving both D_0 and D_0^T). Thus, it is useful for black-box solvers to be tuned to the case of multiple right-hand sides.

3.5.7. The partitioned *LU* update. Recall that the PF approach accumulates updates in a single file, while the BG and FT methods seek to reduce the storage required for the updates by updating two separate factors (one implicitly through a file of updates, the other explicitly). Here we suggest leaving L_0 and U_0 unaltered (in effect, treating them as two "black boxes" for solving linear systems), and accumulating *two* files of updates. In place of the block factorization (3.19) we can write

$$\begin{pmatrix} B_0 & V_k \\ I_k & \end{pmatrix} = \begin{pmatrix} L_0 & \\ R_k & C_k \end{pmatrix} \begin{pmatrix} U_0 & W_k \\ & I \end{pmatrix} \tag{3.24}$$

with the same definition (3.20) of C_k. After the kth update, the new column of W_k and row of R_k satisfy

$$L_0 w_k = v_k \quad \text{and} \quad U_0^T r_k = e_{l_k}. \tag{3.25}$$

The similarity of (3.25) with the equations (3.16) for the FT update leads us to suppose that the storage requirements would be at least as low as for the FT update. Apart from the need to store and update C_k, all implementation advantages are retained (in fact improved upon, since U_0 is not altered). As with the PF and SC updates, the stability depends primarily on the condition of B_0. We could therefore regard the factorization (3.24) as a practical and stable alternative to the FT update.

3.5.8. Avoiding access to $\boldsymbol{B_0}$. In active-set methods, it is often necessary to solve the equations $B_k x = v$, where v is a column of the matrix $\mathcal{A}$. Although v will not be a column of B_k, it *could* be a column of B_0. If B_0 were not stored in main memory, it would be desirable to access its columns as seldom as possible. In this section we shall show that with the PF update or the Schur-complement updates, the elements of B_0 need not be accessed once the initial factorization has been completed.

Assume that v is the lth column of B_0, so that $v = B_0 e_l$ by definition. For the PF update it follows by substituting the expression for v in (3.13) that

$$T_1 \cdots T_k x = e_l,$$

which gives an equation for x that does not involve v or B_0. With the Schur-complement approach, (3.21a) reduces to $w = e_l$, while (3.23) can be rearranged to give $B_0(y - e_l) = -V_k z$. In either case, when solving for x we can avoid not only an explicit reference to the elements of B_0 but also a solve with B_0.

Similarly, it is often necessary to solve $B_k^T y = d$ and then to form $\gamma_j = y^T v_j$ for each column v_j that has been replaced in B_0. (The quantities γ_j are the reduced costs or reduced gradients for variables that have been removed from B_0.) If t denotes the product $B_0^T y$, then by definition of v_j it follows that $y^T v_j = t^T e_{l_j}$. With both the PF and the Schur-complement updates, t is a by-product of the procedure for computing y. Thus, t and all relevant values γ_j are available at no cost.

These results confirm that B_0 need exist only in the form of a "black box" for solving linear systems.

3.6. Other applications of the Schur-complement update. Historically, the formulation LCP has been used because it involves only column updates to B_k, which have appeared to be the least difficult kind of update to implement for sparse problems. However, the Schur-complement approach also applies to more general sequences of related square systems. As with column replacement, the key is to solve a partitioned system that involves the original matrix.

3.6.1. Unsymmetric rank-one updates. Consider the case in which B_0 undergoes a sequence of rank-one modifications:

$$B_k = B_{k-1} + v_k s_k^T \equiv B_0 + V_k S_k^T.$$

The solution of $B_k x = b$ is part of the solution of the extended system

$$\begin{pmatrix} B_0 & V_k \\ S_k^T & -I \end{pmatrix}\begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \tag{3.26}$$

(Kron (1956), Bisschop and Meeraus (1977)). Given factorizations of B_0 and the Schur complement $C_k \equiv -I - S_k^T B_0^{-1} V_k$, the solution may be obtained from

$$C_k z = -S_k^T w, \qquad B_0 x = b - V_k z,$$

where $B_0 w = b$. An alternative that would require more storage but less work could be obtained by using $B_0 = L_0 U_0$ and storing the vectors defined by $L_0 w_k = v_k$, $U_0^T r_k = s_k$. Let R_k denote the matrix whose jth column is r_j, and similarly for W_k. In this case, the solution of (3.26) would be obtained from

$$C_k z = -R_k^T v \qquad U_0 x = v - W_k z,$$

where $L_0 v = b$. Either approach is an alternative to updating a factorization of B_k itself (e.g., Gille and Loute (1981), (1982)), which is even more difficult to implement than the BG update.

We emphasize that column or row replacements are best treated as a special case, *not* as a sequence of general rank-one modifications.

3.6.2. A symmetric Schur-complement update. It was observed in § 3.1 that in some circumstances the search direction can be computed by solving the linear system (3.2) involving the augmented matrix

$$M_k = \begin{pmatrix} H_k & A_k^T \\ A_k & \end{pmatrix}. \tag{3.27}$$

Within an active-set method, changes in the status of fixed and free variables lead to changes in H and A. When a variable becomes fixed, the corresponding row and column of M_k are *deleted*; when a variable is freed, a new row and column of M_k are *added*.

Instead of updating a factorization of M_k, we can start with some M_0 and work with an augmented system of the form

$$\begin{pmatrix} M_0 & S_k \\ S_k^T & \end{pmatrix}.$$

If a variable is fixed at the kth change, the kth column of S_k is an appropriate coordinate vector; if the lth variable is freed, the column is

$$s_k = \begin{pmatrix} h_l \\ a_l \end{pmatrix},$$

where h_l is obtained from the lth column of the full Hessian, and a_l is the lth column of $\mathscr{A}$. The solution of the augmented system corresponding to the kth working set can then be obtained using a factorization of M_0 and a factorization of the Schur complement $C_k = -S_k^T M_0^{-1} S_k$.

3.7. Linear and quadratic programming. Two important special cases of LCP are linear and quadratic programs. Since there are no user-supplied functions, the computation in linear and quadratic programming methods involves primarily linear algebraic operations.

3.7.1. Large-scale linear programming. Large-scale linear programs occur in many important applications, such as economic planning and resource allocation. Methods and software for large-scale LP have thus achieved a high level of sophistication, and many of the techniques discussed in § 3 were designed originally for use within the simplex method.

Much research has involved linear programs with special structure in the constraint matrix—for example, those arising from networks or time-dependent systems. It is impossible to summarize methods for specially-structured linear programs in a survey paper of this type. However, to illustrate the flavor of the work, we consider staircase linear programs (which were used in the examples of § 3.4). These arise in modeling time-dependent processes; the recent book edited by Dantzig, Dempster and Kallio (1981) is entirely devoted to such problems. It has long been observed that the simplex method tends to be less efficient on staircase problems than on general LPs. To correct this deficiency, work has tended to proceed in two directions. First, the simplex method can be adapted to take advantage of the staircase structure, by using special techniques for factorizing, updating, and pricing (Fourer (1982)). Second, special-purpose methods can be designed to exploit particular features of the problem. For staircase problems, several variations of the *decomposition* approach (Dantzig and Wolfe (1960)) have been suggested. The basic idea is to solve the problem in terms of smaller, nearly independent, subproblems.

3.7.2. Large-scale quadratic programming. A general statement of the quadratic programming problem is

$$\begin{aligned} &\underset{x\in\Re^n}{\text{minimize}}\; c^Tx+\tfrac{1}{2}x^THx \\ &\text{subject to } \mathcal{A}x=b, \\ &\qquad\qquad l\leqq x\leqq u, \end{aligned}$$

where H is a symmetric matrix.

An early approach to quadratic programming was to transform the problem into a linear program, which is then solved by a modified LP method (e.g., Beale (1967)). The most popular quadratic programming algorithms are now based on the active-set approach described in § 3.1 (for a comprehensive survey of QP methods, see Cottle and Djang (1979)), and the search direction is defined by the subproblem (3.1). Efficient methods for *sparse* quadratic programs thus involve specializing the techniques discussed in § 3.3 for the special case when the Hessian is *constant.*

4. Nonlinearly constrained optimization. The nonlinearly constrained optimization problem is assumed to be of the following form:

$$\text{NCP}\qquad \begin{aligned} &\underset{x\in\Re^n}{\text{minimize}}\; F(x) \\ &\text{subject to } c(x)=0, \\ &\qquad\qquad l\leqq x\leqq u, \end{aligned}$$

where $c(x)$ is a vector of m nonlinear constraint functions. We shall assume that these constraints are "sparse", in the sense that the $m \times n$ Jacobian matrix $A(x)$ of $c(x)$ is sparse. For simplicity, we shall usually not distinguish between linear and nonlinear constraints in $c(x)$. However, it is usually considered desirable to treat linear and nonlinear constraints separately.

Problems with nonlinear constraints are considerably more difficult to solve than those with only linear constraints. There is an enormous literature concerning methods for nonlinear constraints; recent overviews are given in Fletcher (1981) and Gill, Murray and Wright (1981). In this section, we shall concentrate on the impact of sparsity rather than attempt a thorough discussion of the methods.

One aspect of NCP that is directly relevant to sparse matrix techniques is that any superlinearly convergent algorithm must consider the curvature of the nonlinear constraint functions, and thus the Hessian of interest is the Hessian of the *Lagrangian function* rather than the Hessian of F alone. Let the Hessian of the Lagrangian function be denoted by $W(x, \lambda) \equiv H(x) - \sum_{i=1}^{m} \lambda_i H_i(x)$, where H_i is the Hessian of c_i. At first, it might appear unlikely that this matrix would be sparse, since it is a weighted sum of the Hessians of the objective function and the constraints. However, sparsity in the gradient of a nonlinear constraint *always* implies sparsity in its Hessian matrix. For example, if the gradient of $c_i(x)$ contains five nonzero components, the corresponding Hessian matrix $H_i(x)$ can have at most 25 nonzero elements. Furthermore, there is often considerable overlap in the positions of nonzero elements in the Hessians of different constraints. Thus, *in practice the Hessian of the Lagrangian function is often very sparse.*

The usual approach to solving NCP is to construct a sequence of unconstrained or linearly constrained subproblems whose solutions converge to that of NCP. Early methods included unconstrained subproblems based on penalty and barrier functions (see Fiacco and McCormick (1968)). Unfortunately, these methods suffer from inevitable ill-conditioning; they have for the most part been superseded by more efficient methods.

4.1. Augmented Lagrangian methods. Augmented Lagrangian methods were motivated in large part by the availability of good methods for unconstrained optimization. The original idea was to minimize an approximation to the Lagrangian function that has been suitably augmented (by a penalty term) so that the solution is a local *unconstrained* minimum of the augmented function (Hestenes (1969), Powell (1969)).

In particular, an augmented Lagrangian method can be defined in which x_{k+1} is taken as the solution of the subproblem

$$\begin{aligned} &\underset{x \in \Re^n}{\text{minimize}}\ L_A(x, \lambda_k, \rho_k) \\ &\text{subject to } l \leqq x \leqq u, \end{aligned} \tag{4.1}$$

where the *augmented Lagrangian function* L_A is defined by

$$L_A(x, \lambda, \rho) \equiv F(x) - \lambda^T c(x) + \frac{\rho}{2} c(x)^T c(x). \tag{4.2}$$

The vector λ_k is an estimate of the Lagrange multiplier vector, and ρ_k is a suitably chosen nonnegative scalar. Alternatively, it is possible to treat any general linear constraints by an active-set method (§ 3.1), and to include only *nonlinear* constraints in the augmented Lagrangian function. Whatever the definition of the subproblem, the algorithm has a *two-level* structure—"outer" iterations (corresponding to different subproblems) and "inner" iterations (within each subproblem).

The Hessian of interest when solving (4.1) is the Hessian of L_A (4.2), which is $W(x, \lambda_k)+\rho_k A(x)^T A(x)$. The sparsity patterns of $W(x, \lambda)$ and the Hessian matrix of L_A are sometimes very similar. Hence, techniques designed to use an explicit sparse Hessian may be applied to (4.1).

The Jacobian matrix $A(x)$ need not be stored explicitly in order to solve the subproblem (4.1). If a fairly accurate solution of (4.1) is computed, an improved Lagrange multipler estimate may be obtained without solving any linear systems involving $A(x)$. However, in several recent augmented Lagrangian methods, (4.1) is solved only to *low accuracy* in order to avoid expending function evaluations when λ_k is a poor estimate of the optimal multipliers; in this case, some factorization of the matrix $A(x_{k+1})$ *is* required to obtain an improved Lagrange multiplier estimate (by solving either a linear system or a linear least-squares problem). The relevance of the storage needed for the Jacobian and/or a factorization depends on the number of nonlinear constraints and the sparsity of the Jacobian.

4.2. Linearly constrained subproblems. The solution of NCP is a minimum of the Lagrangian function in the subspace defined by the gradients of the active constraints. This property leads to a class of methods in which linearizations of the nonlinear constraints are used to define a *linearly constrained subproblem*, of the form

$$\begin{aligned} &\underset{x\in\Re^n}{\text{minimize}}\ F(x)-\lambda_k^T(c(x)-A_k x) \\ &\text{subject to } A_k(x-x_k)=-c_k, \\ &\qquad l\leqq x\leqq u, \end{aligned} \tag{4.3}$$

where c_k and A_k denote $c(x_k)$ and $A(x_k)$ (Robinson (1972), Rosen and Kreuser (1972)). With this formulation, the Lagrange multipliers of the kth subproblem may be taken as the multiplier estimate λ_{k+1} in defining the next subproblem, and will converge to the true multipliers at the solution. When $c(x)$ contains both linear and nonlinear functions, only the nonlinear functions need be included in the objective function of (4.3). Under suitable assumptions, the solutions of the subproblems converge *quadratically* to the solution of NCP. A further benefit of the subproblem (4.3) is that linear constraints may be treated explicitly.

One of the important conditions for convergence with the subproblems (4.3) is a "sufficiently close" starting point; thus, some procedure must be used to prevent divergence from a poor value of x_0. Rosen (1980) suggested a two-phase approach, starting with a penalty function method. In the MINOS/AUGMENTED system of Murtagh and Saunders (1982), the objective function of the subproblem is defined as a *modified augmented Lagrangian* of the form

$$\tilde{L}_A(x, \lambda_k, \rho_k)\equiv F(x)-\lambda_k^T\tilde{c}_k(x)+\frac{\rho_k}{2}\tilde{c}_k(x)^T\tilde{c}_k(x), \tag{4.4}$$

where

$$\tilde{c}_k(x)=c(x)-(c_k+A_k(x-x_k)).$$

Methods based on solving (4.3) have several benefits for sparse problems. The ability to treat linear constraints explicitly is helpful for the many large problems in which most of the constraints are linear. As noted in the Introduction, it is often a feature of sparse problems that the cost of evaluating the problem functions is dominated by the sparse matrix operations. The superiority of SQP methods (§ 4.3.2) for dense problems results from the generally lower number of function evaluations compared to methods based on (4.3); for sparse problems, however, the function evaluations required to solve (4.3) may be insignificant compared to the savings that would result from solving fewer subproblems. If an active-set method of the type described in § 3.3.1 is applied to (4.3), only the *projected Hessian* needs to be stored (rather than the full Hessian). Thus, methods based on (4.3) will tend to be more effective than augmented Lagrangian methods for problems in which the Hessian of the Lagrangian function is not sparse and the projected Hessian can be stored as a dense matrix.

4.3. Methods based on linear and quadratic programming. We now consider two classes of methods in which the subproblems are solved *without evaluation of the problem functions* (in contrast to the methods of §§ 4.1 and 4.2).

4.3.1. Sequential linear programming methods. Because of the availability and high quality of software for sparse linear programs, a popular technique for solving large-scale problems has been to choose each iterate as the solution of an LP subproblem; we shall call these *sequential linear programming* (SLP) methods. They were first proposed by Griffith and Stewart (1961); for a recent survey, see Palacios-Gomez, Lasdon and Engquist (1982).

One crucial issue in an SLP method is the definition of the linear functions in the subproblem. A typical formulation is

$$\begin{aligned} &\underset{x\in\Re^n}{\text{minimize}}\ g_k^T(x-x_k)\\ &\text{subject to } A_k(x-x_k)=-c_k,\\ &\qquad l\leqq x\leqq u.\end{aligned}$$

With some formulations, the LP may not be well posed—for example, there may be fewer constraints than variables. The usual way of ensuring a correctly posed subproblem is to include additional constraints on the variables, such as bounds on the change in each variable. In general, the latter are also needed to ensure convergence.

SLP methods have the advantage that the subproblems can be solved using all the technology of sparse LP codes. They tend to be efficient on two types of problems: those with nearly linear functions, particularly slightly perturbed linear programs; and those in which the functions can be closely approximated by piecewise linear functions (e.g., the objective function is separable and convex). Unfortunately, on general problems SLP methods are at best linearly convergent unless the number of active constraints at the solution is equal to the number of variables. Furthermore, the speed of convergence critically depends on the technique that defines each subproblem.

Recently, some of the techniques used in SQP methods (§ 4.3.2) have been applied to the SLP approach—such as the use of a merit function to ensure progress after each outer iteration. Such techniques cannot be expected to improve the asymptotic rate of convergence of SLP methods, but they should improve robustness and overall effectiveness.

Beale (1978) has given a method that is designed to make extensive use of an existing LP system. The nonlinearly constrained problem is assumed to be of the form

$$\begin{aligned} &\underset{x,y}{\text{minimize}}\ c(x)^Ty\\ &\text{subject to } A(x)y=b(x),\\ &\qquad l\leqq x\leqq u,\\ &\qquad v\leqq y\leqq w.\end{aligned}\tag{4.5}$$

A special nonlinear algorithm is then used to adjust x; for each value of x, a new estimate y is determined by solving an LP.

4.3.2. Sequential quadratic programming methods. The most popular methods in recent years for dense nonlinearly constrained problems are based on solving a sequence of quadratic programming subproblems (see Powell (1982) for a survey). At iteration k, a typical QP subproblem has the form

$$\begin{aligned} &\underset{p\in\Re^n}{\text{minimize}}\ \tfrac{1}{2}p^TH_kp+g_k^Tp \\ &\text{subject to } A_kp=-c_k \\ &\qquad\qquad l-x_k\leqq p\leqq u-x_k, \end{aligned}$$

where H_k is an approximation to the Hessian of the Lagrangian function. The solution of the QP subproblem is then used as the *search direction* p_k in (1.1). The step α_k is chosen to achieve a suitable reduction in some *merit function* that measures progress toward the solution. In the dense case, the most popular method is based on taking H_k as a positive-definite quasi-Newton approximation to the Hessian (Powell (1977)). However, the many options in defining the QP subproblem have yet to be fully understood and resolved (see Murray and Wright (1982), for a discussion of some of the critical issues).

Further complex issues are raised when applying an SQP method to sparse problems (see, e.g., Gill et al. (1981)). The general development of methods has been hampered because methods for sparse quadratic programming are only just being developed, and are not yet generally available for use within a general nonlinear algorithm. However, Escudero (1980) has reported some success with an SQP implementation in which a sparse quasi-Newton approximation is used for H_k (see also § 3.7.2).

REFERENCES

J. Abadie and J. Carpentier (1969), *Generalization of the Wolfe reduced-gradient method to the case of nonlinear constraints*, in Optimization, R. Fletcher, ed., Academic Press, London and New York, pp. 37–49.

O. Axelsson (1974), *On preconditioning and convergence acceleration in sparse matrix problems*, Report 74-10, CERN European Organization for Nuclear Research, Geneva.

R. H. Bartels (1971), *A stabilization of the simplex method*, Numer. Math., 16, pp. 414–434.

R. H. Bartels and G. H. Golub (1969), *The simplex method of linear programming using the LU decomposition*, Comm. ACM, 12, pp. 266–268.

R. H. Bartels, G. H. Golub and M. A. Saunders (1970), *Numerical techniques in mathematical programming*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian and K. Ritter, eds., Academic Press, London and New York, pp. 123–176.

E. M. L. Beale (1967), *An introduction to Beale's method of quadratic programming*, in Nonlinear Programming, J. Abadie, ed., Academic Press, London and New York, pp. 143–153.

——— (1978), *Nonlinear programming using a general mathematical programming system*, in Design and Implementation of Optimization Software, H. J. Greenberg, ed., Sijthoff and Noordhoff, Netherlands, pp. 259–279.

J. Bisschop and A. Meeraus (1977), *Matrix augmentation and partitioning in the updating of the basis inverse*, Math. Prog., 13, pp. 241–254.

——— (1980), *Matrix augmentation and structure preservation in linearly constrained control problems*, Math. Prog., 18, pp. 7–15.

K. W. Brodlie (1977), *Unconstrained optimization*, in The State of the Art in Numerical Analysis, D. Jacobs, ed., Academic Press, London and New York, pp. 229–268.

G. C. Brown and W. G. Wright (1981), *Automatic identification of embedded structure in large-scale optimization models*, in Large-Scale Linear Programming, G. B. Dantzig, M. A. H. Dempster and M. J. Kallio, eds., IIASA, Laxenburg, Austria, pp. 781–808.

T. F. COLEMAN AND J. J. MORÉ (1982a), *Software for estimating sparse Jacobian matrices*, Report ANL-82-37, Argonne National Laboratory, Argonne, IL.

——— (1982b), *Estimation of sparse Hessian matrices and graph coloring problems*, Report 82-535, Dept. Computer Science, Cornell Univ., Ithaca, New York.

——— (1983), *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20, pp. 187–209.

P. CONCUS, G. H. GOLUB AND D. P. O'LEARY (1976), *A generalized conjugate-gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, London and New York, pp. 309–322.

R. W. COTTLE (1974), *Manifestations of the Schur complement*, Linear Algebra Appl., 8, pp. 189–211.

R. W. COTTLE AND A. DJANG (1979), *Algorithmic equivalence in quadratic programming*, J. Optim. Theory Appl., 28, pp. 275–301.

A. R. CURTIS, M. J. D. POWELL AND J. K. REID (1974), *On the estimation of sparse Jacobian matrices*, J. Inst. Maths. Applics., 13, pp. 117–119.

G. B. DANTZIG (1963), *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, NJ.

G. B. DANTZIG, M. A. H. DEMPSTER AND M. J. KALLIO, eds., (1981), *Large-Scale Linear Programming* (*Vol.* 1), IIASA Collaborative Proceedings Series, CP-81-51, IIASA, Laxenburg, Austria.

G. B. DANTZIG AND W. ORCHARD-HAYS (1954), *The product form of the inverse in the simplex method*, Math. Comp., 8, pp. 64–67.

G. B. DANTZIG AND P. WOLFE (1960), *The decomposition principle for linear programs*, Oper. Res., 8, pp. 110–111.

R. S. DEMBO, S. C. EISENSTAT AND T. STEIHAUG (1982), *Inexact Newton methods*, SIAM J. Numer. Anal., 19, pp. 400–408.

R. S. DEMBO AND T. STEIHAUG (1980), *Truncated-Newton algorithms for large-scale unconstrained optimization*, Working Paper #48, School of Organization and Management, Yale Univ., New Haven, CT.

J. E. DENNIS, JR. AND E. S. MARWIL (1982), *Direct secant updates of matrix factorizations*, Math. Comp., 38, pp. 459–474.

J. E. DENNIS, JR. AND J. J. MORÉ (1977), *Quasi-Newton methods, motivation and theory*, SIAM Rev., 19, pp. 46–89.

J. E. DENNIS, JR. AND R. B. SCHNABEL (1979), *Least change secant updates for quasi-Newton methods*, SIAM Rev., 21, pp. 443–469.

I. S. DUFF (1982), *A survey of sparse matrix software*, Report AERE-R.10512, Atomic Energy Research Establishment, Harwell, England; in Sources and Development of Mathematical Software, W. R. Cowell, ed., Prentice-Hall, Englewood Cliffs, NJ, 1984.

I. S. DUFF AND J. K. REID (1982), *The multifrontal solution of indefinite sparse symmetric linear systems*, Report CSS 122, Atomic Energy Research Establishment, Harwell, England; ACM Trans. Math. Software, 9 (1983), pp. 302–305.

——— (1983), *Direct methods for solving sparse systems of linear equations*, presented at the Sparse Matrix Symposium, Fairfield Glade, Tennessee, 1982; I. S. DUFF, this Journal, this issue, pp. 605–619.

L. F. ESCUDERO (1980), *A projected Lagrangian method for nonlinear programming*, Report G320-3407, IBM Scientific Center, Palo Alto, CA.

A. V. FIACCO AND G. P. MCCORMICK (1968), *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley, New York and Toronto.

R. FLETCHER (1974), *Methods related to Lagrangian functions*, in Numerical Methods for Constrained Optimization, P. E. Gill and W. Murray, eds., Academic Press, London and New York, pp. 219–240.

——— (1980), *Practical Methods of Optimization, Volume* 1, *Unconstrained Optimization*, John Wiley, New York and Toronto.

——— (1981), *Practical Methods of Optimization, Volume* 2, *Constrained Optimization*, John Wiley, New York and Toronto.

R. FLETCHER AND C. M. REEVES (1964), *Function minimization by conjugate gradients*, Comput. J., 7, pp. 149–154.

J. J. H. FORREST AND J. A. TOMLIN (1972), *Updating triangular factors of the basis to maintain sparsity in the product form simplex method*, Math. Prog., 2, pp. 263–278.

R. FOURER (1982), *Solving staircase linear programs by the simplex method*, 1: *Inversion*, Math. Prog., 23, pp. 274–313.

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

J. A. GEORGE AND E. NG (1982), *Solution of sparse underdetermined systems of linear equations*, Report CS-82-39, Dept. Computer Science, Univ. Waterloo, Waterloo, Ontario, Canada.

P. E. GILL, G. H. GOLUB, W. MURRAY AND M. A. SAUNDERS (1974), *Methods for modifying matrix factorizations*, Math. Comp., 28, pp. 505–535.

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

——— (1979), *Conjugate-gradient methods for large-scale nonlinear optimization*, Report SOL 79-15, Dept. Oper. Res., Stanford Univ., Stanford, CA.

P. E. GILL, N. I. M. GOULD, W. MURRAY, M. A. SAUNDERS AND M. H. WRIGHT (1982), *Range-space methods for convex quadratic programming*, Report SOL 82-14, Department of Operations Research, Stanford Univ., Stanford, CA.

P. E. GILL, W. MURRAY, M. A. SAUNDERS AND M. H. WRIGHT (1981), *QP-based methods for large-scale nonlinearly constrained optimization*, in Nonlinear Programming 4, O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds., Academic Press, London and New York, pp. 57–98.

P. E. GILL, W. MURRAY AND M. H. WRIGHT (1981), *Practical Optimization*, Academic Press, London and New York.

P. GILLE AND E. LOUTE (1981), *A basis factorization and updating technique for staircase structured systems of linear equations*, Discussion Paper 8113, CORE, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

——— (1982), *Updating the LU Gaussian decomposition for rank-one corrections; application to linear programming basis partitioning techniques*, Cahier No. 8201, Séminaire de Mathématiques Appliquées aux Sciences Humaines, Facultés Universitaires Saint-Louis, Brussels, Belgium.

R. E. GRIFFITH AND R. A. STEWART (1961). *A nonlinear programming technique for the optimization of continuous processing systems*, Management Science, 7, pp. 379–392.

F. G. GUSTAVSON (1972), *Some basic techniques for solving sparse systems of linear equations*, in Sparse Matrices and their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, pp. 41–52.

M. T. HEATH (1982), *Some extensions of an algorithm for sparse linear least squares problems*, this Journal, 3, pp. 223–237.

E. HELLERMAN AND D. RARICK (1971), *Reinversion with the preassigned pivot procedure*, Math. Prog., 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.

M. R. HESTENES (1969), *Multiplier and gradient methods*, J. Optim. Theory Appl., 4, pp. 303–320.

——— (1980), *Conjugate Direction Methods in Optimization*, Springer-Verlag, New York.

M. R. HESTENES AND E. STIEFEL (1952), *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49, pp. 409–436.

G. KRON (1956), *Diakoptics*, MacDonald, London.

H. M. MARKOWITZ (1957), *The elimination form of the inverse and its applications to linear programming*, Management Sci., 3, pp. 255–269.

E. S. MARWIL (1978), *Exploiting sparsity in Newton-type methods*, Ph.D. Thesis, Cornell Univ., Ithaca, New York.

S. T. MCCORMICK (1983), *Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem*, Math. Prog., 26, pp. 153–171.

J. J. MORÉ AND D. C. SORENSEN (1982), *Newton's method*, Report ANL-82-8, Argonne National Laboratory, Argonne, IL.

W. MURRAY AND M. H. WRIGHT (1982), *Computation of the search direction in constrained optimization algorithms*, Math. Prog. Study, 16, pp. 62–83.

B. A. MURTAGH AND M. A. SAUNDERS (1977), MINOS User's Guide, Report SOL 77-9, Dept. Operations Research, Stanford University, Stanford, CA.

——— (1978), *Large-scale linearly constrained optimization*, Math. Prog., 14, pp. 41–72.

——— (1980), MINOS/AUGMENTED *User's Manual*, Report SOL 80-14, Dept. Operations Research, Stanford University, Stanford, CA.

——— (1982), *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Math. Prog. Study, 16, pp. 84–118.

S. G. NASH (1982), *Truncated-Newton methods*, Ph.D. Thesis, Computer Science Department, Stanford Univ., Stanford, CA.

D. P. O'LEARY (1982), *A discrete Newton algorithm for minimizing a function of many variables*, Math. Prog., 23, pp. 20–33.

C. C. PAIGE AND M. A. SAUNDERS (1975), *Solutions of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12, pp. 617–629.

F. PALACIOS-GOMEZ, L. S. LASDON AND M. ENGQUIST (1982), *Nonlinear optimization by successive linear programming*, Management Sci., 28, 10, pp. 1106–1120.

M. J. D. POWELL (1969), *A method for nonlinear constraints in optimization problems*, in Optimization, R. Fletcher, ed., Academic Press, London and New York, pp. 283–297.

M. J. D. POWELL (1976), *A view of unconstrained optimization*, in Optimization in Action, L. C. W. Dixon, ed., Academic Press, London and New York, pp. 117–152.

——— (1981), *A note on quasi-Newton formulae for sparse second derivative matrices*, Math. Prog., 20, pp. 144–151.

——— (1982), *State-of-the-Art Tutorial on "Variable metric methods for constrained optimization"*, Report DAMTP 1982/NA5, Dept. Applied Mathematics and Theoretical Physics, University of Cambridge, England.

M. J. D. POWELL AND P. L. TOINT (1979), *On the estimation of sparse Hessian matrices*, SIAM J. Numer. Anal., 16, pp. 1060–1074.

J. K. REID (1971), *On the method of conjugate gradients for the solution of large sparse systems of linear equations*, in Large Sparse Sets of Linear Equations, J. K. Reid, ed., Academic Press, London and New York, pp. 231–254.

——— (1976), *Fortran subroutines for handling sparse linear programming bases*, Report AERE-R8269, Atomic Energy Research Establishment, Harwell, England.

——— (1982), *A sparsity-exploiting variant of the Bartels–Golub decomposition for linear programming bases*, Math. Prog., 24, pp. 55–69.

S. M. ROBINSON (1972), *A quadratically convergent algorithm for general nonlinear programming problems*, Math. Prog., 3, pp. 145–156.

J. B. ROSEN (1978), *Two-phase algorithm for nonlinear constraint problems*, in Nonlinear Programming 3, O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds., Academic Press, London and New York, pp. 97–124.

J. B. ROSEN AND J. KREUSER (1972). *A gradient projection algorithm for nonlinear constraints*, in Numerical Methods for Non-Linear Optimization, F. A. Lootsma, ed., Academic Press, London and New York, pp. 297–300.

M. A. SAUNDERS (1976), *A fast, stable implementation of the simplex method using Bartels-Golub updating*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, pp. 213–226.

L. K. SCHUBERT (1970), *Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian*, Math. Comp., 24, pp. 27–30.

D. F. SHANNO (1980), *On variable metric methods for sparse Hessians*, Math. Comp., 34, pp. 499–514.

D. C. SORENSEN (1982), *Collinear scaling and sequential estimation in sparse optimization algorithms*, Math. Prog. Study, 18, pp. 135–159.

T. STEIHAUG (1980), *Quasi-Newton methods for large-scale nonlinear problems*, Working Paper #49, School of Organization and Management, Yale Univ., New Haven, CT.

——— (1982), *On the sparse and symmetric least-squares secant update*, Report MASC TR 82-4, Dept. Mathematical Sciences, Rice University, Houston, TX.

B. STOTT, O. ALSAC AND J. L. MARINHO (1980), *The optimal power flow problem*, in Electric Power Problems: The Mathematical Challenge, A. M. Erisman, K. W. Neves and M. H. Dwarakanath, eds., Society for Industrial and Applied Mathematics, Philadelphia, pp. 327–351.

M. N. THAPA (1980), *Optimization of unconstrained functions with sparse Hessian matrices*, Ph.D. Thesis, Stanford Univ., Stanford, California.

P. L. TOINT (1977), *On sparse and symmetric matrix updating subject to a linear equation*, Math. Comp., 31, pp. 954–961.

——— (1978), *Some numerical results using a sparse matrix updating formula in unconstrained optimization*, Math. Comp., 32, pp. 839–851.

——— (1979), *On the superlinear convergence of an algorithm for solving a sparse minimization problem*, SIAM J. Numer. Anal., 16, pp. 1036–1045.

J. A. TOMLIN (1975), *An accuracy test for updating triangular factors*, Math. Prog. Study 4, pp. 142–145.

J. H. WILKINSON (1965), *The Algebraic Eigenvalue Problem*, Oxford Univ. Press, London.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

THE SOFTWARE SCENE IN THE EXTRACTION OF EIGENVALUES FROM SPARSE MATRICES*

B. N. PARLETT†

Abstract. The class of users of programs in the area is discussed and split into the intensive and the sporadic subsets. Available software for each group is reviewed and some current developments are outlined.

Key words. eigenvalues, software, sparse matrices, users

1. Introduction. It is quite possible to discuss software without reference to the applications for which it is needed. For basic supportive calculations, such as extracting roots and evaluating trigonometric functions, the set of users is too diverse to warrant description. Computations with small matrices for normal scientific use are in such a good state that there is really no need to know the application in order to understand the algorithm. At the other extreme it is essential to have some grasp of the problems of weather prediction in order to appreciate the software that has been developed for that daunting task.

Although eigenvalue calculations involving large sparse matrices are not as difficult as solving the coupled nonlinear differential equations governing the weather, they are at the frontier of our expertise. Moreover the cost of a big computation depends on the billing algorithm in the operating system and this can be complicated (not to say arbitrary). Unfortunately the price of generality in programs is high. Consequently it does help to know something of what today's users really want.

For those readers unacquainted with the state of the art in small eigenvalue computations we tell the EISPACK story in § 2. EISPACK is a definitive set of routines for dense matrices. The next two sections deliver the result of our inquiry into the community of users of sparse eigenvalue programs. The sad conclusion is that there are a good number of such programs, they are heavily used, they were often developed outside the numerical analysis community and would not be judged adequate by EISPACK standards. Nevertheless these programs seem to be tuned to their users' needs and perform adequately inside packages addressed to specific ambitious tasks. Nevertheless greater reliability is needed before human beings need not be concerned with the sparse matrix phase of their challenging computations.

Section 5 describes the striking difference between methods for dense matrices and methods for sparse matrices.

Section 6 does describe the contributions of the numerical analysts to the sparse eigenvalue problem. At the time of this writing none of them sits inside a large applications package and so their usage is very light on the national scene. These codes are important standard bearers but should be regarded as the front runners in a race that is not yet over. The massive testing phase which characterized EISPACK has not been applied to the codes mentioned in this section.

Section 7 contrasts subspace iteration with the Lanczos algorithm, § 8 mentions a couple of simple points whose importance has become more widely appreciated

* Received by the editors December 27, 1982, and in final form September 12, 1983. The author gratefully acknowledges support from the Office of Naval Research under contract N00014-76-C-0013. This essay arose from a talk delivered at the Sparse Matrix Symposium held at Fairfield Glade, Tennessee in October, 1982.

† Department of Mathematics, University of California at Berkeley, Berkeley, California 94720.

during the five years since the 1978 sparse matrix meeting. Finally § 9 describes some software that is still under development.

Terminology. It is vital to science that one not be more precise than is necessary for the purpose in hand. In this essay words like small, large and sparse play a key role. Yet their meaning will depend on the user's computing system.

We say that an $n \times n$ matrix is *small* if two $n \times n$ arrays can be held in the fast store, in addition to any programs that are needed; otherwise the matrix is *large.* This binary distinction should not be pressed too hard or it will crumble. The purists say that a matrix is sparse if it has only a few nonzero elements in each row, not more than 50 say. By this definition a matrix with 90% of its elements zero would not be sparse because $(1/10)n^2 > 50n$ for large enough $n(n > 500)$.

Our definition suggests that it is reasonable to use similarity transformations to help find the eigenvalues of small matrices. Our definition also suggests that one should cherish the zero elements of sparse matrices.

The exploitation of sparsity in the execution of triangular factorization has given rise to a valuable and vigorous research area called sparse matrix technology. The reviews of Duff describe it well (Duff (1983)). It turns out that this technology is not directly relevant to eigenvalue problems. There are two classes of methods: those that employ explicit nondiagonal similarity transformations and those that do not. That is all. If it is attractive to use explicit similarity transformations then please use them. Our concern here is with the other class, whatever the character of the matrix.

G.W. Stewart (1976) gave a concise review of numerical methods for sparse eigenvalue problems. That reference covers the background of most of the software we discuss here. Since then there have been important refinements to the Lanczos algorithm. Both Davidson's method and the ideas sketched in the section on recent developments are of more recent vintage.

2. EISPACK. EISPACK is an organized collection of some 40 FORTRAN subroutines which compute some or all eigenvalues of a given matrix with or without the corresponding eigenvectors. This large number of subroutines reflects the yearning for efficiency. EISPACK has special techniques for real matrices, for symmetric matrices, for tridiagonal matrices ($a_{ij} = 0$ if $|i-j| > 1$), and for upper Hessenberg matrices ($a_{ij} = 0$ if $i-j > 1$).

The first issue of EISPACK appeared in 1974. The package was distributed by the code center of the Argonne National Laboratory, Argonne, Illinois, and the indispensable EISPACK guide was published by Springer-Verlag (Smith et al. (1974)). To complete most eigenvalue calculations it is necessary to invoke more than one subroutine. For example, a nonsymmetric matrix B might be first "balanced" by a diagonal similarity transformation, $B \rightarrow DBD^{-1} =: C$; next C is reduced to Hessenberg form, $C \rightarrow P^t CP =: H$ with P orthogonal, and then H is reduced to quasi-triangular form T (to within working accuracy), $H \rightarrow Q^t HQ =: T$ with Q orthogonal. The eigenvalues of B are found along the diagonal of T.

A second edition of EISPACK appeared in 1977 (Garbow et al. (1977)) and a third appeared in 1983. Each edition removed blemishes found in some subroutines and added new programs. We enlarge on this topic further on.

A remarkable aspect of EISPACK is that its first version deliberately eschewed the production of new algorithms. The goal was "simply" to translate into FORTRAN some of the ALGOL programs in the famous *Handbook for Automatic Computation, Volume* II, *Linear Algebra*, J. H. Wilkinson and C. Reinsch (1971). The authors were the acknowledged leaders in the art of matrix computations. The Handbook appeared in 1971 and represented a truly international cooperation. Not all the programs were written by Wilkinson or Reinsch but each contribution was carefully scrutinized by them and, more often than not, improvements were incorporated into the final versions. Moreover most of the programs had already been published in Numerische Mathematik and so had been refereed, and tested, and were available in the public domain.

Why mention these details? Because the exercise of translating these ALGOL programs into FORTRAN was far from trivial and, incredible as it may seem, blemishes were found in a number of the Handbook's programs. Of course, the EISPACK team was aiming high. One aim of the enterprise was to examine the variations in performance when the programs were used on *most* of the operating systems available at that time. For example, there are virtual memory systems in which storage is split up into pages. The system's algorithm for fetching pages, when they are not present, can have a strong effect on the time needed to execute a matrix computation. Fortunately it was possible to express the algorithms so that they would perform adequately on all the major systems.

The next aspect of EISPACK that should be remembered is the massive effort at testing the programs, in an adversary manner, before their release. About 15 computing groups in the USA and Canada agreed to install and test the trial programs they received from Argonne. As bugs and blemishes were uncovered there were several iterations on the programs.

A nasty thought must be stirring in the reader's mind. How can an algorithm be published in Numerische Mathematik after careful refereeing, be subject to scrutiny and testing by Wilkinson and Reinsch prior to inclusion in the Handbook, be translated into FORTRAN by the EISPACK team with close attention to detail, be tested by various sites charged with that duty, and yet retain a bug or even a blemish? Either the people involved in this development are not truly competent or there is more to the production of software for the mass market than meets the eye. If the latter, then what precisely are the difficulties? We all await a succinct description of the intellectual problems that would establish mathematical software as a genuine subdiscipline of computer science.

Last, but not least, we should emphasize the effort expended by the team on the documentation and uniform coding style. It is a mistake to speak of "good" documentation because documentation which satisfies A may not be suitable for B. It *is* reasonable to speak of documentation being suitable for a given class of users. The EISPACK Guide has intimidated some people but, at the time it appeared, it glowed with virtue in comparison with most other samples of documentation. Moreover nothing on this scale had been attempted before by the experts on mathematical software.

Inadvertently EISPACK has provided a common vocabulary (the names of its subroutines) to eigenvalue hunters. EISPACK was a highly visible distillation of what had been learnt by the matrix eigenvalue community during the previous 20 years. It was good for public relations. It predated LINPACK by 3 or 4 years and yet most people consider the eigenvalue problem to be significantly more difficult than the linear equations problem. We will not try to explain this anomaly.

To numerical analysts EISPACK seemed to be the solution to the practical eigenvalue problem. What else remained?

1. EISPACK routines are not fast enough for signal processing applications where the output is needed in microseconds, see Speiser and Whitehead (1982).

2. EISPACK manipulates matrices as conventional two-dimensional arrays or uses a conventional one-dimensional representation of symmetric matrices. Except for the tridiagonal and Hessenberg form, EISPACK does not try to exploit zero elements

in the original matrix. Moreover the stable similarity transformations used by most of the subroutines will quickly destroy any initial sparsity. The fixed in-core data structure makes EISPACK codes nearly independent of the computer and operating system. This crucial property permits a definitive package of FORTRAN programs to be of general use.

The field is not dead by any means. As users become more sophisticated they are finding more and more need for eigenvalues and eigenvectors, usage grows.

As the fast storage capacity of many computer systems continues to grow so does the domain of EISPACK. Sound advice to a casual user is to use EISPACK whenever possible, even if the matrix is 500 by 500 and sparse.

Who will then remain unsatisfied? The next section supplies an answer.

3. Who wants eigenvalues of large, sparse matrices?

EISPACK subroutines have been used quite extensively and it has been assumed that the population of users is so large and so diverse that there is little point in examining the market more closely. Yet conversations with a variety of users over several years have forced the author to the following surprising explanation of the current state of affairs.

The market for eigenvalue programs can be divided into two quite different camps: *intensive* users and *sporadic* users. The intensive users already spend (in total) a few million dollars a year on the extraction of eigenvalues because spectral analysis is essential to their daily work. They need efficiency and have already crafted their programs to exploit the special features of their tasks. General purpose programs with meticulous code designed to cope with any difficulty which might arise are not cost effective for this group. Of course, the more enlightened intensive users will collaborate with experts as their special purpose software evolves to meet more exacting demands. Actually the class of intensive eigenvalue hunters also splits into two quite distinct subclasses: those with small, dense matrices in signal processing and those who generate larger and larger matrices as they make their mathematical models more realistic. The signal processors may be driven to solve their problems with hardware rather than software and we will concentrate here on the large matrix problem.

The *sporadic* user has neither the incentive nor the inclination to study matrix methods. The need for some eigenvalues arises and the user wants to obtain them with minimal fuss. Reliability is more important than efficiency and EISPACK is the answer to his prayers. We suggest that the number of sporadic users whose matrices are too large for EISPACK is currently very low.

The foregoing remarks do not lessen the value of developing good methods for all sorts of large eigenvalue problems. The number of sporadic users will increase.

4. Intensive users.

4.1. Structural engineers.

Most eigenvalue calculations arise at the heart of analyses of structures subject to small vibrations. There is an $n \times n$ global stiffness matrix K and an $n \times n$ global mass matrix M. Both are symmetric and real, K is positive definite. The usual task is to find all the eigenvalues λ_i in a given interval at the left end of the spectrum (i.e., near 0), together with their eigenvectors z_i,

$$(K - \lambda_i M) z_i = 0, \qquad i = 1, 2, \cdots.$$

The z_i determine the shapes of the fundamental modes of vibration of the structure and the λ_i determine the associated natural frequencies $2\pi/\sqrt{\lambda_i}$, $i = 1, 2, \cdots$.

The nonzero elements of K and M are integrals and more arithmetic operations are needed to form K and M than to compute a few eigenvalues.

In 1978 one company paid \$12,000 to obtain 30 eigenvalue/vector pairs for a problem with $n = 12{,}000$. This cost excludes program development. A good finite element package was used with the technique called subspace iteration for the eigenvalue extraction. Professor E. Wilson (Civil Engineering Department, University of California, Berkeley) estimates that structural engineers spent about \$10 million in 1978 on eigenvalue computations. A typical problem today will have $n = 500$, and 20 modes will be computed. Nevertheless, there is continued demand to analyze more complicated structures in greater detail. Problems with $n > 10{,}000$ and 400 modes have been solved. There is current interest in the vibration of piping systems in modern nuclear reactors.

4.2. Quantum chemists. The method of configuration interaction (CI) has become the preferred technique for deducing observable properties of real (or hypothetical) molecules from their detailed electronic structure. The CI method approximates the solution of the clamped nuclei Schrödinger equation by expanding it in terms of orthogonal functions made out of products of single and double electron spin orbitals. More details are given in Shavitt (1977) and Davidson (1983). These papers show that interesting, difficult, special eigenvalue problems are being solved quite independently of the numerical analysis community. Great ingenuity goes into the calculation of the real symmetric matrix H (H for Hamiltonian). The nonzero elements of H are multiple integrals and constitute only 10% of the positions. Unfortunately they are scattered over the matrix in a way that precludes any simple structural template. Each eigenvector represents a wave function and its eigenvalue is the energy of the associated state.

In these chemical computations the determination of H requires 10 to 100 times more work than the extraction of the eigenvector belonging to the smallest (i.e., most negative) eigenvalue. Davidson estimates that \$100,000 is spent per year in the USA on eigenvalue/vector computations in chemistry. Usually only the smallest pair is required. A typical calculation has the order $n = 10{,}000$, but this is expected to increase sharply. During the summer of 1982 the ethylene molecule C_2H_4 was analyzed in collaborative work at Cambridge University and the University of California, Berkeley. For this problem

$$n = 1{,}046{,}758.$$

This calculation demonstrated convincingly the need to include excited states in the expansion. In other words the simpler models in CI are not adequate for the detail required in current investigations.

It is sad that most numerical analysts have never heard of the special eigenvalue methods invented by the chemists. These are described in Davidson (1983).

4.3. Plasma physicists. An interesting nonsymmetric application occurs in resistive magneto-hydrodynamics (MHD) theory that combines Maxwell's and fluid flow equations. The hot plasma is confined by magnetic fields. MHD equilibria and their stability are of special interest. Resistivity ($\eta \neq 0$) is the most important nonideal effect since the related instabilities cause the plasma to break away from the magnetic field.

The spectrum of the normal modes connected with linearized perturbations around an equilibrium state is to be computed. The equilibria considered have cylindrical symmetry and depend only on the radius r. This symmetry permits separation of variables in the perturbed state x

$$x(r, \vartheta, z, t) = e^{\gamma t} x(r) \exp [i(m\vartheta + nz/L)],$$

where γ is the crucial growth rate, L is a length, and m, n characterize the perturbation. Then $x(r)$ satisfies the equation

$$\gamma x = f(x, \eta, \text{equilibrium}),$$

where f is a nonhermitian operator involving space derivatives only. This equation is usually solved by applying a finite element Galerkin procedure that yields the matrix eigenvalue problem

$$Fc = \gamma Mc,$$

where c is the vector expansion coefficient of $x(r)$ and γ is the (complex) growth rate, F is nonhermitian, M is symmetric positive definite and each matrix is block tridiagonal. Their order is $d = 12N$, where N denotes the number of radial mesh points.

In one center (the Max Planck Institute of Munich) N is currently taken as 50 and the problem is solved by band QR (on a Cray-1S)! If any eigenvalue γ has a positive real part then there is an exponentially growing instability in the equilibrium. The spectrum of the operator f is quite complicated, containing different branches: namely the fast, slow and Alfven modes. It has been found that only in ideal MHD ($\eta \equiv 0$) do the Alfven and slow modes form continua associated with singular eigenfunctions. For *any* finite resistivity these modes are damped, i.e., the eigenvalues γ leave the imaginary axis.

The physicists need to examine the stability of two-dimensional toroidal (Tokamak) equilibria where the different m-harmonics in the perturbations all couple. The associated matrices F and M are still banded but their order exceeds 10^4. Efficient computation of the eigenvalue with largest real part is required.

This group is potentially intensive but at present they await viable software for their 2D problems.

There are interesting nonsymmetric eigenvalue problems which arise in economics and operations research, see Karlin (1959), Kleinrock (1976), Seneta (1981), but it is not clear that these users are in the intensive class—yet.

The present intensive users have developed their own eigenvalue software. In fact the structural engineers discovered the method of simultaneous iteration for themselves but gave it a more descriptive name, subspace iteration. The method itself is quite obvious and what counts is the implementation. It is sad that the beautiful program RITZIT developed by H. Rutishauser in 1968/69 and published in the handbook of Wilkinson and Reinsch, had no influence on the structural engineers working in the USA, although in Britain the work of Jennings did employ some of the techniques embodied in RITZIT. By the time the RITZIT quality does creep into the finite element packages subspace iteration will have been displaced by modern versions of the Lanczos algorithm.

5. How to exploit sparsity. It is possible to approximate an eigenvalue of a linear operator by "sampling" at well chosen points in its domain. Thus the simple power method samples the action of a matrix A on the sequence of column vectors x, Ax, A^2x, A^3x, $\cdots$. The sequence of Rayleigh quotients of these vectors converges quite rapidly to the dominant eigenvalue of A. Recall that the Rayleigh quotient of a vector v is v^tAv/v^tv. Consequently there are methods that need only be given a subroutine, call it OP for operator, which returns Av for any given vector v.

This is the *only* way in which A appears in the method. The structure and sparsity of A can be exploited by the user in writing code for OP. In other words, the buck is passed to the user! He, or she, is in the best position to take advantage of A's characteristics to speed up the formation of Av. When v has 10,000 components one pays attention to this matrix-vector product. All the software we describe below actually ignores sparsity completely; that feature is hidden in OP.

This situation is in stark contrast to EISPACK and to the direct solution of linear equations where a number of clever devices are used to take advantage of zero elements. Some of these sparse techniques may be used by the subprogram OP, but none of that work appears explicitly in the eigenvalue codes. Nevertheless the development of methods based on the user-supplied subprogram OP represents a beautiful division of labor: the method is not obscured by details of A's structure.

This is the place to mention a serious confusion that has arisen in the past in the assessment of methods. We focus on symmetric matrices for the rest of this section. At a certain level of abstraction the power method is identical to inverse iteration. One technique works with A, the other with A^{-1}. Inverse iteration performs the useful task of finding the eigenvalue nearest to 0 but pays the significant price of factoring A (to LU) in order to form $\omega = A^{-1}v$ (by solving $Lu = v$ and $U\omega = u$). If a sparse eigenvalue program is used to compute eigenvalues of A it makes an *enormous* difference whether the subroutine OP delivers Av or $A^{-1}v$.

Subspace iteration finds eigenvalues close to σ by factoring $A-\sigma I$ and letting OP deliver $(A-\sigma I)^{-1}v$. When $\sigma = 0$ the eigenvalues near 0 are computed first and quite rapidly.

The Lanczos algorithm is somewhat different. It produces eigenvalues at *both* ends of the spectrum of the linear operator represented by OP, the more extreme ones emerging before the interior ones. Thus Lanczos offers the hope of computing the eigenvalues of A nearest 0 without the pain of invoking some sparse factorization of A. In this sense Lanczos has been compared with subspace iteration. The performance depends quite strongly on the matrix but as the order n grows ($n > 100$ say) Lanczos fares worse and worse and is soon eclipsed by subspace iteration. Lanczos will have computed perhaps 50 unwanted eigenvalues near ∞ for every eigenvalue near 0. This is because, in most given problems, the larger eigenvalues are much better separated than the small ones. (OP is approximating an unbounded operator.) Although Lanczos is optimal in certain important respects it is a hopeless task to compute the smallest eigenvalue without either factoring A, or solving $Av = b$ iteratively, or having a good approximation to the wanted eigenvector. This is certainly the case when $n > 1{,}000$.

The solution is easy. Give Lanczos the same OP subroutine as subspace iteration. Then the power of Lanczos reveals itself quickly. Both methods are then working with $(A-\sigma)^{-1}$ to find eigenvalues near σ.

6. Software for the masses. By the time a matrix expert has seen his program appear in a refereed journal he probably never wants to see it again. The program comes to dominate its creator. Dr. R. C. Ward (ORNL) is well aware of the consequent difficulty of getting good matrix programs into the hands of nonexpert users. In conjunction with the Sparse Matrix Symposium of 1982 in Fairfield Glade, Ward, and his colleagues at Oak Ridge, created a public catalog of software (Heath (1982)) for sparse matrix problems, including eigenvalue extraction. This helped to focus the production of good software.

A nonexpert will still be annoyed at seeing perhaps six programs all designed for the same task and all based on, say, the Lanczos algorithm. His annoyance is forgivable but unwarranted. He should understand that the sparse eigenvalue problem is still a research topic. The experts do not yet know the best way to implement the Lanczos algorithm (to block or not to block, to orthogonalize the Lanczos vectors a lot, a little, or not at all), or how to handle secondary storage. Thus it is good that there are several rival programs until a reasonable consensus is reached. None of the codes has been subjected to widespread testing in a variety of computer systems.

It is difficult to appreciate the difficulties of software production without some acquaintance with the variety of computing systems in which the software may be used. Here is a typical example. Professor Ruhe's program STLM (see Table 1) was developed in Sweden. It was transferred to an identical CDC computer running the "same" operating system at Boeing Computer Services. STLM did not work at first but in fact the discrepancies were minor and easily fixed by the experts present. That is not the point of the story. What happened next was that STLM appeared to be a very expensive way to compute the desired eigenvectors. A little investigation revealed that the billing algorithm in the operating system at Boeing was quite different from the one used in Sweden. In particular there were heavy charges both for the quantity of data transferred from the fast memory and for the number of transfers. A far from trivial modification of the program was needed to adapt it to the local pricing system.

In general it is difficult for the author of a program to know the full range of billing algorithms. Even if it were possible to code an algorithm so that it never did badly on any current pricing mechanism it is likely that the code would never do well either. One possibility is to require certain pricing ratios as arguments to a subroutine. The defect with that approach is the severe increase in complexity of such programs. The good ship Portability may well founder on the rock called Operating Systems.

I now turn to the programs in Tables 1 and 2 and apologize to the authors of good published Lanczos programs that are not there. All the programs compute one or more eigenvalue/vector pairs of the symmetric problem $(A-\lambda B)z=0$, unless the contrary is stated. Comments are given at the side. The programs are in FORTRAN, most in the ANSI standard version of FORTRAN 66. All are portable to some extent,

TABLE 1
Symmetric problems, Lanczos based.

Author	Name	Lines	Distributor	Comments
J. Cullum and R. Willoughby	LMAIN LVMAIN	2,100	authors (IBM)	$A-\lambda I$ only no orthogonalization
J. Cullum and R. Willoughby	BLMAIN	1,000	IBM	$A-\lambda I$ only block, limited reorthogonalization
D. S. Scott	LAS02	3,288	NESC*	Block selective orthogonalization
T. Ericsson and A. Ruhe	STLM	6,500	authors (Sweden)	Shift and invert strategy (see § 8) no orthogonalization
B. N. Parlett and J. Reid	EA15AD	648	Harwell**	All eigenvalues in a given interval no eigenvectors no orthogonalization $A-\lambda I$ only

* National Energy Software Center, Room C-235, Bldg. 221, Argonne National Laboratory, Argonne, IL 60439.

** Harwell Library, Bldg. 8.9, AERE, Harwell, Oxfordshire, England.

TABLE 2
Symmetric problems, other methods. Few of the programs stand alone.

Author	Name	Lines	Distributor	Comments
I. Duff	EA12	427	Harwell*	$A-\lambda I$ only RITZIT inspired
P. J. Nikolai	SIMITZ	550	IMSL** or author	$A-\lambda B$ RITZIT inspired (TOMS***)
A. Jennings	SI	~500	author (Ireland)	$A-\lambda B$ (Internat. J. Numer. Meth. Engrg.)
J. A. Wisniewski and A. H. Sameh	TRACMN	586	authors	$A-\lambda B$ minimizes the trace of sections of $A-\lambda B$
E. R. Davidson	EIGEN	1,000	author (Univ. Washington Dept. Chemistry)	perturbation technique $A-\lambda I$ stands alone
		Nonsymmetric problems.		
A. Jennings and W. Stewart	LOPSI	~500	authors (Ireland)	subspace iteration (TOMS***)

* Harwell Library, Bldg. 8.9, AERE, Harwell, Oxfordshire, England.
** International Mathematical Statistical Library, GNB Bldg., 7500 Bellaire, Houston, TX 77036.
*** ACM Transactions on Mathematical Software.

some are completely portable. Between 30 and 60% of the lines are COMMENTS. A potential user should consult the catalog (Carpenter (1982) list) for more information on the programs.

6.1. Programs based on Lanczos. The modern versions of the Lanczos algorithm are not simple and most of them are described in Parlett, (1980, chapt, 13). A brief summary will be given here, but please see the discussion in § 8 for some important details.

Lanczos algorithms are iterative in nature. A starting vector (or block of vectors) is chosen and at each step a new vector (or block) is added to the sequence. In exact arithmetic these vectors are mutually orthogonal and of unit length. A characteristic and pleasing feature is that only the two most recent vectors (or blocks) are needed in the computation of the next one. In addition to these vectors the Lanczos method builds up a symmetric (block) tridiagonal matrix T, each step adds a new row (and column) to T. Quite soon some eigenvalues of T begin to approximate some eigenvalues of the operator hidden in OP. Almost always it is the extreme eigenvalues which are well approximated.

Sometimes an extreme eigenvalue is approximated to full working precision (say 15 decimals) after only 30 Lanczos steps. It is rare, however, that one can solve a linear system correct to 4 decimals after only 30 steps of the conjugate gradient algorithm (which is intimately related to the Lanczos algorithm), unless an excellent preconditioner is used. Thus it seems "easier" to find a few extreme eigenvalues and eigenvectors of A than to solve $Ax = b$!

On some systems EISPACK routines can be called automatically and it is natural to use them. However in order to stand alone such a program must explicitly include these subroutines. This is a small irritation.

The codes. All of the programs in Table 1 stand alone, they do not invoke other packages explictly. This feature turns out to be attractive to new customers.

The only program which can be obtained, independently of the author, from the Argonne Code Center (now called NESC, the National Energy Software Center) is Scott's LAS02. Documentation is available on an accompanying file. This code has been used extensively at Oak Ridge National Laboratory on a variety of structural analysis problems.

Documentation for the Cullum and Willoughby codes is obtainable in book form (Cullum and Willoughby (1983)). Their programs use little or no reorthogonalization of the Lanczos vectors but have developed ingenious ways to identify which eigenvalues of the auxiliary tridiagonal matrix actually belong to the original matrix. Their code is shorter than the rival codes.

The Swedish program uses blocks but does little reorthogonalization. Its chief feature is a sophisticated mechanism for choosing the shifts σ in the spectral transform technique launched by Ruhe and described in § 8. A careful comparison of LAS02 with STLM would be very interesting for the small band of specialists in matrix eigenvalue computations and would help in progress towards a preferred implementation of the Lanczos algorithm.

The program EA15AD is much shorter than the others because it does not use blocks, does not do any reorthogonalization, and finds eigenvalues, only, not even their multiplicity. The user selects the interval to be explored. If the interval happens to be empty the code will report that fact in reasonable time. This is noteworthy because the code assumes that OP delivers Av and so triangular factorization is not available. Thus the standard technique for checking the number of eigenvalues in an interval is not available. The latest version also permits the computation of eigenvectors by an auxiliary package EA15ED.

6.2. Programs not based on Lanczos. There are dozens of programs for computing eigenvalues, see the catalog Carpenter (1982, p. 37) for a list from the year 1981 alone. I apologize to neglected authors for the bias in my selection.

There are a number of long, sophisticated, implementations of subspace iteration buried in finite element packages. As such they are beyond the limits of this survey. However the first three programs in Table 2 are available realizations of subspace iteration.

This paragraph sketches the method. An initial set of orthogonal vectors is chosen. The number of vectors in the set is the dimension of the subspace. Call it p. The proper choice of p is important and difficult. These vectors may be considered as the columns of an $n \times p$ matrix X^0. At step j the subroutine OP is used to compute $Y := (A - \sigma B)^{-1} X^{j-1}$. Next the Rayleigh–Ritz approximations from the column space of Y are computed. These Ritz vectors go into the columns of the matrix X^j. They provide the best orthonormal set of approximate eigenvectors that can be obtained by taking linear combinations of the columns of Y. After a while some of the columns of X^j provide excellent approximations to some eigenvectors of the pair (A, B). There are a number of variants of this scheme and several clever tricks in the implementation. See Parlett (1980, Chapt. 14), Jennings (1981) or Stewart (1976) for more details.

The Achilles' heel of the technique is the selection of block size. The first two programs are by numerical analysts and it would be interesting to see how they compare with the codes embedded in the finite element (FEM) packages mentioned in § 2. It is likely that they are shorter, cleaner, more robust and more efficient than their FEM rivals. On the other hand they have not been fine tuned for structural analysis problems, and that might make a difference.

The entry TRACMN is the product of recent research. It is based on the fact that

$$\text{trace}\,[F^t(A-\sigma B)F],$$

is minimized over all $n\times p$ matrices F when, and only when, the columns of F are the eigenvectors belonging to the p eigenvalues closest to σ. At each step, the current F is adjusted in order to reduce the trace. We hope that the authors will compare it with LASO2 or some similar rival technique.

LOPSI is the only program offered in the software catalog Heath (1982) for the nonsymmetric problem. The program has been published in ACM Transactions on Mathematical Software (a severe test) and employs subspaces iteration.

One version of Davidson's method is realized in EIGEN and was listed in the catalog of the now defunct National Resource for Computation in Chemistry (LBL, Berkeley). The general reader is warned that Davidson's method has been developed for problems in quantum chemistry. The matrix must be strongly diagonally dominant in order for this perturbation technique to be justified. When all the diagonal elements of H are the same then Davidson's method reduces to Lanczos. The difference is as follows. At step j the Rayleigh–Ritz technique is used to produce the best approximation v_j to the fundamental eigenvector that can be obtained as a linear combination of the current basis vectors $b_1,\cdots,b_j$. Let $\rho(x)$ be the Rayleigh quotient, $\rho(x)=x^tHx/x^tx$. The residual vector

$$r_j := (H-\rho(v_j))v_j.$$

is proportional to the gradient vector $\nabla\rho$ at v_j.

Perturbation theory now says that if v_j is a good approximation then an even better one is

$$a_j := (\rho(v_j)-\text{diag}\,(H))^{-1}r_j.$$

Davidson proposed to take as b_{j+1} the normalized component of a_j orthogonal to $b_1,\cdots,b_j$.

We make three observations.

(1) The "interaction" matrix or projection B^tHB is not tridiagonal but full.

(2) The method aims at a particular eigenvalue/vector pair.

(3) When diag $(H)=h_{11}I$ then a_j is a multiple of r_j and Lanczos is recovered. This last assertion is not obvious, since $v_j\neq b_j$, but it is true.

Observation 2 shows that Davidson is not really a rival to Lanczos. The methods address different tasks. Moreover Davidson exploits the fact that good starting vectors are available from chemistry.

7. Lanczos versus subspace iteration.

These two rival techniques are both good and address the same task. A comparison of them is given in Parlett, Nour-Omid and Taylor (1983) and the conclusion there is that a modern, properly used version of the Lanczos algorithm should be an order of magnitude faster than a good subspace iteration program on problems with $n>500$. The larger the number of modes wanted the greater the advantage of the Lanczos algorithm.

The fundamental theoretical advantage of the Lanczos is that it never discards any information whereas subspace iteration, at each step, overwrites a set of approximate eigenvectors with a better set. What is surprising is that the simple Lanczos algorithm needs only 5 or 6 n-vectors in the fast store at each step. Morever, it can find multiple eigenvalues if some form of orthogonalization is used.

There are enough poor implementations of Lanczos available to complicate comparisons. The engineers who have devoted themselves to steady improvements of subspace iteration are loyally defending the virtues of their codes. It will be interesting to see what happens.

8. What have we learnt since 1978? Much could be said in answer to this question but two items suggest themselves.

1. The importance to new users of stand alone programs. This point was made in § 5.

2. The spectral transformation: The Lanczos algorithm requires that a general linear problem

$$[K-\lambda M]x=0$$

be reduced to standard form. There are several ways to accomplish this and it is vital, for large problems, to choose a good one. Although the facts given below are elementary and have been known to the specialists in matrix computations for a long time it is fair to say that until A. Ruhe pointed out their implications in Ericsson and Ruhe (1980) they were not given the proper emphasis. Of course, subspace iteration has employed spectral transformations from the earliest days.

Ruhe proposed that the original problem be rewritten in the standard form

$$[M^{1/2}(K-\sigma M)^{-1}M^{1/2}-\nu I]y=0$$

with $y=M^{1/2}x$, and σ a shift into the interior of the interval which is to be searched for eigenvalues λ. This is quite different from the usual recommendation

$$[L^{-1}KL^{-t}-\lambda I]z=0,$$

where $M=LL^t$ and $z=L^tx$.

In the majority of large structural problems (with displacement formulation) M is singular as well as diagonal, perhaps $\frac{1}{3}$ of its diagonal elements are zero. The shift σ is not fixed precisely so that there is no loss in assuming that $K-\sigma M$ is nonsingular and can be factored in a stable manner without any row and column interchanges. If $K-\sigma M=LDL^t$ then the product $s=M^{1/2}(K-\sigma M)^{-1}M^{1/2}u$ is formed in stages:

$$v\leftarrow M^{1/2}u,\quad \text{solve } Lw=v,\quad \text{solve } DL^tx=w,\quad s\leftarrow M^{1/2}x.$$

If the factorization of $K-\sigma M$ is too costly then in principle $(K-\sigma M)y=M^{1/2}u$ could be solved by an iterative method. The point is that the subprogram OP in Lanczos should return $M^{1/2}(K-\sigma M)^{-1}M^{1/2}u$ when given u. Sparse techniques can be used in factoring $K-\sigma M$. This reduction of the problem transforms the spectrum according to

$$\nu_i=\frac{1}{\lambda_i-\sigma}$$

and so the eigenvalues λ_i closest to σ turn into the eigenvalues ν_i closest to ∞. These are the ones which will appear first in a run of Lanczos algorithm. One way to recover x from y is by solving $(K-\sigma M)x=My$. For large n the reduction in the number of steps is so satisfactory that it will offset the cost of factorization. Without the spectral transformation it will often be necessary to take $\frac{1}{2}n$ or more steps of the Lanczos algorithm in order to approximate the smallest 10 eigenvalues. Yet the Lanczos algorithm is most effective when used with fewer than 200 steps.

9. Work in progress. It is surprising, at first, that the software reviewed above is so narrowly focussed. So it is natural that a few investigators are working to "fill the gaps", to provide robust programs for all the requests that might conceivably be made for eigenvalues of various types of large matrix.

Normal matrices enjoy a full set of orthonormal eigenvectors and most methods designed for real symmetric matrices extend naturally to normal ones. The most important subclass of normal but nonsymmetric matrices are those that are skew and Ward has developed efficient software for them, as described in Ward and Gray (1978).

Work on nonnormal sparse matrices is still in the experimental stage. The experts are exploring the problem and there is no consensus on a preferred technique. Y. Saad has adapted Arnoldi's method for sparse problems. This is the natural extension of the Lanczos idea but subject to the constraint of using orthonormal bases for the associated Krylov subspaces. The auxiliary matrix generated in the course of the algorithm is Hessenberg (i.e., $h_{ij}=0$ if $i>j+1$) rather than tridiagonal. This is not a serious problem provided that fairly short runs of Lanczos are used, say 50 steps at most. Saad is experimenting with variations on this method in which orthogonality is given up in return for a banded structure in the Hessenberg matrix, see Saad (1980).

The Lanczos algorithm was generalized to the nonnormal case by Lanczos himself; the procedure is obvious. Unfortunately it can break down. In Parlett and Taylor (1981) it is shown how to restore a large measure of stability in return for a modest increase in complexity. Some feature of the original, strict Lanczos algorithm must be discarded if the breakdown is to be avoided. The new method looks ahead at every step and decides whether to enlarge the Krylov subspace by one or two dimensions. The auxiliary matrix is not quite triangular, there is a bulge every time the dimension increases by two. Column and row eigenvectors are given equal status and condition numbers are automatically computed.

Axel Ruhe is experimenting with a two-sided Arnoldi process but this work is at an early stage.

What impedes the production of pleasing software for nonnormal problems is the potential ill-condition of the problem itself. Expectations have been set by the symmetric case where the theory is most satisfactory. In the general case it is difficult to generate useful error estimates, let alone error bounds, and this affects the selection of stopping criteria. Numerical analysts would want to deliver condition numbers along with the eigenvalues, but users do not want this information, especially if it increases the cost by a noticeable amount.

We terminate this digression with a reminder that Jenning's program LOPSI has been published and is available to the public. LOPSI adapts subspace iteration to the nonsymmetric case.

The most promising developments are close to the symmetric case. Many problems in dynamics arise in the form

$$M\ddot{x}+C\dot{x}+Kx=f,$$

where $\dot{x}$ is the time derivative of the vector field $x(t)$. The proper choice of the damping matrix C is a challenge. The Scott and Ward (1982) program looks to the future and presents software for the associated quadratic eigenvalue problem

$$(\lambda^2M+\lambda C+K)u=0.$$

Their method is based on the Rayleigh quotient iteration. It does not take the easy way out and reduce the quadratic problem to a linear one of twice the size.

The Lanczos programs in the software catalog Heath (1982) work very well in combination with the spectral transformation discussed in § 8. That approach is not possible when the user cannot, or will not, allow the solution of linear systems of the form

$$(K-\sigma M)v=\omega.$$

What can be done?

Scott observed (Scott (1981)), that when σ is closest to the eigenvalue $\sigma+\delta$ of the pair (K, M) then the vector x belonging to the eigenvalue δ closest to zero for the standard problem

$$(K-\sigma M-\lambda I)x=0$$

is an approximate eigenvector of (K, M) belonging to σ. He formulates an iterative method in which a standard sparse eigenvalue problem is solved at each step for (γ, x) and $\sigma+\gamma$ converges monotonically to an eigenvalue of (K, M).

10. Conclusions. The eigenvalue problem for large, sparse matrices is not yet understood well enough, in all its computational ramifications, to permit the rapid deployment of impeccable software to the masses in the style set by EISPACK. Indeed it is doubtful that the public is impatient for the arrival of this facility. Those with an urgent need for such software prize efficiency above generality. They have developed their own programs independently of the numerical analysts and will continue to do so unless the matrix experts go to them and demonstrate that they can be useful.

The wide variety of computing environments is going to play havoc with naive concepts of portability. "Transmission of thought" is one of the two basic themes of science. The interesting and difficult task is to find an appropriate level at which sparse software can be transmitted usefully. Perhaps each FORTRAN program should mention the computing environments in which it performs dreadfully?

REFERENCES

J. A. CARPENTER (1982), KWIC *index for numerical linear algebra*, Rept. 106, Computer Science Department, Oak Ridge National Lab., Oak Ridge, TN.

J. CULLUM AND R. A. WILLOUGHBY (1983), *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Progress in Scientific Computing Series, Vol. X, Golub et al., eds., Birkhauser, Boston.

E. R. DAVIDSON (1975), *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real symmetric matrices*, J. Comp. Phys., 17, pp. 87–94.

——— (1983), *Matrix eigenvector methods*, Proc. NATO Advanced Study Institute on Methods in Computational Molecular Physics.

J. J. DONGARRA et al. (1979), LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia.

I. DUFF (1983), *A survey of sparse matrix software*, in Sources and Development of Mathematical Software, W. R. Cowell, ed., Prentice-Hall, Englewood Cliffs, NJ, 1984.

T. ERICSSON AND A. RUHE (1980), *The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmeric eigenvalue problems*, Math. Comp., 35, pp. 1251–1268.

B. S. GARBOW, et al. (1977b), *Matrix Eigensystem Routines*-EISPACK *Guide Extensions*, Lecture Notes in Computer Science 51, Springer-Verlag, New York.

K. K. GUPTA (1974), *Eigenvalue solution of damped structural systems*, Internat. J. Numer. Meth. Engrg., 8, pp. 877–911.

M. HEATH, ed. (1982), *Sparse Matrix Software Catalog*, available from Mathematics and Statistics Research Department, Oak Ridge National Laboratory, Oak Ridge, TN.

A. JENNINGS (1981), *Eigenvalue methods and the analysis of structural vibration*, in Sparse Matrices and Their Uses, I. Duff, ed., Academic Press, New York, pp. 109–138.

S. KARLIN (1959) *Mathematical Methods and Theory in Games, Programming, and Economics, Vol.* I, Addison-Wesley, Reading, MA.

L. KLEINROCK (1976), *Queuing Systems, Vol.* 2: *Computer Applications*, John Wiley, New York.

P. J. NIKOLAI (1979), *Algorithm* 538. *eigenvectors and eigenvalues of real centralized symmetric matrices by simultaneous iteration*, ACM Trans. Math. Software, 5, pp. 118–125.

B. N. PARLETT (1980), *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ.

B. N. PARLETT, B. NOUR-OMID AND R. L. TAYLOR, *Lanczos versus subspace iteration for solution of eigenvalue problems*, Internat. J. Numer. Meth. Engrg., 17 (1983), to appear.

B. N. PARLETT AND J. REID (1981), *Tracking the progress of the Lanczos algorithm for large, symmetric eigenproblems*, IMA. J. Numer. Anal., 1, pp. 135–155. The associated code is EA14XX in the Harwell Library.

B. N. PARLETT AND D. TAYLOR (1981), *Lookahead Lanczos algorithm for nonnormal matrices*, Rept. No. 43, Center for Pure and Applied Mathematics.

H. RUTISHAUSER, *Simultaneous iteration method for symmetric matrices*, Handbook for Automatic Computation, Vol. II, p. 284. See Wilkinson and Reinsch (1971).

Y. SAAD (1980), *Variations on Arnoldi's method for computing eigenelements of large symmetric matrices*, Linear Alg. Appl., 34, pp. 269–295.

P. SAXE, D. J. FOX, H. F. SCHAEFER III AND N. C. HANDY (1983), *The shape-driven graphical unitary group approach to the election correlation problem application to the ethylene molecule*, to appear.

D. S. SCOTT (1981), *Solving sparse symmetric generalized eigenvalue problems without factorization*, SIAM J. Numer. Anal., 18, pp. 102–110.

D. S. SCOTT AND R. C. WARD (1982), *Solving quadratic λ-matrix problems without factorization*, this Journal, 3, pp. 58–67.

E. SENETA (1981), *Computing the stationary distribution for infinite Markov chains* in Large Scale Matrix Problems, Bjorck, Plemmons and Schneider, eds., Elsevier, North-Holland, Amsterdam.

I. SHAVITT (1977), *The method of configuration iteration*, in Methods of Electronic Structure Theory, H. F. Schaeffer III, ed., Plenum, New York.

B. T. SMITH et al. (1974), *Matrix Eigensystem Routines*-EISPACK *Guide*, Lecture Notes in Computer Science 6, Springer-Verlag, New York.

J. M. SPEISER AND H. J. WHITEHOUSE (1982), *Parallel processing algorithms and architectures for real time signal processing*, Society of Photo-Optical Instrumentation Engineers, vol. 298 on Real-Time Signal Processing IV.

G. W. STEWART (1976), *A bibliographic tour of the large sparse generalized eigenvalue problem*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, pp. 113–130.

W. J. STEWART AND A. JENNINGS (1981), *Algorithm* 510. LOPSI: *a simultaneous iteration algorithm for real matrices*, ACM Trans. Math. Software 7, pp. 230–232.

R. C. WARD AND L. J. GRAY (1978), *Eigensystem computation for skew-symmetric matrices and a class of symmetric matrices*, ACM Trans. Math. Software, 4, pp. 278–285.

J. H. WILKINSON AND C. REINSCH (1971), *Handbook for Automatic Computation, Volume* II—*Linear Algebra*, Springer-Verlag, New York.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

DIRECT METHODS FOR SOLVING SPARSE SYSTEMS OF LINEAR EQUATIONS*

IAIN S. DUFF†

Abstract. We survey algorithms and software for solving sparse systems of linear equations by matrix factorization, paying particular attention to recent developments. We classify the various algorithms according to the type of system they solve (i.e. unsymmetric, symmetric definite, symmetric indefinite, unsymmetric but with symmetric pattern) and whether they perform pivoting for numerical stability. We consider both algorithms which work in main memory and those which use auxiliary storage.

We illustrate the performance of the major approaches which we discuss by runs on test problems.

Key words. sparse matrices, sparse linear equations, Gaussian elimination, sparse factorization, direct methods

1. Introduction. In this survey, we study algorithms and software for solving the system of linear equations

$$Ax = \mathbf{b} \tag{1.1}$$

where the coefficient matrix, A, is large and sparse. We consider matrix factorization methods for solving (1.1) concentrating on Gaussian elimination which computes the decomposition

$$PAQ = LU, \tag{1.2}$$

where P and Q are permutation matrices and L and U are (unit) lower and upper triangular matrices respectively. If P and Q are chosen on the basis of the sparsity pattern of A, without reference to actual numerical values, we refer to this process as ANALYZE and to the subsequent numerical factorization of PAQ as FACTOR. When the choice of P and Q and the actual factorization are performed together, we talk of ANALYZE-FACTOR. Given the factors, we then SOLVE by forward substitution

$$L\mathbf{y} = P\mathbf{b}, \tag{1.3}$$

followed by back substitution

$$UQ^T\mathbf{x} = \mathbf{y}. \tag{1.4}$$

We divide our survey according to the property of the matrix A. In § 2 we consider the case when A is symmetric and positive definite, while in § 3, we discuss the case when A is structurally symmetric (including numerically symmetric but indefinite systems). The solution of systems with an unsymmetric matrix is examined in § 4 and band and profile methods, which are efficient on matrices with a regular pattern, are considered in § 5. Some concluding remarks are made in § 6.

We emphasize recent advances and concentrate more on algorithms than software. We do, however, indicate which software in the catalog edited by Heath (1982) is pertinent to our algorithmic discussion. A recent systematic survey of sparse matrix software has been conducted by Duff (1982a). Although we comment on the merits of various approaches, we are in no sense attempting to compare software. A more

* Received by the editors December 30, 1982, and in revised form October 8, 1983. This paper was written by an employee of the United Kingdom Atomic Energy Authority. © UKAEA.

† Computer Science and Systems Division, AERE Harwell, Didcot, Oxfordshire, England OX11 0RA.

critical comparison of features in software for solving sparse linear systems is given by Duff (1979). The runs reported in this paper are used to illustrate particular points. The test matrices used are taken from the data base discussed by Duff et al. (1982), and the examples have been chosen to illustrate a wide range of cases.

We assume that the reader has some basic knowledge of sparsity and its exploitation. For example, we assume that the phenomenon of fill-in is appreciated. For people without this background, we recommend the book by George and Liu (1981), the chapter by Duff (1982b) or the excellent SIAM News articles by George (1980).

In an article of this breadth some intentional omissions must be made. We do not discuss in any detail the impact of new machine architectures and do not consider algorithms and software specifically designed for particular architectures. We do not describe direct methods based on Fourier transformations or cyclic or total reduction (so called fast direct methods) nor do we give details of the increasing interrelationship of direct and iterative methods through the use of preconditioning. There has been substantial algorithmic and code development on the manipulation of sparse systems, for example on bandwidth minimization. Although we discuss some preorderings in § 4, we do not consider such auxiliary algorithms or routines in detail.

2. Symmetric positive definite systems. In this section we consider the case when the coefficient matrix, A, is positive definite and so can be factored as

$$PAP^T = LL^T \tag{2.1}$$

where P is a permutation matrix and L is a lower triangular matrix. Although the Cholesky factorization (2.1) is numerically satisfactory for positive definite A, the diagonal entries are often stored separately. This can save indexing storage, avoids taking square roots, and is valid (even if potentially unstable) when A is not positive definite. Our decomposition then becomes

$$PAP^T = LDL^T, \tag{2.2}$$

where L is now a unit lower triangular matrix and D is a diagonal matrix. Sometimes D^{-1} rather than D is stored to avoid any divisions during the SOLVE phase.

Because the decompositions (2.1) and (2.2) are stable for any permutation matrix P when A is positive definite, it is possible to choose P without reference to the actual numerical values of A. For sparse matrices, this separation of determining P (the ANALYZE phase) from the numerical factorization (FACTOR) is extremely significant, since the permutation P can be computed on sparsity grounds alone.

There are several criteria which can be used in choosing P. Most of them either try to reduce the number of nonzeros in L or constrain the structure of L. An example of the latter is the variable-band scheme where all entries (including zeros) between the first nonzero in each row and the diagonal are held. Variable-band solvers have a low integer overhead at the expense of more nonzeros in the factors and are discussed further in § 5. Since it has been shown that the problem of finding an ordering to minimize the fill-in is NP complete (Yannakakis (1981)) as is the minimization of bandwidth (Papadimitriou (1976)), algorithms for choosing P are heuristic. There are many heuristics available and the SPARSPAK package of George et al. (1980) implements five. Of these, the most widespread, successful and general heuristic is that of minimum degree. We will concentrate on this ordering strategy for the rest of this section.

The minimum degree ordering strategy chooses as pivot at each stage the diagonal entry from a row with least number of nonzeros, including fill-ins from previous pivotal steps. It is evident that such a choice minimizes the multiplication count at each step and limits the number of potential fill-ins. Although it can be shown (for example, Duff and Reid (1974)) that minimum degree need not minimize the total multiplication count, it is cheap and easy to implement and performs very well in practice. Recent implementations are very efficient and run in $O(\tau)$ space, where τ is the number of nonzeros in the original matrix. Although in practice the time is also typically $O(\tau)$, such a bound has not been established except for some very special matrix structures. The principal observation from which this high efficiency is obtained is that a full submatrix of a sparse matrix can be represented compactly by an index list of its rows and columns (K integers representing $K(K+1)/2$ reals). If it is only the structure which is required, then the nonzero values themselves need not be stored. These full submatrices occur naturally in Gaussian elimination since the submatrix whose rows (and columns) are those with nonzeros in the pivot column (row) will be full after that elimination step. Subsequent eliminations will require the merging of all index lists that involve the pivot in order to form the index set for the pivot row. Since the sets merged in will not be needed thereafter, there is no possibility of failure for lack of storage and indeed the storage demanded is very modest. Similarly the work is modest when we work with index lists of length K in place of triangular arrays of order K. This is illustrated in Table 2.1, where times are shown for the three phases of code MA27 from the Harwell Subroutine Library (Duff and Reid (1982b)).

TABLE 2.1
Times in seconds on an IBM 3033 *for three phases of the solution of positive definite systems.*

Order	1,561	1,176	900
Nonzeros	6,121	9,864	4,322
ANALYZE	.428	.295	.300
FACTOR	1.101	.341	.575
SOLVE	.118	.043	.066

In sparse codes prior to 1976 such an observation was not used and the ANALYZE phase mimicked FACTOR and was therefore slower since it also had to find and store fill-ins. The table also indicates the time for SOLVE. This time is proportional to the number of nonzeros in the factors and will usually be much less than for ANALYZE or FACTOR.

After P has been determined, additional symbolic manipulation (sometimes called SYMBOLIC-FACTOR) is performed just once to expedite subsequent FACTORs. We have included the time for this in the ANALYZE figures in Table 2.1. Recent work by George and Liu (1980) and others has shown that the symbolic factorize phase can be performed in time and space proportional to the number of nonzeros in the factors. This symbolic processing ensures that numerical factorization is dominated by the arithmetic operations themselves.

Perhaps the most dramatic feature of the recent improvements to the implementation of the minimum degree algorithm has been the ability to use the fast and cheap symbolic phases (which execute in a predetermined amount of space) to forecast the much greater storage and work which will be required subsequently by the numerical factorization. Thus it is possible to discover at fairly low cost whether it is worthwhile to pursue a direct method of solution. Another significant advance in the minimum degree algorithm has been the reduction in integer overhead by observing that because of fill-in, a row of the factors, L^T, may well have the same structure as the tail of the previous row (Sherman (1975)). For example, in a factorization using the MA27 code there are 17798 nonzeros in the factors of the nine-point discretization of the Laplacian on a 30×30 grid but the number of integer indices required is only 5580. This saving in overhead storage is illustrated again in § 5 when we compare the minimum degree ordering with more specialized techniques.

Software implementing the minimum degree algorithm efficiently is readily available for symmetric positive definite systems. Examples are YSMP (Eisenstat et al. (1977a), (1982)), SPARSPAK (George et al. (1980), George and Liu (1981)), MA27 (Duff and Reid (1982a), (1982b)) and SYMFAC-NUMFAC-BKSLVE (Gustavson (1972)). A comparison of the first three packages is given by Duff and Reid (1982a). To our knowledge, the only software available for positive-definite complex Hermitian matrices is ME27 (the complex counterpart to MA27) and CSPARSPAK (a complex version of SPARSPAK written by Lewis for Boeing Computer Services).

3. Symmetric patterns. We now consider the case when the coefficient matrix may sensibly be represented as having a symmetric pattern, possibly with a few zero values in positions that we regard as part of the structure.

We first discuss the case when A is symmetric (i.e. $a_{ij} = a_{ji}$) but it not necessarily definite. Here we could, of course, take a chance and perform the decomposition (2.2) using the algorithms and software discussed in § 2. Although this approach will work in many instances, it is not satisfactory in general because of possible numerical instability or breakdown. We can, of course, use a code for unsymmetric systems but then we lose any potential benefit from symmetry. We illustrate this loss in Table 3.1 where we can see the penalty incurred by using a more general code both in terms of time for ANALYZE-FACTOR and FACTOR and in storage which is more than doubled when symmetry is ignored. We give ANALYZE-FACTOR rather than ANALYZE times here because the unsymmetric code cannot perform ANALYZE by itself.

A stable method of decomposing indefinite systems while preserving symmetry was developed by Bunch and Parlett (1971) and this was extended to the sparse case by Duff et al. (1979). Stability is maintained by using pivots of the form

$$\begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix} \tag{3.1}$$

in addition to 1×1 pivots from the diagonal (a_{kk} say). A 2×2 pivot of the form (3.1) is used if the 1×1 pivots a_{kk} and a_{ll} are small relative to other entries in their rows. Our stability criterion in the sparse case is that a_{kk} is suitable for use as pivot if

$$|a_{kk}| > u \cdot \max_j |a_{kj}|,$$

where the threshold parameter u is set by the user to a value in the range $[0, \frac{1}{2})$. A 2×2 pivot is satisfactory if

$$\left\| \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix}^{-1} \right\|_\infty \max_{i \neq k,l} \{\max\{|a_{ik}|, |a_{il}|\}\} < u^{-1}.$$

The threshold parameter u is chosen to give freedom to pivot for maintenance of sparsity and must be less than $\frac{1}{2}$ to ensure that a pivot will always be available. Experience has shown that a value of around 0.1 is suitable. Munksgaard (1977) implemented the suggestions of Duff et al. (1979) but the FACTOR is performed at the same time as ANALYZE to enable the stability tests to be performed and so the times are high. Recently, Duff and Reid (1982a) have implemented 2×2 pivoting within a multifrontal scheme. In this approach an analysis is performed exactly as for the positive definite case and stability is ensured by perturbing this pivot choice during numerical factorization. The perturbations are performed only within small submatrices and the overhead for such pivoting is low (Duff and Reid (1982a)). The results for the symmetric code in Table 3.1 were obtained using this multifrontal approach. An added benefit is that pivoting for stability is being performed during FACTOR, not possible in conventional codes that stick to the predetermined sequence. The gains of using the multifrontal approach over Munksgaard's code are evident from the results given by Duff and Reid (1982a).

TABLE 3.1
Use of general code on symmetric systems. The codes used are MA28 (*unsymmetric*) *and* MA27 (*symmetric*) *from the Harwell Subroutine Library.*

Order	147	900	292	199
Nonzeros	1,298	4,322	1,250	536
Time (seconds on an IBM 3033)				
ANALYZE-FACTOR				
Symmetric code	.12	.94	.17	.09
Unsymmetric code	.60	7.43	.58	.20
FACTOR				
Symmetric code	.08	.64	.09	.04
Unsymmetric code	.17	.96	.14	.05
SOLVE				
Symmetric code	.008	.067	.012	.006
Unsymmetric code	.011	.055	.014	.006
Storage (words on an IBM 3033)				
Symmetric code	6,950	46,728	7,616	3,727
Unsymmetric code	15,020	108,029	17,166	7,468

When the coefficient matrix has a symmetric structure but is unsymmetric, the choices are very similar. We can either risk using diagonal pivots chosen on sparsity grounds alone, use a general unsymmetric code, or locally perturb a symbolically chosen pivotal sequence. This latter approach is discussed by Duff and Reid (1982c) and is similar to the multifrontal scheme discussed above except that the perturbation now allows off-diagonal pivots within the full submatrices. The stability criterion is then of the form

$$|a_{lk}| > u \cdot \max_i |a_{ik}|, \qquad u \in [0, 1). \tag{3.2}$$

Both YSMP and SPARSPAK, which we discussed in § 2, will solve either class of problem using diagonal pivoting with no stability control. The code of Munksgaard (1977), INDANL, is available from Lyngby and the multifrontal codes MA27 (indefinite) and MA37 (unsymmetric) are in the Harwell Subroutine Library. We defer our discussion of software for the more general case until § 4.

4. Unsymmetric systems.

4.1. Preordering to block triangular form. If the coefficient matrix is unsymmetric, then there is an added degree of freedom available inasmuch as there is no need to use the same permutations of rows and columns. Sometimes it is possible to permute the original system so that subsequent elimination can be restricted to submatrices of the permuted structure. A good example of such a preordering is to permute the matrix to block triangular form where the permuted matrix can be partitioned so that the resulting block matrix is triangular. We illustrate this below for the case with 3 blocks on the diagonal.

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix}. \tag{4.1}$$

The system (4.1) can then be solved in the order

$$\begin{aligned} A_{33}\mathbf{x}_3 &= \mathbf{b}_3, \\ A_{22}\mathbf{x}_2 &= \mathbf{b}_2 - A_{23}\mathbf{x}_3, \\ A_{11}\mathbf{x}_1 &= \mathbf{b}_1 - A_{12}\mathbf{x}_2 - A_{13}\mathbf{x}_3, \end{aligned}$$

so that the elimination operations are only performed within the blocks $A_{ii}(i = 1, \cdots, 3)$ and the off-diagonal blocks are only used for forward substitution and are not modified during the solution process. Matrices which can be permuted into the form (4.1) are called reducible and occur frequently in chemical engineering, econometrics and linear programming. Algorithms for obtaining the block triangular form are normally divided into two phases. The first phase permutes the matrix to ensure the diagonal is zero-free and the second applies a symmetric permutation to get the required form. The first phase is the maximum matching problem for which algorithms based on a depth first search have a worst case asymptotic complexity of $O(n\tau)$ where n is the order of the system and τ the number of nonzeros. On typical problems, however, the matching phase executes in $O(n)+O(\tau)$ time and this is the complexity of the second phase of the preordering. Harwell Subroutines MC21A (Duff (1981a)) and MC13D (Duff and Reid (1978)) effect the two stages respectively and are employed by the Harwell unsymmetric linear equation solver MA28 (Duff (1977)) to preorder the matrix prior to Gaussian elimination. Gustavson (1976) also has software (FULL ASSIGN and BLTF) for performing this preordering. We illustrate in Table 4.1, the low costs of block triangularization. The third case illustrates the benefits which can be obtained. The runs in this table were performed using Harwell Subroutine MA28. Further gains might be obtained on computing systems where advantage can be taken of the fact that the decompositions of the blocks on the diagonal can be performed simultaneously.

Not all systems can be permuted to a nontrivial block triangular form. For example, the matrix in the first column of Table 4.1 is irreducible. Common extensions to block

TABLE 4.1
Times in seconds on an IBM 3033 *for block triangularization.*

Order Nonzeros	900 4,380	199 701	822 4,841
Block triangularization	0.3	.03	.25
ANALYZE-FACTOR (after block triangularization)	11.4	.21	.48
ANALYZE-FACTOR (without block triangularization)	11.4	.24	1.52

triangularization are preorderings to bordered triangular or bordered block triangular form but, although algorithms have been proposed for these orderings (for example, Hellerman and Rarick (1971)), they have not yet established themselves as useful general tools partly because the border tends to be rather wide. The Hellerman and Rarick algorithm has been incorporated in a general linear equation solver by Stadtherr and Wood (1983).

4.2. Pivoting considerations. It is tempting to use the fast ordering algorithms of §§ 2 and 3 to obtain a pivotal sequence for unsymmetric problems. A common way of doing this is to employ a minimum degree ordering on the pattern of $A+A^T$ or the symmetric pattern whose upper (or lower) triangle is the same as the upper (or lower) triangle of A. This is the diagonal pivoting approach adopted by the unsymmetric version of the YSMP code (Eisenstat et al. (1977b)). Diagonal pivoting will produce sparse factors if the pattern is nearly symmetric and will be stable if the matrix is diagonally dominant. However, fill-in can be high if the structure is far from symmetric and severe instability or breakdown can occur when diagonal pivoting is used on general systems. We illustrate these points in Table 4.2, reproduced from Duff (1979). Diagonal pivoting performs well on a matrix with a nearly symmetric pattern (first column) but does very badly on the more unsymmetric pattern of order 822. In both cases, the accuracy of diagonal pivoting is poorer.

TABLE 4.2
Diagonal pivoting on unsymmetric matrices. The codes used were YSMP (*diagonal*) *and* MA28 (*general*).

Order	541	822
Nonzeros	4,285	5,607
Nonzeros in factors		
Diagonal	14,025	95,292
General	13,623	6,653
Times in seconds on an IBM 370/168		
ANALYZE-FACTOR		
Diagonal	1.98	31.89
General	3.99	14.98
FACTOR		
Diagonal	.29	16.18
General	.56	2.72
SOLVE		
Diagonal	.03	.21
General	.05	.03
l_2 norm of error		
Diagonal	8×10^{-6}	8×10^{-9}
General	4×10^{-9}	1×10^{-11}

The unsymmetric versions of the YSMP code or the SYMFAC/NUMFAC code (Gustavson (1972)) do not perform any pivoting for stability. It is possible, however, to perturb an initial pivotal sequence (either diagonal or unsymmetric) in a similar way to the multifrontal approach mentioned earlier. The MA37 code discussed in § 3 can be used in exactly this way. Another approach is to process the rows in pivotal order and within each row to choose the entry in the pivot column at that stage if it satisfies a threshold test

$$|a_{ik}| > u \cdot \max_j |a_{ij}|, \qquad 0 \leqq u < 1. \tag{4.2}$$

Otherwise, some other nonzero in the row which satisfies the test is used as pivot. The code NSPFAC of Sherman, which is an improvement on his earlier code NSPIV (Sherman (1978)), uses this strategy. Kaufman (1982) has incorporated code based on the ideas of YSMP and NSPFAC within the framework of the PORT library. Unfortunately, there is as yet no implementation of symbolic unsymmetric pivoting which is as efficient as the symmetric ordering algorithms discussed in § 2 so there is no cheap way of obtaining an ordering to drive these routines when the coefficient matrix is far from symmetric.

The most established and robust technique for general unsymmetric matrices is to combine a threshold test of the form (4.2) with some sparsity control and determine a pivotal sequence by executing an ANALYZE-FACTOR phase. For the unsymmetric case, the sparsity analogue of minimum degree is to choose as pivot a nonzero for which the product of the number of other nonzeros in its row with the number of other nonzeros in its column is a minimum. This Markowitz (1957) criterion then minimizes the number of multiplications/additions for the step and limits the fill-in. It can be combined with the threshold test (4.2) in the following way. The nonzeros are searched in approximate order of increasing Markowitz count and the one with lowest Markowitz count satisfying the threshold criterion is chosen as pivot. The ordered access to the nonzeros requires a fairly elaborate data structure. Several codes use variants of this strategy including MA28 (Duff (1977)), SSLEST (Zlatev et al. (1978)) and Y12M (Zlatev et al. (1981)). A variant of MA28 is incorporated in the NAG library as subroutines F01BSF and F04AXF.

A major difference between these unsymmetric codes and the symmetric ones discussed earlier is that the ANALYZE-FACTOR times are now greater. This is seen clearly if we compare the runs in Table 4.3 with those of Table 2.1. As in the symmetric case, the SOLVE times are substantially smaller than those for ANALYZE-FACTOR or FACTOR. The Harwell code MA28 was used for the runs in Table 4.3.

TABLE 4.3
Times in seconds on an IBM 3033 *for three phases of the solution of unsymmetric systems.*

Order	199	822	541
Nonzeros	701	4,841	4,285
ANALYZE-FACTOR	.11	1.33	1.72
FACTOR	.03	.19	.36
SOLVE	.01	.02	.03

There are two modifications to the Markowitz/threshold strategy which are worth discussing. Because of fill-in, the reduced matrix during sparse Gaussian elimination becomes progressively denser and so the use of a sparse matrix code with its complicated data structures and integer overhead is increasingly inefficient. The SLMATH code DMOOP (IBM (1976)) switches to a code for full matrices when the reduced matrix is full while the Harwell code MA31 (Munksgaard (1980)) for symmetric positive definite systems and an experimental version of MA28 (Duff (1982c)) switch at a user set density. Surprisingly, not only is there a significant speed advantage in switching at quite low densities (dependent on the structure of the matrix and the machine used), but there is also an overall reduction in storage since the increase due to storing explicit zero values is more than offset by the reduction in integer overhead (Duff (1982c)). The other modification involves restricting the number of rows or columns which will be searched when looking for the nonzeros with lowest Markowitz count. While for many systems good implementations of a Markowitz strategy involve little searching (Duff (1979)), on systems with a regular structure the pivot search can be expensive. For example, on a five diagonal matrix of order 2000 with bandwidth 42, the time taken for ANALYZE-FACTOR by the MA28 code on an IBM 3033 was reduced from 51.0 to 25.9 seconds when searching was restricted to three rows (the choice of three has been found empirically to be optimal over a wide range of problems). Furthermore, the number of nonzeros in the factors and the scaled residual were not significantly altered by this restriction. The codes SSLEST and Y12M implement restricted Markowitz as does a new version of MA28 (Duff et al. (1984)) where the option of an efficient implementation of full Markowitz has been retained because it has been found to be slightly but consistently better on irregular structures.

4.3. Use of drop tolerances. The principal problem with direct methods for solving sparse linear systems is in storing the factors which, because of fill-in, are usually far denser than the original matrix. If fill-ins which are substantially smaller than original entries are dropped from the structure, the storage will be reduced and the factorization obtained will be exact for a matrix

$$B=A+E$$

where the error matrix E will have small norm relative to A although it will not be at rounding error level. It is then possible to use this factorization of B as a preconditioning for an iterative method. Drop tolerances have been employed by MA31, where the iterative scheme used was conjugate gradients, and by Y12M and the new version of MA28 where iterative refinement is used as the iterative scheme. The subject of partial factorizations and preconditionings for iterative methods is also discussed by Eisenstat (1983).

4.4. Use of orthogonal decompositions. It is possible to use a QU (Q orthogonal, U upper triangular) rather than an LU decomposition to solve the system (1.1). Although the stability is very satisfactory, such methods have not been greatly favored for full systems because of the extra work involved in the factorization. In the sparse case, QU factorizations seem even less promising since there is considerably more fill-in than with Gaussian elimination. For example, in a plane rotation between two rows fill-in occurs in both the pivot and nonpivot rows and orthogonalization methods based on elementary reflectors (e.g. Householder's method) are even worse for fill-in (Duff and Reid (1975)). If we can avoid storing Q, however, the situation is not so gloomy. For the original system, Q^T can operate simultaneously on the matrix and the right-hand side(s) before being discarded, while subsequent systems can be solved through the reduced normal equations

$$U^TU\mathbf{x}=A^T\mathbf{b}. \tag{4.3}$$

Although the premultiplication by A^T in (4.3) can cause some information loss, the conditioning of (4.3) is usually tolerable. George and Heath (1980) (see also, Heath (1983)) make use of the observation that U is identical to the Cholesky factor of the normal equations matrix by performing the symbolic phases on A^TA to get the structure of U and a column ordering for the QU decomposition. Although this approach is primarily intended for the solution of linear least squares, it can also be used to solve general unsymmetric systems. We show, in Table 4.4, some results sent to us by Mike Heath from runs of MA28 (LU) and a prototype version of SPARSPAK-B (QU), an extension to SPARSPAK.

TABLE 4.4
Use of QU decomposition to solve $A\mathbf{x}=\mathbf{b}$.

Order	113	199	541
Nonzeros	655	701	4,285
Total storage in words			
LU	3,897	6,493	39,846
QU	5,494	5,167	31,798
Time for ANALYZE-FACTOR-SOLVE in seconds on an IBM 3033			
LU	.07	.11	1.22
QU	.30	.25	5.31

Although the storage for A which is necessary for solving further systems has not been included in the figures for the QU decomposition, it is evident that the storage is competitive with that required by the LU decomposition. The LLSS01 code of Zlatev and Nielsen (1979), which computes U row by row in order, could be used to solve linear systems using a QU decomposition.

4.5. Compiled code approach. All the techniques we have described so far can be classified as a looping index approach. Other classes of methods are the compiled code approach, where a loop free code for decomposing the matrix and solving the system is generated in the ANALYZE phase (Gustavson et al. (1970)), and the interpretative approach, where a simple code indicates the operations to be performed (Bending and Hutchison (1973)). These techniques were compared by Duff (1979). Their main problem lies in storing and executing the compiled code which can be very long (typically several megabytes). Also, GNSO (Gustavson et al. (1970)) produces its loop free code in Fortran, the length of which causes most Fortran compilers to fail. However, for small problems for which the compiled code fits into main storage, the speed during FACTOR and SOLVE is unsurpassed.

5. Regular patterns. The discretization of partial differential equations and the finite element analysis of engineering structures usually gives rise to matrices which can be ordered so that few nonzeros are far from the diagonal. This permits advantage to be taken of variable-band storage (Jennings (1966)), also known as "profile" or "skyline" storage, where all entries between the first nonzero in a row or column and the diagonal are stored explicitly. Only one integer need be held per row or column and all numerical processing is on full vectors. The disadvantage is that more reals have to be stored and processed. Recent advances in the minimum degree algorithm, outlined in § 2, mean that the matrix must be well-suited for the variable-band method for it to perform better. We illustrate this observation in Table 5.1, from Duff (1982b), where we show results on a two-dimensional partial differential equation and on a large power network. In the first case the methods are very comparable whereas in the second the minimum degree algorithm is clearly superior. For problems for which it is suited, the variable-band scheme has the advantages of much shorter code, good vectorization possibilities and ease of implementation out-of-core. General-purpose

TABLE 5.1
Minimum degree and profile scheme using reverse Cuthill–McKee ordering. The SPARSPAK *code was used for both orderings.*

Order	1,009	1,723
Nonzeros	3,937	4,117
Problem class	PDE	Network
Nonzeros in factors		
Minimum degree	19,352	6,313
Profile	26,254	80,983
Total storage in words		
Minimum degree	33,189	23,729
Profile	34,328	94,769
Time in seconds on an IBM 3033		
ANALYZE		
Minimum degree	.649	.733
Profile	.239	.263
FACTOR		
Minimum degree	.761	.220
Profile	.764	2.957

software is available from NAG (F01MCF and F04MCF, written by Cox of NPL) for in-core working and Harwell (MA15, written by Reid (1972)) for both in-core and out-of-core working. Although these codes solve definite systems, it is not difficult to extend variable-band techniques to solve nondefinite systems using partial pivoting.

The choice of ordering is important for the success of the variable-band technique. A good automatic procedure is that of Gibbs, Poole and Stockmeyer (1976) and an improved implementation has recently been published by Lewis (1982). As an example of the gains that may be obtained, Loden (1980) used the Gibbs–Poole–Stockmeyer algorithm to reduce, in a large scale production code where the variables were originally ordered by hand from the underlying geometry, the number of reals stored from 83530 to 45375 and the operation count from 5.7 million to 1.7 million on a structures problem with 691 degrees of freedom.

A closely related scheme is frontal elimination (Irons (1970), Hood (1976)). This is normally used for finite element problems to permit the matrix assembly (addition of the contributions from the individual elements) and elimination to take place together in a single pass, without the overall matrix A ever being stored. However, it can also be adapted to treat nonelement problems or already-assembled element problems by accepting the matrix row-by-row (Duff (1981c)). The advantage over the band scheme is principally organizational, but there can be sparsity gains since each variable is eliminated as soon as its row and column is fully assembled. As for the band scheme, high computational speeds are possible on vector machines (Duff (1981c), (1982c)). The Harwell Subroutine Library code MA32 (Duff (1981b)) implements a general purpose frontal scheme for unsymmetric matrices.

We indicate the usefulness of these approaches by two examples in Tables 5.2 and 5.3. The first is due to Loden (1980) and compares a general sparse out-of-core solver (Reid (1978)), which uses a local minimum-fill criterion within a multifrontal approach, with a variable-band solver (Felippa (1978)). Although the general sparse approach reduces storage by the factor 2.7 and operation count by the factor 6.1, its

TABLE 5.2
Comparison of two out-of-core codes on a large problem.

	General sparse	Variable-band
Order	3,423	
Nonzeros	113,062	
Nonzeros in factors	229,605	618,039
Operations ($\times 10^6$)	9.4	57.5
Factor time	50	158
I/O time	415	86
Total program size	71,056	57,343

TABLE 5.3
Comparison of codes on a pde discretization.

Order	2,500
Nonzeros	9,800
FACTOR time	
MA32 (frontal)	6.0
MA15 (variable-band)	3.5
SPARSPAK (sparse in-core)	3.0
SOLVE time	
MA32 (frontal)	0.64
MA15 (variable-band)	0.52
SPARSPAK (sparse in-core)	0.29
Core storage (k-bytes)	
MA32 (frontal)	90
MA15 (variable-band)	40
SPARSPAK (sparse in-core)	740

additional input–output make its overall cost greater. In Table 5.3 we show a comparison by Duff (1981c) between his frontal code MA32 for unsymmetric matrices, Reid's (1972) band-solver MA15 for symmetric matrices and SPARSPAK of George et al. (1980) for treating general symmetric matrices in-core. Since MA32 is designed for unsymmetric matrices, its storage and factorization time must be expected to be about double that of the other two. The saving of core storage by the first two codes is impressive. It is at the expense of little extra time for FACTOR, but some for SOLVE because the nonzeros of the factors must be read from auxiliary storage.

6. Conclusions. We have emphasized some recent developments which excite us, particularly the availability of cheap ANALYZE codes that do not unexpectedly run out of storage and can predict the storage needed for actual factorization, along with improvements to some old friends such as variable band and frontal codes and the Markowitz algorithm for unsymmetric systems. There remains scope for further improvement, for example in extending cheap ANALYZE to unsymmetric patterns and in better exploitation of parallel hardware, but the overall picture is healthy. Good reliable code is available for solving sparse sets of equations.

Acknowledgment. I would like to thank John Reid for making several detailed comments on the manuscript and, in particular, for contributions to § 5 concerning the comparisons of variable-band and multifrontal schemes.

REFERENCES

M. J. BENDING AND H. P. HUTCHISON (1973), *The calculation of steady state incompressible flow in large networks of pipes*, Chem. Eng. Sci., 28, pp. 1857–1864.

J. R. BUNCH AND B. N. PARLETT (1971), *Direct methods for solving symmetric indefinite systems of linear equations*, SIAM J. Numer. Anal., 8, pp. 639–655.

I. S. DUFF (1977), MA28—*a set of Fortran subroutines for sparse unsymmetric linear equations*, AERE Report R.8730, HMSO, London.

——— (1979), *Practical comparisons of codes for the solution of sparse linear equations*, in Sparse Matrix Proceedings 1978, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, pp. 107–134.

——— (1981a), *On algorithms for obtaining a maximum transversal*, ACM Trans. Math. Software, 7, pp. 315–330, pp. 387–390.

——— (1981b), MA32—*A package for solving sparse unsymmetric systems using the frontal method*, AERE Report R.10079, HMSO, London.

——— (1981c), *Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core*, AERE Report CSS 89; this Journal, 5 (1984), pp. 270–280.

——— (1982a), *A survey of sparse matrix software*, AERE Report R.10512; in Sources and Development of Mathematical Software, W. R. Cowell, ed., Prentice-Hall, Englewood Cliffs, NJ, 1984.

——— (1982b), *Research directions in sparse matrix computations*, AERE Report R.10547; MAA Study on Numerical Analysis, G. H. Golub, ed., 1984.

——— (1982c), *The solution of sparse linear equations on the* CRAY-1, AERE Report CSS 125, presented at the Cray Research Inc. Symposium, Science, Engineering, and the CRAY-1, Minneapolis, April 5–7, 1982.

I. S. DUFF, R. G. GRIMES, J. G. LEWIS AND W. G. POOLE (1982), *Sparse matrix test problems*, ACM SIGNUM Newsletter, 17 (2).

I. S. DUFF, N. MUNKSGAARD, H. B. NIELSEN AND J. K. REID (1979), *Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite*, J. Inst. Math. Appl., 23, pp. 235–250.

I. S. DUFF AND J. K. REID (1974), *A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination*, J. Inst. Math. Appl., 14, pp. 281–291.

——— (1975), *On the reduction of sparse matrices to condensed forms by similarity transformations*, J. Inst. Math. Appl., 15, pp. 217–224.

——— (1978), *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Trans. Math. Software, 4, pp. 137–147, pp. 189–192.

——— (1982a), *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Trans. Math. Software, 9, pp. 302–325.

——— (1982b), MA27—*A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, AERE Report R.10533, HMSO, London.

——— (1982c), *The multifrontal solution of unsymmetric sets of linear equations*, AERE Report CSS 133, presented at Sparse Matrix Symposium, 1982, Fairfield Glade, TN, October 24–27, 1982; this Journal, this issue, pp. 633–641.

I. S. DUFF, J. K. REID, K. SCHAUMBURG, J. WASNIEWSKI AND Z. ZLATEV (1984), *New pivoting strategies in* MA28, *the Harwell code for solving sparse linear equations*, AERE Report, to appear.

S. C. EISENSTAT (1983), *Iterative methods for solving large sparse linear systems*, invited paper at Sparse Matrix Symposium 1982, Fairfield Glade, TN, October 24–27, 1982.

S. C. EISENSTAT, M. C. GURSKY, M. H. SCHULTZ AND A. H. SHERMAN (1977a), *Yale sparse matrix package* I, *The symmetric codes*, Report 112, Dep. Computer Science, Yale Univ., New Haven, CT.

——— (1977b), *Yale sparse matrix package.* II. *The nonsymmetric codes*, Research 114, Dept. Computer Science, Yale Univ., New Haven, CT.

——— (1982), *Yale sparse matrix package* I: *The symmetric codes*, Int. J. Numer. Meth. Engng., 18, pp. 1145–1151.

C. A. FELIPPA (1978), *Skymatrix processing library* (SKYPUL) *user's manual*, Report LMSC D623146, Lockheed Palo Alto Research Laboratory, Palo Alto, CA.

A. GEORGE (1980), *Direct methods for the solution of large sparse systems of linear equations. Parts* I *and* II, SIAM News, 13 (3) and (4).

A. GEORGE AND M. T. HEATH (1980), *Solution of sparse linear least squares problems using Givens rotations*, Lin. Algebra and Appl., 34, pp. 69–83.

A. GEORGE AND J. W-H. LIU (1980), *A fast implementation of the minimum degree algorithm using quotient graphs*, ACM Trans. Math. Software, 6, pp. 337–358.

——— (1981), *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ.

A. George, J. Liu and E. Ng (1980), *User Guide for* SPARSPAK: *Waterloo Sparse Linear Equations Package*, Report CS-78-30 (revised January 1980), Dept. Computer Science, Univ. Waterloo, Waterloo, Ontario, Canada.

N. E. Gibbs, W. G. Poole, Jr and P. K. Stockmeyer (1976), *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM J. Numer. Anal., 13, pp. 236–250.

F. G. Gustavson (1972), *Some basic techniques for solving sparse systems of linear equations*, in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum, New York, pp. 41–52.

——— (1976), *Finding the block lower triangular form of a matrix*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, pp. 275–289.

F. G. Gustavson, W. M. Liniger and R. A. Willoughby (1970), *Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations*, J. Assoc. Comp. Mach., 17, pp. 87–109.

M. T. Heath, ed. (1982), *Sparse Matrix Software Catalog*, prepared by Oak Ridge National Laboratory in conjunction with Sparse Matrix Symposium 1982 at Fairfield Glade, TN, October 24–27, 1982.

——— (1983), *Numerical methods for large sparse linear least squares problems*, invited paper at Sparse Matrix Symposium 1982, Fairfield Glade, TN, October 24–27, 1982, this Journal, this issue, pp. 497–513.

E. Hellerman and D. Rarick (1971), *Reinversion with the preassigned pivot procedure*, Math. Programming, 1, pp. 195–216.

P. Hood (1976), *Frontal solution program for unsymmetric matrices*, Int. J. Numer. Meth. Engng., 10, pp. 379–399.

IBM (1976), IBM *System*/360 *and System*/370 IBM 1130 *and* IBM 1800, Subroutine Library—Mathematics, User's Guide, Program Product 5736-XM7, IBM catalogue # SH12-5300-1.

B. M. Irons (1970), *A frontal solution program for finite element analysis*, Int. J. Numer. Meth. Engng., 2, pp. 5–32.

A. Jennings (1966), *A compact storage scheme for the solution of symmetric linear simultaneous equations*, Comput. J., 9, pp. 281–285.

L. Kaufman (1982), *Usage of the sparse matrix programs in the* PORT *library*, Report 105, Bell Laboratories, Murray Hill, NJ.

J. G. Lewis (1982), *Implementation of the Gibbs–Poole–Stockmeyer and Gibbs–King algorithms*, ACM Trans. Math. Software, 8, pp. 180–189 and pp. 190–194.

W. A. Loden (1980), *A comparison of two sparse matrix processing techniques for structural analysis applications*, in Supplementary Study of the Sensitivity of Optimized Structures by P. S. Jensen and W. A. Loden, Report LMSC-D777859, Lockheed Palo Alto Research Laboratory, Palo Alto, CA.

H. M. Markowitz (1957), *The elimination form of the inverse and its application to linear programming*, Management Sci., 3 (1957), pp. 255–269.

N. Munksgaard (1977), *Fortran subroutines for direct solution of sets of sparse and symmetric linear equations*, Report 77.05, Numerical Inst., Lyngby, Denmark.

——— (1980), *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Software, 6, pp. 206–219.

Ch. H. Papadimitriou (1976), *The* NP-*completeness of the bandwidth minimization problem*, Computing, 16, pp. 263–270.

J. K. Reid (1972), *Two Fortran subroutines for direct solution of linear equations whose matrix is sparse, symmetric and positive-definite*, AERE Report R.7119, HMSO, London.

——— (1978), *A package of subroutines for solution of very large sets of linear finite-element equations*, AERE Report M.2947, HMSO, London.

A. H. Sherman (1975a), *On the efficient solution of sparse systems of linear and non-linear equations*, Ph.D. thesis, Dept. Computer Science Report # 46, Yale Univ., New Haven, CT.

——— (1978), *Algorithms for sparse Gaussian elimination with partial pivoting*, ACM Trans. Math. Software, 4, pp. 330–338, pp. 391–398.

M. A. Stadtherr and E. S. Wood (1983), *Sparse matrix method for equation based chemical process flowsheeting:* I. *Reordering phase.* II. *Numerical phase*, Report from Chemical Engineering Dept., Univ. Illinois, Urbana, to appear.

M. Yannakakis (1981), *Computing the minimum fill-in is* NP-*complete*, SIAM J. Alg. Disc. Meth., 2, pp. 77–79.

Z. Zlatev, V. A. Barker and P. G. Thomsen (1978), SSLEST: *A Fortran* IV *subroutine for solving sparse systems of linear equations, User's guide*, Report NI-78-01, Numerisk Institut, Lyngby, Denmark.

Z. ZLATEV AND H. B. NIELSEN (1979), LLSS01: *Fortran subroutine for solving linear least squares problems*, Report NI-79-07, Numerisk Inst., Lyngby, Denmark.

Z. ZLATEV, J. WASNIEWSKI AND K. SCHAUMBURG (1981), Y12M *Solution of Large and Sparse Systems of Linear Algebraic Equations*, Lecture Notes in Computer Science 121, Springer-Verlag, New York.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

ORDERING SCHEMES FOR PARALLEL PROCESSING OF CERTAIN MESH PROBLEMS*

DIANNE P. O'LEARY†

Abstract. In this work, some ordering schemes for mesh points are presented which enable algorithms such as the Gauss–Seidel or SOR iteration to be performed efficiently for the nine-point operator finite difference method on computers consisting of a two-dimensional grid of processors. Convergence results are presented for the discretization of $u_{xx}+u_{yy}$ on a uniform mesh over a square, showing that the spectral radius of the iteration for these orderings is no worse than that for the standard row by row ordering of mesh points. Further applications of these mesh point orderings to network problems, more general finite difference operators, and picture processing problems are noted.

Key words. parallel computation, mesh problems, finite difference methods, finite element methods, nine-point operator

1. Introduction. Consider the standard uniform mesh finite difference approximation to the equation

$$u_{xx}+u_{yy}=f(x, y)$$

in a square domain with appropriate boundary conditions. The equation for each mesh point involves data at that point and at its north, south, east, and west neighbors. The Jacobi iterative method for this problem converges, and the iteration matrix has a spectral radius of $\cos(\pi/n)=1-O(1/n^2)$, when the mesh is $n\times n$. The successive overrelaxation method (SOR) with optimal choice of the relaxation parameter gives an iteration an order of magnitude faster, with spectral radius $[1-\sin(\pi/n)]/[1+\sin(\pi/n)]=1-O(1/n)$. Thus, for this problem SOR is preferred over the Jacobi method for standard computers, since both take time proportional to n^2 per iteration. These standard results can be found, for example, in [23].

However, the Jacobi method has undergone a renaissance recently with the development of computers with parallel design. On a computer with n^2 processors connected in a two-dimensional grid with local communication only, one iteration of Jacobi can be completed in time independent of n, while SOR still requires $O(n)$ for the first iteration if the mesh points are ordered row by row. (However, successive iterations can overlap the first, and be completed in time independent of n.) More details on these implementations will be given in § 2.

SOR can be speeded for parallel computation by reordering the mesh points. For example, using the checkerboard ordering (Fig. 5a: all even numbered mesh points ordered after all odd points), the time per iteration using n^2 processors is again independent of n and the convergence rate is unchanged. The mesh can also be ordered by lines into a block scheme, so that all new values on a line are determined at once. If k lines are grouped together, the spectral radius is $1-O(k/n)$, but iteration time increases with n [18].

The checkerboard ordering does not work so well for more complicated elliptic equations or alternate approximation strategies. Whenever a finite difference mesh point (or finite element unknown) is linked to one of its diagonal neighbors, the checkerboard trick fails. The line methods are often still useful.

* Received by the editors December 27, 1982, and in revised form November 19, 1983. This work was supported by the Air Force Office of Scientific Research under Grant AFOSR-82-0078.
† Computer Science Department, University of Maryland, College Park, Maryland 20742.

One purpose of this work is to develop ordering strategies for use in solving certain discretizations of elliptic equations on parallel processors. These strategies are applicable to any problem in which the equation for each mesh point involves data at that point and any subset of its north, south, east, west, northeast, northwest, southeast, and southwest neighbors. The goal is to make iteration time independent of n without sacrificing convergence speed.

A second purpose is to note that these orderings are also useful in several classes of problems unrelated to partial differential equations.

This work is related to work performed independently by Adams [1]. In that paper, the four-color ordering of Fig. 5a is presented for the nine-point finite difference operator, and some multicolored orderings for other couplings of mesh points are also given. No theoretical results concerning rate of convergence are given, but numerical experiments on elliptic partial differential equations are reported.

There has also been other work on parallel iterative methods (see, for example, [8]). Most recent work (see [24] for an exception) has centered around implementation of the conjugate gradient algorithm and appropriate preconditionings. Sameh [22] discusses preconditioning partial differential equation problems by block Jacobi with line red/black ordering. Kowalik, Kumar, and Lord [10] discuss block Jacobi, and Kumar in her thesis [12] considers other preconditionings and examples. Lichnewsky [16] discusses preconditioning with an incomplete Cholesky factorization under the nested dissection ordering. Parter and Steuerwalt [19], [20] discuss convergence properties of various preconditionings based on block iterative methods.

Another aspect of the problem is the mapping of irregular mesh problems onto regular arrays of processors. One heuristic approach is given in [3]. The measure of success is taken to be maximizing the number of problem edges that match processor connections. In [7], the mapping problem is studied for adaptive local refinements of regular meshes.

In § 2 we present some background on parallel computation and mesh problems. In § 3 we present orderings for mesh points and discuss convergence rates for the system of equations corresponding to the nine-point finite difference approximation to the operator $u_{xx}+u_{yy}$. In § 4 we discuss implications for more complicated problems, including nonlinear systems of equations and constrained optimization problems.

2. Parallel computation of mesh problems. In this section, we consider sources of mesh problems and the implementation of the Jacobi, Gauss–Seidel, and conjugate gradient algorithms on parallel processors.

By a parallel computer system, we mean a set of processors, possessing some local memory, capable of performing some arithmetic operations and connected in some network so that each processor can communicate with "neighboring" processors and perhaps with common memory. Examples of parallel processors include the Denelcor HEP, the ILLIAC [2], DAP, BSP [11], FEM [9], the ZMOB [21], systolic arrays [13], wavefront array processors [14], [15], and plans for the Japanese Fifth Generation Computer System [17].

The examples we consider will assume that the processors are arranged in a two-dimensional grid. Each processor should have at least one connected neighbor in each adjacent row and column. This structure is of interest because in many sparse matrix problems, the graph of the matrix has the structure of a planar mesh. Such problems arise from diverse applications areas. Three are described below.

1) The discretization of elliptic partial differential equations imposes a regular or irregular grid on the region. In two dimensions, a finite difference method often results

in a rectangularly oriented grid in which each unknown is directly coupled to some subset of eight compass-point neighbors. Finite element methods over irregular regions produce less patterned grids, but are still characterized by local coupling only. Graded grids, which introduce refinement into some subregions, also occur commonly in such problems.

2) Network problems also exhibit a mesh structure, arising from limited connectivity between nodes of the network. Such problems are typical in electrical power system analysis, queuing theory models of communication networks, and geodesy.

3) Digital image processing problems also have mesh structure. In this case, the grid is usually quite regular, resulting from digital coding of a gray level or color level for each "pixel" or picture element. Typical pictures have 10,000–100,000 pixels. Key problems are noise smoothing, feature extraction (e.g., finding region boundaries), and scene analysis (e.g., determining the position of the light source). Often the problem is formulated as a constrained optimization problem

$$\min_{c \leqq u \leqq d} u^T A(u) u + u^T b.$$

In noise smoothing, for example, u is the vector of digitized color levels, c and d represent bounds on meaningful digitized colors, and A has the structure of a 9-point operator, since a color at one point is most closely coupled to the eight neighboring colors.

Jacobi-type iterative methods have been widely used for parallel computation of mesh problems. Application to partial differential equations and network problems seem to share a common heritage, but the developments for image processing were independent. Such iterations take the form

$$u_i^{(k+1)} = \Psi\{u_j^{(k)}: j \text{ is a mesh neighbor of } i\},$$

where Ψ is a function of several variables. Under a reasonable assignment of mesh values to processor nodes, the ith process can access neighboring values efficiently, update its own value, and be ready for the next iteration in time independent of the size of the mesh. An example is given in Fig. 1. Here we have a rectangular grid of processors, with nearest neighbor connections, applying the Jacobi iteration to a five-point operator. In many applications, however, these schemes are only slowly convergent, and more sophisticated methods are required.

Gauss–Seidel-like methods have also been considered. They take the form

$$u_i^{(k+1)} = \Psi[\{u_j^{(k)}: j \text{ is a mesh neighbor of } i \text{ and } j \geqq i\}$$
$$\cup \{u_j^{(k+1)}: j \text{ is a mesh neighbor of } i \text{ and } j < i\}].$$

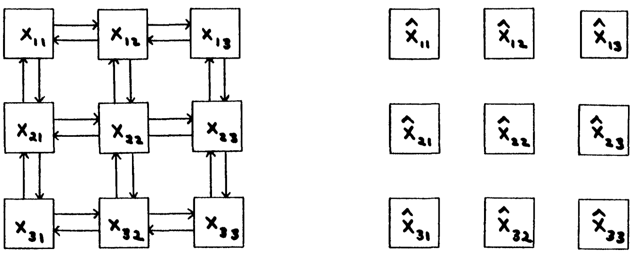


FIG. 1. *The Jacobi iteration for a five-point operator on a $n \times n$ grid of processors.* (a) *Step* 1: *Each processor passes its current mesh value to each of its neighbors.* (b) *Step* 2: *Each processor updates its mesh value.*

When expressed in this form, u_i cannot be updated until all of its neighbors with lower indices are updated. This method is illustrated in Fig. 2 for the same problem as Fig. 1 with row by row ordering of mesh points. The first iteration takes time proportional to n for an $n\times n$ grid, but each successive iteration takes only two more time units. Alternate orderings of mesh points are much better than this; we discuss them in the next section.

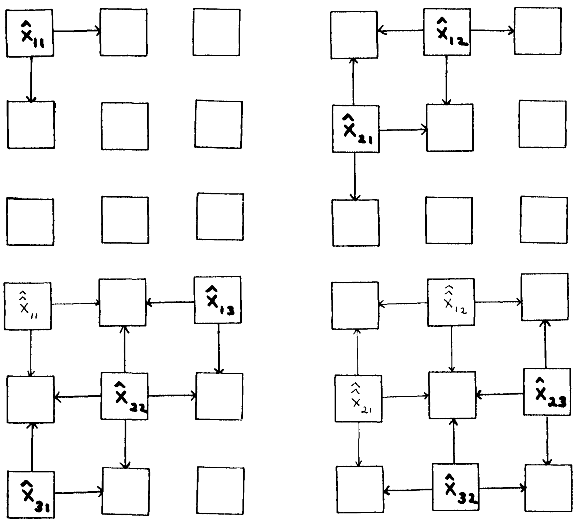


FIG. 2. *The Gauss–Seidel iteration for a five-point operator on a $n\times n$ grid of processors, using row by row ordering of the mesh points.*

An alternate way to consider Gauss–Seidel-type algorithms is to express them in iteration matrix form. To solve $Au=b$, for example, the iteration takes the form

$$u^{(k+1)}=(D-L)^{-1}[b+Uu^{(k)}]$$

where $A=D-L-U$, L is strictly lower triangular, and U is strictly upper triangular. Some researchers have proposed explicitly forming $(D-L)^{-1}$ or some approximation to it so that the iteration can be performed completely in parallel.

Because of success in solving problems on standard machines, methods like conjugate gradients are attractive candidates for parallel processors. They impose one further requirement on machine architectures, however: in addition to easy access to mesh neighbors, it is also necessary to accumulate inner products. On a rectangular $n\times n$ grid of processors with only nearest neighbor connections, this is an $O(n)$ process, quite slow for large grids. Some additional processor communication channels are necessary. Some alternatives follow:

(1) One common proposal is to add to each column of processors the ability to accumulate an inner product quickly using a bus.

(2) Perfect shuffle connections among processors in each row and column reduce inner product time to $O(\log n)$. Connections for a single column of $n=16$ processors are shown in Fig. 3. Information in a processor is redistributed as if it were on a card being shuffled in a deck. For $n=16$ processors, the successive reorderings are

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16
1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16
1	3	5	7	9	11	13	15	2	4	6	8	10	12	14	16

The cycle repeats every log n steps. Thus, if originally each even processor j accumulates the jth and $(j+1)$st elements in the inner product, then at stage 2, processors numbered $2, 4, \cdots, 16/2$ can accumulate four terms. At the third stage, 8-term partial sums can be accumulated, and the 16-term inner product is available after log 16 steps. It can then be communicated to each processor in another log n steps.

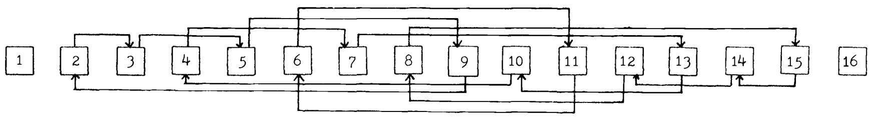


FIG. 3. *Perfect shuffle connections among* 16 *processors.*

(3) An arrangement with the same speed for inner products but with fewer connections and fewer wire crossings is shown in Fig. 4. In this incomplete interchange, the even processors send their information to the top of the grid in reverse order. The successive reorderings for $n = 16$ are

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
16	14	12	10	8	6	4	2	*	*	*	*	*	*	*	*
*	*	*	*	2	6	10	14	*	*	*	*	*	*	*	*
*	*	*	*	14	6	*	*	*	*	*	*	*	*	*	*

Again, inner products can be accumulated in log n steps and broadcast along the reverse pathways.

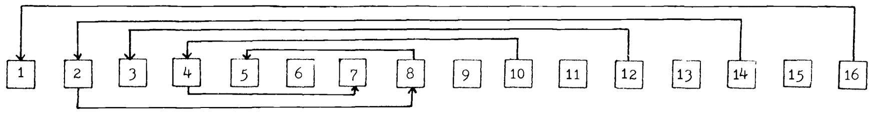


FIG. 4. *Incomplete interchange connections among* 16 *processors.*

It is interesting to note that either the perfect shuffle or the incomplete interchange connections shown above make multigrid iterations [4] possible on a nearest neighbor grid, since examination of the permutation pattern shows that within log n steps, rearrangements are made which could be used to place in proximity every other mesh point, every fourth mesh point, every eighth, etc.

3. Orderings for nine-point operators. Figure 5 shows orderings of mesh points which can make algorithms like SOR practical for parallel computation when the equation at each mesh point depends on the point itself and any subset of its eight immediate neighbors. To make the discussion clear, we will use the P^3 scheme as an example. The other schemes are similar and, in many cases, simpler.

Note in Fig. 5b that we have divided the mesh points into three groups. Those labeled "1" are to be ordered before those labeled "2", and those labeled "3" are last. Within each group, neighboring points—those in the same "P"—are numbered consecutively in an arbitrary way. The matrix corresponding to the mesh in Fig. 5b has the sparsity structure shown in Fig. 6. Notice that the pattern is

$$\begin{bmatrix} D_1 & A & B \\ A^T & D_2 & C \\ B^T & C^T & D_3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{1}$$

1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3
1	2	1	2	1	2	1	2	1	2
4	3	4	3	4	3	4	3	4	3

FIG. 5a. *Four colors are necessary for checkerboard ordering for a nine-point operator.*

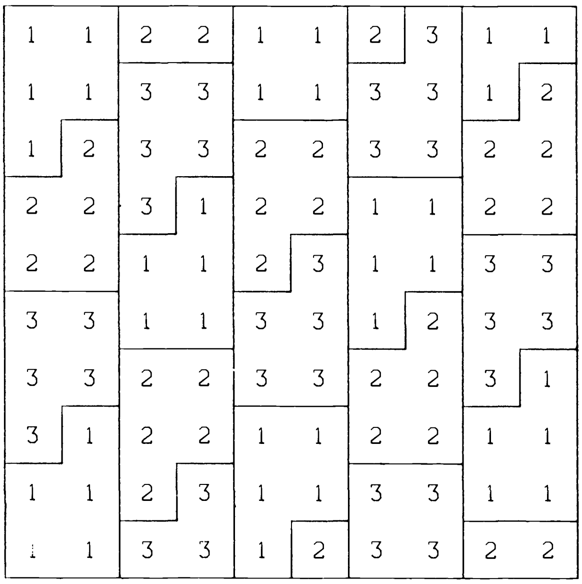


FIG. 5b. *The P^3 ordering.*

where D_1, D_2, and D_3 are block diagonal matrices with blocks of size 5×5 or less, and the vector u_i consists of all variables numbered "i". For parallel processing, the $(k+1)$st step of this scheme would be as follows:

(1) Perform an iteration of block SOR on the first group of equations:

$$u_1^{(k+1)}=(1-\omega)u_1^{(k)}+\omega D_1^{-1}(v_1-Au_2^{(k)}-Bu_3^{(k)}).$$

Note that each "P" group can be processed independently and concurrently by a

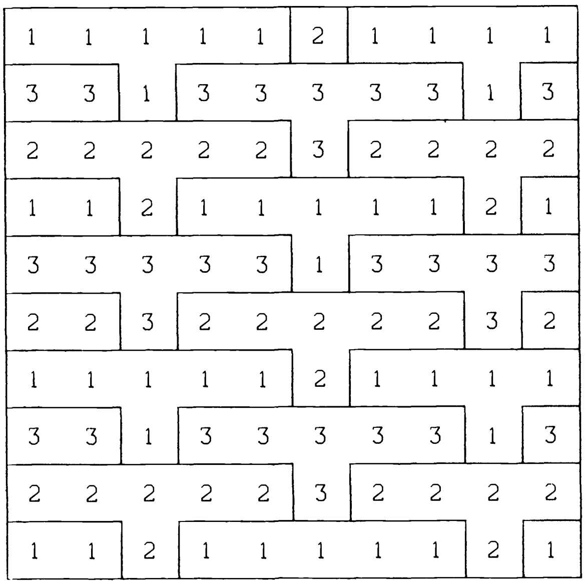


FIG. 5c. *The T^3 ordering.*

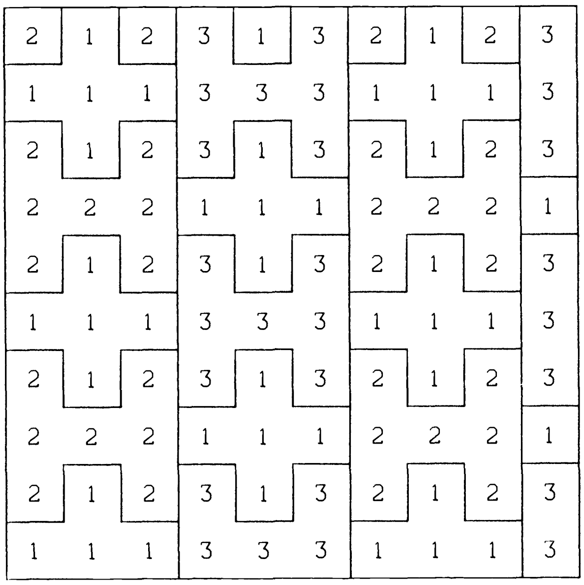


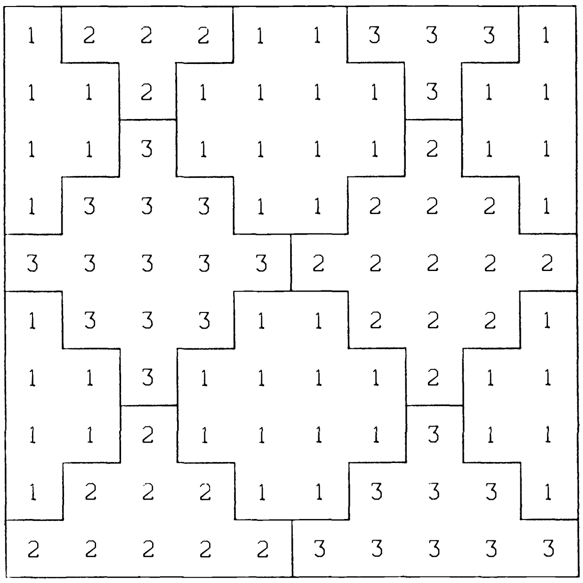
FIG. 5d. *The $H+H$ ordering.*

separate processor or group of 5 processors. Only 5×5 linear systems need to be solved directly.

(2) Process the second group of equations similarly:

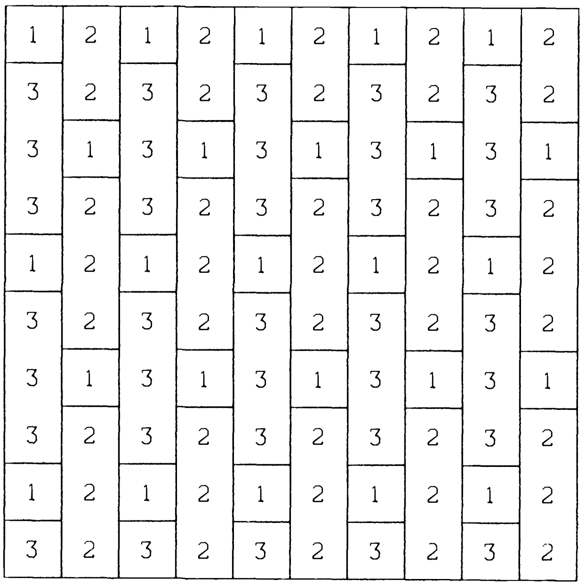
$$u_2^{(k+1)}=(1-\omega)u_2^{(k)}+\omega D_2^{-1}(v_2-A^Tu_1^{(k+1)}-Cu_3^{(k)}).$$

Part of this computation could overlap (1).



1	2	2	2	1	1	3	3	3	1
1	1	2	1	1	1	1	3	1	1
1	1	3	1	1	1	1	2	1	1
1	3	3	3	1	1	2	2	2	1
3	3	3	3	3	2	2	2	2	2
1	3	3	3	1	1	2	2	2	1
1	1	3	1	1	1	1	2	1	1
1	1	2	1	1	1	1	3	1	1
1	2	2	2	1	1	3	3	3	1
2	2	2	2	2	3	3	3	3	3

FIG. 5e. *The Cross ordering.*



1	2	1	2	1	2	1	2	1	2
3	2	3	2	3	2	3	2	3	2
3	1	3	1	3	1	3	1	3	1
3	2	3	2	3	2	3	2	3	2
1	2	1	2	1	2	1	2	1	2
3	2	3	2	3	2	3	2	3	2
3	1	3	1	3	1	3	1	3	1
3	2	3	2	3	2	3	2	3	2
1	2	1	2	1	2	1	2	1	2
3	2	3	2	3	2	3	2	3	2

FIG. 5f. *The Box ordering.*

(3) Process the third group:

$$u_3^{(k+1)} = (1-\omega)u_3^{(k)} + \omega D_3^{-1}(v_3 - B^T u_1^{(k+1)} - C^T u_2^{(k+1)}).$$

Computation of $(1-\omega)u_3^{(k)}$ could overlap (1), and computation of $v_3 - B^T u_1^{(k+1)}$ could overlap (2).

This scheme can be implemented efficiently on computers with n^2 processors (one per point), $n^2/5$ processors (one per "P"), or $n^2/15$ processors (one per cluster of

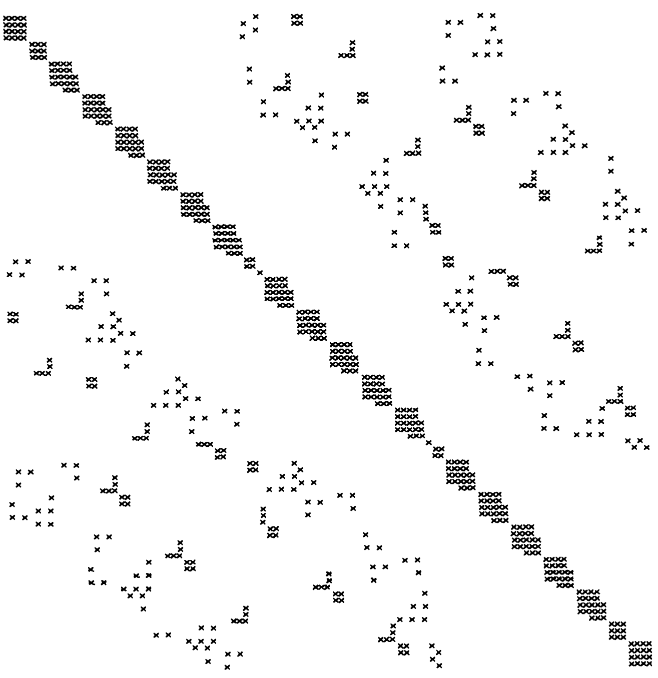


FIG. 6. *Matrix sparsity structure for the P^3 ordering.*

three "P's" numbered 1, 2, and 3). Communication is local: each P communicates with at most six of its neighbors, and by distributing the mesh points in the natural way, these will be on neighboring processors.

For illustration, we describe the algorithm for a two-dimensional grid of processors with communication connections to horizontal and vertical neighbors only, assigning three vertically adjacent "P's" to each processor. A processor's view of a typical iteration is as follows:

(1) For its block of 5 equations for $u_1^{(k+1)}$, each processor accumulates the terms involving points in u_1 and u_2 from information it already has. When the necessary u_3 values from the previous iteration arrive from (a subset of) the north, south, east, and west neighboring processors, then the u_3 terms are computed, a 5×5 linear system is solved, and then u_1 can be updated. Appropriate subsets of the 5 new u_1 values are then sent north, south, east, and west.

(2) Next, in a similar way, the processor accumulates terms for its 5 components of $u_2^{(k+1)}$ which involve points in u_2 and u_3 and completes the update after u_1 information arrives from the 4 neighbors. Then the 5 new u_2 values are sent north, south, east, and west as appropriate.

(3) The third set of 5 points is updated and communicated in the same way.

The iteration is synchronized by the data flow rather than by any global communication. If fewer processors are available, the "P's" can be enlarged, at the cost of solving linear systems larger than 5×5: each number in the "P" can represent a $j \times l$ group of mesh points for any integers j and l, giving $5jl \times 5jl$ systems to be solved. The iteration can be terminated after a fixed number of iterations or by a convergence test. If the communication required for a convergence test requires m times the time of an iteration, it could be performed roughly that often. Any global communication paths in the grid of processors (such as the ones described for inner products in § 2) are idle during the iteration process, and thus could be dedicated to convergence communication.

To study the convergence rate of the P^3 SOR method (as well as the others in Fig. 5) we list some properties of the matrix of (1) corresponding to the nine-point difference approximation to the Laplacian. Let D denote the block diagonal part; $-L$ the strictly lower triangular part; and $-U$ the strictly upper triangular part. The main tools we use are

(T1) [23, p. 91] For a regular splitting M–N of a Stieltjes matrix (one which is symmetric, positive definite, and nonpositive off the diagonal), if M and N have no common nonzeros, then putting more nonzero elements in M monotonically improves the spectral radius of the iterative method

$$u \leftarrow M^{-1}Nu - M^{-1}v.$$

(T2) [23, p. 124] For irreducible Stieltjes matrices, the "SOR theory" holds "approximately"; i.e., for

$$\omega_{\text{opt}} = \frac{2}{1+\sqrt{1-\rho^2(J)}}$$

where $\rho(J)$ denotes the spectral radius of the Jacobi iteration matrix, then, for the SOR iteration using this value of the relaxation factor ω,

$$\omega_{\text{opt}} - 1 < \rho(\text{SOR}) < \sqrt{\omega_{\text{opt}} - 1}.$$

We have the following properties:

(a) The original matrix is an irreducible Stieltjes matrix.

(b) The P^3 Jacobi method ($M = D$, $N = L + U$) and the P^3 Gauss–Seidel method ($M = D - L$, $N = U$), are regular splittings and thus convergent. They are also p-regular splittings.

(c) By (T1), the rate of convergence for each P^3 method is better than that for the corresponding standard method.

(d) Consider dividing the mesh of Fig. 5b into blocks, each containing two vertical lines of mesh points. By (T1), the spectral radius for the P^3 Jacobi method is not less than that for the two-line Jacobi method (since it is independent of ordering) and not greater than the standard point-Jacobi method. Thus

$$\rho(J_{2\text{-line}}) \leqq \rho(J_{P^3}) \leqq \rho(J_{\text{point}}),$$

and, since $\rho(J_{2\text{-line}})$ and $\rho(J_{\text{point}})$ are both $1 - O(1/n^2)$ [18], so is $\rho(J_{P^3})$.

(e) By (T2), there is a value of ω for which $\rho(\text{SOR}_{P^3}) = 1 - O(1/n)$. Thus, in using the P^3 ordering we have not sacrificed rate of convergence.

The P^3 scheme is the most complicated of the schemes in Fig. 5. The pattern repeats a shifted scheme of 4 columns. The other schemes are more regular.

The P^3 and T^3 schemes are balanced, dividing the mesh points into three groups of equal size. The $H+H$ and *Cross* schemes have about twice as many "1" points as "2's" or ("3's"), and the *Box* scheme has three times as many "2's" (or "3's") as "1's".

The patterns can be enlarged in various ways; for example, the $H+H$ scheme can be stretched in both dimensions, with each number in Fig. 5c representing a $j \times l$ block; stretched vertically, with each number in a crossbar representing a $2j \times l$ block while all other numbers represent $j \times l$ blocks; stretched horizontally, with each number in a vertical bar representing a $j \times 2l$ block while all other numbers represent a $j \times l$ block; or in a variety of other proportions.

4. Further applications. We have shown how the mesh orderings of Fig. 5 can be used to make the time per iteration of the standard stationary iterative methods on a two-dimensional grid of processors independent of n without sacrificing rate of convergence. We now discuss implications for more complicated algorithms and problems. In particular, we consider further cases in which iteration time is independent of n.

A. Other connectivities and geometries. It is not critical that the mesh be equally spaced, that the region be a square, or that each variable be directly coupled to all 8 neighbors. For example, hexagonal connections, and piecewise linear finite elements over regular triangles, also fall within this scheme.

B. Nonlinear problems. The iterations of § 2 can also be applied to nonlinear systems of equations $f(u)=0$, where the Jacobian matrix of f has nine-point connectivity structure.

C. Gradient methods. These mesh orderings can also be used to simplify steepest descent or conjugate gradient algorithms for the problem

$$\min_u u^TAu-u^Tb,$$

or a nonquadratic version of it, where A is positive definite and has nine-point structure. The standard algorithms require inner products over vectors of length n^2. On two-dimensional grids of processors with nearest neighbor connections only, this is an $O(n)$ process. An iteration can be derived, however, which holds two sets of variables constant while solving problems involving the third. Each iteration would take the form:

```
For i = 1, 2, 3
  Decrease the function by changing $u_i$, holding the other variables fixed. ("Decrease"
  could mean solving the subproblem exactly or simply reducing the objective function
  by several iterations of a gradient method.)
```

This breaks the problem into three parallel sets of small problems (5 unknowns each, for the P^3 ordering) which can be solved using local communication only.

This is a descent algorithm, but does not have the finite termination property of conjugate gradients.

The mesh orderings can also be used with the standard preconditioned conjugate gradient algorithm (see, for example, [5]). In this case the preconditioning operator, iterations of the SSOR iteration, for example, could be applied in time independent of n using only local communication, although the conjugate gradient iteration would still require inner products of length n^2.

D. Constrained problems. Free boundary problems for partial differential equations can lead to minimization problems with upper and lower bounds on the variables [6]: for example,

$$\min_u u^TAu-u^Tb, \qquad c\leqq u\leqq d,$$

or a similar problem with nonquadratic objective function. Iterations as in § 2 (Jacobi, Gauss–Seidel, SOR) are still applicable as long as each variable change is truncated if necessary to keep the variable within range.

Gradient methods are also often used for constrained problems. In addition to the inner products used to determine parameters, an additional global check is ordinarily necessary to calculate the maximum step which keeps the variables within range. The orderings of § 2 can be used to produce an algorithm in which the inner products and step length checking for gradient methods are reduced to local operations. The algorithm is analogous to that in (C) above.

E. Three-dimensional problems. These methods also extend to the solution of three-dimensional problems on two-dimensional arrays of processors. For an $n \times n \times p$ grid, an iteration using the ordering schemes above crossed with a line scheme in the third dimension would produce algorithms with iteration time proportional to p on a two-dimensional grid of n^2 processors with local connections.

Acknowledgments. Seymour Parter, G. W. Stewart, and David Young have made helpful suggestions related to this work.

REFERENCES

[1] L. M. ADAMS, *Iterative algorithms for large sparse linear systems on parallel computers*, NASA Contractor Report 166027, NASA Langley Research Center, Hampton, VA, 1982.

[2] G. H. BARNES, R. M. BROWN, M. KATO, D. J. KUCK, D. L. SLOTNICK AND R. A. STOKES, *The* ILLIAC IV *computer*, IEEE Trans. Comput., C-17 (1968), pp. 746–757.

[3] SHADID H. BOKHARI, *On the mapping problem*, IEEE Trans. Comput. C-30 (1981), pp. 207–214.

[4] A. BRANDT, *Multigrid solvers on parallel computers*, in Elliptic Problem Solvers, M. Schultz, ed., Academic Press, New York, 1971.

[5] P. CONCUS, G. H. GOLUB AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309–322.

[6] C. W. CRYER, P. M. FLANDERS, D. J. HUNT, S. F. REDDAWAY AND J. STANSBURY, *The solution of linear complementarity problems on an array processor*, Tech. Sum. Rep. 2170, Mathematics Research Center, Univ. Wisconsin, Madison, 1981.

[7] DENNIS GANNON, *On mapping non-uniform P.D.E. structures and algorithms onto uniform array architectures*, in Proc. Internat. Conf. on Parallel Processing, Ming T. Liu and Jerome Rothstein, eds., IEEE Computer Society, 1981.

[8] DON HELLER, *A survey of parallel algorithms in numerical linear algebra*, SIAM Rev. 20 (1978), pp. 740–777.

[9] H. JORDAN, *A special purpose architecture for finite element analysis*, Proc. 1978 Conference on Parallel Processing, IEEE Computer Society, pp. 263–266.

[10] J. S. KOWALIK, S. P. KUMAR AND R. E. LORD, *Solving linear algebraic equations on a* MIMD *computer*, Proc. International Conference on Parallel Processing, IEEE Computer Society, 1980.

[11] DAVID J. KUCK AND RICHARD A. STOKES, *The Burroughs scientific processor* (BSP), IEEE Trans. Comput. C-31 (1982), pp. 363–375.

[12] S. P. KUMAR, *Parallel algorithms for solving linear equations on* MIMD *computers*, Ph.D. thesis, Computer Science Dept., Washington State Univ., Pullman, 1982.

[13] H. T. KUNG AND C. E. LEISERSON, *Systolic arrays (for* VLSI), in Sparse Matrix Proceedings 1978, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1979, pp. 256–282.

[14] S. Y. KUNG, VLSI *array processor for signal processing*, Proc. Conference on Advanced Research in Integrated Circuits, Massachusetts Institute of Technology, Cambridge, 1980.

[15] S. Y. KUNG, R. J. GAL-EZAR, K. S. ARUN AND D. V. BHASKAR RAO, *Wavefront array processor: architecture, language, and applications*, IEEE Trans. Comput., C-31 (1982), pp. 1054–1066.

[16] A. LICHNEWSKY, *Solving some linear systems arising in finite element methods on parallel processors*, Tech. Rep., Université de Paris-Sud and INRIA, 1982.

[17] T. MOTO-OKA, ed., *Fifth Generation Computer Systems*, North-Holland, New York, 1982.

[18] SEYMOUR V. PARTER, *On estimating the "rates of convergence" of iterative methods for elliptic difference equations*, Trans. Amer. Math. Soc., 114 (1965), pp. 320–354.

[19] SEYMOUR V. PARTER AND MICHAEL STEUERWALT, *Another look at iterative methods for elliptic difference equations*, Computer Sciences Dept. Technical Report 358, Univ. of Wisconsin, Madison, 1979.

[20] ———, *Block iterative methods for elliptic and parabolic difference equations*, SIAM J. Numer. Anal., 19 (1982), pp. 1173–1196.

[21] CHUCK RIEGER, ZMOB: *Hardware from a user's viewpoint*, Proc. IEEE Computer Society Conference on Pattern Recognition and Image Processing, August, 1981, pp. 399–408.

[22] AHMED H. SAMEH, *Parallel algorithms in numerical linear algebra*, CREST Conference 1981, Bergamo, Italy, Academic Press, to be published.

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

[24] YEHUDA WALLACH AND V. KONRAD, *On block parallel methods of solving linear equations*, IEEE Trans. Comput., C-29 (1980), pp. 354–359.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

THE MULTIFRONTAL SOLUTION OF UNSYMMETRIC SETS OF LINEAR EQUATIONS*

I. S. DUFF† AND J. K. REID†

Abstract. We show that general sparse sets of linear equations whose pattern is symmetric (or nearly so) can be solved efficiently by a multifrontal technique. The main advantages are that the analysis time is small compared to the factorization time and that analysis can be performed in a predictable amount of storage. Additionally, there is scope for extra performance during factorization and solution on a vector or parallel machine. We show performance figures for examples run on the IBM 3081K and CRAY-1 computers.

Key words. frontal solution, Gaussian elimination, sparse matrices, unsymmetric equations, vectorization

1. Introduction. We consider the direct solution of sparse unsymmetric sets of n simultaneous linear equations

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

by an approach that is economical in the case where the pattern of nonzeros is symmetric or nearly so. For symmetric positive definite matrices A any choice of diagonal pivots for Gaussian elimination is numerically stable so they can be chosen on sparsity grounds alone. This choice can be made symbolically much faster and in much less storage than is needed for the actual numerical operations (George and Liu (1981)) and several good codes exist that do this (George et al. (1980), Eisenstat et al. (1977), (1982), Duff and Reid (1983)). By using frontal elimination (Irons (1970) and Hood (1976)) for the actual numerical operations and by including interchanges and the use of both 1×1 and 2×2 pivots, Duff and Reid (1983) extended this approach to symmetric indefinite matrices without risking numerical instability. Here we further extend the approach to unsymmetric matrices, assuming that the reader is familiar with our earlier paper (Duff and Reid (1983)). Note that the conventional approach to factorizing an unsymmetric sparse matrix with numerical pivoting is to perform the analysis and factorization together, which is typically about six times more costly than the analysis of a symmetric pattern. Note also that the original frontal method of Irons and Hood is usually unable to use a pivotal sequence that is as economical in overall storage or total number of floating-point operations. We illustrate this by showing some results of using a frontal code in § 4.

Our overall strategy is outlined in § 2 and we explain our numerical factorization in § 3. Comparisons with other codes are made in § 4 and conclusions drawn in § 5.

When performing the numerical experiments for this paper, we have used sets of test matrices covering a wide range of sparsity patterns and applications. Among the test matrices used were those in Table 1.1. We used more matrices in our experiments, but the above sample is representative of these and is sufficient to illustrate our points.

2. Overall strategy. Our overall strategy is to use the analysis algorithm of our previous work on symmetric matrices (Duff and Reid (1983)) on the pattern of the matrix $A+A^T$, creating an assembly tree and performing a depth-first search of it; then to factorize A in a similar way to our earlier work, except that now the whole of each frontal matrix is stored since it is unsymmetric and threshold pivoting with

* Received by the editors February 3, 1983, and in final revised form January 15, 1984. This article was written by an employee of the United Kingdom Atomic Energy Authority. © UKAEA.

† Computer Science and Systems Division, AERE Harwell, Oxfordshire, England OX11 0RA.

TABLE 1.1
Test matrices.

Order	Source	Application
147	Johansson, Lund	Eigenvalue calculations in atomic physics
1176	Erisman, Boeing	Electrical circuit analysis
199	Willoughby, IBM	Stress analysis
292	Ashkenazi, Nottingham	Normal equations matrix from surveying
130, 541	Curtis, Harwell	Stiff ODE Jacobians from laser research and from atmospheric pollution
406, 1561	George and Liu (1978)	Triangulation of L-shaped region
144, 1072	Marro, Cannes	Aircraft structures
532, 1224	Ponting, Harwell	Oil reservoir modelling
225	Bogle, Imperial College	Hydrocarbon separation problem in chemical engineering
1454	Lewis, Boeing	Power network
900		9-point discretization of the Laplacian on a 30×30 grid

row interchanges is introduced for numerical stability (see § 3 for details); finally the solution phase is similar except that now the upper triangular factor is no longer the transpose of the lower triangular factor.

Notice that analyzing $A + A^T$ implies that a zero a_{ij} for which $a_{ji} \neq 0$ is treated as a nonzero which has the value zero. Thus our approach will work for any pattern, but is likely to be inefficient for very unsymmetric patterns, and this is borne out by our experience (see Tables 3.2 and 3.3).

An important feature of our approach is that we can predict the storage and arithmetic requirements that would be needed for numerical factorization if no interchanges for stability are performed. This is particularly important since such a prediction may well determine whether it is computationally feasible or attractive to continue with the direct solution of the system.

Of course, the forecast may be optimistic for systems where interchanges are required but our experience (see Table 3.2) is that for symmetrically structured problems this inaccuracy is slight, although Ruhe and Ericsson (private communication) report a more significant difference when using our symmetric code (Duff and Reid (1983)) on problems with relatively small diagonal entries. This phenomenon of lack of diagonal dominance is also indicated by the experiments of Duff (1983).

3. The numerical factorization. Because of the depth-first search of the tree produced by our analysis routine we can, as in the symmetric case (Duff and Reid (1983)), store frontal matrices corresponding to generated elements in a stack.

The basic step of the numerical factorization is then:

i) the assembly of the appropriate number of generated elements from the top of their stack together with the so far unassembled nonzeros of the original matrix which are in the rows and columns now available as pivots;
ii) the elimination of as many as possible of these rows and columns using row interchanges, if necessary;
iii) the storage of the pivot rows and columns (which are part of the factorization);
iv) the placing of the reduced frontal matrix on the top of the stack of generated elements.

The basic scheme is thus that of Duff and Reid (1983) but there are some significant operational and organizational differences which we now discuss.

Because of our experience in the symmetric case (Duff and Reid (1983)) we store the frontal matrix afresh at every stage. We ensure that pivot candidates are the leading rows and columns, thereby reducing the number of permutations required. That is, the sort is done during the assembly rather than after it.

Since we follow our normal procedure (for example, used also in Harwell code MA28) of allowing the user to input the nonzeros in any order, we must first sort the matrix to avoid gross inefficiencies in the factorization. We will require the matrix in tentative pivot order by arrow-heads. That is to say that the nonzeros in a row of the upper triangular part of the matrix (permuted according to tentative pivot order) immediately precede those of the corresponding column of the strictly lower triangular part. An example is shown in Fig. 3.1. Sorting into this order enables a simple reuse of the storage occupied by the sorted matrix.

$$\begin{array}{ccccc}
1 & & 2 & & 3 \\
 & 6 & & 7 & \\
4 & & 9 & 10 & 11 \\
 & 8 & 12 & 14 & \\
5 & & 13 & & 15
\end{array}$$

FIG. 3.1. *The positions of the nonzeros of a* 5×5 *matrix when ordered by arrowheads.*

The storage organization we have used is illustrated for the reals in Figure 3.2 where the arrows indicate the directions in which the boundaries move. The associated integers are held similarly in a separate array. We have chosen this scheme because we wish to use a single array for the numerical processing and we also want to keep garbage collections infrequent and simple. The simpler scheme used in the symmetric case is not possible because we now need to store the pivot row and column (i.e. an arrow-head form) in the factor space while wishing to hold the front matrix in normal rectangular order for convenient row interchanges and eliminations.

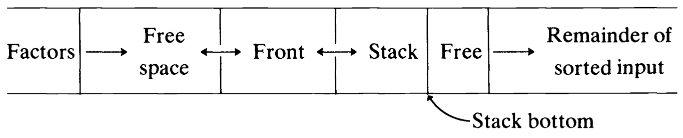


FIG 3.2. *Storage scheme for the reals during numerical factorization where the arrows indicate the directions in which the boundaries move.*

We know, in advance, the space required for the numerical factorization when no interchanges are necessary and additionally we know how much less space is required if we allow the stack to overwrite the already processed input when the free space is exhausted. This is the only form of garbage collection which we have and is particularly simple. Garbage collections on the real and integer files are performed similarly but completely independently. Table 3.1 shows examples of real and integer storage needed to avoid garbage collections, the minimum storage needed and the number of collections when minimum storage is used. The timing differences are comparable with the uncertainty of the IBM timer, thanks to the few collections needed and their simplicity.

Since there are many cases when it is known a priori that the decomposition is numerically stable for any sequence of diagonal pivots (for example, when the matrix is diagonally dominant), we provide an option which does no numerical tests other than ensuring that the pivots are all nonzero. If this option is not invoked then we

TABLE 3.1
The effect of running with minimum space.

Order	147	1,176	292	406	1,561	144
Nonzeros	2,449	18,552	2,208	2,716	10,681	1,296
Total storage required Reals × 1000						
No compress	7.7	45	8.0	16	85	3.3
Minimum	5.7	31	5.8	14	74	2.2
Total storage required Integers × 1000						
No compress	4.3	28	5.4	8	32	2.8
Minimum	2.8	21	2.9	5	19	1.7
Number compresses						
Real	9	2	10	8	10	9
Integer	4	3	8	4	5	8
Time (IBM 3081K secs.)						
Plenty	.065	.43	.057	.16	1.34	.115
Min. space	.068	.44	.059	.16	1.34	.118

require that a pivot must satisfy the threshold criterion

$$|a_{jk}| > u \cdot \max_i |a_{ik}| \tag{3.1}$$

where u is a preset parameter in the range $[0, 1)$. The nonzero a_{jk} must, of course, be available as a pivot candidate and note that this implies that there are no nonzeros in column k outside the front so the maximum can be evaluated within the front. This numerical test has been widely used in direct methods for sparse matrices and has been found satisfactory, the parameter u permitting an adequate balance between stability and sparsity preservation to be maintained. For a full discussion of stability and threshold pivoting, see Reid (1977). We have found that the value 0.1 is usually satisfactory.

TABLE 3.2
Storage for factors (excluding extra space needed during factorization but not during solution) and time to find them, with and without pivoting (symmetric pattern examples). Times are for actual factorization only.

Order	147	1,176	292	1,561	1,454
Nonzeros	2,449	18,552	2,208	10,681	5,300
Real storage (words × 1000)					
No numerical pivoting	4.62	20.6	5.28	66.3	10.3
With pivoting, $u = 0.1$	4.64	21.0	5.43	68.5	11.2
Integer storage (words × 1000)					
No numerical pivoting	1.46	6.85	2.56	18.4	9.30
With pivoting, $u = 0.1$	1.47	6.86	2.57	18.4	9.38
Time (IBM 3081K secs.)					
No numerical pivoting	.065	.43	.057	1.34	.031
With pivoting, $u = 0.1$	.068	.45	.062	1.42	.035

The results in Table 3.2 indicate that for symmetric patterns there is only slightly more storage required when numerical stability considerations delay pivoting and increase the front size. Thus the estimates from the analysis and tree search are a good guide to that actually required. In order to see the effect of badly-behaved problems, Duff (1983) ran our code on a sequence of five-diagonal matrices with decreasing diagonal entries. The increase in storage over the positive definite case was less than 50%.

TABLE 3.3
Time and storage for matrices with unsymmetric patterns. The two cases of order 541 *have quite different numerical values. Here the sort time is included.*

Order	199	130	541	541	225	1,224	532
Nonzeros	701	1,282	4,285	4,285	1,308	9,613	3,474
Asymmetry ratio	.94	.24	.32	.32	.90	.39	.26
Reals in factors (in 1000s)							
MA28 ($u=0.1$)	1.6	1.3	12.6	14.7	1.7	43	9.56
new code ($u=0$)	9.8	1.6	15.7	15.7	7.1	91	9.71
new code ($u=0.03$)	17.9	1.6	15.7	39.3	11.4	109	9.71
new code ($u=0.1$)	18.4	1.6	17.5	53.2	12.0	140	9.71
new code ($u=0.3$)	19.1	1.6	20.7	77.6	12.0	173	9.71
Factor time (IBM 3081K secs.)							
MA28 ($u=0.1$)	.026	.022	.18	.23	.02	1.1	.13
new code ($u=0$)	failed	.032	.26	.26	failed	3.4	.14
new code ($u=0.03$)	.74	.033	.26	2.16	.28	4.7	.15
new code ($u=0.1$)	.76	.033	.34	3.70	.30	7.8	.15
new code ($u=0.3$)	.79	.033	.54	6.40	.31	10.7	.15

The situation can be quite different if the pattern is very nonsymmetric, as the results in Table 3.3 show. We follow Duff (1983) in quantifying this asymmetry by the ratio of the number of unmatched off-diagonal entries ($a_{ij}=0$ but $a_{ji}\neq 0$) to the total number of off-diagonal entries so that a symmetric matrix has ratio 0.0. (The measure used by Erisman et al. (1983) is essentially the reverse of this.) Pivoting (i.e., $u>0$) may be essential if the code is not to fail (the matrices of orders 199 and 225 have some zeros on the diagonal) and yet it may drastically increase the storage and computing requirements. We show results for three values of the threshold parameter where, for some examples, the sensitivity to u is apparent. We have not shown in the table any measure of the accuracy of the results. The reason for this is that when failure does not occur, there is little difference between the runs. This would argue for a lower value of u to be the default, but we prefer 0.1 because of the added safety, our experience with other codes, and the generally small increase in requirements over lower values. Really these very nonsymmetric cases are outside the scope for which the new algorithm has been designed. A code, such as MA28, which treats the matrix as genuinely unsymmetric and uses a pivot sequence that takes advantage of the unsymmetric sparsity pattern is likely to be more successful and comparative figures are included in Table 3.3. It is apparent that MA28 performs particularly well when the asymmetry ratio is high. Note, however, that MA28 generally has much higher analyze times (see Table 4.1). Note also that in two cases (orders 130 and 532) our new code behaved perfectly satisfactorily.

The storage structure which we use for the factors greatly reduces the integer overhead as is evident in Tables 3.1–3.2 where sometimes substantially less words of storage are required for the integers than for the reals. This feature, which is shared by many codes for positive definite systems, is present because only one set of row and column indices are stored for each set of eliminations on the one frontal matrix.

4. Comparisons with other codes. We have developed a code based on the ideas in this paper and have placed it in the Harwell Subroutine Library under the name MA37.

YSMP (Yale Sparse Matrix Package, Eisenstat et al. (1977)) includes a code for unsymmetric matrices A based on an analysis of the sparsity of the upper triangle of A reflected in the diagonal. Since no numerical pivoting is performed it may fail, so it is not as general a code as MA37 but we use it as a basis for comparison. To prevent its failing we gave it only diagonally dominant matrices.

The conventional approach for unsymmetric sparse matrices involves taking account of numerical values during the analysis of the sparsity structure and we have run the Harwell code MA28 (Duff (1977)) as representative of this technique.

Another approach is that of George and Heath (1980) who use orthogonal reduction to triangular form, based on the analysis of the structure of the matrix A^TA. Code based on this will be available under the name SPARSPAK-B, but unfortunately was not available at the time of writing.

We have chosen to measure separately the three phases of analysis of the structure (ANALYZE), numerical input and factorization (FACTOR), and solution using the factors (SOLVE) since these correspond to the operations required once for a given structure, once for given matrix values, and once for each right-hand side respectively. We present the times in double precision on the IBM 3081K for these three phases in Table 4.1 and the storage required in Table 4.2.

The MA28 and MA37 ANALYZE and FACTOR entries include sorting of the user's input matrix whereas YSMP requires the nonzeros to be ordered by rows. We have therefore added the MA37 sorting times to the YSMP times in both cases. The sort time is also displayed so the YSMP run time for a sorted case can be found by a simple subtraction.

In Table 4.2, we have included all the storage required including overhead and permutations and have given values in words for both the IBM, where reals occupy two words and some integers (for example, row and column indices) occupy only half a word, and the CDC where reals and integers both occupy one word. Both versions are available in MA37 and MA28 but YSMP does not offer a half-word integer version although comments are included to allow rapid conversion between precisions. Addi-

TABLE 4.1
Times (IBM 3081K *seconds*) *for three phases.*

Order	147	1,176	292	1,561	1,454	1,072	532	900
Nonzeros	2,449	18,552	2,208	10,681	5,300	12,444	3,474	4,322
ANALYZE								
MA37 sort	.018	.14	.016	.08	.04	.09	.03	.06
MA37	.033	.25	.049	.19	.15	.30	.07	.19
YSMP	.054	.40	.062	.45	.22	.46	.32	.27
MA28	.290	1.50	.240	21.0	.44	15.0	.76	2.60
FACTOR								
MA37 sort	.020	.18	.018	.1	.05	.12	.03	.07
MA37	.085	.61	.075	1.4	.16	1.01	.14	.72
YSMP	.075	.57	.058	1.4	.11	.99	.40	.76
MA28	.091	.68	.073	4.1	.11	1.80	.13	.77
SOLVE								
MA37	.0058	.030	.0078	.080	.023	.054	.014	.044
YSMP	.0060	.026	.0067	.078	.017	.052	.037	.043
MA28	.0060	.029	.0074	.112	.019	.065	.014	.043

tionally, the data structure used by parts of the YSMP code involves pointers which require full integers (unless the number of nonzeros in the factors is severely limited) so full-word integer storage has been assumed throughout. We have, however, not included storage for the matrix reals in the YSMP ANALYZE because minor changes to some statements in a sort routine yield a version that does not need the real arrays.

The examples displayed in Tables 4.1 and 4.2 were chosen from our test set to be a representative sample of problems for which it is sensible to use MA37, that is the structurally symmetric or nearly symmetric cases. All but the case of order 532 are structurally symmetric and that one is nearly so, but YSMP chose a poor pivot sequence, presumably because it bases its choice only on the upper triangular part of the matrix.

On ANALYZE timings MA37 is consistently better than YSMP and much better than MA28. The FACTOR times are usually slightly worse than the less general YSMP code and are broadly comparable with MA28. The SOLVE times are comparable with MA28 and not as good as YSMP.

TABLE 4.2
Storage in thousands of Fortran words, with MA37 *in minimum space.*

Order	147	1,176	292	1,561	1,454	1,072	532	900
Nonzeros	2,449	18,552	2,208	10,681	5,300	124,444	3,474	4,322
ANALYZE (IBM)								
MA37	3	24	4	18	13	18	6	12
YSMP	8	52	9	84	22	61	39	49
MA28	15	68	17	282	35	159	30	100
ANALYZE (CDC)								
MA37	6	47	7	34	22	33	11	23
YSMP	8	52	9	84	22	61	39	49
MA28	15	73	176	263	41	148	29	92
FACTOR (IBM)								
MA37	14	84	15	163	30	117	25	91
YSMP	23	128	25	241	57	177	113	142
MA28	16	79	18	271	39	160	32	102
FACTOR (CDC)								
MA37	11	73	11	102	27	80	19	60
YSMP	15	86	17	162	39	118	76	95
MA28	16	88	18	234	42	146	31	94
SOLVE (IBM)								
MA37	10	48	13	146	29	97	23	79
YSMP	15	67	17	203	35	135	100	115
MA28	13	58	15	257	31	146	27	93
SOLVE (CDC)								
MA37	6	29	8	87	22	59	16	48
YSMP	10	45	11	136	24	90	67	77
MA28	11	48	13	208	27	118	22	75

Because of the overhead of assembling the frontal matrices in the multifrontal approach, MA37 will do better on larger problems where the factors are significantly denser than the original matrix. Thus we would expect MA37 to perform well on matrices from discretizations of partial differential equations (examples of order 1561, 1072, 532 and 900) or from finite element based discretizations (order 147, 1176) but less well on network problems (order 1454). This is broadly borne out by the results in Table 4.1.

On our set of test examples, our new code almost always requires less storage.

Finally, we demonstrate, in Table 4.3, the vectorization of our FACTOR and SOLVE codes by comparing them with YSMP on the CRAY-1 computer. YSMP uses indirect addressing throughout and hence does not vectorize which makes it less competitive.

Most routines which vectorize better on the CRAY-1 than MA37 do so at the expense of many more multiplications. Codes based on variable-band or frontal techniques fall into this category. We ran the Harwell frontal code MA32 (Duff (1981a)), which includes row interchanges but works with a single front, on the examples in our table. We found that the FACTOR and SOLVE times for MA32 were significantly slower than MA37 on the IBM 3081K. For example, on the problems of orders 1454 and 532 in Table 4.1, the MA32 FACTOR times were 1.93 and 1.32 seconds while the MA32 SOLVE times were .13 and .14 seconds. We were unable to solve several of our problems because the front-size became too large, but for the more banded structures (like those of the last two columns) the storage demands were quite modest. On these two examples the in-core requirements of the FACTOR entry of MA32 were only 33 and 15 thousand words on the IBM and 17 and 8 thousand words on the CDC.

The better vectorization of MA32 on the CRAY-1 (the vectors encountered are much longer than for MA37), yields more competitive times on that machine where the MA32 FACTOR times for the same problems (orders 1454 and 532) were .269 seconds and .260 seconds respectively. Evidence of greater vectorization is more clearly seen if the thousands of multiplications required by the algorithms are compared. They are 79 and 579 for MA37 as against 1,680 and 811 for MA32.

TABLE 4.3
CRAY-1 *timings in seconds of* MA37 *and* YSMP.

Order	147	1,176	292	1,561	1,454	1,072	532	900
Nonzeros	2,449	18,552	2,208	10,681	5,300	12,444	3,474	4,322
FACTOR								
MA37 (sort)	.023	.17	.021	.10	.051	.12	.03	.07
MA37	.047	.33	.052	.44	.146	.37	.09	.25
YSMP	.055	.39	.046	.81	.087	.59	.25	.46
SOLVE								
MA37	.0023	.015	.0045	.029	.0186	.021	.0083	.017
YSMP	.0030	.015	.0037	.040	.0092	.027	.0194	.023

5. Conclusions.

5. Conclusions. We have described algorithms and code for the solution of sparse linear equations using a multifrontal approach. Our software is primarily designed for systems which are structurally symmetric or nearly so. We have, however, shown that it can be used on quite general systems although it is better to use code specifically designed for the general case. Our code incorporates numerical pivoting and so is more robust than codes which pivot on the sparsity pattern alone. Nevertheless, we have shown our software to be competitive in speed and storage on the IBM 3081K with these less powerful codes and have demonstrated that, because of the vectorization of the inner loop, it will generally outperform them on the CRAY-1.

Acknowledgments. We would like to thank the associate editor, Alan George, and the referees for their helpful comments which have improved the presentation of this work.

REFERENCES

I. S. DUFF (1977), MA28—*a set of Fortran subroutines for sparse unsymmetric linear equations*, AERE Report R.8730, HMSO, London.

——— (1981a), *Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core*; this Journal, 5 (1984), pp. 270–280.

——— (1981b), MA32—*a package for solving sparse unsymmetric systems using the frontal method*, AERE Report R.10079, HMSO, London.

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

I. S. DUFF (1983). *The solution of nearly symmetric sparse linear equations*, AERE Report CSS 150, AERE Harwell; Proc. Sixth International Symposium on Computing Methods in Engineering and Applied Sciences, Versailles. North-Holland, Amsterdam, 1984.

S. C. EISENSTAT, M. C. GURSKY, M. H. SCHULTZ AND A. H. SHERMAN (1977), *The Yale sparse matrix package*, I, *The symmetric codes*; II *The nonsymmetric codes*, Reports 112 and 114, Dept. Computer Science Yale Univ., New Haven, CT.

——— (1982), *Yale sparse matrix package* I: *The symmetric codes*, Int. J. Numer. Meth. Engng, 18, pp. 1145–1151.

S. C. EISENSTAT, M. H. SCHULTZ AND A. H. SHERMAN (1981), *Algorithms and data structures for sparse symmetric Gaussian elimination*, this Journal, 2, pp. 225–237.

A. M. ERISMAN, R. G. GRIMES, J. G. LEWIS, W. G. POOLE AND H. D. SIMON (1983), *Evaluation of orderings for unsymmetric sparse matrices*, Report MM-5, ETA Division, Boeing Computer Services, Seattle, WA.

A. GEORGE AND J. W-H. LIU (1978), *An automatic nested dissection algorithm for irregular finite element problems*, SIAM J. Numer. Anal., 15, pp. 1053–1069.

A. GEORGE AND M. T. HEATH (1980), *Solution of sparse linear least squares problems using Givens rotations*, Lin. Alg. and Appl., 34, pp. 69–83.

A. GEORGE, J. W. LIU AND E. NG (1980), *User guide for* SPARSPAK: *Waterloo Sparse Linear Equations Package*, Research Report CS-78-30 (revised January 1980), Dept. Computer Science, Univ. Waterloo, Waterloo, Ontario, Canada.

A. GEORGE AND J. W. LIU (1981), *Computer Solution of Large Sparse Positive Definite Systems* Prentice-Hall, Englewood Cliffs, N.J.

P. HOOD (1976), *Frontal solution program for unsymmetric matrices*, Int. J. Numer. Meth. Engng, 10, pp. 379–400.

B. M. IRONS (1970), *A frontal solution program for finite element analysis*, Int. J. Numer. Meth. Engng, 2, pp. 5–32.

J. K. REID (1977), *Sparse matrices*, in The State of the Art in Numerical Analysis. D. A. H. Jacobs, ed., Academic Press, New York, pp. 85–146.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

A PROGRAM FOR FITTING RATE CONSTANTS IN GAS PHASE CHEMICAL KINETICS MODELS*

G. D. BYRNE,† A. J. DEGREGORIA† AND D. E. SALANE†‡

Abstract. This paper describes a program for developing gas phase chemical kinetics mechanisms. Program input is a proposed chemical reaction mechanism specified in the usual stoichiometric form and associated thermodynamic data. The program generates and solves the corresponding rate equations and adjusts reaction rate constants to fit a user-provided data base of experimental values of concentrations of certain chemical species. Examples are given.

Key words. kinetics, parameter estimation, inverse problems

1. Introduction. In this paper we outline a computer code which can be applied to various gas phase chemical kinetics models. The basic idea is as follows. The user develops an input file to describe the stoichiometry in a familiar chemical notation. The user can then call the code which fits the specified parameters in the rate coefficients so that the species concentrations fit user-supplied data. These data may represent pilot plant data, laboratory data, or the data from a full-scale operating system.

The program uses the Chemical Kinetics package CHEMKIN [15] to translate the reaction mechanism into a FORTRAN subroutine containing the rate equations. The ordinary differential equation (ODE) solver LSODE [14], [16] is used to solve the rate equations, and a constrained optimization procedure we have developed is used to adjust reaction rate constants to fit an experimental data base. The optimization algorithm has been implemented in a way that allows the user to fit fairly extensive data bases.

Since the program uses CHEMKIN as an interface between the reaction mechanism and the rate equations, the program can be used to model and find parameters for a large class of proposed mechanisms. The mechanisms can be easily changed and parameters selected for those reactions for which precise estimates are not available. The program has been used successfully to measure rate constants in the Thermal DeNO$_x$ process [9], [17], [18] where only a few of the several chemical species could be measured experimentally.

The idea of solving the inverse problem described is not new. Deuflhard et al. [8], Gear [12], Edsberg [11], and Chance et al. [7] are but a few of the workers in this area. However, this is the first code to use CHEMKIN in an automatic way, to our knowledge. In the actual application, we solved the same system of ODE's or kinetics model with several different sets of initial concentrations to get each set of rate parameters. This complication does not appear in other examples we have seen. In the Exxon Thermal DeNO$_x$ process we tuned several rate constants so that predicted species concentrations fit an experimental data base which included data for a wide range of temperatures. The modeling details of the computations were further complicated by the small (short) time constants and the relatively few, but large measurement times.

In what follows, the equations for gas phase kinetics are developed in § 2. The formal problem for the computation of the reaction rate parameters is given in § 3. Section 4 outlines a constrained Gauss–Newton method for solving the formal problem.

* Received by the editors March 2, 1983, and in revised form May 16, 1983.

† Exxon Research and Engineering Company, Clinton Township, Annandale, New Jersey 08801.

‡ Present address: Sandia National Laboratories, Albuquerque, New Mexico 87185.

The code sketch is in § 5, while numerical results are given in § 6. Appendix 1 gives the divided difference scheme we used for computing gradients and Jacobians. Appendix 2 is a description of the active constraint strategy. Finally, the derivation of the termination condition for Algorithm 2 appears in Appendix 3.

2. The rate equations for elementary gas phase reactions. Elementary reversible gas phase chemical reactions can be written in the form

$$\sum_{k=1}^{K} \nu'_{k,i}\chi_k \rightleftarrows \sum_{k=1}^{K} \nu''_{k,i}\chi_k \tag{2.1}$$

for $i=1,2,\cdots,I$, where I is the total number of reactions in the mechanism, K is the number of chemical species in the mechanism, $\nu'_{k,i}$ is the stoichiometric coefficient of the kth reactant species in the ith reaction, $\nu''_{k,i}$ is the stoichiometric coefficient of the kth product species in the ith reaction, and χ_k is the chemical symbol for the kth species.

The molar concentration of the kth chemical species at time t will be denoted by $X_k(t)$ and the rate of production of the species by $\dot{X}_k(t)$. For elementary reactions of the form (2.1), the rate of production of the kth chemical species is defined by the equations

$$\dot{X}_k(t)=\sum_{i=1}^{I} \nu_{k,i}q_i \tag{2.2}$$

where

$$q_i=k_{f_i}\prod_{k=1}^{K}(X_k)^{\nu'_{k,i}}-k_{r_i}\prod_{k=1}^{K}(X_k)^{\nu''_{k,i}}$$

and $\nu_{k,i}=\nu''_{k,i}-\nu'_{k,i}$.

The forward reaction rate constant for the ith reaction, k_{f_i}, is determined by the generalized Arrhenius expression

$$k_{f_i}=A_iT^{\beta_i}\exp\left(\frac{-E_i}{RT}\right) \tag{2.3}$$

where A_i is the pre-exponential factor, β_i is the temperature exponent, and E_i the activation energy. In (2.3), R is the universal gas constant and T denotes temperature. The reverse reaction rate k_{r_i} is defined by the equation $k_{r_i}=k_{f_i}/k_{c_i}$ where k_{c_i} denotes the equilibrium constant for the ith reaction. The equilibrium constant is given by the expression

$$k_{c_i}=\exp\left(\sum_{k=1}^{K}\frac{\nu_{k,i}}{R}\left(S_k^0-\frac{H_k^0}{T}\right)\right)\left(\frac{\bar{P}}{T}\right)^{a_i}$$

where $a_i=\sum_{k=1}^{K}\nu_{k,i}$. S_k^0 and H_k^0 are the standard state entropy and enthalpy, respectively, of the kth species and $\bar{P}$ denotes atmospheric pressure.

For certain reactions a third body may be required for the reaction to proceed and this will change the form of the rate of progress variable, q_i. For example, if all species have the same third body effect in a reaction, the rate of progress variable is

$$\frac{P}{RT}q_i$$

where P denotes pressure and q_i is given by (2.2). For the form of the rate of progress variable when the reaction includes third-body effects with enhancements for certain species, we refer the reader to Kee, Miller, and Jefferson [15].

For given values of temperature, pressure, and species concentrations at time t_0, (2.2) is an initial value problem consisting of K nonlinear ordinary differential equations. Generally speaking, systems of differential equations associated with chemical kinetics models are often quite stiff, since different chemical species in a reaction mechanism may approach steady state on widely varying time scales. More precisely, if the ratio of the length of the interval of integration to the smallest time constant is large for a nonoscillatory system, the system is stiff. See the papers by Byrne and Hindmarsh [4], [5] for a discussion of stiff systems in chemical kinetics.

LSODE was selected as the integrator, because it has many desirable features—general error control, robust and efficient algebra routines, good documentation, dynamic storage allocation, and a good pedigree.

3. The minimization problem. A major consideration in chemical kinetics modeling is the choice of the forward reaction rate constants or, equivalently, the choice of the parameters A_i, β_i, and E_i in the generalized Arrhenius expression (2.3). In complex chemical mechanisms such as the Thermal DeNO$_x$ mechanism, experimental methods and theoretical methods cannot provide sufficiently accurate estimates of all the required rate constants, especially over a wide range of temperatures. If experimental data are available, these data can be used to estimate unknown rate constants and to obtain more accurate estimates of rate constants whose quoted errors are large.

Often, accurate estimates will be available for certain pre-exponentials, temperature exponents, and activation energies. We will let z denote a vector whose components are the Arrhenius parameters for which precise estimates are not known. Parameters for which accurate values are known will be held fixed while we try to determine z so that the model "best fits" the experimental data base.

For the lth set of experimental conditions, let T_l denote temperature, P_l will be used for pressure, and $X^l(t_0)$ will denote the initial concentrations of the chemical species. We use $Y^l(t)$ to denote the chemical concentrations obtained by experiment with the lth set of experimental conditions, and $X^l(t, z)$ will denote the solution of the initial value problem (2.2) at time t and for parameters z.

The residual vector f^l for the lth experiment is given by

$$f^l(z) = X^l(t^l, z) - Y^l(t^l)$$

where t^l is the residence time associated with the lth experiment. When the errors in the components of f^l are normally distributed, weighted nonlinear least squares is the most commonly used method for fitting the solution of (2.2) to available experimental data [1]. More specifically, we wish to find z that minimizes the objective function

$$\tfrac{1}{2}F^*(z)\,W_F F(z) \tag{3.1}$$

where[1]

$$F^* = ((f^1)^*, (f^2)^*, \cdots, (f^N)^*).$$

N is the total number of experiments, and W_F is the inverse of the variance–covariance matrix of the vector F or a matrix of suitable scaling factors.

We, as well as others, have observed that if a vector of parameters z minimizes (3.1), several parameters may be physically unrealistic (e.g., the wrong size or algebraic

[1] * denotes the transpose of a matrix or a vector.

sign). The experimental data may not be sufficient to correctly determine these parameters, the reaction mechanism may be inaccurate or incomplete, or the objective function may have several local minima. It is useful to know how well existing data can be fit if the parameters are subject to physically realistic constraints. Instead of (3.1), we consider the constrained nonlinear least squares problem

$$\min_z \tfrac{1}{2}F^*(z)F(z), \tag{3.2}$$

or equivalently,

$$\min_z \tfrac{1}{2}\sum_{l=1}^{N}\sum_{j=1}^{\bar{K}}(X_j^l(t^l, z) - Y_j^l(t^l))^2,$$

subject to the constraints, $a_i \leqq z^*e_i \leqq b_i$ for $i = 1, 2, \cdots, L$ where L is the number of parameters selected for optimization. $\bar{K}$ denotes the number of species in each experiment for which experimentally measured values are available. In the inequalities in problem (3.2), e_i is a unit vector with a one in the ith component (and zero elsewhere) and a_i and b_i are, respectively, the lower and upper bounds for the ith parameter.

We note that if the matrix W_F is a positive definite and symmetric matrix, there is no loss of generality in rewriting the objective function in (3.1) in the form that it appears in (3.2). Unfortunately, in practice experiments are not often repeated; hence, an accurate estimate of the variance-covariance matrix is seldom available.

4. The parameter estimation procedure. In choosing a parameter estimation procedure, we knew that the program would be used to find parameters in the following situations. First, measurements would be available for only a few of the K chemical species in the mechanism and only at several different residence times. Second, the chemical mechanisms would be moderately sized—30 to 40 reactions and involving 20 to 30 species. Third, the number of experiments in a given data base would also be large—usually around 80 experiments. As many as 80 systems of 20–30 stiff ODE's would need to be solved for each evaluation of the objective function or least squares residual function. Consequently, evaluations of $F(z)$ would be expensive.

For solving problem (3.2) we chose a Gauss–Newton method with a so-called active constraint strategy [13] which we present in Appendix 2. This is, of course, a standard method for solving nonlinear least squares problems subject to linear inequality constraints. However, if the method is to be effective in the current application, several details of implementation require careful attention.

Let $\phi(z)$ denote the objective function to be minimized in problem (3.2). Most methods for minimizing ϕ are of the form

$$\bar{z} = z - \alpha B^{-1}\nabla\phi(z) \tag{4.1}$$

where $\nabla\phi$ is the gradient of ϕ, B is an approximation to $\nabla^2\phi$, the second derivative or Hessian of $\phi(\nabla^2\phi = [\partial^2\phi/\partial z_i\, \partial z_j])$, and α is a positive number chosen to insure that $\phi(\bar{z}) < \phi(z)$. For problem (3.2),

$$\nabla\phi = J_F^*F \tag{4.2}$$

and

$$\nabla^2\phi = J_F^*J_F + \sum_{i=1}^{N}\sum_{j=1}^{\bar{K}} f_j^i\nabla^2 f_j^i \tag{4.3}$$

where J_F is the Jacobian of the residual vector F, and f_j^i is the jth component of the residual vector for the ith experiment. If the double summation term in (4.3) is dropped, and the approximation $B = J_F^* J_F$ is used in iteration (4.1) along with (4.2), the iterative scheme (4.1) is the Gauss–Newton method.

The dimensions of the matrix J_F are $N \cdot \bar{K}$ by L where $\bar{K}$ is the number of chemical species for which experimental values of the concentrations are available. In order to avoid excessive storage requirements when large experimental data bases are to be fit, we note that

$$J_F^* J_F = \sum_{i=1}^{N} J_{f^i}^* J_{f^i} \tag{4.4}$$

and

$$J_F^* F = \sum_{i=1}^{N} J_{f^i}^* f^i \tag{4.5}$$

where J_{f^i} is the Jacobian of the residual vector for the ith experiment. If (4.4) and (4.5) are used to compute the gradient and Hessian, a vector of length L and an L by L array are the storage needed for the derivative information. Since computation of J_F is very costly, it is interesting to note that terms in (4.4) and (4.5) could be computed simultaneously if several processors were available.

At each iteration the value of α is usually determined by solving the one-dimensional minimization problem,

$$\min_{\alpha>0} \phi(z - \alpha B^{-1} \nabla \phi(z)).$$

See, for example, Ortega and Rheinboldt [20, Chap. 8]. Although this method of calculating α is an important safeguard in general minimization packages [6], [10], [19], it can in certain problems add considerably to the number of function evaluations required in the parameter estimation. For the reaction mechanisms we examined, $\alpha = 1$ gave good results.

On the other hand, in the current application the accurate approximation of J_{f^i} is critical to the success of the Gauss–Newton method. We use the standard forward difference approximation

$$(\hat{J}_{f^i}) e_l = \frac{f^i(z + (\Delta^* e_l) e_l) - f^i(z)}{\Delta^* e_l} \tag{4.6}$$

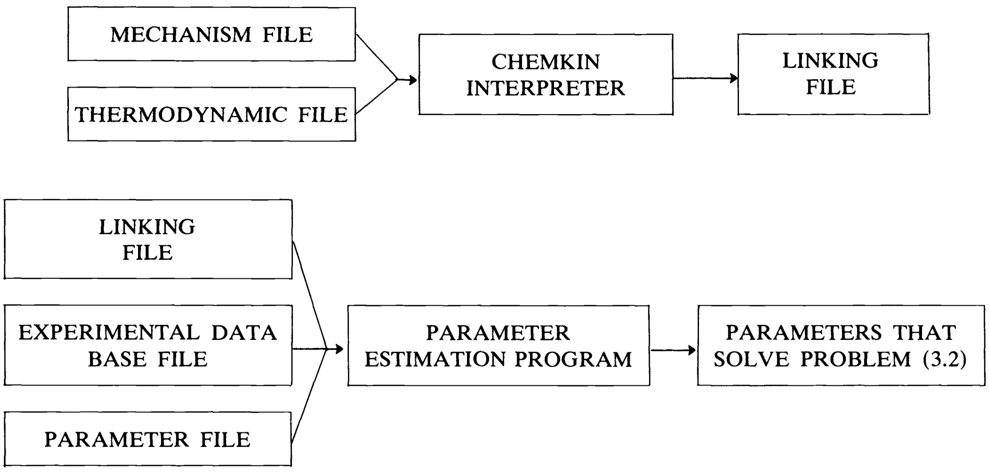
where Δ is a vector of small positive numbers and $\hat{J}_{f^i}$ denotes the difference approximation to J_{f^i}. The program we have developed uses the algorithm in Appendix 1 to compute the vector Δ. In the current program, Algorithm 1 provides a way of relating the accuracy of the underlying stiff ODE solver LSODE, and the choice of step size in the forward difference approximation (4.6). The key is the recognition of the level of the inherent error in the ODE solution and hence the concentration levels. This error level must be accommodated in the differencing scheme in the Gauss–Newton method.

An active constraint method was selected for solving problem (3.2) because such a method forces all approximations to the optimal parameters to remain within the physically realistic region. Also, in many problems a minimum exists within the constrained region. In these cases, if a starting solution is sufficiently close to the optimal solution, the constrained Gauss–Newton algorithm presented in Appendix 2 reduces to an unconstrained Gauss–Newton method and the additional costs of a constrained optimization procedure are avoided. Furthermore, an active constraint algorithm allows the user to make very minimal assumptions concerning the prior distribution of the parameters to be estimated.

5. A CHEMKIN-based program for analyzing gas phase kinetics. Fig. 5.1 gives the basic outline of the program we have developed. The elements, chemical species, and elementary reactions that govern the reaction are specified in the MECHANISM FILE. For each reaction, the corresponding Arrhenius parameters must also be given by the user if they have not been selected for optimization.

The THERMODYNAMIC FILE contains the information required to compute the entropy and enthalpy for each species in the mechanism. Our program uses the thermodynamic data base provided by CHEMKIN. This data base is known to be adequate for a large class of gas phase reactions. The THERMODYNAMIC FILE and the MECHANISM FILE are the same format as in other CHEMKIN applications; for further details see the CHEMKIN user's guide.

The MECHANISM FILE and THERMODYNAMIC FILE are read by the CHEMKIN INTERPRETER. The stoichiometric information in these files is assembled in the LINKING FILE. CHEMKIN routines are used in the FORTRAN program to read the stoichiometric information in the LINKING FILE into work arrays in the FORTRAN program.


MECHANISM FILE
THERMODYNAMIC FILE
CHEMKIN INTERPRETER
LINKING FILE
LINKING FILE
EXPERIMENTAL DATA BASE FILE
PARAMETER FILE
PARAMETER ESTIMATION PROGRAM
PARAMETERS THAT SOLVE PROBLEM (3.2)

Parameter Estimation Program includes:

CHEMKIN ROUTINES—To read the BINARY FILE which contains the stoichiometry for the mechanism and to evaluate the rate equations.
Algorithm 2—To perform the constrained optimization.
LSODE—To solve the rate equations.

FIG. 5.1. CHEMKIN-*based program for fitting rate constants.*

In the PARAMETER FILE the user indicates the reactions involving Arrhenius parameters to be optimized, and specifies which parameters in a particular reaction are to be optimized. The user provides initial estimates for these parameters as well as upper and lower bounds. In the EXPERIMENTAL DATA FILE the user provides the data base of experiments to be fit. Each experiment consists of a set of initial conditions (i.e., temperature, pressure, and initial concentrations) and the corresponding species concentrations at some residence time.

We have added a routine to the CHEMKIN SUBROUTINE Library which computes $X^l(t^l, z)$. In other words, for the lth set of initial conditions, the routine computes the species concentrations as a function of time and the parameters specified by the user for optimization. The CHEMKIN subroutine, CKRAT, is used in the subroutine that calculates $X^l(t^l, z)$. CKRAT evaluates the rate equations (2.2) and LSODE is used to solve them. The constrained optimizer, Algorithm 2, repeatedly calls $X^l(t^l, z)$ when the algorithm requires residual function values. The final program output is a set of parameters that solve problem (3.2) and an evaluation of the model, with the optimal parameters, for each set of initial conditions.

Note that the reaction mechanism can be changed and an optimization can be performed to try to fit the new mechanism to the data base. No coding changes are required. The following section shows examples of several reaction mechanisms in which certain Arrhenius parameters were determined by our program.

6. Test examples. Example 1 shows the program being used to determine parameters in a well-behaved reaction mechanism where precise data is used in the fitting. This "controlled" example is presented to illustrate the behavior of the numerical algorithms in the program. Example 2 is more typical of the problems encountered in practice. Here the program is used to determine parameters in a large reaction mechanism where noise is present in the data to be fit, and measured values are available for only several of the species in the mechanism.

Example 1. *Hydrogen burn reaction.*

In this example the program is used to recover the pre-exponentials in reactions 2, 3, 4, and 5 of the hydrogen burn reaction mechanism shown in Table 6.1. We generated the experimental data to be fit by evaluating the reaction mechanism at residence time .05 seconds, the Arrhenius parameters given in Tables 6.1 and 6.2, and the following nonzero initial conditions: $H_2 = .0002$, $O_2 = .02$, $N_2 = .97988$, $P = 1$ atm, and $T = 1000$ K (initial species values are in mole fractions). In Table 6.2 we show

TABLE 6.1
MECHANISM FILE *for hydrogen burning reaction.*

ELEMENTS			
H O N			
END			
SPECIES			
H_2 O_2 N_2 H O OH HO_2 H_2O			
END			
REACTIONS			
1 $H_2+O_2=2OH$	1.7×10^{13}	0.0	47,780.0
2 $H_2+OH=H_2O+H$	···	0.0	6,500.0
3 $H+O_2=OH+O$	···	−0.907	16,620.0
4 $O+H_2=OH+H$	···	1.0	8,826.0
5 $H+O_2+M=HO_2+M$ $H_2O/20./$†	···	0.0	−870.0
6 $OH+HO_2=H_2O+O_2$	1.2×10^{13}	0.0	0.0
7 $H+HO_2=2OH$	6.0×10^{13}	0.0	0.0
8 $O+HO_2=O_2+OH$	1.0×10^{13}	0.0	0.0
9 $2OH=O+H_2O$	1.7×10^{6}	2.03	−1,190.0
10 $H_2+M=2H+M$ $H_2O/5/$	2.23×10^{12}	0.500	92,600.0
11 $O_2+M=2O+M$	1.85×10^{11}	0.500	95,560.0
12 $H+OH+M=H_2O+M$ $H_2O/20.0/$	7.5×10^{23}	−2.6	0.0
13 $H+HO_2=H_2+O_2$	1.3×10^{13}	0.0	0.0
END			

† This indicates a third-body enhancement for the reaction (see [15]).

TABLE 6.2
Trace of the algorithm for the unconstrained problem.

Iteration	Objective function	Gradient norm ($\|\cdot\|_\infty$)
1	2.90	5.60
2	.250	.689
3	.290	3.18
4	$.977\times10^{-2}$	.332
5	$.279\times10^{-4}$	$.150\times10^{-1}$
6	$.305\times10^{-8}$	$.151\times10^{-3}$

	Initial parameter approximation	Exact parameters†	Computed parameters
z_1	$.20\times10^{18}$	$.122\times10^{18}$	$.1220025\times10^{18}$
z_2	$.10\times10^{11}$	$.180\times10^{11}$	$.1800007\times10^{11}$
z_3	$.32\times10^{14}$	$.520\times10^{14}$	$.5200005\times10^{14}$
z_4	$.30\times10^{16}$	$.200\times10^{16}$	$.2000014\times10^{16}$

† Parameters used in reactions 3, 4, 2, and 5, respectively, to generate experimental data.

TABLE 6.3
Fit to data.

Species	Computed values of species	|Computed–experimental|†
H_2	$.343333\times10^{-4}$	1.5×10^{-9}
O_2	$.199168\times10^{-1}$	3.0×10^{-8}
N_2	$.979880\times10^{-1}$	3.5×10^{-7}
H	$.498736\times10^{-7}$	2.6×10^{-12}
O	$.177672\times10^{-5}$	2.9×10^{-11}
OH	$.889328\times10^{-6}$	1.9×10^{-11}
HO_2	$.124121\times10^{-5}$	3.4×10^{-11}
H_2O	$.164593\times10^{-3}$	1.9×10^{-9}

† The values in this column are the absolute values of the differences between the values for the given species and the values of the species predicted by the model using the optimal parameters.

the performance of the algorithm, and in Table 6.3 we compare the fit to the given data. The objective function we used is given by (3.1) with W_F set to a diagonal matrix whose nonzero entries are the reciprocals of the concentrations of the species to be fit.

For each iteration the value of Δ used in (4.6) is $10^{-3}\times z$. Algorithm 1 was used to compute r at the initial parameter approximation, and this value was used throughout the algorithm. In LSODE, the absolute error tolerance was set to 10^{-15} and the relative error tolerance to 10^{-4} (see [14]).

In Table 6.4, we show the results of an optimization where z_3 and z_4 are subject to the constraints, $z_3 \leqq .48\times10^{14}$ and $z_4 \geqq .25\times10^{16}$. All other initial conditions and values of parameters controlling the optimization algorithm are the same as in the unconstrained problem.

Example 2. *Nitrogen chemistry reaction.*

The program has been used successfully to fit a realistic nitrogen chemistry mechanism to an extensive experimental data base. Table 6.5 gives the mechanism for

TABLE 6.4
Trace of the algorithm for the constrained problem.

Iteration	Active constraints	Objective function	Gradient norm ($\|\cdot\|_\infty$) (active variables only)
1	None	2.906543	4.186
2	None	.249895	6.89×10^{-1}
3	z_4 (lower)	.110626	4.51×10^{-1}
4	z_4 (lower) z_3 (upper)	$.37371\times10^{-1}$	1.46×10^{-1}
5	z_4 (lower) z_3 (upper)	$.265896\times10^{-1}$	1.23×10^{-2}
6	z_4 (lower) z_3 (upper)	$.26368\times10^{-1}$	2.17×10^{-3}
7	z_4 (lower) z_3 (upper)	$.26356\times10^{-1}$	8.89×10^{-4}

	Final parameters	Final gradient
z_1	$.196939\times10^{18}$	8.89×10^{-4}
z_2	$.158350\times10^{11}$	8.76×10^{-5}
z_3	$.480\times10^{14}$	-6.53×10^{-3}
z_4	$.250\times10^{16}$	8.20×10^{-1}

TABLE 6.5
MECHANISM FILE *for reduced nitrogen chemistry.*

ELEMENTS
H O N HE
END

SPECIES
NO NH_3 O_2 H_2O HE NO_2 H OH N_2 HO_2 NH_2 NH HNO O
END

REACTIONS

1	$NH_3+O=NH_2+OH$	$.1500\times10^{13}$	0.0	6,040.0
2	$NH_3+OH=NH_2+H_2O$	$.3260\times10^{13}$	0.0	2,120.0
3	$HNO+M=NO+H+M$	$.1860\times10^{17}$	0.0	48,680.0
4	$HNO+OH=NO+H_2O$	$.3600\times10^{14}$	0.0	0.0
5	$NH_2+NO=N_2+H+OH$	$.6000\times10^{20}$	−2.46	1,866.0
6	$NH_2+O=NH+OH$	$.1546\times10^{14}$	0.0	1,000.0
7	$NH_2+OH=NH+H_2O$	$.6000\times10^{11}$	0.68	1,290.0
8	$NH+O_2=HNO+O$	$.3000\times10^{14}$	0.0	3,400.0
9	$H+O_2=OH+O$	$.2200\times10^{15}$	0.0	16,800.0
10	$O+HO_2=O_2+OH$	$.4800\times10^{14}$	0.0	1,000.0
11	$OH+HO_2=H_2O+O_2$	$.5000\times10^{14}$	0.0	1,000.0
12	$OH+OH=O+H_2O$	$.6300\times10^{13}$	0.0	1,090.0
13	$HO_2+NO=NO_2+OH$	$.3430\times10^{13}$	0.0	−260.0
14	$O+NO_2=NO+O_2$	$.1000\times10^{14}$	0.0	600.0
15	$H+O_2+M=HO_2+M$ $H_2O/21./$	$.1500\times10^{16}$	0.0	−995.0
16	$NH_3+M=NH_2+H+M$	$.4800\times10^{17}$	0.0	93,929.0
17	$NH_2+NO=N_2+H_2O$	$.9000\times10^{20}$	−2.46	1,866.0

END

a simplified nitrogen chemistry model. Although the reactions and rate constants are all physically plausible, the mechanism, as a whole, only describes nitrogen chemistry in a qualitative manner.

Using the mechanism in Table 6.5, we have manufactured the exact data base shown in Table 6.6. The data base is comprised of 15 constant pressure, constant temperature "experiments," which could be performed in a flow tube reactor. In all experiments, 5 of the 14 species are initially nonzero. After the indicated residence

TABLE 6.6
Example data base 1—no scatter.

Initial nonzero species concentrations (mole fractions):

NO = 0.00025; NH_3 = 0.0004; O_2 = 0.04; H_2O = 0.1; He = 0.85935.

Pressure = 1.1 atm.

Temperature (K)	Residence times (s)	NO	NH_3
1,000.0	0.5	0.0002379	0.0003903
1,000.0	1.0	0.0001432	0.0003139
1,000.0	1.5	0.0001181	0.0002937
1,000.0	2.0	0.0001096	0.0002868
1,000.0	2.5	0.0001057	0.0002837
1,200.0	0.5	0.0000533	0.0001360
1,200.0	1.0	0.0000309	0.0001009
1,200.0	1.5	0.0000218	0.0000851
1,200.0	2.0	0.0000168	0.0000751
1,200.0	2.5	0.0000136	0.0000680
1,400.0	0.5	0.0001842	0.0000012
1,400.0	1.0	0.0001837	0.0000000
1,400.0	1.5	0.0001836	0.0000000
1,400.0	2.0	0.0001836	0.0000000
1,400.0	2.5	0.0001836	0.0000000

TABLE 6.7
Example data base 2—with 1.8% *scatter.*

Initial nonzero species concentrations (mole fractions):

NO = 0.00025; NH_3 = 0.0004; O_2 = 0.04; H_2O = 0.1; He = 0.85935.

Pressure = 1.1 atm.

Temperature (K)	Residence time (s)	NO	NH_3
1,000.0	0.5	0.0002378	0.0003851
1,000.0	1.0	0.0001437	0.0003178
1,000.0	1.5	0.0001188	0.0002955
1,000.0	2.0	0.0001095	0.0002829
1,000.0	2.5	0.0001068	0.0002898
1,200.0	0.5	0.0000543	0.0001379
1,200.0	1.0	0.0000312	0.0001014
1,200.0	1.5	0.0000219	0.0000846
1,200.0	2.0	0.0000167	0.0000757
1,200.0	2.5	0.0000137	0.0000695
1,400.0	0.5	0.0001851	0.0000013
1,400.0	1.0	0.0001824	0.0000000
1,400.0	1.5	0.0001821	0.0000000
1,400.0	2.0	0.0001827	0.0000000
1,400.0	2.5	0.0001847	0.0000000

times, 2 of the 14 species are measured. In Table 6.7 we show the same data base with 1.8% r.m.s. scatter introduced in the output concentrations using a normally distributed random number generator.

Table 6.8 summarizes the results of the optimization algorithm when applied to these data bases. In both cases, three pre-exponentials were significantly perturbed in the mechanism of Table 6.5, resulting in large initial objective functions. (The objective function is defined by (3.1), where we choose W_F to be a diagonal matrix, each diagonal element being the inverse of the corresponding species concentration at time zero.) In both cases, the algorithm was unconstrained and convergence occurred after a few iterations.

TABLE 6.8
Summary of unconstrained optimization examples.

Data base	Iteration	Pre-exponential parameters Reaction number			Objective function
		7	11	17	
1	0	$.5000\times10^{11}$	$.7000\times10^{14}$	$.8000\times10^{20}$	$.5030\times10^{00}$
1	12	$.5920\times10^{11}$	$.5918\times10^{14}$	$.8876\times10^{20}$	$.8229\times10^{-5}$
2	0	$.5000\times10^{11}$	$.7000\times10^{14}$	$.8000\times10^{20}$	$.4956\times10^{00}$
2	5	$.5866\times10^{11}$	$.5851\times10^{14}$	$.8795\times10^{20}$	$.3859\times10^{-3}$
1	0	$.5866\times10^{11}$	$.5851\times10^{14}$	$.8795\times10^{20}$	$.6411\times10^{-5}$

In fitting the exact data, the three pre-exponentials returned to within 1.5% of their exact values, while the ratios of one to another returned to within 0.05% of their exact values. The objective function is insensitive to changes which correspond to multiplication of the three pre-exponentials by an overall constant, while it is sensitive to changes in their ratios. The same behavior occurs with the scattered data. We attempted an example with 5% scatter in the data and found that the algorithm did not converge with the three parameters; however, it converged with any two of the three. Since the objective function is so flat in one direction of the three-parameter space, we lose the minimum when there is only a moderate amount of scatter in the data. In order to obtain convergence with the three parameters when there is larger scatter, we must increase the size of the data base.

In Table 6.8, we show the objective function on the exact data using the model obtained with the scattered data. This objective function is as good as the one obtained in fitting the exact data. This indicates that noise in the data can be filtered out even in a case where the objective function is not that well behaved.

7. Concluding remarks. We have presented the outline of a general program for analyzing gas phase chemical kinetics when a large data base of experiments is available. The program used a constrained Gauss–Newton procedure that takes advantage of the structure of the objective function and the fact that the constraints are simple interval constraints. We have focused in this report on several important details—the storage efficient implementation and a method of choosing a differencing factor for computing sensitivities which realizes the underlying error in the numerical solution of the ODES that govern the reaction mechanism. The flexibility of the chemical kinetics package, CHEMKIN, is also illustrated.

The program was initially developed to fit parameters in a mechanism where chemical species concentrations were available for only several of the responses in the model. However, the number of experiments in the data base was quite large. When a large amount of significant data is available, weighted least squares is an appropriate fitting procedure. In other cases, an effective fitting procedure should take into consideration the fact that errors in different components of each residual f^l may be correlated. A priori knowledge of the parameters should be utilized in determining rate constants. Parameter fitting procedures such as those presented in Box and Draper [2], Box et al. [3], and Stewart [22] should be used instead of nonlinear least squares. We are currently adding these estimators to the program so the user has a choice of fitting procedures.

Parameter estimators are available for cases where the variance–covariance matrix is unknown and the errors in the residuals of a given experiment may be correlated (see, for example, Stewart [22]). These estimators, however, require the minimization of an objective function which is not least squares. Although there is now excellent software [6] available for handling such problems, the general optimization problem is more difficult. When the objective function is given by nonlinear least squares, for many problems $J_F^* J_F$ is an accurate Hessian approximation; hence, the Gauss–Newton method for nonlinear least squares is usually quite effective. Furthermore, the Hessian approximation in the Gauss–Newton method can be augmented with a quasi-Newton correction (see Dennis et al. [10]) to produce an even better algorithm. It should be noted that when a nonlinear least squares objective function is used, fitting a large data base may be practical. This may not be the case if a different objective function is used. In realistic problems where experimentally measured values are not available for all the chemical species in the model, it is usually not possible to make use of a system of stiff ODEs (see the papers by Varah [23] and Yermakova et al. [25]).

Appendix 1.

ALGORITHM 1. Computation of a difference parameter.

```
Step 1. Choose the set of initial conditions from the experimental data base which
        includes the smallest residence time $t$. Let $z$ be the initial estimate of the
        parameters selected for optimization.
        Set $r = 0$.
Step 2. $r = r + 1$.
        For $l = 1, 2, \cdots, L$,
            $\Delta_l^r = (z^* e_l)(.1)^r$.
        $\Delta^r = (\Delta_1^r, \cdots, \Delta_L^r)^*$.
        For $l = 1, 2, \cdots, L$,
            $\Lambda^r e_l = \frac{X(z + \Delta^r e_l) - X(z)}{\Delta_l^r}$.
Step 3. If $r \leqq 2$ go to step 2.
        If for some $l$, $1 \leqq l \leqq L$,
            $\|\Lambda^{r+1} e_l - \Lambda^r e_l\| > \|\Lambda^r e_l - \Lambda^{r-1} e_l\|$,
        then set $\Delta = (\Delta_1^{r-1}, \Delta_2^{r-1}, \cdots, \Delta_L^{r-1})^*$ and stop.
Step 4. Go to step 2.
```

In Step 1 of the algorithm, the smallest residence time is chosen, because at this time it is likely that many of the species concentrations will not yet have reached steady state, and instabilities in the differencing scheme are most likely to occur. Initial values from several experiments and different choices of z could be used to determine a suitable value for r, but we found no significant advantage to this approach. Schemes such as Algorithm 1 are being used to check the accuracy and reliability of command and control software [24].

Appendix 2.

ACTIVE CONSTRAINT ALGORITHM. In an active constraint method, lists of those parameters that are equal to their upper or lower constraints are maintained and updated at each iteration. We use Γ_k^u to denote the set of indices of those parameters that are equal to their upper bounds at the kth iteration and Γ_k^l will be the set of indices of those parameters that have attained their lower bounds.

Let z_k be an approximation to the solution of problem (3.3) and assume z_k satisfies the constraints. When a new approximation, $z_k - \lambda p$, is computed, λ is chosen so that

(1) $a_i \leqq (z_k - \lambda p)^* e_i \leqq b_i$

for $i = 1, 2, \cdots, L$. If at the kth iteration,

(2a) $(z_k - \lambda p)^* e_i = b_i$

then $\Gamma_{k+1}^u = \Gamma_k^u \cup \{i\}$;
if

(2b) $(z_k - \lambda p)^* e_i = a_i$

then $\Gamma_{k+1}^l = \Gamma_k^l \cup \{i\}$.

The following rule is used to determine if constraints should be excluded from the new active sets. For any index i, if

(3a) $F^* J_F(z_k) e_i > 0$ and $i \in \Gamma_k^u$

then $\Gamma_{k+1}^u = \Gamma_k^u - \{i\}$;
if

(3b) $F^* J_F(z_k) e_i < 0$ and $i \in \Gamma_k^l$

then $\Gamma_{k+1}^l = \Gamma_k^l - \{i\}$.

The conditions in rule (3) indicate a new value for $z_k^* e_i$ can be found which decreases the objective function and satisfies constraints. A similar criterion for dropping constraints can be found in Powell [21].

The algorithm can terminate successfully in either of two ways. A local minimum of the objective function can be found within the constraint set. This situation can occur if

(4a) $\|J_F^* F(z_k)\|_\infty < \text{TOL}$

and

(4b) $a_i < z_k^* e_i < b_i$

for $i = 1, 2, \cdots, L$. In condition (4a), TOL is a small positive number.

The other possibility is that the local minimum is on the boundary of the constraint set. In this case some constraints are active and this situation is determined by the following check. For each i, $i = 1, 2, \cdots, L$, determine if one of the following three conditions holds:

(5a) $i \in \Gamma_k^u$ and $F^* J_F(z_k) e_i < 0$

or

(5b) $i \in \Gamma_k^l$ and $F^* J_F(z_k) e_i > 0$

or

(5c) $|F^* J_F(z_k) e_i| < \mathrm{TOL}$.

If for each component of z_k, one of the conditions (5a), (5b), or (5c) is satisfied, then the vector z_k cannot be perturbed in any way that significantly decreases the objective function and satisfies constraints. For a derivation of these conditions, see Appendix 3. We now give the Gauss–Newton algorithm which uses this active constraint strategy.

ALGORITHM 2. Constrained Gauss–Newton method.

Step 1. Choose z_0 such that

$$a_i \leqq z_0^* e_i \leqq b_i$$

for $i = 1, 2, \cdots, L$. Let

$$\Gamma_0^l = \{i\colon a_i = z_0^* e_i\}$$

and $\Gamma_0^u = \{i\colon b_i = z_0^* e_i\}$.

Repeat Steps 2–5, for $k = 0, \cdots, \mathrm{MAX}$ where MAX denotes the maximum number of iterations allowed.

Step 2. Use Algorithm 1 to find a suitable differencing factor Δ for computing required derivatives at step k.

Step 3. Compute $\hat{J}_F^* \hat{J}_F(z_k)$ and $\hat{J}_F^* F(z_k)$ according to (4.4), (4.5), and (4.6).

Step 4. (i) Apply condition (4) to determine if an unconstrained minimum was located. If (4) is satisfied, stop.

(ii) Apply the check (5) to determine if a constrained minimum was located. If satisfied, stop.

(iii) Check conditions (3a) and (3b) to determine if any constraints can be deleted from active sets.

Step 5. Compute the solution p_k of the system

$$\hat{J}_F^* \hat{J}_F p_k = \hat{J}_F^* F,$$

and set

$$z_{k+1} = z_k - \lambda p_k$$

where λ is chosen so that condition (1) is satisfied for $i = 1, 2, \cdots, L$. Choose the new active sets Γ_{k+1}^l and Γ_{k+1}^u according to rule (2).

Appendix 3. In this appendix we show that if Algorithm 2 reaches a point z such that conditions (5a), (5b), and (5c) of Appendix 2 are satisfied with $\mathrm{TOL} = 0$, then the algorithm has found a constrained local minimum for problem (3.2). Throughout this section f will denote a twice continuously differentiable function with domain in R^n and range in R. Further, $\nabla^2 f$ will be assumed to be positive definite.

The following theorem concerning the Kuhn–Tucker conditions is given in Powell [21]. This theorem provides conditions for the solution of the following problem:

(1) $\min\limits_z f(z)$

subject to the constraints

(2) $v_i^* z \geqq s_i$ for $i = 1, 2, \cdots, L$

where $v_i \in R^L$ and $s_i \in R$ for $i = 1, 2, \cdots, L$.

THEOREM 1. *Let z satisfy* (2). *Let only the first $n \leqq L$ of the constraints be satisfied as equalities. Then z is a strong local minimum of f if and only if*

(3) $\nabla f(z) = \sum_{i=1}^n \lambda_i v_i$, $\lambda_i \geqq 0$.

To show that z is a constrained local minimum if (5a), (5b), and (5c) of Appendix 2 hold, we assume, without loss of generality, that Algorithm 2 has reached a z such that

$$z^* e_i = a_i \quad \text{for } i = 1, 2, \cdots, m,$$

$$z^* e_i = b_i \quad \text{for } i = m+1, \cdots, n,$$

$$a_i < z_i^* e_i < b_i \quad \text{for } i = n+1, \cdots, L.$$

Suppose the termination conditions hold. In this case,

(4) $\nabla f(z)^* e_i \geqq 0$ for $i = 1, 2, \cdots, m$,

(5) $\nabla f(z)^* e_i \leqq 0$ for $i = m+1, \cdots, n$,
$\nabla f(z)^* e_i = 0$ for $n+1, \cdots, L$.

Let $\bar{e}_i = -e_i$; note that

(6) $\nabla f(z) = \sum_{i=1}^m (e_i^* \nabla f(z)) e_i + \sum_{i=m+1}^n (\bar{e}_i^* \nabla f(z)) \bar{e}_i$.

From equation (6), conditions (4) and (5), and Theorem 1, we see that z is a solution to the problem

$$\min_z f(z)$$

such that

$$z^* e_i \geqq a_i \quad \text{for } i = 1, 2, \cdots, m,$$

and

$$z^* e_i \leqq b_i \quad \text{for } i = 1, 2, \cdots, n.$$

The conditions (5a), (5b), and (5c) of Appendix 2 guarantee that if the Hessian of the objective function in problem (3.2) is positive definite at z, then there exists no local descent direction which satisfies the constraints in problem (3.2). We note that the termination conditions indicate a constrained local minimum even in certain cases where $\nabla^2 f(z)$ is not positive definite (see Powell [21, Thm. 1.5]).

Acknowledgments. We thank Dr. R. J. Kee of Sandia National Laboratories for several useful conversations which enabled us to make the necessary modifications to subroutines in the CHEMKIN package. We are grateful to Mrs. B. S. Nemerofsky who did an excellent job typing the manuscript. Finally, we appreciate the thoughtful and expeditious comments of the referees.

REFERENCES

[1] Y. BARD, *Nonlinear Parameter Estimation*, Academic Press, New York, San Francisco, and London, 1974.

[2] G. E. P. BOX AND N. R. DRAPER, *The Bayesian estimation of common parameters from several responses*, Biometrika, 52 (1965), pp. 355–365.

[3] G. E. P. BOX, W. G. HUNTER, J. F. MACGREGOR AND J. ERJAVEC, *Some problems associated with the analysis of multiresponse data*, Technometrics, 15 (1973), pp. 33–51.

[4] G. D. BYRNE, *Software for differential systems and applications involving chemical kinetics*, Computers and Chemistry, 5 (1981), pp. 151–158.

[5] G. D. BYRNE AND A. C. HINDMARSH, *Numerical Solution of Stiff Ordinary Differential Equations*, AICHE Today Series, American Institute of Chemical Engineers, New York, 1977.

[6] R. L. CRANE, R. E. HILLSTROM AND M. MINKOFF, *Solution of the general nonlinear programming problem with subroutine* VMCON, Argonne National Laboratory Report ANL-80-64, Argonne, IL, 1980.

[7] E. M. CHANCE, A. R. CURTIS, I. P. JONES AND C. R. KIRBY, FACSIMILE: *A computer program for flow and chemistry simulation and general initial value problems*, Report AERE-B8775, AERE Harwell, 1977.

[8] P. DEUFLHARD, G. BARDER AND U. NOWAK, LARKIN—*A software package for the numerical simulation of large systems arising in chemical kinetics*, Preprint 100, Institut für Angewandte Mathematik, University of Heidelberg, Nov. 1980.

[9] A. J. DEGREGORIA AND A. M. DEAN, *A detailed kinetic model of the thermal* DeNO$_x$ *process*, CR-SL Report CR.5EB.82, 1982.

[10] J. E. DENNIS, D. M. GAY AND R. E. WELSCH, *An adaptive nonlinear least squares algorithm*, ACM Trans. Math. Softw., 7 (1982), pp. 348–368.

[11] EDSBERG, KEMPEX II, *A Program Package for Interactive Simulation of Some Chemical Reactors*, Report TRITA-NA-7504, Dept. Numerical Analysis and Computing Science, The Royal Institute of Technology, Stockholm, 1975.

[12] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[13] P. E. GILL, W. MURRAY AND M. H. WRIGHT, *Practical Optimization*, Academic Press, New York, 1981.

[14] A. C. HINDMARSH, LSODE *and* LSODEI, *two new initial value ordinary differential equation solvers*, ACM Signum Newsletter, 15 (1980), pp. 10–11.

[15] R. J. KEE, J. A. MILLER AND T. H. JEFFERSON, CHEMKIN: *A general-purpose, problem-independent, transportable, Fortran chemical kinetics code package*, Sandia National Laboratories, Report SAND80-8003, Livermore, CA, 1980.

[16] A. C. HINDMARSH, ODE *solvers for use with the method of lines*, in Advances in Computer Methods for Partial Differential Equations IV, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, Dept. Computer Science, Rutgers University, New Brunswick, NJ, 1981.

[17] R. K. LYON, *The* NH_3–NO–O_2 *reaction*, Int. J. Chem. Kin., 8 (1976), p. 315.

[18] J. A. MILLER, *Current status in the development of a chemical kinetic model for the selective reduction of nitric oxide by ammonia*, presented at the Combustion 1981 Fall Technical Meeting/Eastern States Section.

[19] JORGE J. MORÉ, BURTON S. GARBOW AND KENNETH E. HILLSTROM, *User Guide for Minpack*-1, Argonne National Laboratory, Report ANL-80-74, Argonne, IL, 1980.

[20] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solutions of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[21] M. J. D. POWELL, *Introduction to constrained optimization*, in Numerical Methods for Constrained Optimization, P. E. Gill and W. Murray, eds., Academic Press, London and New York, 1974, Chapter 1.

[22] W. E. STEWART, *Computer-aided modeling of reaction networks*, in Foundations of Computer-Aided Process Design, Vol. II, 1981, Engineering Foundation, New York, pp. 335–366.

[23] J. M. VARAH, *A spline least squares method for numerical parameter estimation in differential equations*, this Journal, 3 (1982), pp. 28–47.

[24] M. L. WATKINS, *A technique for testing command and control software*, Comm. ACM, 25 (1982), pp. 228–232.

[25] A. YERMAKOVA, S. VAJDA AND P. VALKO, *Direct integral method via spline-approximation for estimating rate constants*, Applied Catalysis, 2 (1982), pp. 139–154.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

COMPUTING A FEW EIGENVALUES AND EIGENVECTORS OF A SYMMETRIC BAND MATRIX*

DAVID S. SCOTT†

Abstract. This paper presents a new combination of the bisection algorithm and the Rayleigh quotient iteration for computing a few eigenvalues of a symmetric band matrix. Both global and local convergence results are proved and numerical examples are presented. The modification of the algorithm needed to handle generalized eigenvalue problems is described.

Key words. eigenvalues, eigenvectors, symmetric band matrices, Rayleigh quotient iteration

1. Introduction. This paper describes a new algorithm for computing a few specified eigenvalues and eigenvectors of a symmetric band matrix. The algorithm was developed for use in conjunction with the symmetric block (or band) Lanczos algorithm. At each step of the Lanczos algorithm it is necessary to compute a few eigenvalues (and eigenvectors) of a symmetric band matrix. The eigenvalues are specified by index and an approximate eigenvector is available which is often accurate, although it can occasionally be completely wrong.

The new algorithm can compute eigenvalues by index or by location (i.e. all the eigenvalues between zero and one) and can take advantage of approximate eigenvectors if they are available. The local and global convergence rates of the algorithm are given. Some numerical tests are described. Finally the changes necessary to make the algorithm applicable to banded generalized eigenvalue problems are described.

2. The algorithm. The use of the bisection algorithm to compute eigenvalues of tridiagonal matrices was introduced by Givens [1] in 1954. Bisection uses the Sturm sequence property to compute the number of eigenvalues of a symmetric matrix A less than a number σ by computing the triangular factorization of $(A-\sigma I)$. Thus given any interval known to contain the desired eigenvalue, it is possible to compute the eigenvalue count at some point in the interval and thereby shrink the size of the interval known to contain the desired eigenvalue. In the absence of any ancillary information the best choice for the shift σ is the midpoint of the interval, which is what the bisection algorithm uses. By repeating the process the eigenvalue can be computed to any desired accuracy up to the roundoff error threshold.

Bisection of tridiagonal matrices is remarkably stable even without pivoting as shown by Wilkinson [3, p. 302]. A careful implementation of tridiagonal bisection is given [4, p. 249]. For larger band widths special pivoting is necessary to get a reliable eigenvalue count. An ALGOL code (BANDET2) implementing this pivoting technique is described in [4, p. 70]. For some reason a FORTRAN version of this code has not been incorporated into EISPACK.

Bisection gives a completely reliable way of computing eigenvalues. The eigenvectors can then be obtained by inverse iteration. The main drawbacks with bisection are that its asymptotic convergence rate is only linear and that there is no convenient way to take advantage of approximate eigenvectors. Of course it is possible to switch from bisection to some faster technique once an eigenvalue has been isolated. Wilkinson mentions this [3, p. 306] although he adds the observation that for tightly

* Received by the editors March 30, 1982, and in revised form April 12, 1983.

† Computer Sciences Department, University of Texas at Austin, Austin, Texas 78712.

clustered eigenvalues the advantage of switching will be slight, since most of the work will be done in the isolation phase.

The basic idea of the new algorithm is to use a faster technique before the desired eigenvalue has been isolated. Instead of just bisecting the current containment interval, it is possible to maintain an approximate eigenvector and use the Rayleigh quotient of the vector as the shift rather than the midpoint of the interval. A step of inverse iteration applied to the approximate eigenvector can be performed at the same time that the latest eigenvalue count is obtained. This is just the Rayleigh quotient iteration (see [2, p. 69]) except that if the Rayleigh quotient is outside the containment interval, the approximate eigenvector should be replaced by a random vector and the bisection shift should be used.

There are two possible reasons why such an algorithm has not been described before. One is the known existence of stagnation points of the Rayleigh quotient iteration. The stagnation points are numerically unstable, but it is still impossible to prove a global convergence result for an algorithm using Rayleigh quotient shifts. A more practical problem is the fact that the Rayleigh quotient iteration often converges monotonically. All the computed shifts are then on one side of the eigenvalue and the containment interval does not become small. This is fatal if interval widths are used for termination. Furthermore if the desired eigenvalues are specified by index, then many steps of the algorithm may be taken before it is discovered to be converging to one of the wrong eigenvalues. This false convergence can significantly degrade the performance of an algorithm based on Rayleigh quotient shifts.

The surprising fact is that both problems can be cured by the same mechanism. It would be best to have a way of choosing the shift so that a desired eigenvalue is always in the smaller subinterval. This is not always possible but it is possible to choose a shift so that the eigenvalue closest to the Rayleigh quotient is in the smaller subinterval. What is needed is a bound on the distance from the Rayleigh quotient to the nearest eigenvalue. Two potentially useful bounds are available. Given a normalized vector x and its Rayleigh quotient θ, then there is an eigenvalue λ of A which satisfies (see [2, pp. 69, 222])

$$|\theta-\lambda|\leqq\delta$$

and

$$|\theta-\lambda|\leqq\delta^2/\gamma$$

where

$$\delta=\|(A-\theta I)x\|$$

is the residual norm of x and the gap γ is the distance from θ to the next closest eigenvalue of A.

The vector x and its Rayleigh quotient θ are already available from the algorithm and the residual norm δ can be easily computed. Thus the first bound is always computable. The second bound is an improvement over the first whenever $\delta<\gamma$. In general the true gap is not known but a computable lower bound on the gap may be available from the eigenvalue counts obtained from previous factorizations. Provided that θ lies in an interval known to contain only one eigenvalue, then a lower bound on γ can be computed as the distance from θ to the nearest endpoint of this interval. We now define γ to be this computable lower bound (if it exists). Thus β, the bound on the distance from θ to the nearest eigenvalue of A, will be just δ unless γ exists and $\delta<\gamma$ in which case $\beta=\delta^2/\gamma$.

If instead of using θ directly as the shift, it is perturbed toward (but not past) the midpoint of the interval by β, then the interval known to contain the eigenvalue closest to θ will be shrunk at least by a factor of two and perhaps much more. If the eigenvalue closest to θ is a desired eigenvalue, then convergence may proceed at a much faster pace than bisection, while if the eigenvalue closest to θ is not a desired eigenvalue, then this fact is discovered immediately and the current approximate eigenvector can be abandoned. If the Rayleigh quotient lies outside the current containment interval, then the vector should be replaced by a random vector and the bisection shift should be used. We call this the BPRQS (Bisection with Perturbed Rayleigh Quotient Shift) algorithm. A formal description of the algorithm (for finding one eigenvalue and eigenvector) is as follows:

BPRQS ALGORITHM

Given an interval $[\kappa_1, \zeta_1]$ known to contain the desired eigenvalue, a normalized vector x_1, θ_1 the Rayleigh quotient of x_1, δ_1 the residual norm of x_1, and γ_1 the computable lower bound on the gap between θ_1 and the second closest eigenvalue of A (if it exists):

```
For $j = 1, 2, \cdots$ until convergence do 1–4
1. Compute the minimum error bound $\beta_j$:
   If ($\gamma_j$ does not exist or $\gamma_j < \delta_j$)
      Then $\beta_j = \delta_j$
      Else $\beta_j = \delta_j^2/\gamma_j$
2. Select a shift $\sigma_j$:
   Let $\mu_j = (\kappa_j + \zeta_j)/2$.
   If $\theta_j < \kappa_j$ or $\theta_j > \zeta_j$
      Then $\sigma_j = \mu_j$ and set $x_j$ to a random vector
      Else If $\theta_j < \mu_j$
         Then $\sigma_j = \min\{\mu_j, \theta_j + \beta_j\}$
         Else $\sigma_j = \max\{\mu_j, \theta_j - \beta_j\}$
3. Factor $(A - \sigma_j)$ to get the new eigenvalue count, solve
      $(A - \sigma_j I)y_{j+1} = x_j$, and normalize $y_{j+1}$ to get $x_{j+1}$
4. Update the values:
   If the desired eigenvalue is smaller than $\sigma_j$
      Then $\kappa_{j+1} = \kappa_j$ and
$$\zeta_{j+1} = \sigma_j.$$
      Else $\kappa_{j+1} = \sigma_j$ and
$$\zeta_{j+1} = \zeta_j$$
   $\theta_{j+1} = x_{j+1}{}^T A x_{j+1}$
   $\delta_{j+1} = \|(A - \theta_{j+1}1)x_{j+1}\|$
   Update $\gamma$ (if possible) from $\theta_{j+1}$ and the previous
      eigenvalue counts.
```

Termination is based on the width of the containment interval or on the error bound once the computed gap is larger than the residual norm.

3. Global convergence. The BPRQS algorithm converges globally to a particular eigenvalue with an asymptotic convergence rate which is at least linear with rate constant at least SQRT($\frac{1}{2}$), as shown by the following theorem.

THEOREM 1. *For all steps j of the* BPRQS *algorithm, the containment interval after two more steps, that is $[\kappa_{j+2}, \zeta_{j+2}]$, will satisfy at least one of*:

1. $\zeta_{j+2}-\kappa_{j+2} \leqq (\zeta_j-\kappa_j)/2$
2. $[\kappa_{j+2}, \zeta_{j+2})$ *contains fewer eigenvalues than* $[\kappa_j, \zeta_j]$.

Proof. If θ_j is not in $[\kappa_j, \zeta_j]$, then the first step is a bisection step and the theorem is true. Without loss of generality we now assume that θ_j is in $[\kappa_j, \mu_j]$.

If $\theta_j+\beta_j > \mu_j$, then the next step is a bisection step and the theorem is true. So we may assume that the next shift will be $\sigma_j=\theta_j+\beta_j$.

If the desired eigenvalue is in $[\kappa_j, \sigma_j]$, then the theorem is true. So we may assume that $\kappa_{j+1}=\sigma_j$ and $\zeta_{j+1}=\zeta_j$.

If θ_{j+1} is not in $[\kappa_{j+1}, \zeta_{j+1}]$, then the next step will be a bisection step and the theorem will be true. So we may assume that θ_{j+1} is in $[\kappa_{j+1}, \zeta_{j+1}]$.

Assume for now that $\theta_{j+1} \leqq \mu_{j+1}$.

If $\theta_{j+1}+\beta_{j+1} \geqq \mu_{j+1}$, then the next step will be a bisection step and the theorem is true. So we may assume that $\sigma_{j+1}=\theta_{j+1}+\beta_{j+1}$.

If the desired eigenvalue is in $[\kappa_{j+1}, \sigma_{j+1}]$, then the theorem is true. So we may assume that $\kappa_{j+2}=\sigma_{j+1}$ and $\zeta_{j+2}=\zeta_{j+1}$.

If there is any eigenvalue in $[\kappa_j, \kappa_{j+2}]$, then the theorem is true. The bound β is computed using γ only if the interval $[\theta-\delta, \theta+\delta]$ is known to contain only the desired eigenvalue. In this case $[\theta-\beta, \theta+\beta]$ also contains only the desired eigenvalue. Since we have assumed that the desired eigenvalue is not in this interval at both step j and step $j+1$, it follows that both $\beta_j=\delta_j$ and $\beta_{j+1}=\delta_{j+1}$.

The following lemma is needed to prove under the above assumptions that there is an eigenvalue in the interval $[\kappa_j, \kappa_{j+2}]$.

LEMMA 2. *Let θ be the Rayleigh quotient of the normalized vector x and let δ be its residual norm ($\|A-\theta I)x\|$). Let $\mu y=(A-\sigma I)^{-1}x$ for some number σ, where μ ($\|(A-\sigma I)^{-1}x\|$) is chosen to normalize y, let α be the Rayleigh quotient of y, and let ρ be its residual norm. Then if $\theta \leqq \sigma \leqq \alpha$ then*[1]

$$\rho \leqq \delta$$

and

$$\alpha-\sigma \leqq \delta.$$

Proof.

$$\|(A-\theta I)x\| \geqq y^T(A-\theta I)x$$

by the Cauchy–Schwarz inequality,

$$\begin{aligned}&=y^T(A-\sigma I)x+(\sigma-\theta)y^Tx\\&=y^T(A-\sigma I)x+\mu(\sigma-\theta)y^T(A-\sigma I)y\\&=y^T(A-\sigma I)x+\mu(\sigma-\theta)(\alpha-\sigma)\\&\geqq y^T(A-\sigma I)x\end{aligned}$$

by hypothesis,

$$=\|(A-\sigma I)y\|$$

[1] This generalizes Kahan's monotone residual theorem for the Rayleigh quotient iteration. See [2, p. 75].

since x is parallel to $(A-\sigma I)y$,

$$\geqq \|(A-\sigma I)y\|$$

since the Rayleigh quotient minimizes the residual norm.

This proves the first result. To get the second we apply the Cauchy–Schwarz inequality to the next to last line above to get

$$\geqq y^T(A-\sigma I)y$$
$$=\alpha-\sigma.$$

By elementary linear algebra there is an eigenvalue in $[\theta_{j+1}, \delta_{j+1}, \theta_{j+1}+\delta_{j+1}]$. $\beta_{j+1}=\delta_{j+1}$ as shown above, $\delta_{j+1}\leqq\delta_j$ by the lemma, $\sigma_j=\theta_j+\beta_j$ by construction, and $\theta_j\geqq\kappa_j$. Therefore

$$\begin{aligned}[\kappa_j, \kappa_{j+2}] &= [\kappa_j, \theta_{j+1}+\beta_{j+1}]\\ &= [\kappa_j, \theta_{j+1}+\delta_{j+1}]\\ &\supseteq [\theta_{j+1}, \delta_{j+1}, \theta_{j+1}+\delta_{j+1}]\end{aligned}$$

and hence contains an eigenvalue.

Finally assume $\theta_{j+1}\geqq\mu_{j+1}$. The new containment interval will be smaller than $[(\sigma_j, \theta_{j+1}]$ and by the lemma, $\theta_{j+1}-\sigma_j\leqq\delta_j$ and $\delta_j\leqq(\kappa_j-\zeta_j)/2$ by assumption (since the first shift was less than a bisection shift. Therefore $\zeta_{j+2}-\kappa_{j+2}\leqq(\zeta_j-\kappa_j)/2$ and the theorem is true.

This completes the proof.

4. Local convergence. The previous section showed that the BPRQS algorithm converges globally with asymptotic rate of convergence no worse than linear with rate constant SQRT($\frac{1}{2}$). With a careful analysis it is possible to improve the rate constant to $\frac{1}{2}$ but impossible to prove that the asymptotic convergence rate is better than linear. The reason for this is that the global convergence result only guarantees convergence to the eigenvalue by the bisection process; it does not guarantee convergence to the eigenvector. This convergence failure can happen only if the starting vector x_0 is orthogonal to the desired eigenvector. Furthermore during the algorithm it is very likely that the current vector would be periodically discarded and a new random vector generated. To prevent convergence to the desired eigenvector, each of these subsequent vectors would also have to be orthogonal to the desired eigenvector. This event has probability zero. We now assume that the vectors x_j converge to the desired eigenvector. With this assumption it is possible to prove that the convergence of the BPRQS algorithm is asymptotically quadratic or cubic.

THEOREM 3. *Assume that the sequence of vectors x_j generated by the* BPRQS *algorithm converges to the desired eigenvector z. Let λ be the eigenvalue associated with z. If λ is a simple eigenvalue, then the convergence is asymptotically cubic. Otherwise the convergence is asymptotically quadratic.*

Proof. Assume that the projection of the vector x_j onto the eigenspace associated with λ is the vector z and assume that

$$x_j=\nu z+\varepsilon w$$

where w and z are unit vectors, w is orthogonal to z, and $\nu=\text{SQRT}(1-\varepsilon^2)$. The Rayleigh quotient of x_j is

$$\theta_j=\lambda+\varepsilon^2(w^TAw-\lambda)$$

which is obviously $\lambda+O(\varepsilon^2)$.

The residual norm of x_j is

$$\|(A-(\lambda+O(\varepsilon^2))I)x_j\|=\varepsilon\|(A-\lambda I)w\|+O(\varepsilon^2).$$

If λ is a multiple eigenvalue, then the next shift σ_j will be within $O(\varepsilon)$ of λ and the next vector will be

$$y_{j+1}=(A-(\lambda+O(\varepsilon))I)^{-1}x_j.$$

Thus the z component of x_j will be amplified by $1/\varepsilon$ while the other components will be amplified by at most $1/\gamma$ where γ is the gap between λ and the distinct eigenvalue closest to λ. If $\varepsilon \ll \gamma$, then after normalization

$$x_{j+1}=z+O(\varepsilon^2).$$

If λ is a simple eigenvalue, then eventually the minimum error bound will be δ_j^2/γ_j and thus the next shift σ_j will be within $O(\varepsilon^2)$ of λ. The next vector y_{j+1} will have its z component amplified by $O(1/\varepsilon^2)$ while the others remain bounded so that after normalization

$$x_{j+1}=z+O(\varepsilon^3)$$

and the theorem is proved.

5. Numerical results. The implementation of the algorithm used in this section computes the desired eigenvalues sequentially from left to right. Containment intervals for all the desired eigenvalues are updated after each factorization. Vectors discarded in the course of computing one eigenvalue are saved if appropriate for computing later eigenvalues.

This section gives the results of a few numerical experiments. In every test except the last, the matrix A was of the form HDH where $H=1-2ww^T/w^Tw$ was a Householder reflection matrix with a random vector w, and D was a diagonal matrix containing the eigenvalues. This form allows the easy specification of the eigenvalues without suffering from the special rounding error characteristics of using a diagonal matrix directly. Of course the resulting matrix is not banded but $(A-\sigma I)^{-1}$ can be computed as $H(D-\sigma I)^{-1}H$, which keeps the cost of the tests down. The tests were run on a DEC-20 computer in double precision.

The first example had $D=\operatorname{diag}(1,2,3,\cdots,50)$. The smallest eigenvalue was computed from an initial containment interval $[-6,55]$ and a random vector. The behavior of the algorithm is given in Table 1.

As can be seen from Table 1, it takes a while for the shift to get nearer to 1 than to any other eigenvalue, at which point the cubic rate of convergence of the algorithm becomes apparent.

It is more efficient in terms of factorizations per eigenvalue to compute several adjacent eigenvalues since the information gained while locating the first eigenvalue gives a head start on finding the rest. Table 2 shows the number of factorizations needed to compute the first ten eigenvalues of the same matrix A using the same initial containment interval.

To test the effects of multiple eigenvalues on the algorithm, the second test matrix had the same eigenvalues as the first except that five extra eigenvalues were added to make the eigenvalues 1, 2, 3, 4, and 5 double. The behavior of the algorithm on computing the smallest eigenvalue of the matrix, starting with the interval $[-5,56]$ and a random vector, is shown in Table 3. The quadratic convergence can be seen easily as well as the additional penalty for terminating due to interval width rather than due to small residual.

TABLE 5.1
Behavior of BPRQS *algorithm.*

Step	RQ	Residual	Error bound	Shift
1	27.204	13.50	13.50	25.5000
2	25.715	2.09	2.09	10.2500*
3	10.096	1.87	1.87	8.2220
4	9.232	.837	.837	1.6611*
5	2.825	3.11	3.11	−1.6945
6	4.756	7.00	7.00	−.0417*
7	3.267	4.80	4.80	.7846*
8	3.964	5.78	5.78	1.1978*
9	1.319	1.24	1.24	.9912
10	1.000	.0555	.00505	1.00534
11	1.000	.515e−4	.434e−8	.999999972
12	1.000	.962e−13	.304e−15	terminate

* indicates a bisection shift and randomization of the eigenvector approximation

TABLE 5.2
Number of factorizations per eigenvalue.

Eigenvalue	Number of factorizations
1.	11
2.	8
3.	6
4.	4
5.	7
6.	3
7.	4
8.	5
9.	4
10.	3

TABLE 5.3
Behavior of BPRQS *algorithm on multiple eigenvalues.*

Step	RQ	Residual	Error bound	Shift
1	19.957	14.9	14.9	25.500*
2	25.098	4.03	4.03	21.067
3	21.048	.514	.514	20.535
4	20.998	.075	.075	7.767*
5	8.264	4.73	4.73	1.384*
6	1.685	1.96	1.96	−1.808*
7	3.972	6.05	6.05	−2.12*
8	3.464	4.68	4.68	586*
9	1.201	1.39	1.39	985*
10	1.000	.498e−2	.498e−2	1.005
11	1.000	.227e−4	.227e−4	.999977
12	1.000	.511e−9	.511e−9	1.0000000005107
13	1.000	.334e−15	.334e−15	.99999999999999666
14	1.000	.334e−15	.334e−15	1.00000000000000334
15	1.000	termination due to interval width		

* indicates a bisection shift and randomization of the eigenvector approximation

As before in the simple eigenvalue case, the number of factorizations per eigenvalue was reduced if several eigenvalues were computed at once. The number of factorizations needed for each of the ten smallest eigenvalues for the second test matrix are given in Table 4. Each second eigenvalue was already known to the desired accuracy; the factorization was needed only in computing the corresponding eigenvector. The total number of factorizations needed to compute the ten eigenvalues was 54. This compares to the 55 factorizations needed to compute all the eigenvalues in Table 2.

TABLE 5.4
Number of factorizations per eigenvalue.

Eigenvalue	Number of factorizations
1.0	14
1.0	1
2.0	10
2.0	1
3.0	9
3.0	1
4.0	7
4.0	1
5.0	9
5.0	1

The behavior of the algorithm on close but not multiple eigenvalues is similar. The third test matrix had eigenvalues i^{-3}, $i = 1, 2, \cdots, 50$. The code took 60 factorizations to compute the ten smallest eigenvalues which are bunched closely together near zero. When asked to compute the ten largest eigenvalues, the code needed 61 factorizations.

The final test matrix used was one of the tridiagonal matrices used to test the tridiagonal bisection code [4, p. 255]. The off diagonal elements are all ones while the diagonal is diag$(100, 90, 80, \cdots, 10, 0, 10, \cdots, 100)$. All twenty-one eigenvalues were computed to fifteen figures of accuracy using 93 factorizations starting with random vectors and the containment interval $[-1, 101]$, which is known to contain all the eigenvalues by the Gerschgorin circle theorem. The bisection algorithm took 345 factorizations to compute the eigenvalues to seven figures of accuracy.

6. Generalized eigenvalue problems. Both the bisection algorithm and the Rayleigh quotient iteration can be applied to the generalized eigenvalue problem

$$(A - \lambda M)z = 0$$

provided that both A and M are symmetric and M is positive definite. The only problem is that given an approximate eigenvector x with $x^TMx = 1$ and its Rayleigh quotient $\theta = x^TAx$, the residual norm needed by the algorithm is the M^{-1} norm of $(A - \theta M)x$. Computing an M^{-1} norm requires the factorization of M. However neither the time needed to compute this factorization nor the space needed to store it make a significant increase in the time and space needed for the algorithm in the standard case. Once the factorization of M is available the algorithm proceeds as described above. Thus the algorithm is equally applicable to generalized eigenvalue problems.

7. Conclusions. This paper has described and analyzed a new combination of the Rayleigh quotient iteration and the bisection algorithm and has shown that it is an effective way to compute a few eigenvalues and eigenvectors of a symmetric band matrix or a symmetric definite banded generalized eigenvalue problem.

8. Acknowledgment. The author wishes to thank the referees for their many helpful comments.

REFERENCES

[1] W. GIVENS, *Numerical computation of characteristic values of a real symmetric matrix*, Technical Report ORNL-1574, Oak Ridge National Laboratory, Oak Ridge, TN, 1954.
[2] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.
[3] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965.
[4] J. H. WILKINSON AND C. REINSCH, *Handbook for Automatic Computation*, Springer-Verlag, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

A SIMPLIFIED MOVING FINITE DIFFERENCE SCHEME: APPLICATION TO DENSE GAS DISPERSION*

E. J. KANSA†, D. L. MORGAN, JR.†, AND L. K. MORRIS†

Abstract. A simplified moving grid scheme has been developed and applied to a system of six partial differential equations (PDEs) that describe the dispersion of a cloud of cold, dense gas resulting from a spill of liquefied natural gas (LNG).

The grid velocity at each point is determined by the physics represented by the governing PDEs. By a minimization process, a grid velocity is found that transforms the PDEs into a system of ordinary differential equations (ODEs) in the least-squares sense. At a shock, the grid velocity is automatically the shock velocity given by the Rankine–Hugoniot relations.

The ill-posed problems associated with moving grids are handled by three regularization schemes. No constraints are placed upon gradients. In the event of a zero gradient, a special matrix regularization is applied. Likewise, no constraints were placed on the grid velocity. Problems of grid tangling and/or violation of minimum point separation were handled by a second regularization scheme. A third regularization is also used to equally distribute the third-order spatial truncation errors. Results are presented using 21 grid points. Shock structure is well resolved. A comparable spatial resolution in a fixed-frame calculation of equally spaced grids would have required in excess of 670 grid points.

Key words. moving grids, grid regularization, adaptive grids, implicit finite scheme, Rankine–Hugoniot relations, moving frame

1. Introduction.

Modeling of two-phase fluid flow is hampered by strong nonlinearities, and by disparate temporal and spatial scales. Explicit numerical techniques traditionally used for hydrodynamical problems are quite inefficient, especially when three-dimensional problems are considered. This report discusses a scheme that gives accurate and reliable results at a reasonable cost.

Because such systems of PDEs may admit solutions with propagating steep wave fronts, several authors have proposed procedures that follow and resolve such fronts. Dwyer et al. [5] and White [18] introduced coordinate transformations such that no large derivatives appear in the transformed space. Hu and Schiesser [11] used a dynamic mesh-refinement procedure in an Eulerian frame; successively finer meshes are added to minimize spatial truncation error, and fine meshes are deleted where the solution is smooth. Davis and Flaherty [4] used a fixed number of grid points and adapted the grid to equipartition the spatial truncation errors.

The MFE (Moving Finite Element) approach [9], [16], [17] also uses a fixed number of grid points. However, the MFE approach provides a mathematical methodology for optimally moving grids. The dependent variables are discretized by piecewise linear polynomials. The polynomial coefficients and grid positions are assumed to be unknown functions of time whose solutions are found by minimizing the least-squares residual. Grid motions are controlled by penalty functions that force grid points away from regions of closest separation or prevent three or more dependent variables from lying upon a straight line segment. More refined penalty functions also prevent points from concentrating about a steep front and force them into other regions of high curvature. The principal drawback to this approach is that the penalty functions have many adjustable parameters that must be fine tuned for each problem. However,

* Received by the editors August 4, 1982, and in revised form May 6, 1983. 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.

† Liquefied Gaseous Fuels Program, Lawrence Livermore National Laboratory, Livermore, California 94550.

unlike some adaptive techniques, the MFE approach can be extended into two or three dimensions.

The motivation for this paper was to find computationally simpler methods for choosing grid velocities and controlling grid motion while retaining as much of the power of MFE approach as possible. The criterion for choosing grid velocities is based upon an intuitive approach, rather than on an equidistribution scheme.

This paper presents a moving-grid method for a finite difference scheme in R^1 involving six hyperbolic PDEs with nonlinear source terms. These PDEs arise from the Zeman [19] and SLAB [6] models which treat the time-dependent dispersion of a cloud of heavy gas.

This paper discusses the following topics: the criteria for the selection of grid velocities including regularization, the time integration scheme, and numerical results for a case in which the cloud is a time-dependent mixture of LNG vapor and air.

2. Criterion for selecting moving grid velocities. First-order parabolic conservation laws are used to model many unsteady systems whose behavior is determined by both time and position. Because of the nonlinear nature of the PDEs, such effects as convection and chemical reactions may cause steep fronts to evolve. The standard numerical approach has been to spread out such fronts and to use unrealistically large amounts of dissipation.

Following Bird et al. [2], the conservation laws in the laboratory frame may be written as

$$\frac{\partial \Phi}{\partial t} + \nabla \cdot \mathbf{F} = S \tag{1}$$

where

$$\Phi = [\rho, \rho\mathbf{u}, \varepsilon, \{\rho_i\}]^\dagger \qquad (i = 1, NS-1), \tag{2}$$

$$\mathbf{F} = [\rho\mathbf{u}, (\rho\mathbf{uu} + pI + \tau), (\mathbf{u}(p+\varepsilon) + \mathbf{q} + \tau : \mathbf{u}), \{(\rho_i\mathbf{u} + \mathbf{j}_i)\}]^\dagger, \tag{3}$$

$$S = [0, \rho\mathbf{g}, \rho\mathbf{g} \cdot \mathbf{u} + \dot{Q}_c, \{\dot{R}_i\}]^\dagger, \tag{4}$$

where ρ is the total density, $\mathbf{u}$ is the convective velocity, ε is the total energy density, ρ_i is the density of the ith species, τ is the Newton stress tensor, I is the identity tensor, $\mathbf{q}$ and $\mathbf{j}_i$ are the flux vectors corresponding to Fourier's and Fick's laws, respectively, p is the pressure determined from the ideal gas law, $\mathbf{g}$ is the gravitational vector; $\dot{Q}_c$ is the heat due to chemical reactions, $\dot{R}_i$ is the rate of the ith chemical reaction, NS is the number of chemical species, and $[\ ]^\dagger$ denote transpose of a vector.

The basic idea for grid velocity selection in this paper is that, rather than solving the PDEs (1) in the fixed laboratory frame, we solve the PDEs in a moving frame, $\mathbf{v}$, in which the solutions may be easier to obtain. In this frame $\mathbf{v}$, the conservation equations become

$$\frac{\partial \Phi}{\partial t} + \nabla' \cdot \mathbf{F} - \mathbf{v} \cdot \nabla' \Phi = S \tag{5}$$

and

$$\frac{d\mathbf{r}}{dt} = \mathbf{v} \tag{6}$$

where $\mathbf{r}'$ is the position vector, and the moving frame (denoted by primed variables) is related to the fixed frame by means of the transformation tensor, Γ. It should be noted that (5) is not completely general because the Coriolis terms arising from local grid rotations are missing. In our one-dimensional application there is no grid rotation. (See Bird et al. [2] for the rules of transforming vectors and tensors.) For notational simplicity, the primes are dropped, and it is understood (unless otherwise stated) that we are in the moving coordinate system.

If we could choose the grid velocity so that

$$\mathbf{v} \cdot \nabla \Phi = \nabla \cdot \mathbf{F}, \tag{7}$$

then in the moving frame $\mathbf{v}$, (5) and (6) would become

$$\frac{d\Phi}{dt} = S \tag{8}$$

and

$$\frac{d\mathbf{r}}{dt} = \mathbf{v}. \tag{9}$$

At a shock, (7) is a restatement of the Rankine–Hugoniot jump conditions, and $\mathbf{v}$ is the shock speed; see [3]. Similarly, jump conditions can be obtained for flames (see Lewis and Von Elbe [13]). Defining a burning velocity and expansion ratio particular to the chemical composition of the flame, a flame will appear steady in the Galilean frame moving at the flame speed. Analogous jump conditions exist for the mass, momentum and energy fluxes. Consequently, we choose $\mathbf{v}$ so (7) is satisfied in the least-squares sense. Thus, we minimize

$$\sum_{i=1}^{\mathrm{NEQ}} (\mathbf{v} - \mathbf{v}_i)^2 \tag{10}$$

where

$$\mathbf{v}_i = \frac{(\nabla \cdot \mathbf{F}_i) \nabla \Phi_i}{|\nabla \Phi_i|^2} \tag{11}$$

and NEQ is the number of PDEs in the system. We shall introduce a regularization when $|\nabla \phi_i|$ becomes too small.

In summary, the grid velocity is chosen so that a system of PDEs transforms into ODEs in some least-squares sense. At a shock, however, this velocity automatically is the shock velocity given by the Rankine–Hugoniot jump conditions. Like the MFE scheme of Miller [16], [17], the scheme presented here is readily extended into higher dimensions.

3. The finite difference scheme and determination of grid velocity. This section is presented in two parts. The first outlines the numerical time-integration scheme used to determine the dependent variables. The second part examines the difficulties associated with determining the grid velocities as an ill-posed problem. Rather than using the Tikhonov-type regularization, as in Miller [16], [17], we use a regularization based on the LU decomposition.

The problem to be solved in one spatial dimension can be stated as follows:

$$\frac{\partial \Phi(i)}{\partial t} + \frac{\partial F(i)}{\partial x} - \frac{v \partial \Phi(i)}{\partial x} - S(i) = 0, \qquad (i = 1, \cdots, \mathrm{NEQ}), \tag{12}$$

$$\frac{dx}{dt} - v = 0, \tag{13}$$

where

$$\sum_{i=1}^{\mathrm{NEQ}} (v - v_i)^2 = \text{minimum} \tag{14}$$

and

$$v_i = \frac{\partial F(i)/\partial x}{\partial \Phi(i)/\partial x}, \tag{15}$$

and we will use regularization if $|\partial\Phi/\partial x|$ becomes too small. It is convenient to adopt the notation

$$\Phi_j^n(i) = \Phi(i, t^n, x_j^n) \tag{16}$$

where i refers to the ith dependent variable, n refers to the nth time step, and j refers to the jth spatial node at time, t^n.

For convenience, we write

$$G_i = G(\Phi_i, x) = \frac{\partial F(i)}{\partial x} - \frac{v\partial\Phi(i)}{\partial x} - S(i). \tag{17}$$

Then

$$\frac{\partial \Phi(i)}{\partial t} + G_i = 0 \tag{18}$$

and

$$\frac{dx}{dt} - v = 0. \tag{19}$$

The time marching scheme used here is a two-level implicit scheme which is correct to second order for the dependent variables:

$$\Phi_j^{n+1} - \Phi_j^n + \Delta t\{\theta G_j^{n+1} + (1-\theta)G_j^n\} - \left(\frac{2\theta - 1}{2}\right)\left(\frac{\Delta t^2}{2}\right)\left\{\frac{\partial G}{\partial \Phi} G - \frac{\partial G}{\partial x} v\right\}_j^{n+1} = 0, \tag{20}$$

and

$$x_j^{n+1} = x_j^n + \Delta t\{\theta v_j^{n+1} + (1-\theta) v_j^n\}, \tag{21}$$

$$\tfrac{1}{2} \leqq \theta \leqq 1. \tag{22}$$

The last term in brackets in (20) is the second-order temporal correction to the trapezoidal rule; it behaves like a dissipative term (see [7]). This term is similar to the dissipation terms deduced by Majda and Osher [15].

The dissipation term used in (20) is given a floor value equal to typical molecular values in gases of the same viscosity, thermal conductivity and species diffusion. Likewise, a ceiling on the numerical diffusion is controlled by the time step, Δt, and the parameter θ. The code was run with $\theta = 0.51$. The PDEs solved are the one-dimensional, SLAB-averaged, Navier–Stokes equations presented in Appendix A.

Since the spatial points $\{x_j^{n+i}\}$ move at different velocities, fixed spacing difference schemes are not appropriate. The first and second spatial partial derivatives were obtained from a second-order accurate, three-point collocation polynomial. The derivatives in the interior regions at the point x_j are written as

$$\left.\frac{\partial \Phi}{\partial x}\right|_{x_j} = \frac{(x_j - x_{j-1})(\Phi_{j+1} - \Phi_j)}{(x_{j+1} - x_{j-1})(x_{j+1} - x_j)} + \frac{(x_{j+1} - x_j)(\Phi_j - \Phi_{j-1})}{(x_{j+1} - x_{j-1})(x_j - x_{j-1})}, \tag{23}$$

$$\left.\frac{\partial^2 \Phi}{\partial x^2}\right|_{x_j} = \frac{2(\Phi_{j+1} - \Phi_j)}{(x_{j+1} - x_{j-1})(x_{j+1} - x_j)} - \frac{2(\Phi_j - \Phi_{j-1})}{(x_{j+1} - x_{j-1})(x_j - x_{j-1})}. \tag{24}$$

Substituting (23)–(24) into (20)–(21) yields a block tridiagonal system of nonlinear finite difference equations. The equations were linearized about the current iterate using the Powell's modification of the Newton–Raphson scheme; see [12]. Typically at most two iterations were required, during the highly transitory early phases. Davis and Flaherty [4] also used a similar time-integration technique.

The grid velocity and dependent variables could be solved simultaneously or sequentially. For this particular application of the SLAB PDEs, analysis of the Jacobian matrix showed the coupling of the dependent variables and spatial positions at the advanced time step to be weak. Because of this weak coupling, it is possible to solve for the advanced time-dependent variables and spatial position sequentially rather than simultaneously as in the generalized MFE method of Miller [16], [17]. Davis and Flaherty [4] also use the sequential approach. The sequential solution method offers computational advantages which will be elaborated later on in this section.

As stated in the previous section, the principle by which a common grid velocity is selected is such that a moving frame is sought in which the PDEs behave in some average sense as ODEs. The common grid velocity v is selected by requiring

$$\sum_{i=1}^{\text{NEQ}} (v - v_i)^2 = \text{minimum} \tag{25}$$

where

$$v_i = \frac{\partial F(i)/\partial x}{\partial \Phi(i)/\partial x} \tag{26}$$

and

$$\frac{dx_j}{dt} = v_j. \tag{27}$$

The minimization procedure gives rise to a simple tridiagonal matrix involving v_{j-1}, v_j, and v_{j+1} because of the use of the three-point collocation formulae for spatial derivatives. Note that the off-diagonal elements are of order Δt.

The above scheme, (25)–(27), produces a nonsingular tridiagonal matrix as long as the gradients are nonzero at each point. At straight line segments and maxima or minima, the matrix is either singular or very ill-conditioned.

By solving for the grid velocities or advanced spatial positions sequentially, the ill-posed problems associated with moving grids arise only in the grid-velocity determination. The ill-posed problems are found whenever gradients are zero or very close to zero and whenever the zone size goes to zero. Furthermore, it is desirable to force points to regions of high curvature and to equidistribute the truncation errors.

In order to handle singular or highly ill-conditioned matrices that arise in the determination of grid velocities, regularization is used in several stages. The first regularization scheme is used in determining grid velocities. A special routine for tridiagonal matrices was developed that employs Gaussian elimination with row pivoting. If after pivoting the jth diagonal matrix element is zero or almost zero, the

Gaussian elimination is temporarily halted. The matrix is modified so that v_j becomes the weighted average of v_{j-1} and v_{j+1}, and the elimination is continued. The code permits an analogous modification for any number of consecutive rows of the matrix, but this feature was not needed in the routine.

Further regularization is needed whenever points come within a specified distance of one another. If the spatial separation becomes zero, the system of equations becomes ill posed. Likewise, points are forbidden to cross.

In an earlier version we used penalty functions to prevent the spatial separation from going to zero. A repulsive force penalty function was added to (20)–(22) to decrease the grid velocity as grid separation decreased near the developing shock. This has several drawbacks. On one hand, the minimization procedure forces the grid velocity to become the shock velocity in the vicinity of the shock. But on the other hand, the retarding force tends to decrease the grid velocity. In reality, slowing or stopping the grid motion makes the solution near the shock behave much like an Eulerian PDE calculation on a fine grid rather than like an ODE calculation. To preserve stability, the time step had to be progressively decreased to an uneconomically small level.

In the current version of the code, we use a second regularization which permits larger time steps. If the minimium grid separation is violated at the end of a time step, the dependent variables and grid velocities are mapped onto new points that satisfy the minimum separation criterion. The mapping was done using a cubic spline routine developed by Forsythe et al. [8]. As evidenced by the shock profiles to be presented, this mapping procedure appears to introduce minimum smoothing. The advantage of this procedure is that at a shock, the grid velocity is the shock speed and the Rankine–Hugoniot relations are satisfied.

The third stage of regularization consists of forcing points to seek regions of high curvature and preventing points from concentrating only about the shock. The unmodified minimization procedure attracts grid points to a shock front, leaving the remainder of the domain poorly resolved. The same mapping scheme is also applied to force points back to regions of high curvature and to distribute them more uniformly.

The criterion for redistributing points after unrestricted time advancement is a modification of the methods of Dwyer et al. [5] and Davis and Flaherty [4]. Denote U_j''' as the Euclidian norm of the third derivative,

$$\tilde{\Phi}_{xxx}(k), \qquad k=1,\cdots,6, \qquad U_j'''=\left(\sum_{k=1}^{6}\frac{\tilde{\Phi}^2_{xxx}(k)}{6}\right)^{1/2}. \tag{28}$$

Following Davis and Flaherty [4], we demand that

$$(\Delta x_j)^3|U_j'''|=\text{constant} \qquad (j=1,2,\cdots,nx-1) \tag{29}$$

where nx is the number of grid points; following Dwyer et al. [5] it was demanded that

$$\frac{1}{C}\leqq \Delta x_j \leqq C \qquad (C<1), \tag{30}$$

where

$$\Delta x_j=(x_{j+1}-x_j)\geqq \delta, \tag{31}$$

where δ is a minimum separation distance. If a grid separation comes within the minimum separation distance, then it is translated back a distance $\Delta t v_j$. The sum of the grid intervals is constrained to be the normalized length of unity.

After the new set of grid locations is determined by the above scheme, the six dependent variables and grid velocities are calculated at these new locations by interpolation. As an initialization procedure before the next time cycle, this regridding procedure is performed after the dependent variables and grid velocities are determined. The grid velocities at the advanced time are determined by (23)–(26) rather than by the methods of Dwyer et al. [5] or Davis and Flaherty [4]. A modification of their methods is used only as part of the regularization at the end of a time step.

The numerical procedure for determining the advanced time-dependent variables and grid velocities attempted to keep the number of adjustable constants to a minimum. The first constant is θ, which weights the relative contributions of the old and new time solutions and which controls the amount of stabilizing dissipation added to the scheme. (θ was chosen to equal 0.51 in this application.)

Likewise, a floor value was placed upon the numerical dissipation added to the scheme. Typical molecular values for gases were used. Other adjustable parameters are the number of grid points, the minimum point separation, and the constants used in forcing points to seek regions of high curvature, requiring three additional adjustable constants, cf. (24)–(25).

With this regridding procedure, it was found that the time steps were not constrained by the grid motion. Rather, the time steps were chosen to keep the maximum error of the third-order temporal truncation terms below 10^{-7}. Typical iteration errors for the dependent variables and the grid velocity calculations were of the order of 10^{-10} to 10^{-12}. While Miller [16], [17] and Gelinas et al. [9] used higher-order time integration schemes such as an implicit Runge–Kutta or the GEAR package, this paper uses a second-order block implicit stiff integration scheme; see [12]. The advantage of this lower-order scheme is that the Jacobian has a block tridiagonal structure which saves storage and can be treated by fast existing packages. Although smaller time steps are needed with the stiff block implicit scheme, as compared to higher-order implicit schemes, the code developed here ran with time steps 23 to 52 times greater than the explicit CFL condition, even with the steep profile at the leading edge. We believe that a lower-order block implicit scheme becomes more advantageous in higher dimensions where memory limitations are important.

4. Presentation of the results. A system of nonlinear hyperbolic equations can develop a steepening front. In the absence of physical dissipation, this shock front is infinitesimally thin, resulting in a true mathematical discontinuity. The SLAB PDEs presented in Appendix A were solved.

Because the turbulent dissipation in the SLAB model has been accounted for by the entrainment mechanism, molecular dissipation was added. Assuming that the Schmidt and Prandtl numbers were unity, the diffusion coefficient was taken to be $2.2\,10^{-5}\,\mathrm{m}^2/\mathrm{sec}$.

The results presented are a simulation of a 40-m^3 spill of LNG at a spill rate of 0.13 kg/sec. The ambient wind speed was 5.7 m/sec. The primary purpose of this computer simulation was to test the numerics of the simplified adaptive gridding scheme. A detailed discussion of the physics will appear in a future paper.

The domain of existence of the SLAB PDEs is restricted to lie within the boundaries of the dense vapor cloud. However, the cloud grows in time with the injection of further mass, and it spreads out because of the gravity interaction and wind convection. The computational domain was restricted to lie within the vapor cloud by means of a coordinate transformation that moves with the cloud boundaries (see Appendix B). The results presented here were calculated in a transformed system, in which both internal points and cloud boundaries moved in time.

The code was tested first using SLAB equations for a problem whose analytical solution was known. If air is injected into air at the same temperature and density, then the width, density, temperature and lateral momentum of the front should be constant. After 10 seconds of simulation time the model results were within 3% of the analytic answers.

Figures 1 and 2 depict the grid and the downwind particle velocities versus the entire downwind distance at 5 and 60 seconds respectively. Figures 3 and 4 depict the five points nearest the right-hand boundary for the grid and downwind particle velocities at 5 and 60 seconds.

These figures show that the grid velocities are considerably larger than the downwind particle velocity. Only near the left-hand boundary is the grid velocity Lagrangian in any sense. The maximum grid velocity occurs near the formation of the steep front.

At $t=0$, all the dependent variables were centered symmetrically about $x=0$ m. In 5 seconds, the maxima or minima of the variables had shifted to about 4.4 m downwind with the source turned on. The inflection in v at 4.4 m corresponds to the shift in the nearly coincident extrema. However, by 60 seconds, the extrema for the dependent variables were no longer coincident. Consequently, the inflection was considerably spread out and reduced.

Figures 3 and 4 show the combined effect from physical and second-order temporal diffusion; see (23). The sharp front has broadened from 10 cm out of 83 m at 5 seconds to 64 cm out of 397 m at 60 seconds. The steep front is broadening at an average rate

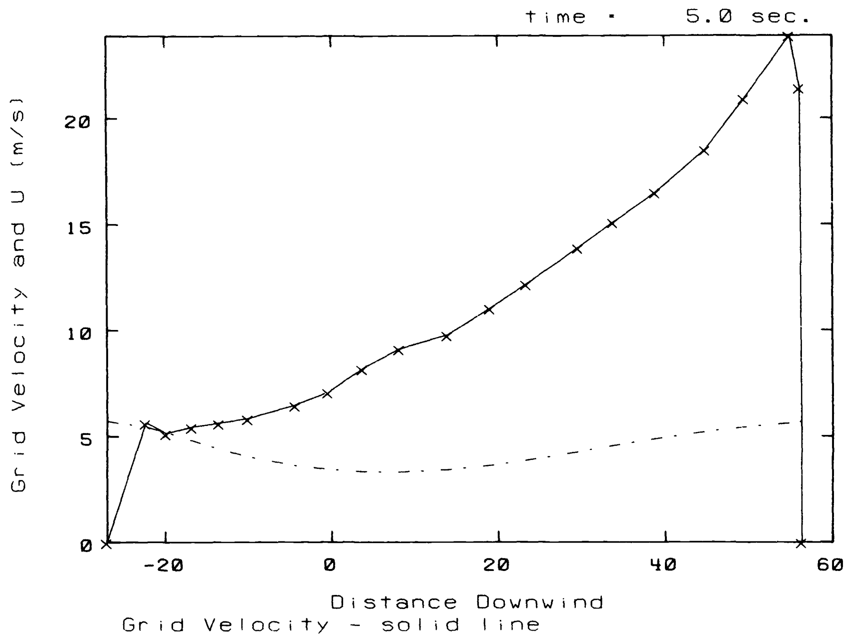


FIG. 1. *Advection and grid velocities versus distance at* 5.0 *sec.*

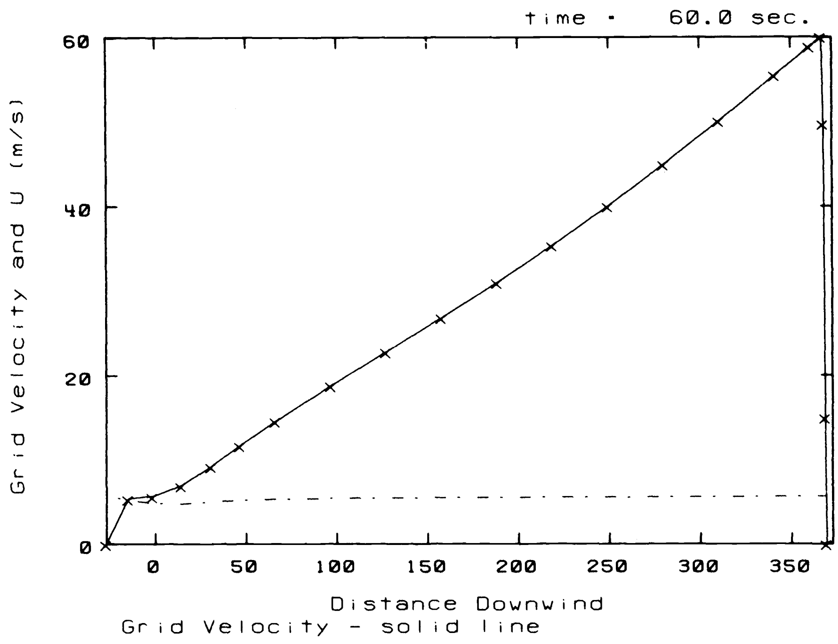


FIG. 2. *Advection and grid velocities versus distance at* 60.0 *sec.*

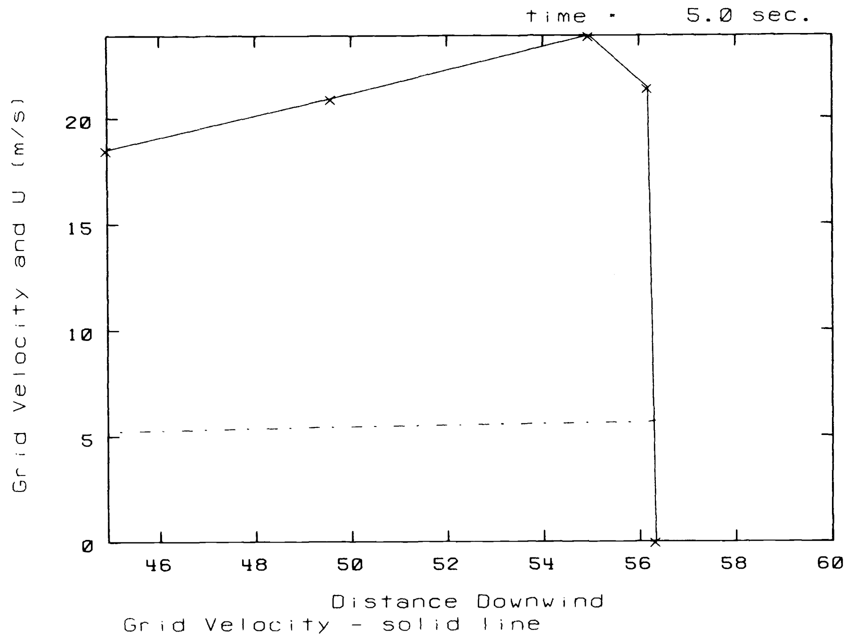


FIG. 3. *Detailed advection and grid velocities versus distance at* 5.0 *sec.*

of 0.98 cm/sec. Considering the length scales involved, both the width and the rate of broadening of this front are negligible.

Figures 5 and 6 depict the SLAB-averaged natural gas concentrations versus downstream distance at 5 and 60 seconds. Figures 7 and 8 depict the SLAB-averaged height of the cloud versus downstream distance. Both the grid velocity and regridding

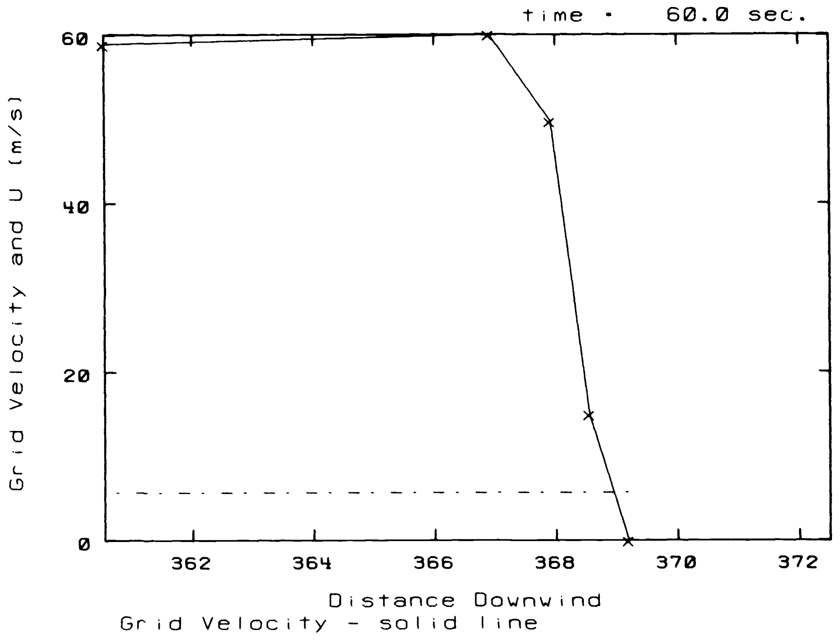


FIG. 4. *Detailed advection and grid velocities versus distance at* 60.0 *sec.*

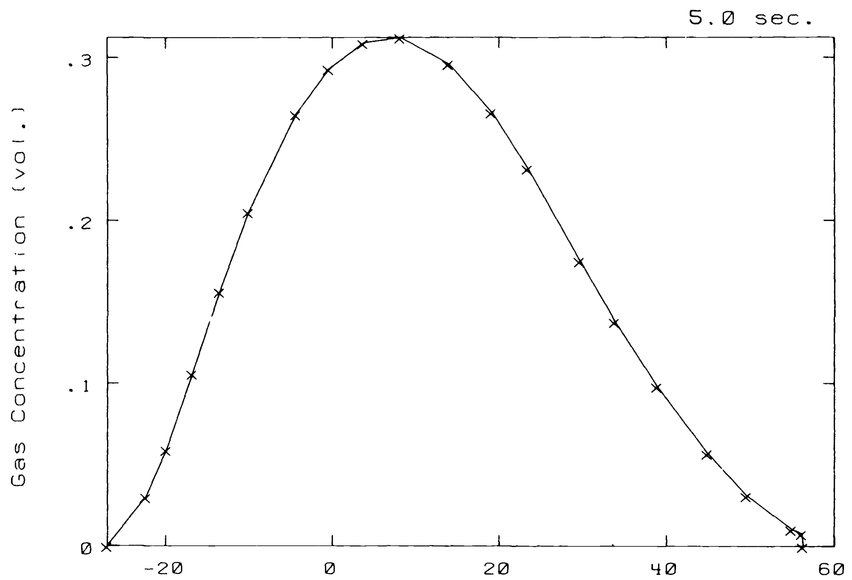


FIG. 5. *Gas concentration versus distance at* 5.0 *sec.*

phases of the calculation placed equal weights on the concentration of the various dependent variables. Early into the spill, the extrema over the spill site, $x = -27$ m to $x = 27$ m, are well resolved. However, at 60 seconds into the spill, although the height over the spill point is well resolved, the maximum in gas concentration needs more resolution. Air entrainment is continuously diluting the LNG vapor cloud. This resolution problem may indicate that more points are needed or that in a detailed kinetic

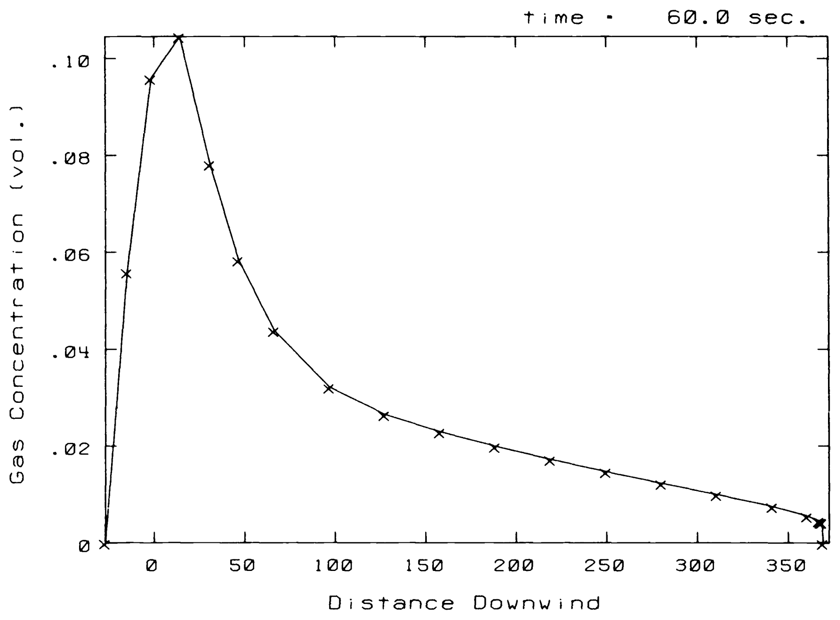


FIG. 6. *Gas concentration versus distance at* 60.0 *sec.*

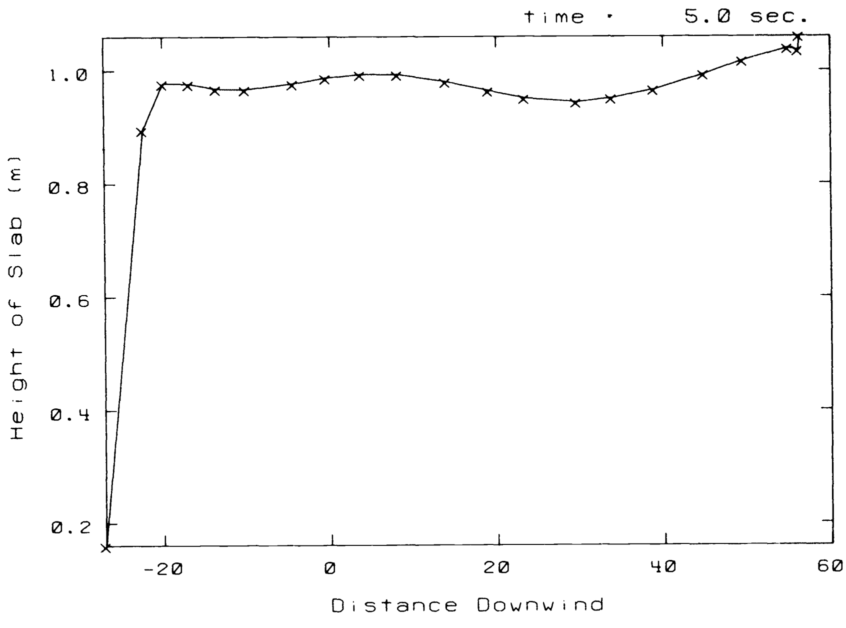


FIG. 7. *Height of cloud versus distance at* 5.0 *sec.*

scheme the best resolution will occur if each dependent variable carries its own grid. Note that the plotting routine connects points linearly.

Figures 9 and 10 depict the five points nearest the right-hand boundary. Because a very small amount of physical diffusion and second-order temporal diffusion were incorporated to avoid true mathematical discontinuities, a parabolic set of PDEs require boundary conditions at both the left- and right-hand boundary. From the compatibility

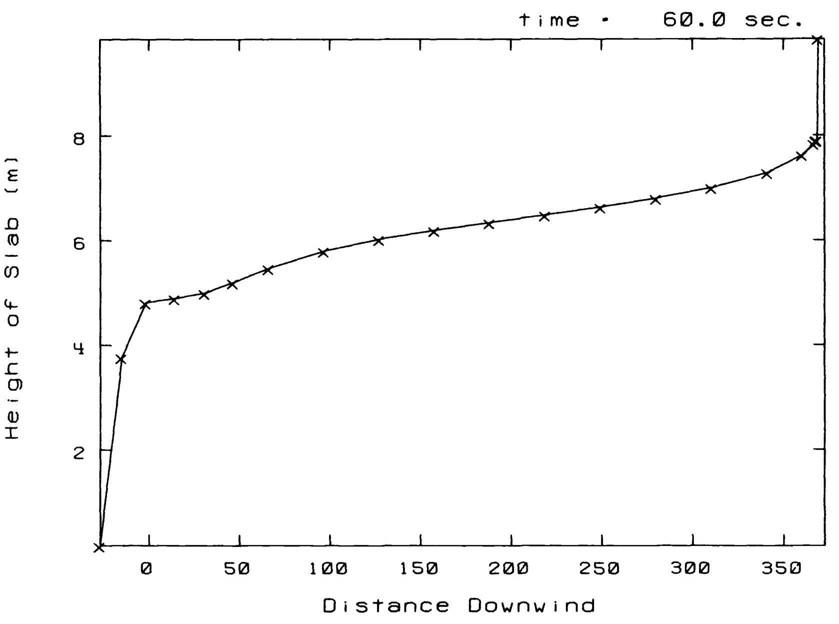


FIG. 8. *Height of cloud versus distance at* 60.0 *sec.*

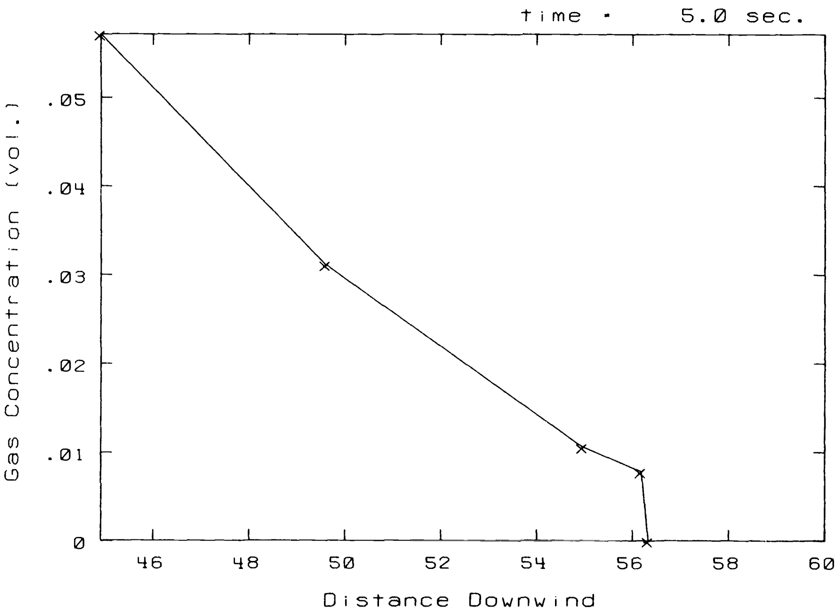


FIG. 9. *Detailed gas concentration versus distance at* 5.0 *sec.*

equations, and using the method of characteristics (see Hedstrom [10]), five compatibility relations were derived. The sixth relation was obtained by specifying the concentration to be zero at the right-hand boundary. Because of a small amount of ground friction and ground heating velocity through the source terms, a steepening front develops at the right-hand boundary. In a previous paper using PDECOL [17], in a fixed frame, it was necessary to use, on the average, a diffusion about 100,000 larger

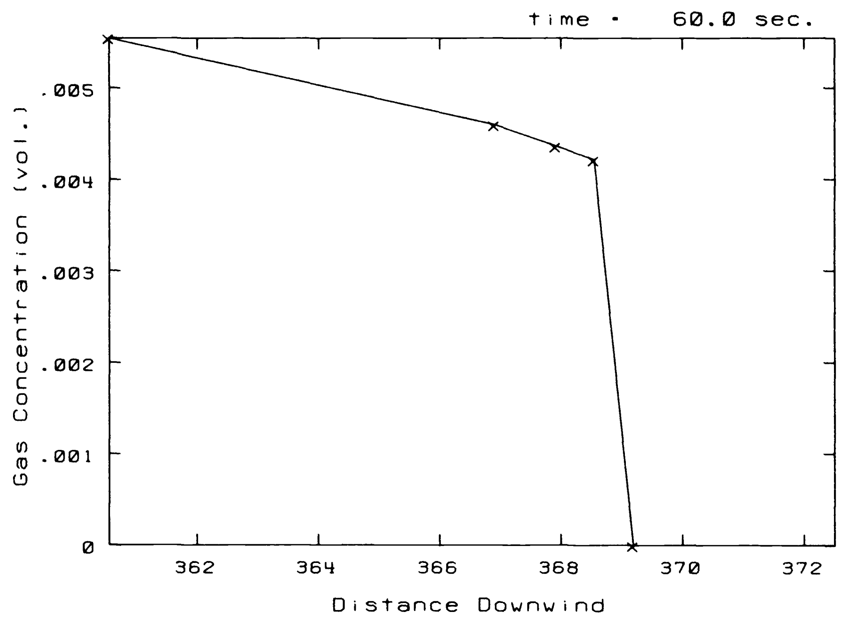


FIG. 10. *Detailed gas concentration versus distance at* 60.0 *sec.*

than used in this calculation in order to avoid destabilizing numerical oscillations. However when the grids are permitted to move, the principal mechanism of diluting the natural gas vapor cloud is air entrainment which is approximately 10,000 times larger than the diffusive processes in the moving finite difference (MFD) scheme. Typical values of first and second derivatives were 3,500 and 300,000 respectively. From 5 to 60 seconds the steep front has broadened at a mean rate of 0.98 cm/sec due to the small amount of physical and numerical diffusion in the MFD scheme.

5. Concluding remarks. The MFE scheme of Miller [16], [17] and Gelinas et al. [9] have shown that very accurate solutions to difficult physical problems can be achieved. This work has found some simplifications to the MFE approach and has perhaps made it easier to implement.

The basic concept for determining grid velocities is that a moving frame is sought which, in the least-squares sense, transforms a system of PDEs into ODEs. At a shock, this frame is defined to move at the shock velocity which is determined by the Rankine–Hugoniot conditions. Somewhat analogous jump relations can be used at flame fronts (see Lewis and von Elbe [13]).

As with the MFE scheme of Miller [16], [17] and Gelinas [9], regularization is necessary for the problem of vanishing small gradients, grid points that come within the minimum separation distance, and for preventing grid points from accumulating at a shock. However, rather than using penalty functions that modified the grid velocity, alternative methods of regularization were used.

Optimally, grids at a shock should move at the shock speed. A penalty function that retards the grid motion to prevent points from coming within some minimum separation distance forces the dependent-variable time-marching scheme to behave more like an Eulerian calculation on a fine grid. To prevent the uncontrolled growth of numerical instabilities, the time step must be severely cut.

Alternative methods of regularization were used, abandoning the penalty function approach. First, if a vanishingly small gradient is encountered at the position x_j, the

Gaussian elimination procedure is stopped. The grid velocity v_j is taken to be a weighted average of its neighbors, the matrix equation is modified and the Gaussian elimination process is permitted to continue.

Second, violations of the minimum grid separation criterion are handled in a remapping procedure before the next time step loop. Because the shock spread by only 2 parts per hundred thousand, in 60 seconds of simulation time, it was concluded that mapping had a small impact in dissipating the shock.

Third, remapping was also used to prevent points from accumulating near a shock by equipartitioning the third spatial derivative. In addition, for the application under consideration, the dependent variable and the grid velocities need not be solved simultaneously, but sequentially. Finally, the regridding procedure uses a minimal number of adjustable constants. No fine tuning is used.

Using the grid selection criterion (10)–(11), shocks are adequately resolved within a preselected minimum point separation. At the shock, grid points move at the shock velocity. However, if the grid velocities are set to zero, instabilities quickly developed if the time steps are not reduced and more numerical dissipation is not added.

The above mentioned observation at the shock front is not thoroughly understood. It is speculated that because the PDEs are in a frame moving at the shock velocity the PDEs behave as quasi-stationary ODEs with source terms that are solved to sufficient accuracy even by the second-order time-marching scheme.

Extensions into higher dimensions are feasible. However, it is highly recommended that one should have an additional degree of freedom by allowing the number of grid points to vary. In two or three dimensions, Berger's [1] adaptive mesh refinement scheme, which permits successively finer grids to overlay coarser grids in both translated and rotational degrees of freedom, can be merged with moving grid methods.

The authors concur with Miller [16], [17] and Gelinas et al. [9] that the extra degree of freedom of grid motion and resolution of physical processes to their proper length scales is definitely a very economical method for producing results determined by the physics rather than by numerics.

Appendix A. The SLAB model. The SLAB model treats the dispersion of a well-defined cloud of LNG vapor of height h and half-width B. Inside the cloud, the properties are assumed to be uniform in the crosswind plane, so that they vary only with time and in the downwind direction. This assumption allows the cloud to be described in terms of SLAB or layer-averaged properties, such as

$$\text{(A1)} \qquad \bar{\rho}(x) = \frac{1}{Bh} \int_0^B dy \int_0^h dz\, \rho(x, y, z)$$

for the cloud density. The variable x refers to the downwind distance, y refers to the horizontal crosswind distance, and z refers to the height above the ground. The assumption of uniformity justifies the use of the approximation that the average of the product is equal to the product of the averages (i.e., $\overline{\rho U} = \bar{\rho} \cdot \bar{U}$) in deriving the conservation equations. Cloud dispersion is assumed to occur because of the entrainment of air at the top and sides of the cloud and due to gravity spread, which is treated using the hydrostatic approximation. At the cloud/ground interface, heat and momentum exchange are also allowed to occur. The PDEs are properly defined for nonzero h and B.

The above assumptions and approximations are used to derive the layer-averaged conservation equations for mass, LNG vapor, momentum, and energy. These equations, along with a rate equation for the cloud half-width, are:

$$\frac{\partial \Phi}{\partial t}+\frac{\partial F}{\partial x}=S, \tag{A2}$$

where

$$\Phi=[m, BB, ml, \varepsilon, PX, PY]^{\dagger}, \tag{A3}$$

$$F=[PX, \{BB\cdot PX/m\}, \{PX\cdot ml/m\}, \{\varepsilon\cdot PX/m\}, \{PX^2/m+P\}, \{PX\cdot PY/m\}]^{\dagger}, \tag{A4}$$

and

$$S=[S_m, S_{BB}, S_{ml}, S_e, S_{PX}, S_{PY}]^{\dagger} \tag{A5}$$

where

$$S_m=S_s+S_a, \tag{A6}$$

$$S_s=\rho_s w_s B_s, \tag{A7}$$

$$S_a=\rho_a[V_e h+w_e BB/m], \tag{A8}$$

$$S_{BB}=\left(\frac{BB}{m}\right)\cdot S_m+2\cdot PY+m\cdot V_e, \tag{A9}$$

$$S_{ml}=S_s, \tag{A10}$$

$$S_e=C_{pa}T_a\cdot S_a+C_{ps}T_s\cdot S_s+QJ+\left(\frac{BB}{m}-B_s\right), \tag{A11}$$

$$\begin{aligned} S_{PX}=U_a\cdot S_a+U_s\cdot S_s-f_x\cdot\left(\frac{BB}{m}-B_s\right)\\ +g\cdot\frac{BB}{m}\cdot h\cdot(\rho-\rho_a)\cdot\sin\alpha, \end{aligned} \tag{A12}$$

$$\begin{aligned} S_{PY}=V_a\cdot S_a+V_s\cdot S_s-f_y\cdot\left(\frac{BB}{m}-B_s\right)\\ +\frac{1}{2}\cdot g\cdot h^2\cdot(\rho-\rho_a), \end{aligned} \tag{A13}$$

where

$$\rho=\rho_a\cdot C_{pa}\cdot T_a\cdot\left(\frac{m}{\varepsilon}\right)\cdot Y\left(\frac{ml}{m}\right), \tag{A14}$$

$$h=\frac{m^2}{(BB\cdot\rho)}, \tag{A15}$$

and

$$P=\frac{1}{2}gh^2\frac{BB}{m}(\rho-\rho_a),$$

where α = the angle due to elevation change in the downwind distance,

$$y\left(\frac{ml}{m}\right)=\frac{\{1+(C_{ps}/C_{pa}-1)\cdot ml/m\}}{\{1+(M_a/M_s-1)\cdot ml/m\}}, \tag{A16}$$

(A17) $$\varepsilon = C_p \cdot m \cdot T; \qquad C_p = C_{pa} + (C_{ps} - C_{pa}) \cdot \frac{ml}{m}.$$

The above equations, together with the ideal gas approximation for the equation of state and specific heat form the SLAB model. The main cloud variables are: the total mass (per unit length) m, and the product of mass and half-width BB, the thermal energy (per unit length) ε, mass of species ml (per unit length), momentum (per unit length) in the direction of the wind PX, and the mean crosswind cloud momentum (per unit length) PY. The bar over a quantity to designate a layer average has been dropped, since it is understood that all quantities are averaged in this manner. Other parameters are the acceleration due to gravity g, the vertical NG source velocity W_s, the source half-width B_s, the specific heat C_{pi}, the molecular weight M_i ($i = a$ or s), the vertical and horizontal entrainment rates w_e, and v_e, and the surface momentum and heat fluxes f, and QJ. Downwind, crosswind, and vertical velocities are denoted by subscripted u's. v's, and w's, respectively. The subscripts s and a designate an NG source-related property, or an ambient air property, respectively.

The entrainment rate equations have been modified from those proposed by Zeman [19]. For the present, the entrainment rates are taken to be constants whose values are chosen to be average values of those calculated.

Appendix B. Limitation of the PDE domain. In a previous paper [6], the six coupled, nonlinear, hyperbolic PDEs of the SLAB model were solved using PDECOL [17] computer software package. PDECOL used finite element collocation methods based on piecewise continuous polynomials for the spatial discretization, and stiff implicit time integration techniques. Because the solutions can involve the occurrence of steep fronts at the cloud edges, it was necessary to smooth out initial conditions and to add a rather large diffusion term to each PDE. These procedures smoothed the leading and trailing edges of the cloud, overcoming the numerical instabilities that often occurred. Although the resultant degree of smoothing was physically reasonable for the problems treated, it might be excessive for a wide range of problems where a greater propensity to instabilities might exist. In addition, running times using PDECOL were thought to be longer than could be achieved by the new method.

Another problem that would probably be present with any packaged integrator, was that the grid domain had to extend to all relevant portions of the cloud for all times of interest. Thus, at early times the ends of the grids extend well beyond the edges of the cloud, with the end portions being pure air. This resulted in unnecessary calculations at early times, and it was not clear that the true edges of the cloud were being treated properly, since the PDEs are defined only within the cloud.

Therefore, it was decided to create an integrator specifically designed for the problem. The only diffusion permitted is the "molecular" diffusion taken to be a constant, $D = 2.2\ 10^{-5}\ \text{m}^2/\text{sec}$, adding to each PDE, casting the purely hyperbolic set of PDEs into a set of parabolic equations that are almost hyperbolic. Among other features, the new code would be suitable for hyperbolic-like equations, not prone to instabilities, and constructed to perform integration only within the cloud. It is also intended as a model of an integrator for PDEs of higher dimensions. If $X_L(t)$ and $X_R(t)$ represent the left- and right-hand edges of the LNG cloud, a new coordinate ζ can be defined as

(B1) $$\zeta = \frac{(X - X_L(t))}{(X_R(t) - X_L(t))},$$

where the domain of dependence of the sytem of PDEs is $0 \leqq \zeta \leqq 1$. For many problems, X_L is fixed during the spill, but $X_R(t)$ moves with particle velocity at the edge of the cloud i.e.,

$$\frac{dX_R}{dt} = U(X_R). \tag{B2}$$

Consequently, in the ζ coordinate system the PDEs have the following forms

$$\frac{\partial \Phi}{\partial t} + \frac{\partial \zeta}{\partial x}\frac{\partial F}{\partial \zeta} - \frac{\partial \zeta}{\partial t}\frac{\partial \Phi}{\partial \zeta} = S(\Phi) + D \cdot \left(\frac{\partial \zeta}{\partial x}\right)^2 \cdot \frac{\partial^2 \Phi}{\partial \zeta^2}, \tag{B3}$$

which is a system of six coupled stiff nonlinear PDEs.

REFERENCES

[1] M. BERGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, Ph.D. dissertation, Stanford Univ., Stanford, CA, 1982.

[2] R. B. BIRD, W. E. STEWART AND E. N. LIGHTFOOT, *Transport Phenomena*, John Wiley, New York, 1965.

[3] R. COURANT AND K. O. FRIEDRICHS, *Supersonic Flow and Shock Wωves*, Interscience, New York 1948.

[4] S. F. DAVIS AND J. E. FLAHERTY, *An adaptive finite element method for initial-boundary value problems for partial differential equations*, this Journal, 3 (1982), pp. 6–27.

[5] H. A. DWYER, M. D. SMOOKE AND R. J. KEE, Report SAND82-8661 Sandia National Laboratory, Albuquerque, NM, 1982.

[6] D. L. ERMAK, S. T. CHAN, D. L. MORGAN AND L. K. MORRIS, *A comparison of dense gas dispersion with the Burro series* LNG *spill test results*, J. Hazardous Materials, 6 (1982), pp. 129–160.

[7] B. FORNBERG, *On the instability of leap-frog and Crank–Nicolson approximations of a nonlinear partial differential equation*, Math. Comp. 27 (1973), pp. 45–57.

[8] G. E. FORSYTHE, M. A. MALCOLM AND C. B. MOLER, *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[9] R. J. GELINAS, S. K. DOSS AND K. MILLER, *The moving finite element method: applications to general partial differential equations with multiple large gradients*, J. Comp. Phys., 40 (1981), pp. 202–249.

[10] G. W. HEDSTROM, *Nonreflecting boundary conditions for nonlinear hyperbolic systems*, J. Comp. Phys., 30 (1979), pp. 222–237.

[11] S. S. HU AND W. E. SCHIESSER, *Advances in Computer Methods for Partial Differential Equations* IV, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, New Brunswick, NJ, 1981.

[12] E. J. KANSA, *An algorithm for multidimensional combusting flow problems*, J. Comp. Phys., 42 (1981), pp. 152–194.

[13] B. LEWIS AND G. VON ELBE, *Combustion, Flames and Explosions of Gases*, Academic Press, New York 1961.

[14] N. K. MADSEN AND R. F. SINCOVEC, *Algorithm* 540 PDECOL, *general collocation software for partial differential equations*, ACM Trans. Math. Software, 5 (1979), pp. 326–351.

[15] A. MAJDA AND S. OSHER, *A systematic approach for correcting nonlinear instabilities. The Lax–Wendroff scheme for scalar conservation laws*, Numer. Math., 30 (1978), pp. 429–452.

[16] K. MILLER AND R. MILLER, *Moving finite elements*, I, SIAM J. Numer. Anal., 18 (1981), pp. 1019–1032.

[17] K. MILLER, *Moving finite elements*, II, SIAM J. Numer. Anal., 18 (1981), pp. 1033–1057.

[18] A. B. WHITE, JR., *On the numerical solution of initial/boundary value problems in one space dimension*, SIAM J. Numer. Anal., 19 (1982), pp. 683–697.

[19] O. ZEMAN, *The dynamics and modeling of heavier-than-air cold gas releases*, Atmospheric Environment, 16 (1982), p. 741.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

THE USE OF SINGULAR FUNCTIONS FOR THE APPROXIMATE CONFORMAL MAPPING OF DOUBLY-CONNECTED DOMAINS*

N. PAPAMICHAEL† AND C. A. KOKKINOS†‡

Abstract. Let f be the function which maps conformally a given doubly-connected domain onto a circular annulus. We consider the use of two closely related methods for determining approximations to f of the form

$$f_n(z) = z \exp\left\{\sum_{j=1}^{n} a_j u_j(z)\right\},$$

where $\{u_j\}$ is a set of basis functions. The two methods are respectively a variational method, based on an extremum property of the function

$$H(z) = f'(z)/f(z) - 1/z,$$

and an orthonormalization method, based on approximating the function H by a finite Fourier series sum.

The main purpose of the paper is to consider the use of the two methods for the mapping of domains having sharp corners, where corner singularities occur. We show, by means of numerical examples, that both methods are capable of producing approximations of high accuracy for the mapping of such "difficult" doubly-connected domains. The essential requirement for this is that the basis set $\{u_j\}$ contains singular functions that reflect the asymptotic behavior of the function H in the neighborhood of each "singular" corner.

Key words. conformal mapping, doubly-connected domains, Bergman kernel

1. Introduction.

Let Ω be a finite doubly-connected domain with boundary $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$ in the complex z-plane, where $\partial\Omega_i$, $i = 1, 2$, are closed Jordan curves. We assume that $\partial\Omega_i$, $i = 1, 2$, are respectively the inner and outer components of $\partial\Omega$, and that the origin 0 lies in the "hole" of Ω, i.e. $0 \in \text{Int}(\partial\Omega_1)$.

Let ζ be a fixed point in $\bar{\Omega} = \Omega \cup \partial\Omega$, and let

$$w = f(z), \tag{1.1}$$

be the function which maps conformally Ω onto the circular annulus

$$R = \{w : r_1 < |w| < r_2\}, \tag{1.2}$$

so that $\partial\Omega_i$, $i = 1, 2$, correspond respectively to $|w| = r_i$, $i = 1, 2$, and

$$f(\zeta) = \zeta. \tag{1.3}$$

As is well known, this mapping exists uniquely and the ratio of the two radii, i.e. the number

$$M = r_2/r_1 > 1, \tag{1.4}$$

is the so-called conformal modulus of Ω. This number determines completely the conformal equivalence class of the domain Ω.

The conformal map defined by (1.1) is of considerable practical interest and has important applications in, for example, fluid mechanics, electrostatics and stress analysis. Several such applications to specific physical problems are noted in the survey article by Laura [8]. A more general application, concerns the computer generation of orthogonal curvilinear coordinate systems for the finite difference solution of partial

* Received by the editors March 19, 1982, and in revised form May 3, 1983.

† Department of Mathematics, Brunel University, Uxbridge, England UB8 3PH.

‡ On study leave from the National Technical University of Athens, Greece.

differential equations. This important application of (1.1) has received much interest recently; see e.g. the review paper by Thompson et al. [15].

In the present paper we consider the use of two closely related numerical methods for determining approximations to the mapping function f of the form

$$f_n(z) = z \exp\left\{ \sum_{j=1}^{n} a_j u_j(z) \right\}, \tag{1.5}$$

where $\{u_j(z)\}$ is an appropriate set of basis functions. The two methods are respectively a variational method (VM), based on an extremum property of the function

$$H(z) = f'(z)/f(z) - 1/z, \tag{1.6}$$

and an orthonormalization method (ONM), based on approximating the function H by a finite Fourier series sum. The VM is described with full theoretical details in Gaier [4], whilst the ONM emerges easily from the theory contained in [1], [4] and [11]. The two methods resemble respectively the well-known Ritz and Bergman kernel methods for the mapping of a simply-connected domain onto the unit disc. In fact, the two numerical techniques of the present paper can be regarded as generalizations, to the mapping of the doubly-connected domains, of the Ritz and Bergman kernel procedures studied recently in [12].

The general objectives of the present paper are as follows. To give a summary of the theoretical results on which the VM and the ONM are based, to describe the two numerical techniques and to present a number of illustrative numerical examples. However, our main purpose is to consider the use of the two methods for the mapping of domains involving sharp corners, where branch point singularities occur. For this reason, most of the numerical examples considered in this paper concern the mapping of such difficult domains.

The numerical results given in § 5, as well as results of other numerical experiments not presented in this paper, indicate that both the VM and the ONM are capable of producing approximations of high accuracy. More precisely, our results show that high accuracy is achieved when the domain under consideration is $2n$-fold symmetric, with $n \geqq 2$, provided that the basis set, used for approximating the function H, contains singular functions that reflect the asymptotic behavior of H in the neighborhood of a corner where a singularity occurs. Such a basis can always be constructed, in a manner similar to that used for constructing the basis for the Ritz and the Bergman kernel methods in [12], by introducing appropriate singular functions into the set

$$\{z^j\}_{j=-\infty}^{\infty}, \qquad j \neq -1. \tag{1.7}$$

2. Preliminary results. We let $\mathscr{L}_2(\Omega)$ be the Hilbert space of all square integrable functions which are analytic and possess a single-valued indefinite integral in Ω, and denote the inner product of $\mathscr{L}_2(\Omega)$ by $(\cdot, \cdot)$, i.e.

$$(g_1, g_2) = \iint_\Omega g_1(z)\overline{g_2(z)}\, dx\, dy. \tag{2.1}$$

We also let

$$A(z) = \log f(z) - \log z, \tag{2.2}$$

where f is the function (1.1) mapping Ω onto the circular annulus R. Then, the function A is analytic and single-valued in Ω, and its derivative

$$H(z) = A'(z), \tag{2.3}$$

is the function (1.6). Clearly, $H(z) \neq 0$, $z \in \Omega$, unless Ω is itself a circular annulus with its centre at the origin.

In order to present the results on which the VM is based we let

$$\begin{aligned} \mathcal{H}^{(1)}(\Omega) &= \{u(z)\colon u \in \mathcal{L}_2(\Omega) \text{ and } (u, H) = 1\}, \\ \mathcal{H}^{(0)}(\Omega) &= \{v(z)\colon v \in \mathcal{L}_2(\Omega) \text{ and } (v, H) = 0\}, \end{aligned} \tag{2.4}$$

and, as in Gaier [4, p. 245], we consider the following variational problem.

Problem 2.1. To minimize

$$\|u\|^2 = \iint_\Omega |u(z)|^2 \, dx \, dy, \tag{2.5}$$

over all $u \in \mathcal{H}^{(1)}(\Omega)$.

The following results are proved in [4]:

R2.1. *Problem* 2.1 *has a unique solution* u_0.

R2.2. u_0 *is orthogonal to* $\mathcal{H}^{(0)}(\Omega)$, *i.e.* $u_0 \perp \mathcal{H}^{(0)}(\Omega)$.

R2.3. *The function H is related to* u_0 *by*

$$H(z) = u_0(z)/\|u_0\|^2. \tag{2.6}$$

It is of interest to note that the above results are all special cases of standard results of the theory of Hilbert spaces. This follows from the observations that $\mathcal{H}^{(1)}(\Omega)$ and $\mathcal{H}^{(0)}(\Omega)$ are respectively a closed convex subset and a closed subspace of $\mathcal{L}_2(\Omega)$; see e.g. [14].

In addition to R2.1–R2.3, the following two results, which are proved in [4, p. 250], are needed for the description of both the VM and the ONM.

R2.4. *For each function* $\eta \in \mathcal{L}_2(\Omega)$ *which is continuous on* $\partial\Omega = \partial\Omega_1 \cup \partial\Omega_2$

$$(\eta, H) = i \int_{\partial\Omega} \eta(z) \log |z| \, dz, \tag{2.7}$$

where H is defined by (2.2).

R2.5. *The modulus* $M = r_2/r_1$ *of* Ω *is related to the function H by*

$$\log M = \left\{ \frac{1}{i} \int_{\partial\Omega} \frac{1}{z} \log |z| \, dz - \|H\|^2 \right\} \Big/ 2\pi. \tag{2.8}$$

The result R2.4 is established easily after first expressing the inner product (η, H) as

$$(\eta, H) = \frac{1}{2i} \int_{\partial\Omega} \eta(z) \overline{A(z)} \, dz.$$

This is done by means of the Green's formula

$$(g_1, g_2') = \frac{1}{2i} \int_{\partial\Omega} g_1(z) \overline{g_2(z)} \, dz, \tag{2.9}$$

which is also needed for determining certain other inner products that occur in both the VM and the ONM. As is shown in Bergman [1, p. 96], formula (2.9) is valid for any functions g_1 and g_2 which are analytic in Ω and continuous on $\partial\Omega$. The result R2.5 is established by integration by parts after first applying (2.8) to the norm $\|H\|^2 = (H, H)$; see [4, pp. 250–251] for further details.

We point out that the assumptions concerning the continuity on $\partial\Omega$ of the functions g_1, g_2 in (2.9) and η in (2.7) can be replaced by somewhat weaker requirements. For example, it can be shown that both (2.7) and (2.9) are applicable to "singular" functions of the type considered in § 4.

3. The numerical methods. As was previously remarked the VM is due to Gaier [4, p. 249]. The method emerges by seeking the solution of the finite-dimensional counterpart of Problem 2.1, and resembles closely the Ritz method for the mapping of simply-connected domains. For this reason the VM details given below are similar to those used for the description of the Ritz method in [12].

Let $\{\eta_j(z)\}$ be a basis of $\mathscr{L}_2(\Omega)$ such that $\eta_1 \notin \mathscr{K}^{(0)}(\Omega)$, and denote by $\mathscr{K}_n^{(l)}(\Omega)$; $l = 0, 1$ the n-dimensional counterparts of $\mathscr{K}^{(l)}(\Omega)$, $l = 0, 1$, i.e.

(3.1) $$\mathscr{K}_n^{(l)}(\Omega) = \mathscr{E}_n \cap \mathscr{K}^{(l)}(\Omega), \qquad l = 0, 1,$$

where

(3.2) $$\mathscr{E}_n = \operatorname{span}(\eta_1, \eta_2, \cdots, \eta_n).$$

Then the set $\mathscr{K}_n^{(1)}(\Omega)$ is nonempty for any $n \geqq 1$, and the n-dimensional variational problem corresponding to Problem 2.1 can be stated as follows.

Problem 3.1. To minimize $\|u\|$ over all $u \in \mathscr{K}_n^{(1)}(\Omega)$.

The following results hold.

R3.1. *Problem* 3.1 *has a unique solution* u_n.

R3.2. *The minimal function* u_n *is characterized by the property* $u_n \perp \mathscr{K}_n^{(0)}(\Omega)$.

R3.3. $u_n \to u_0$ *almost uniformly in* Ω. *That is, from* (2.6),

(3.3) $$u_n(z)/\|u_n\|^2 \to H(z),$$

almost uniformly in Ω. (*By almost uniform convergence we mean convergence in every compact subset of* Ω.)

The results R3.1–R3.2 are of course the finite dimensional counterparts of R2.1–R2.2. Like R2.1 and R2.2, they are particular cases of standard results from the theory of Hilbert spaces. R3.3 is a direct consequence of the fact that in $\mathscr{L}_2(\Omega)$ convergence in the norm implies almost uniform convergence, and is established after first showing that

$$\lim_{n \to \infty} \|u_n - u_0\| = 0.$$

Let

(3.4) $$\gamma_j = (H, \eta_j), \qquad j = 1, 2, \cdots,$$

and recall that $\gamma_1 \neq 0$. Then, since

$$\mathscr{K}_n^{(0)}(\Omega) = \{u \in \mathscr{E}_n : (H, u) = 0\},$$

the set

(3.5) $$\bar{\gamma}_1 \eta_j(z) - \bar{\gamma}_j \eta_1(z), \qquad j = 2, 3, \cdots, n,$$

is a basis of $\mathscr{K}_n^{(0)}(\Omega)$. Therefore, if we let

(3.6) $$u_n(z) = \sum_{j=1}^{n} c_j \eta_j(z),$$

the orthogonality property of R3.2 and the condition $(H, u_n) = 1$ lead immediately to the linear system

$$\sum_{j=1}^{n} \{\gamma_1(\eta_j, \eta_i)-\gamma_i(\eta_j, \eta_1)\}c_j=0 \qquad i=2,3,\cdots,n, \tag{3.7}$$
$$\sum_{j=1}^{n} \bar{\gamma}_j c_j=1,$$

where the inner products γ_i, $i=1,2,\cdots,n$ are known by means of (2.7). Thus the coefficients c_j in (3.6) may be determined by solving the $n\times n$ complex linear system (3.7). Then, because of R3.3,

$$H_n(z)=u_n(z)/\|u_n\|^2, \tag{3.8}$$

gives the nth VM approximation to the function $H(z)=A'(z)$ and thus, from (2.2),

$$f_n(z)=z\exp\left\{\int_{\zeta}^{z} H_n(t)\,dt\right\}, \qquad \zeta\in\bar{\Omega}, \tag{3.9}$$

is the nth VM approximation to the mapping function f. Also, from (2.8),

$$M_n=\exp\left\{\left(\frac{1}{i}\int_{\partial\Omega}\frac{1}{z}\log|z|\,dz-\|H_n\|^2\right)\Big/2\pi\right\}, \tag{3.10}$$

is the nth VM approximation to the modulus M of Ω. In fact, it can be easily verified that M_n gives an upper bound to M.

In the ONM the approximation to the mapping function f is determined after first approximating the function H by a finite Fourier series sum. The method emerges easily from the theory contained in [11, p. 373], [1, p. 102] and [4, p. 249].

Let $\{\eta_j^*(z)\}$ be an orthonormal basis of $\mathscr{L}_2(\Omega)$. Then the function H has the Fourier series expansion

$$H(z)=\sum_{j=1}^{\infty}\beta_j\eta_j^*(z), \tag{3.11}$$

where the Fourier coefficients $\beta_j=\overline{(\eta_j^*, H)}$ are known by means of (2.7). The series (3.11) certainly converges in the norm of $\mathscr{L}_2(\Omega)$ and, as in the case of R3.3, this norm convergence implies almost uniform convergence in Ω.

Given a basis $\{\eta_j(z)\}$ of $\mathscr{L}_2(\Omega)$, the above results suggest the following procedure for obtaining a numerical approximation to the mapping function f. The set $\{\eta_j(z)\}_{j=1}^n$ is orthonormalized by means of the Gram–Schmidt process to give the orthonormal set $\{\eta_j^*(z)\}_{j=1}^n$. The series (3.11) is then truncated after n terms to give the approximation

$$H_n(z)=\sum_{j=1}^{n}\beta_j\eta_j^*(z), \qquad \beta_j=\overline{(\eta_j^*, H)}, \qquad j=1,2,\cdots,n \tag{3.12}$$

to the function H. Finally, with this H_n, the equations (3.9) and (3.10) give respectively the nth ONM approximation to the mapping function f and to the modulus M of Ω.

We end this section by observing that Problem 2.1, with the function H defined by (2.3), is a special case of a more general variational problem which is considered fully in Gaier [4, Chap. V]. More specifically the results R2.1–R2.3, together with the corresponding finite-dimensional results of this section, hold for any finitely-connected domain and any function $H\in\mathscr{L}_2(\Omega)$, such that $H(z)\neq 0$, $z\in\Omega$. An interesting example of this is the case where Ω is simply-connected and H is taken to be the Bergman kernel function of Ω. In this case, the results R2.1–R2.3 constitute the so-called property of minimum area, and the variational method of the present section reduces to the Ritz method for the mapping of simply-connected domains. Other choices of H lead to other interesting results concerning the mapping of multiply-connected domains; see Gaier [4, Chap. V] for further details.

4. Choice of basis. A serious drawback of both the VM and the ONM is that severe loss of accuracy may occur during the computation, due to ill-conditioning of the matrix in 3.7 or to numerical instability of the Gram–Schmidt process. For this reason, the success of the methods depends strongly on the rate of convergence of the approximating series, and this in turn depends on the choice of basis functions $\{\eta_j(z)\}$.

An obvious choice of basis is the set

$$\{z^j\}_{j=-\infty}^{\infty}, \qquad j \neq -1. \tag{4.1}$$

This set is complete in $\mathscr{L}_2(\Omega)$ and provides a computationally convenient basis for both the VM and the ONM. Unfortunately, the situation regarding the use of (4.1) is exactly the same as that observed in [10], [12] and [13], in connection with the use of the set $\{z^j\}_{j=0}^{\infty}$ for the mapping of simply-connected domains by means of the Ritz and the Bergman kernel methods. That is, due to the presence of singularities of the function H in the complement of Ω, the convergence of the resulting approximating series is often extremely slow. Because of this, the use of (4.1) does not always lead to approximations of acceptable accuracy. In order to overcome this difficulty, we adopt the procedure first proposed in [10], in connection with the choice of basis for the Bergman kernel method. This involves the use of an "augmented basis," formed by introducing into the set (4.1) singular functions that reflect the main singular behavior of H in compl (Ω).

In [10], [12] and [13], the augmented basis for the mapping of simply-connected domains is formed by considering two types of singularities of the mapping function. These are either poles which lie close to the boundary or branch point singularities on the boundary itself. For the problems considered in [10], [12] and [13], the dominant poles of the mapping function can be determined, by using the symmetry principle, whenever the boundary of the simply-connected domain consists of straight line segments and circular arcs. Unfortunately, in the case of doubly-connected domains we do not know of a systematic way for determining the "poles" of f and the corresponding singularities of the function H, irrespective of the geometry of $\partial\Omega$. For this reason, in the present paper we construct the augmented basis by considering only the branch point singularities of H.

Branch point singularities are corner singularities. They occur, when due to the presence of a corner at a point $z_j \in \partial\Omega$, the asymptotic expansion of the mapping function f in the neighborhood of z_j involves fractional powers of $(z-z_j)$. The question regarding the choice of suitable basis functions for dealing with such singularities can be answered in exactly the same way as in [10], [12], and [13], by using the results of Lehman [9]. For this reason, we state below the formulae which define the singular functions, without repeating the details of their derivation.

Let part of the boundary $\partial\Omega$ consists of two analytic arcs Γ_1 and Γ_2 which meet at a point z_j and form there a corner of interior angle $\alpha\pi$, where $\alpha = p/q > 0$ is a fraction reduced to its lowest terms. (By interior angle we mean interior to the domain Ω.) Then, the asymptotic expansion of H involves terms which can be written in the form

$$\begin{array}{ll} (4.2) \\ (4.3) \end{array} \qquad \eta_{rj}(z) = \begin{cases} \{(1/z - 1/z_j)^{r-1}\}/z^2 & \text{if } z_j \in \partial\Omega_1, \\ (z-z_j)^{r-1} & \text{if } z_j \in \partial\Omega_2, \end{cases}$$

where

$$(4.4) \qquad r = k + l/\alpha, \qquad k = 0, 1, 2, \cdots, \qquad 1 \leqq l \leqq p.$$

Thus, if $p \neq 1$ a branch point singularity occurs at z_j. For this reason, the augmented basis is formed by introducing into the set (4.1) the first few singular functions of the sequences (4.2) or (4.3), corresponding to the first few fractional values of r.

We note that the singular functions defined by (4.3) are the same as those used in [10] and [12], for dealing with the branch point singularities of the interior mapping function for simply-connected domains. Similarly, the singular functions (4.2) are the same as those used in [13], in connection with the exterior mapping problem for simply-connected domains. This choice of singular functions ensures that the η_{rj} always have a single valued integral in Ω. We also note that a branch point singularity might occur at the point z_j even when $p = 1$. This happens because the asymptotic expansion of f might involve logarithmic terms; see [9] and [12, § 4.2]. However, such logarithmic singularities are never very serious and, for this reason, we do not consider them in the present paper. Finally, we note that if $z_j \in \partial\Omega_2$ and the arms Γ_1, Γ_2 of the corner are straight lines then the range (4.4) for the values of r in (4.3) may be replaced by

$$(4.5) \qquad r = l/\alpha, \qquad l = 1, 2, 3, \cdots;$$

see [10, § 2.2] and [12, § 4.2].

5. Computational details and numerical examples. Both the VM and ONM require the evaluation of inner products of the form (η_r, η_s) and (η, H). These are needed for determining the coefficients of the linear system (3.7) for orthonormalizing the set $\{\eta_j(z)\}$ by means of the Gram–Schmidt process, and for evaluating the Fourier coefficients in (3.12). Using Green's formula (2.9), the inner products (η_r, η_s) are expressed as

$$(5.1) \qquad (\eta_r, \eta_s) = \frac{1}{2i} \int_{\partial\Omega} \eta_r(z) \overline{\mu_s(z)}\, dz, \qquad \mu_s'(z) = \eta_s(z),$$

and the integrals in (5.1) are then computed by Gaussian quadrature, in exactly the same way as in [10], [12] and [13]. Similarly, each inner product (η, H) is computed by applying to the integral in (2.7) the Gaussian rule used for the evaluation of (5.1). When performing the quadrature, care must be taken to deal with integrand singularities that occur when, due to the presence of a corner at z_j, the basis set contains singular functions of the form (4.2) or (4.3). In the examples considered below, the arms of the corners are always straight line segments, and any integrand singularities are removed, as explained in [10, § 3], by choosing an appropriate parametric representation for $\partial\Omega$; see also [12, § 5] and [13, § 3].

In the VM, the complex linear system (3.7) is solved by using the NAG Library routine FO4ADF, which is based on Crout's factorization method. In the ONM, the orthonormalization is performed by means of the procedure used in [10], [12] and [13], in connection with the Bergman kernel method. This procedure is based on the standard Gram–Schmidt algorithm.

The approximation M_n to the modulus M of Ω is computed from (3.10), by applying to the integral the Gaussian rule used for the evaluation of the inner products (η_r, η_s) and (η, H).

An estimate of the maximum error in $|f_n(z)|$ is given by the quantity E_n, which is determined as follows. In each example, the fixed point ζ in (1.3) is taken to be a convenient point on the outer boundary $\partial\Omega_2$. Thus, in each case, the outer radius of the annulus is

$$r_2 = |\zeta|, \tag{5.2}$$

and r_2/M_n gives an approximation to the inner radius r_1. Hence, we may take the error estimate to be

$$E_n = \max\left\{\max_j \left| |f_n(z_{1j})| - r_2/M_n \right|, \max_j \left| |f_n(z_{2j})| - r_2 \right| \right\}, \tag{5.3}$$

where $\{z_{1j}\}$ and $\{z_{2j}\}$ are two sets of "boundary test points" on $\partial\Omega_1$ and $\partial\Omega_2$ respectively. We expect E_n to be a reasonable error estimate, because our numerical experiments indicate strongly that, in general, the approximation M_n is much more accurate than $|f_n(z)|$, $z \in \partial\Omega$; see e.g. the numerical results of Example 5.2.

In each example, the ONM results presented correspond to the approximation f_{Nopt}, where $n = Nopt$ is the "optimum number" of basis functions which gives maximum accuracy in the sense explained in [10, § 3]. That is, this number is determined by computing a sequence of approximations $\{f_n(z)\}$, where at each stage the number n of basis functions is increased by one. If at the $(n+1)$th stage the inequality

$$E_{n+1} < E_n \tag{5.4}$$

is satisfied, then the approximation $f_{n+2}(z)$ is computed. When for a certain value of n, due to numerical instability, (5.4) no longer holds then we terminate the process and take $n = Nopt$. Also, in order to safeguard against slow convergence, we do as in [13] and after $n = 19$ we begin to compute the ratios

$$q_M = E_{10+M}/E_M, \qquad M = 10, 20, \cdots.$$

If, for some $M, q_M > 0.5$ then we terminate the process at $n = 10+M$ and write $\overset{*}{Nopt} = 10 + M$.

For the presentation of the results we adopt a format similar to that used in [12]. That is, we denote the two methods respectively by VM/MB and ONM/MB or VM/AB and ONM/AB to indicate whether the "monomial basis" (4.1) or an "augmented basis" is used. For each example we list the singular functions, the boundary test points and the order of the Gaussian quadrature, which are used respectively for augmenting the set (4.1) for determining the error estimate (5.3) and for computing the inner products. As was previously remarked, if the basis set contains singular functions of the form (4.2) or (4.3) then the resulting integrand singularities in the inner products are removed by using special parametric representations for $\partial\Omega$. These representations are similar to those used in [10, § 3]. For this reason, we do no not list them here.

All computations were carried out on a CDC 7600 computer, using programs written in Fortran with single precision working. Single length working on the CDC 7600 is between 13 and 14 significant figures.

Example 5.1. Circle in square, Fig. 5.1.

$$\Omega = \{(x, y): |x| < 1, |y| < 1\} \cap \{z: |z| > a, a < 1\}.$$

Basis. This example does not involve corner singularities. For this reason, we do not need to use an augmented basis.

Because the domain has eightfold symmetry about the origin, the monomial basis set is taken to be

$$z^{k_j(2j+1)}, \qquad k_j = (-1)^{j+1}, \qquad j = 1, 2, 3, \cdots. \tag{5.5}$$

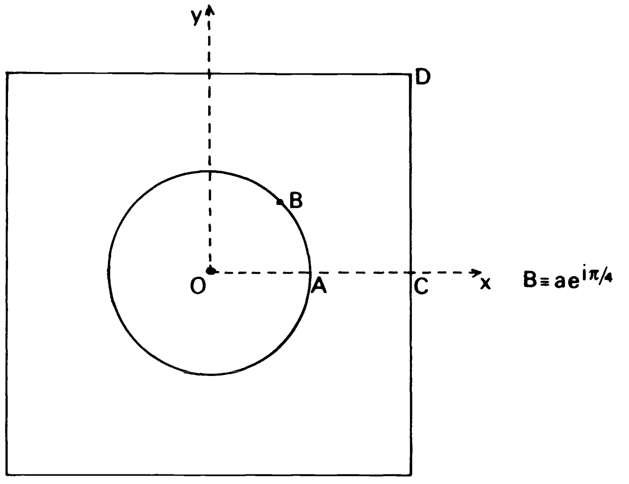


FIG. 5.1

Quadrature. Gauss–Legendre formula with 48 points along each quarter of the circle and each half side of the square.

Boundary test points. Because of the symmetry, we only consider points on AB and CD. On AB the points are defined by $z = ae^{i\tau}$, $\tau = 0(\pi/16)\pi/4$. On CD the points are equally spaced, in steps of 0.25, starting from C.

Numerical results.

		(i) $a = 0.2$	
ONM/MB:	$Nopt = 20$,	$E_{20} = 9.5\times10^{-12}$,	$M_{20} = 5.393\,525\,710\,616$.
VN/MB:		$E_{20} = 9.0\times10^{-12}$,	*
		(ii) $a = 0.4$	
ONM/MB:	$Nopt = 22$,	$E_{22} = 5.2\times10^{-12}$,	$M_{22} = 2.696\,724\,431\,230$.
VN/MB:		$E_{22} = 3.1\times10^{-12}$,	*
		(iii) $a = 0.8$	
ONM/MB:	$Nopt = 28$,	$E_{28} = 1.8\times10^{-10}$,	$M_{28} = 1.342\,990\,365\,599$.
VN/MB:		$E_{28} = 7.0\times10^{-11}$,	*

(* In each case the VM approximation to M agrees with the ONM approximation to the number of figures quoted.)

The numerical results of this example illustrate the remarkable accuracy that can be achieved by the VM/MB and the ONM/MB, when the domain under consideration is highly symmetric and does not involve corner singularities.

Accurate VN/MB approximations for the cases $a = 0.4$ and $a = 0.8$ have also been obtained by Gaier. His approximations to M are quoted to seven significant figures in [5], and agree perfectly with the approximations listed above. The approximation M_{22}, corresponding to the case $a = 0.4$, should also be compared with the value 2.696 727 given in [12]. This value is obtained by a method based on approximating the conformal map onto the unit disc, of the simply-connected domain bounded by the arc AB and the straight lines BD, DC and CA; see [12, Example 6.3].

Example 5.2. Square frame, Fig. 5.2.

Let

$$G_a = \{(x, y): |x| < a, |y| < a\}.$$

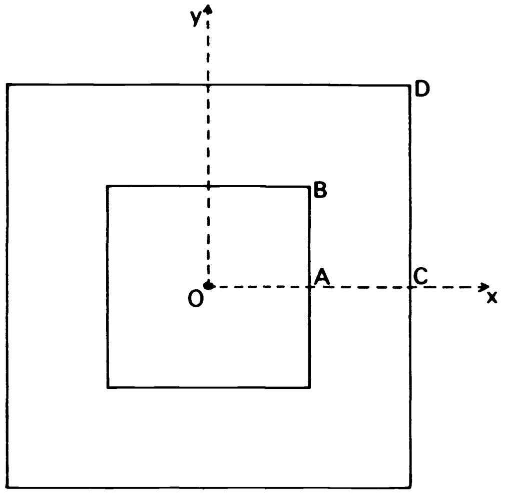


FIG. 5.2

Then

$$\Omega = G_1 \cap \operatorname{compl}(\bar{G}_a), \quad \text{with } a < 1.$$

Augmented basis. Let z_j, $j = 1, 2, 3, 4$, be respectively the four corners of the inner square. Then, the singular functions corresponding to the branch point singularities at the corners z_j, $j = 1, 2, 3, 4$, are respectively the functions $\eta_{rj}(z)$, $j = 1, 2, 3, 4$, given by (4.2) with

(5.6) $$r = k + 2l/3, \qquad k = 0, 1, \cdots, \quad 1 \leqq l \leqq 3.$$

Because of the eightfold symmetry the function $H(z)$ satisfies the property

$$e^{i\pi/2} H(e^{i\pi/2} z) = H(z).$$

For this reason, for each value of r, the four functions $\eta_{rj}(z)$ can be combined into the single function

(5.7) $$\tilde{\eta}_r(z) = \eta_{r1}(z) + \sum_{j=2}^{4} e^{i\theta_j} \eta_{rj}(z),$$

where the arguments θ_j, $j = 2, 3, 4$ are chosen so that

(5.8) $$e^{i\pi/2} \tilde{\eta}_r(e^{i\pi/2} z) = \tilde{\eta}_r(z).$$

It is important to observe that the constants θ_j in (5.7) depend on the branches used for defining the functions $\eta_{rj}(z)$. For this reason, great care must be taken when constructing symmetric singular functions of the form (5.7).

In this example the augmented basis is formed by introducing into the monomial set (5.5) the four singular functions (5.7) corresponding to the values $r = 2/3$, $4/3$, $5/3$, $7/3$.

Quadrature. Gauss–Legendre formula with 48 points along each side of the outer square and each half side of the inner square. In order to perform the integration accurately, the parametric representation of the inner square is chosen to be that used in [13, Example 3.2].

Boundary test points. Five equally spaced points on each of AB and CD.
Numerical results.

		(i) $a = 0.2$	
ONM/MB:	$\overset{*}{N}opt = 30$,	$E_{30} = 1.8 \times 10^{-2}$,	$M_{30} = 4.575\,2 \cdots$.
ONM/AB:	$Nopt = 24$,	$E_{24} = 1.1 \times 10^{-8}$,	$M_{24} = 4.570\,859\,677\,117$.
VM/AB:		$E_{24} = 1.1 \times 10^{-8}$,	$M_{24} = 4.570\,859\,677\,116$.
		Exact value of M	$= 4.570\,859\,677\,215$.
		(ii) $a = 0.5$	
ONM/MB:	$\overset{*}{N}opt = 30$,	$E_{30} = 4.3 \times 10^{-2}$,	$M_{30} = 1.856\,9 \cdots$.
ONM/AB:	$Nopt = 24$,	$E_{24} = 5.0 \times 10^{-8}$,	$M_{24} = 1.847\,709\,011\,217$.
VM/AB:		$E_{24} = 5.0 \times 10^{-8}$,	$M_{24} = 1.847\,709\,011\,216$.
		Exact value of M	$= 1.847\,709\,011\,236$.
		(iii) $a = 0.8$	
ONM/MB:	$\overset{*}{N}opt = 30$,	$E_{30} = 5.0 \times 10^{-2}$,	$M_{30} = 1.205\,2 \cdots$.
ONM/AB:	$Nopt = 26$,	$E_{26} = 3.7 \times 10^{-7}$,	$M_{26} = 1.201\,452\,809\,479$.
VM/AB:		$E_{26} = 4.1 \times 10^{-7}$,	$M_{26} = 1.201\,452\,809\,478$.
		Exact value of M	$= 1.201\,452\,809\,469$.

The exact values of M, listed above, were computed by using the exact formulae of Bowman [2] and [3, p. 104].

VM approximations to M have also been computed by Gaier and his students [6], who used as basis the set (5.5) augmented with the single singular function

$$1/\{z^{11/3}(z^4 + 4a^4)^{1/3}\}.$$

For the case $a = 0.5$, their approximation to M is 1.847 776. For the same case, by using an approximation to the conformal map of the quadrilateral $ABDC$ the method of [12] gives the value 1.847 719; see [12, Example 6.1].

Example 5.3. Rectangle in circle, Fig. 5.3.

Let

$$G_{ab} = \{(x, y): |x| < a < 1, |y| < b < 1\}.$$

Then

$$\Omega = \{z: |z| < 1\} \cap \text{compl}\,(\bar{G}_{ab}).$$

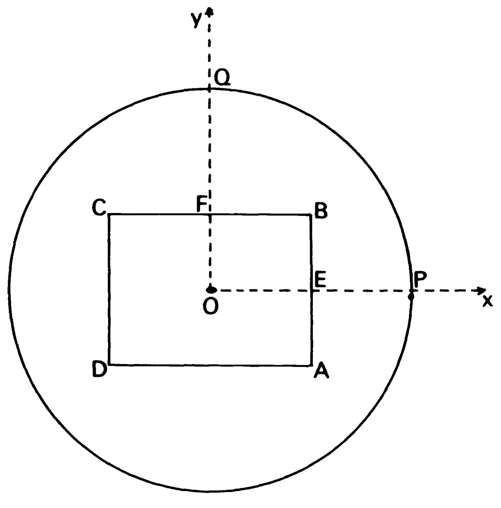


Fig. 5.3

Augmented basis. When $a = b, \Omega$ has eightfold symmetry about the origin and, for this reason, the monomial basis is taken to be the set (5.5).

When $a \neq b, \Omega$ has fourfold symmetriy about 0. Because of this, the function H satisfies

$$e^{i\pi}H(e^{i\pi}z) = H(z),$$

and the monomial basis is taken to be the set

$$z,\ z^{\pm(2j+1)}, \qquad j = 1, 2, 3, \cdots. \tag{5.9}$$

Let z_j, $j = 1, 2, 3, 4$, be respectively the corners A, B, C and D of the rectangle. Then, the singular functions corresponding to the branch point singularities at z_j, $j = 1, 2, 3, 4$, are respectively the functions $\eta_{rj}(z)$, $j = 1, 2, 3, 4$, defined by (4.2) with r given by (5.6).

When $a = b$, for each value of r, the four functions $\eta_{rj}(z)$ can be combined into a single function of the form (5.7)–(5.8). Similarly, because of the fourfold symmetry, when $a \neq b$, for each value of r, the four functions $\eta_{rj}(z)$ can be combined into the two functions

$$\tilde{\eta}_{rj}(z) = \eta_{rj}(z) + e^{i\theta_j}\eta_{r,j+2}(z), \qquad j = 1, 2, \tag{5.10}$$

where the arguments θ_j, $j = 1, 2$, are chosen so that

$$e^{i\pi}\tilde{\eta}_{rj}(e^{i\pi}z) = \tilde{\eta}_{rj}(z). \tag{5.11}$$

In this example we form the augmented basis, for the cases $a = b$ and $a \neq b$, by introducing respectively into the monomial sets (5.5) and (5.9) the four functions (5.7) and the eight functions (5.10) corresponding to the values $r = 2/3$, $4/3$, $5/3$, $7/3$.

Quadrature. Gauss–Legendre formula with 48 points along each half side of the rectangle and each quarter of the circle. In order to perform the integration accurately the parametric representation of the rectangular boundary is chosen to be that used in [13, Example 3.2].

Boundary test points. Five equally spaced points on each of EB, BF and PQ.

Numerical results.

		(i) $a = b = 0.5$	
ONM/MB:	$N\overset{*}{o}pt = 30$,	$E_{30} = 5.1 \times 10^{-2}$,	$M_{30} = 1.702\,0 \cdots$.
ONM/AB:	$Nopt = 24$,	$E_{24} = 7.0 \times 10^{-8}$,	$M_{24} = 1.691\,564\,902\,59$.
VM/AB:		$E_{24} = 7.0 \times 10^{-8}$,	*
		(ii) $a = 0.4, b = 0.2$	
ONM/AB:	$Nopt = 18$,	$E_{18} = 5.0 \times 10^{-6}$,	$M_{18} = 2.849\,771\,072$.
VM/AB:		$E_{18} = 5.0 \times 10^{-6}$,	*
		(iii) $a = 0.6, b = 0.2$	
ONM/AB:	$Nopt = 26$,	$E_{26} = 1.4 \times 10^{-5}$,	$M_{26} = 2.133\,835\,1$.
VM/AB:		$E_{26} = 1.2 \times 10^{-5}$,	*
		(iv) $a = 0.8, b = 0.2$	
ONM/AB:	$Nopt = 22$,	$E_{22} = 2.3 \times 10^{-4}$,	$M_{22} = 1.626\,912\,4$.
VM/AB:		$E_{22} = 2.3 \times 10^{-4}$,	*

(* In each case the VM/AB approximation to M agrees with the ONM/AB approximation to the number of figures quoted.)

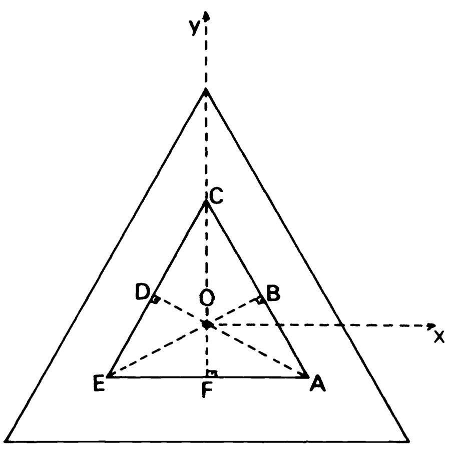


FIG. 5.4

Example 5.4. Triangle in triangle, Fig. 5.4.

Let G_h denote an equilateral triangle of height h, oriented so that its centroid is at the origin and one of its sides is parallel to the real axis.
Then,

$$\Omega = G_3 \cap \operatorname{compl}(G_h) \quad \text{with } h < 3.$$

Augmented basis. Because the domain has sixfold symmetry about the origin, the monomial basis is taken to be the set

$$z^{(3j-1)}, \qquad j = \pm 1, \pm 2, \pm 3, \cdots. \tag{5.12}$$

Let z_j, $j = 1, 2, 3$ be respectively the corners A, C, E of the inner triangle. Then, the singular functions corresponding to the branch point singularities at z_j; $j = 1, 2, 3$ are respectively the functions $\eta_{rj}(z)$, $j = 1, 2, 3$, given by (4.2) with

$$r = k + 3l/5, \qquad k = 0, 1, 2, \cdots, \quad 1 \leqq l \leqq 5.$$

Because of the symmetry, for each value of r, the three functions $\eta_{rj}(z)$ can be combined into the single function

$$\tilde{\eta}_r(z) = \eta_{r1}(z) + \sum_{j=2}^{3} e^{i\theta_j}\eta_{rj}(z), \tag{5.13}$$

where the arguments θ_j, $j = 2, 3$, are chosen so that

$$e^{2\pi i/3}\tilde{\eta}_r(e^{2\pi i/3}z) = \tilde{\eta}_r(z). \tag{5.14}$$

In this example the agumented basis is formed by introducing into the set (5.12) the four functions (5.13) corresponding to the values $r = 3/5$, $6/5$, $8/5$, $9/5$.

Quadrature. Gauss–Legendre formula with 48 points along each half side of the inner and outer triangles. In order to perform the quadrature accurately the parametric representation of the inner triangle is chosen to be that used in [13, Example 3.3].

Boundary test points. Eighteen points distributed along the half sides BC, CD of the inner triangle and the corresponding half sides of the outer triangle.

Numerical results.

		(i) $h=2.25$	
ONM/MB:	$\overset{*}{Nopt}=30$,	$E_{30}=1.3\times10^{-1}$,	$M_{30}=1.227\,0\cdots$.
ONM/AB:	$Nopt=18$,	$E_{18}=7.8\times10^{-5}$,	$M_{18}=1.208\,168\,761$.
VM/AB:		$E_{18}=7.8\times10^{-5}$,	*
		(ii) $h=1.50$	
ONM/AB:	$Nopt=22$,	$E_{22}=6.8\times10^{-7}$,	$M_{22}=1.657\,038\,875$.
VM/AB:		$E_{22}=6.8\times10^{-7}$,	*
		(iii) $h=0.75$	
ONM/AB:	$Nopt=16$,	$E_{16}=7.0\times10^{-6}$,	$M_{16}=3.132\,784\,643$.
VM/AB:		$E_{16}=7.0\times10^{-6}$,	*

(* In each case the VM/AB approximation to M agrees with the ONM/AB approximation to the number of figures quoted.)

Example 5.5. Cross in square, Fig. 5.5.

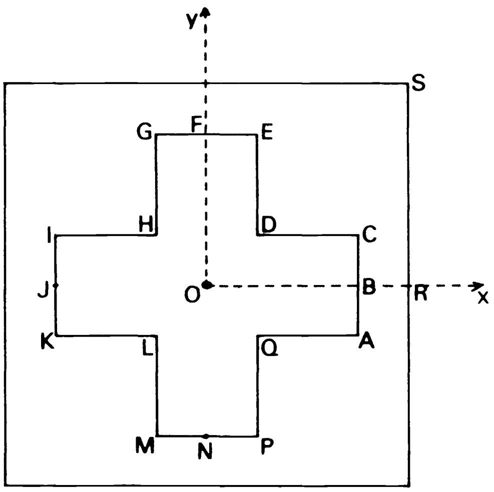


FIG. 5.5

Let

$$G_{ab}=\{(x,y): |x|<a, |y|<b\}\cup\{(x,y): |x|<b, |y|<a\} \tag{5.15}$$

and

$$G_c=\{(x,y): |x|<c, |y|<c\}.$$

Then

$$\Omega=G_c\cap\text{compl}\,(\bar{G}_{ab})\quad\text{with } a<c \text{ and } b<c.$$

Augmented basis. Let $z_j, j=1,2,\cdots,8$ be respectively the corners A, C, E, G, I, K, M and P of the cross-shaped region G_{ab}. Then the singular functions corresponding to the branch point singularities at $z_j, j=1,2,\cdots,8$ are respectively the functions $\eta_{rj}(z), j=1,2,\cdots,8$ defined by (4.2) with r given by (5.6). Because of the symmetry, for each value of r, the eight singular functions $\eta_{rj}(z)$ can be combined into the two functions

$$\tilde{\eta}_{rj}(z)=\eta_{rj}(z)+\sum_{m=1}^{3}e^{i\theta_{2m+j}}\eta_{r,2m+j}(z),\qquad j=1,2, \tag{5.16}$$

where, as in (5.7), the arguments θ_{2m+j} are chosen so that

$$e^{i\pi/2}\tilde{\eta}_{rj}(e^{i\pi/2}z)=\tilde{\eta}_{rj}(z), \qquad j=1,2.$$

In this example the augmented basis is formed by introducing into the monomial set (5.5) the six singular functions (5.16) corresponding to the values $r=2/3, 4/3, 5/3$.

Quadrature. Gauss–Legendre formula with 48 points along each of the segments AB, BC, $CD, \cdots, QA$, of the inner boundary, and each side of the square. In order to perform the quadrature accurately the parametric representation of the inner boundary is chosen to be that used in [13, Example 3.5].

Boundary test points. Seventeen points on the inner boundary segment $BCDEF$ and seventeen points on the side RS of the square.

Numerical results. The upper and lower bounds for the modulus M, listed below, are due to Jauer [7], and were obtained by using a finite-element method. The comparison values $\tilde{M}$ were computed, as in [12, Example 6.4], by using an approximation to the conformal map of the pentagonal domain bounded by the straight lines BR, RS, SD, DC and CB.

(i) $a=0.5, b=1.2\ c=1.5$

ONM/MB:	$Nopt=14$,	$E_{14}=9.5\times10^{-2}$,	$M_{14}=1.349\,0\cdots$.
ONM/AB:	$Nopt=21$,	$E_{21}=3.8\times10^{-5}$,	$M_{21}=1.331\,473\,449$.
VM/AB:		$E_{21}=3.8\times10^{-5}$,	*

Comparison value: $\tilde{M}=1.331\,463$

Bound: $1.331\,003<M<1.331\,944$.

(ii) $a=0.5, b=1.0, c=1.5$

ONM/AB:	$Nopt=27$,	$E_{27}=8.3\times10^{-6}$,	$M_{27}=1.566\,289\,179$.
VM/AB:		$E_{27}=8.3\times10^{-6}$,	*

Comparison value: $\tilde{M}=1.566\,274$.

Bound: $1.565\,602<M<1.566\,978$.

(iii) $a=0.2, b=0.7, c=1.2$

ONM/AB:	$Nopt=25$,	$E_{25}=3.0\times10^{-5}$,	$M_{25}=1.981\,644\,1$.
VM/AB:		$E_{25}=2.8\times10^{-5}$,	*

Comparison value: $\tilde{M}=1.981\,774$.

Bound: $1.979\,574<M<1.983\,722$.

(iv) $a=0.1, b=0.8, c=1.1$

ONM/AB:	$Nopt=23$,	$E_{23}=3.6\times10^{-4}$,	$M_{23}=1.747\,492\,5$.
VM/AB:		$E_{23}=4.0\times10^{-4}$,	*

Comparison value: $\tilde{M}=1.747\,677$.

Bound: $1.745\,050<M<1.749\,940$.

(* In each case the VM/AB approximation to M agrees with the ONM/AB approximation to the number of figures quoted.)

Example 5.6. Circle in cross, Fig. 5.6.

As in Example 5.5, let G_{ab} denote the cross-shaped region defined by (5.15). Then

$$\Omega=G_{3,1}\cap\{z: |z|>c\}.$$

Augmented basis. Let z_j, $j=1,2,3,4$, be respectively the corners A, B, C and D of the outer boundary. Then, the singular functions corresponding to the branch point singularities at z_j, $j=1,2,3,4$, are respectively the functions $\eta_{rj}(z)$, $j=1,2,3$ given by (4.3) with

$$r=2l/3, \qquad l=1,2,3,\cdots.$$

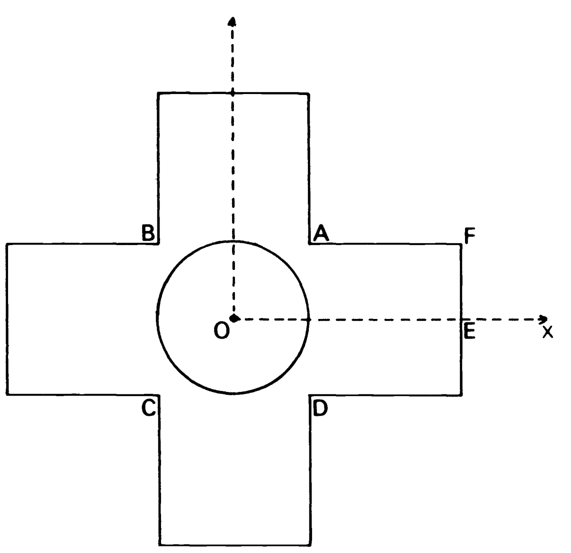


FIG. 5.6

Because of the symmetry, for each value of r, the four functions $\eta_{rj}(z)$ can be combined, as in Examples 5.2 and 5.3, into a single function $\tilde{\eta}_r(z)$ of the form (5.7)–(5.8). The augmented basis is formed by introducing into the monomial set (5.5) the four functions $\tilde{\eta}_r(z)$ corresponding to the values $r = 2/3,\ 4/3,\ 8/3,\ 10/3$.

Quadrature. Gauss–Legendre formula with 48 points along each side of the outer boundary and each quarter of the circle.

Boundary test points. Thirteen equally spaced points on the straight lines EF and FA of the outer boundary, and six points on the circle. The test points on the circle are defined by $ce^{i\tau}$; $\tau = 0(\pi/20)\pi/4$.

Numerical results.

		(i) $c = 0.8$	
ONM/MB:	$\overset{*}{N}opt = 30$,	$E_{30} = 6.1 \times 10^{-1}$,	$M_{30} = 2.356\,0 \cdots$.
ONM/AB:	$Nopt = 22$,	$E_{22} = 1.5 \times 10^{-5}$,	$M_{22} = 2.246\,094\,81$.
VM/AB:		$E_{22} = 1.3 \times 10^{-5}$,	*
		(ii) $c = 0.5$	
ONM/AB:	$Nopt = 24$,	$E_{24} = 7.8 \times 10^{-6}$,	$M_{24} = 3.595\,639\,19$.
VM/AB:		$E_{24} = 8.9 \times 10^{-6}$,	*
		(iii) $c = 0.2$	
ONM/AB:	$Nopt = 24$,	$E_{24} = 8.0 \times 10^{-6}$,	$M_{24} = 8.989\,209\,95$.
VM/AB:		$E_{24} = 8.9 \times 10^{-6}$,	*

(* In each case the VM/AB approximation agrees with the ONM/AB approximation to the number of figures quoted.)

6. Discussion. The results of § 5, as well as results of other numerical experiments not presented here, indicate that both the VM and the ONM are capable of computing approximations of high accuracy. In particular, our results show that the two methods can produce accurate approximations for the mapping of difficult domains, involving sharp corners. The essential requirement for this is that the basis set contains functions that reflect the asymptotic behavior of the function H, in the neighborhood of a corner where a singularity occurs. Regarding computational efficiency, our experiments show that the two methods require approximately the same computational effort for producing approximations of comparable accuracy.

The above remarks apply only to the mapping of domains with $2n$-fold symmetry, $n \geqq 2$, of the type considered in § 5. For such symmetrical domains the number of basis functions used in the numerical process can be reduced considerably and, in general, the two methods are extremely accurate. Unfortunately, in the absence of $2n$-fold symmetry, $n \geqq 2$, the performance of both the VM and the ONM is rather disappointing. If the domain involves "singular" corners then the use of functions of the form (4.2) or (4.3) always leads to some improvement in accuracy. However, if Ω is a nonsymmetrical domain then dealing with corner singularities alone is not sufficient for the methods to produce accurate approximations. The difficulty in this case might be due to the presence of poles of the function H in compl $(\bar{\Omega})$. Unfortunately, as we remarked earlier, we do not know of a way for dealing with such singularities.

Acknowledgments. We are grateful to Professor D. Gaier, who first suggested to us the use of the VM with singular basis functions, for the mapping of doubly-connected domains containing corners. We are also grateful to him for reading the original draft of the manuscript. His detailed comments improved the presentation of this paper.

REFERENCES

[1] S. BERGMAN, *The Kernel Function and Conformal Mapping*, Math. Surveys 5, American Mathematical Society, Providence, RI, 1970.
[2] F. BOWMAN, *Notes on two-dimensional electric field problems*, Proc. London Math. Soc., 39 (1935), pp. 205–215.
[3] ———, *Introduction to Elliptic Functions*, English University Press, London, 1953.
[4] D. GAIER, *Konstructive Methoden der konformen Abbildung*, Springer-Verlag, Berlin, 1964.
[5] ———, *Das logarithmische Potential und die konforme Abbildung mehrfach zusammenhängender Gebiete*, in E. B. Christoffel, The Influence of His Work on Mathematics and the Physical Sciences, Birkhäuser Verlag, Basel, 1981, pp. 290–303.
[6] W. EIDEL, *Konforme Abbildung mehrfach zusammenhängender Gebiete durch Lösung von Variationsproblemen*, Diplomarbeit Giessen, 1979.
[7] H. J. JAUER, *Die Finite-Element-Methode bei Gebieten mit Krummlinigem Rand*, Diplomarbeit Giessen, Justus-Liebig-Universität, Giessen, 1981.
[8] P. A. A. LAURA, *A survey of modern applications of the method of conformal mapping*, Revista de la Unión Mathematica Argentina, 27 (1975), pp. 167–179.
[9] R. S. LEHMAN, *Development of the mapping function at an analytic corner*, Pacific J. Math., 7 (1957), pp. 1437–1449.
[10] D. LEVIN, N. PAPAMICHAEL AND A. SIDERIDIS, *The Bergman kernel method for the numerical conformal mapping of simply-connected domains*, J. Inst. Math. Applics., 22 (1978), pp. 171–187.
[11] Z. NEHARI, *Conformal Mapping*, McGraw-Hill, New York, 1952.
[12] N. PAPAMICHAEL AND C. A. KOKKINOS, *Two numerical methods for the conformal mapping of simply-connected domains*, Comput. Meth. Appl. Mech. Engrg., 28 (1981), pp. 285–307.
[13] ———, *Numerical conformal mapping of exterior domains*, Comput. Meth. Appl. Mech. Engrg., 31 (1982), pp. 189–203.
[14] J. D. PRYCE, *Basic Methods of Functional Analysis*, Hutchinson University Library, London, 1973.
[15] J. F. THOMPSON, Z. U. A. WARSI AND C. W. MASTIN, *Boundary-fitted coordinate systems for numerical solution of partial differential equations—A review*, J. Comput. Phys., 41 (1982), pp. 1–108.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

TECHNIQUES FOR SOLVING BLOCK TRIDIAGONAL SYSTEMS ON RECONFIGURABLE ARRAY COMPUTERS*

R. N. KAPUR† AND J. C. BROWNE‡

Abstract. This paper illustrates the concept of multiphase parallel structuring of algorithms on reconfigurable computers. Reconfigurable network architectured computers are described and a paradigm for programming them is defined. The execution behavior of two linear system solving techniques is determined and compared. This paper does not attempt a traditional analysis of linear system solvers: instead it presents a study of the scheduling and data flow requirements of a selected pair of algorithms.

Key words. parallel algorithms, reconfigurable computers

1. Introduction. This paper gives illustrations of the concept of multitype, multiphase parallel structuring of algorithms. Most previous work on parallel structuring of numerical algorithms has been concerned with mapping of algorithms to a fixed parallel architecture which has a single mode of parallel execution and a single fixed interconnection geometry between the processors. This paper describes reconfigurable network architectured computer systems which can implement multiple types and degrees of parallelism and multiple communication geometries. A paradigm for mapping parallel computation structures to these architectures is developed. We then determine and describe the natural computational structure (degree and type of parallelism and geometry of communication) for odd-even elimination (OEE) and odd-even reduction (OER) algorithms for the solution of block tridiagonal linear systems. The computational structures derived are then mapped upon a reconfigurable network architectured multiprocessor, and the total execution cost in such an environment is determined. The principal results of the work are:

1. Block odd-even elimination and odd-even reduction algorithms require multitype multiphase structuring in order to realize optimal or near optimal total execution costs.
2. It is possible to obtain an execution speed-up linear in the number of processors as long as the number of processors is less than the number of blocks.
3. The overhead cost of data movement, synchronization delays and reconfiguration time can be kept to about 10% of the actual computation costs of the natural geometry of data movement can be realized in the architecture of the host computer.
4. Odd-even elimination becomes the algorithm of choice for parallel architectures for system sizes exceeding approximately 512×512.

There has been historical difficulty in effectively mapping any extensive set of problems or any complex algorithm to execute efficiently on any given parallel architecture. The factors which lead to inefficient execution include the following:

1. Mapping of complex computations to a single architecture often leads to significant portions of the computation being based on high operation count algorithms.
2. It is often awkward to map the several types of data movements required by significant algorithms onto a single interconnection geometry.

* Received by the editors, July 28, 1982, and in final form June 2, 1983. This research was sponsored in part by the National Science Foundation under grant MCS77-20698 and in part by the Department of Energy, Office of Basic Energy Sciences under grant DE-ASOS-81ER10987.

† Bell Laboratories, Murray Hill, New Jersey 07974. The research of this author was performed while he was with the Electrical Engineering Department at the University of Texas at Austin.

‡ Department of Computer Sciences, University of Texas at Austin, Austin, Texas 78712.

3. Mapping to a fixed number of processors often leads to multiple passes through the processing elements for some data unless the number of processors available exceeds the dimensionality of the problem.

4. It is often the case that a computation will pass through several phases, each of which may need a different parallel structure or different degrees of parallelism for efficient realization.

Agerwala and Lint [Age81] have analyzed the communication geometries of a number of significant algorithms. The importance of problem specific interconnection geometry has been described by Gentlemen [Gen78]. He has demonstrated that for the specific case of matrix inversion and matrix multiplication data movement time on the ILLIAC IV mesh-connected communication geometry can easily dominate execution time. Recent investigations of the interconnection geometries required for the efficient execution of such significant tasks as solution of Poisson's equation [Gro79], [Gro80], [Gan81] (which uses the algorithms described in the current work) have shown that no single type of network is suitable for these problems.

There are hardware architectures which implement a complete spectrum of computational geometries. A full cross-bar network between processors and memories removes all restrictions on algorithm formulation, but will be prohibitively expensive even for a moderately large number of processing elements. A common memory architecture with a limited number of paths between memory and processors suffers performance degradation from memory access interference as the number of processors seeking access to the memory becomes large. One solution to this dilemma of requirements for complex interconnection geometries without excessive cost or performance degradation is being sought with development of reconfigurable interconnection networks to link together arrays of processing elements.

2. Reconfigurable network architectured computers. A reconfigurable network architectured computer is a collection of processor, memory and I/O resources and a network. The network configures sets of resources to form computer architectures with specified parallel structure and interconnection geometry. A configuration may be established at job initiation time and may vary during the course of the execution. The architectures dynamically created by configuration of the network are resource partitions which execute *independently* except at specified points of interaction.

There are a number of university projects either building or proposing to build reconfigurable architectures [Lip71], [Vic79], [Kar82], [Sie79], [Sny81].

The availability of such architectures opens new dimensions for the formulation of parallel algorithms. The arrangement of the computations can now be based upon the structure of the algorithm rather than upon a specific available architecture or upon an effectively unrealizable architecture. Resource partitions can be tailored to the computation and the data movement requirements of the algorithm. The computing power of partitions can be varied to minimize synchronization delays.

The Texas Reconfigurable Array Computer (TRAC) is a representative example of such an architecture [Sej80], [Pre80], [Kap80]. TRAC is in this paper used as the target of reconfigurable multiprocessor architecture for the mapping and performance analysis of parallel versions of block OEE and OER algorithms.

The network for TRAC is a banyan network [Lip73] with each node having a fanout of 2 and a spread of 3. Figure 2.1 shows a 4 processor–9 memory configuration of TRAC. Each processor has 8-bit-wide data paths but can execute with precision of up to 256 bytes. The memories are byte addressable and the address space is virtualized. Specific computer architectures are constructed by establishing *circuits* through the

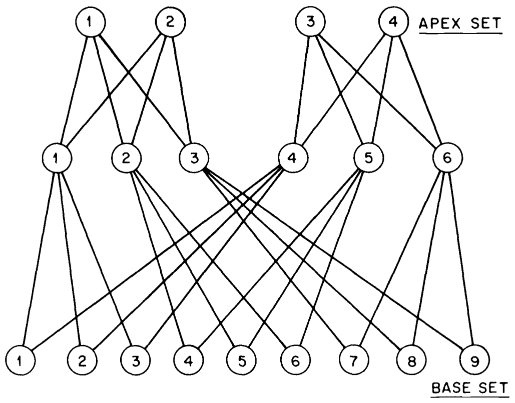


FIG. 2.1. *4 processor–9 memory configuration of* TRAC.

network, connecting processors to memories and processors to processors. The construction proceeds in four steps. (i) Circuits linking processors to memories are established to generate computers. These circuits, once established, behave exactly as conventional memory buses. (ii) Processors may be linked by carry-look-ahead paths to more efficiently execute extended precision arithmetic. (iii) Processors may be coupled by an instruction broadcast bus to give a single-instruction–multiple-data stream configuration [Fly66]. (iv) Finally the individual computers constructed in steps (i)–(iii) may be linked by circuits connecting many processors to a single memory which can be switched between configurations. These steps are illustrated in Fig. 2.2 (formation of a two-configuration partition) where two physical processors are coupled to form a pair of logical computers. Each computer does 32-bit arithmetic. The two computers share a single memory. The heavy solid lines are the circuits connecting processors to memories. The - - - line is the carry linkage coupling physical processors

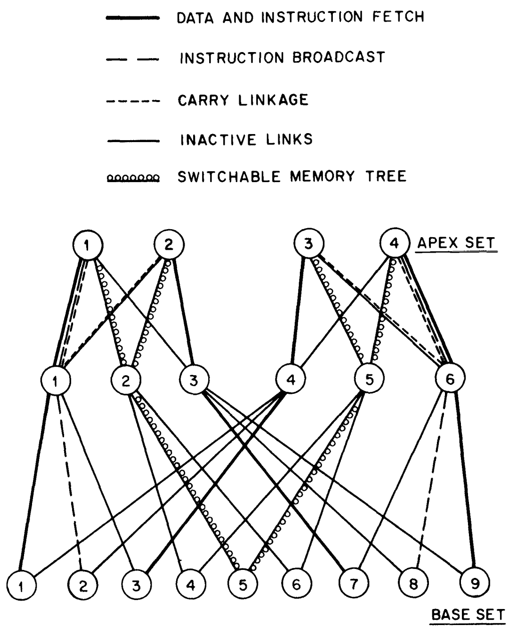


FIG. 2.2. *Formation of a two-configuration partition.*

to form logical processors. The —— lines are the instruction broadcast buses for each logical computer. The ∘∘∘∘∘ lines gives the coupling of each logical computer to the sharable memory. The configuration process for reconfigurable network architectures is described more fully by Browne and Lipovski [Bro82].

3. A paradigm for organizing multitype multiphase parallel structuring. The architecture described in the preceding section gives the algorithm designer the power to define an execution architecture with a type and degree of parallelism and an interaction geometry appropriate to the architecture of the algorithm under consideration. Problems can be decomposed into phases, and suitable configurations consisting of interconnected partitions can be constructed for each phase. Each phase consists of partitions (SISD, single instruction single data, or SIMD, single instruction multiple data) linked according to the required communication geometry. Problems are thus formulated as a set of tasks or a sequence of sets of tasks (MIMD/SISD or MIMD/SIMD mode of operation) rather than as just a sequence of tasks as is the case for a uniprocessor. Each task set may have a different type and degree of parallelism and/or a different interconnection geometry. The execution structure of an algorithm will then consists of a number of partitions interconnected according to the data flow of the job. Processors within a partition are under lock step control of one instruction (SIMD mode) stream. Processors for different partitions are under the control of different instruction (MIMD mode) streams. Algorithm structure will be specified in terms of phases. Each phase consists of partitions (SISD or SIMD) linked according to some data flow specifications. Partitions from phase to phase are also linked according to interphase data flow specifications. This SIMD/MIMD organization of programs has been independently suggested by Siegel [Sie79].

Two types of data transfer requirements can be observed in the SIMD—MIMD organization. Data transfers between partitions are needed within a phase and between phases. A synchronization mechanism is necessary for these transfers. Secondly, data realignment transfers may be necessary within SIMD partitions.

The TRAC architecture, upon which the algorithm described subsequently will be mapped, implements two different modes of interpartition data transfers. One is through the use of switched memory [Pre79]. In this mode a physical memory cell is switched from partition to partition. This type of sharing of switched memory in a resource partitioning computer is not the same as common memory in a multiprocessor such as, for example, C.mmp [Wul72]. A common memory organization shares on an access by access basis with the possibility of interference of access to the entire memory module when more than one processor simultaneously endeavors to access the same memory module. Software level access to shared objects is commonly controlled by synchronization mechanisms for implementing mutual exclusion. The execution of these mechanisms consumes memory bandwidth and interferes with the performance of the sharing resources [Ole78].

Sharing in a reconfigurable network architecture is combined with synchronization by altering the interconnection network to move a physical memory module from one task address space to a different task address space. The acquisition and arbitration circuitry is independent of and parallel to the data path in the switch memory. Acquisition attempts by one processor for a memory held by another processor therefore affects only this one processor rather than blocking or interfering with the operation of the other executing processors.

TRAC also contains a second communication mechanism with characteristics that are quite different from switched memory. A packet switched network is provided—this mechanism bypasses the tree structure described earlier. This is accomplished by multiplexing the packet switched network onto the interconnection network. It is important to note that this multiplexing uses the interconnection network without interfering with the functions that have been described.

Packet switching uses the interconnection network as a store and forward medium. Packets are transmitted by a source processor from a data memory to a target processor which stores them in another data memory. It is possible for packet collisions to occur; i.e., two or more packets may attempt to travel over a single link at the same time. This is resolved by sequentializing the movement of these packets over the appropriate links. Contention of this nature is dependent on the topology of the network and the nature (source, distinction and intensity) of the packet traffic. Data transfers are now possible without prior setup; however, the transfer times are no longer deterministic.

A detailed description of the communication mechanism in TRAC is available in [Pre79].

4. Block tridiagonal systems.

We consider the solution of block tridiagonal linear systems using OEE and OER. This paper concentrates entirely on mapping these algorithms to reconfigurable computer architectures. The issues of numerical mathematics such as stability are assumed to be solved and are not considered. It may be noted, however, that the strong diagonal dominance characteristic of the equations determined by Poisson's equation tends to guarantee stability.

Block tridiagonal matrices occur commonly as linear system coefficient matrices when partial differential equations are discretized [You72], [Hel77]. The solution of Poisson's equation [You72] is an example of a physical problem that yields a block tridiagonal linear system.

This section describes techniques for the solution of properly conditioned block tridiagonal linear systems on a reconfigurable array computer. The methods are resolved into distinct phases, each of which uses a different degree of parallelism and has a different interconnection geometry. This formulation displays advantages for the use of reconfigurable array systems with independent partitions assigned to blocks. These are:

1. Each partition operates independently, therefore independent pivoting is possible.
2. The processing power of each partition can be tailored to the size of the block it is handling so that synchronization waits are minimized.
3. The synchronized nature of the shared data access is well suited to intercommunication mechanisms characteristic of reconfigurable computers.

This analysis assumes that it is always possible to set up configurations representative of the executing phase of a program. The case where insufficient resources are present to set up a requested configuration has not been studied.

Two types of odd-even solvers are described: odd-even elimination (OEE) and odd-even reduction (OER). OER is considered to be a compact version of OEE on uniprocessor architectures. On reconfigurable parallel computers, however, OEE will be seen to be superior to OER in terms of total execution time; the overall computational effort remains greater for OEE (but many more operations can be done in parallel) than for OER, as with uniprocessors [Hel77].

The organization of the rest of this section is as follows. Section 4.1 provides the notation used to represent block tridiagonal systems. OEE is synthesized and analyzed for reconfigurable computers in § 4.2. OER is derived from OEE in § 4.3; it is synthesized and analyzed for reconfigurable computers and its performance is compared with OEE.

4.1. Notation. The matrix A can be represented as

$$A=(a(j), b(j), c(j))_N$$

where $b(i)$ is a $n.i \times n.i$ matrix and $a(1)=0$ and $c(N)=0$.

$$\begin{bmatrix} b(1) & c(1) & & & & & \\ a(2) & b(2) & c(2) & & & & \\ & a(3) & b(3) & c(3) & & & \\ & & \cdot & & & 0 & \\ & & & \cdot & & & \\ & 0 & & & \cdot & & \\ & & & a(N-2) & b(N-2) & c(N-2) & \\ & & & & a(N-1) & b(N-1) & c(N-1) \\ & & & & & a(N) & b(N) \end{bmatrix}$$

$A = (a(i), b(i), c(i))_N$

$a(1) = 0, c(N) = 0$

FIG. 4.1. *Block tridiagonal matrix.*

Odd-even methods are regarded as efficient direct methods for the case where the constituent blocks (submatrices) are small enough to be stored explicitly [Hel77].

4.2. Odd-even elimination. Consider odd-even elimination as described in Heller [Hel77, § 4]: pick three consecutive block equations

$$Ax=v, \qquad A=(a(j), b(j), c(j))_N,$$

$$a(k-1)x(k-2)+b(k-1)x(k-1)+c(k-1)x(k)=v(k-1)\ldots(k-1),$$

$$a(k)x(k-1)+b(k)x(k)+c(k)x(k+1)=v(k)\ldots(k),$$

$$a(k+1)x(k)+b(k+1)x(k+1)+c(k+1)x(k+2)=v(k+1)\ldots(k+1).$$

If we multiply equation $k-1$ by

$$-a(k)b^{-1}(k-1),$$

equation $k+1$ by

$$-c(k)b^{-1}(k+1),$$

and add, the result is:

$$\begin{aligned} &(-a(k)b^{-1}(k-1)a(k-1))x(k-2)\\ &\quad+(b^{-1}(k)-a(k)b^{-1}(k-1)c(k-1)-c(k)b^{-1}(k+1)a(k+1))x(k)\\ &\quad+(-c(k)b^{-1}(k+1)c(k+1))x(k+2)\\ &=(v(k)-a(k)b^{-1}(k-1)v(k-1)-c(k)b^{-1}(k+1)v(k+1)). \end{aligned}$$

For $k=1$ or N there are only two equations involved and the modifications should be obvious. This operation eliminates the odd unknowns for k even and the even unknowns for k odd. By collecting the new equations into the block pentadiagonal system $H.2x = v.2$ (with A defined as $H.1$), it is seen that row elimination has preserved the fact that the matrix has only three nonzero block diagonals, but they are further apart. A similar set of operations is applied combining equations $k-2$, k and $k+2$ in $H.2$ to produce the $H.3x = v.3$ system. This process is repeated until only one block diagonal remains (or in the case of semidirect methods, until some accuracy criteria are fulfilled). The initial coefficients matrix $H.1$ contains $3N-2$ nonzero blocks, while the final matrix consists of N nonzero blocks along the main diagonal.

Solving the N blocks independently gives the required solution.

Figure 4.2 shows the effect of 5 steps of elimination on a 16×16 block tridiagonal system.

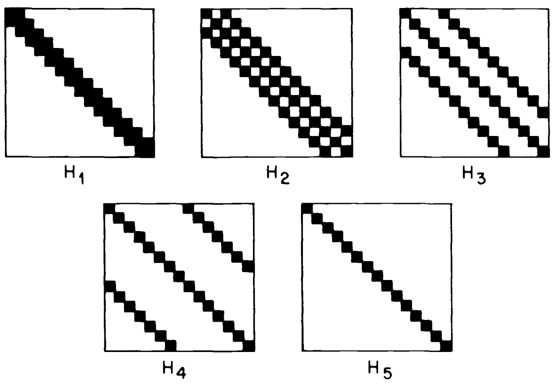


FIG. 4.2. *Five elimination steps* (*from* [Hel77, pp. 39]).

4.2.1. Data flow and implementation. Sections 4.2.1 and 4.2.2 are derived from an earlier paper [Kap81]. The material is extended to include additional modes of parallelism and to include packet based data movement in addition to switched memory based data movement.

In this subsection we will look at the data flow characteristics of odd-even elimination. The computational aspects such as operation counts are well understood; the communication geometry is studied herein and is found to be regular and simple.

Computationally, instead of determining the inverse of $b(i)$ explicitly, LU factorization [You72] of $b(i)$ is generally used:

$$\begin{aligned}
&\text{solve } b(k)[\underset{\sim}{a}(k)\underset{\sim}{c}(k)\underset{\sim}{v}(k)] = [a(k)c(k)v(k)], && 1\leqq k\leqq N,\\
&b(k).2 \leftarrow b(k).1 - a(k).1\underset{\sim}{c}(k-1) - c(k).1\underset{\sim}{a}(k+1), && 1\leqq k\leqq N,\\
&v(k).2 \leftarrow v(k).1 - a(k).1\underset{\sim}{v}(k-1) - c(k).1\underset{\sim}{v}(k+1), && 1\leqq k\leqq N,\\
&a(k). \leftarrow -a(k).1\underset{\sim}{a}(k-1), && 3\leqq k\leqq N,\\
&c(k).2 \leftarrow -c(k).1\underset{\sim}{c}(k+1), && 1\leqq k\leqq N-2.
\end{aligned}$$

4.2.1.1. *Intertask data flow.* The sequence of actions that results in the computation of $H.i+1$ from $H.i$ is referred to as a stage: in this case each stage is shown to consist of three steps and the steps further consist of substeps.

Consider the input data flow for computing $H.2$ and $v.2$. In the first step, the first substep results in the LU factorization of $b(k)$; this is then used in the next substep for computing $\underset{\sim}{a}(k)$, $\underset{\sim}{c}(k)$ and $\underset{\sim}{v}(k)$. N-way parallelism is displayed in this step.

In the second step the computation of $a(k).i+1$ and $c(k).i+1$ requires $a(k).i$, $\underline{a}(k-1).i$ and $c(k).i$, $\underline{c}(k+1).i$ respectively; $b(k).i+1$ and $v(k).i+1$ require data from the $(k-1)$, (k) and $(k+1)$th row equations. Binary operations are performed on the blocks—pairwise access to blocks is sufficient, giving rise to up to $(N/2)$-way parallelism.

Figure 4.3a shows the interconnection geometry needed for the second step for a 16×16 system. The blocks are stored in separately accessible switched memories, one block row per switched memory. The diagram to the right of the 16×16 tridiagonal system is the interconnection pattern with circles representing processors and rectangles, switched memories. The edges represent potential links that are activated in 4 separate patterns, as shown further to the right. The new blocks computed at the end of substep 2 are shown in curly brackets between the patterns of substeps 2 and 3; the remaining new blocks are completed at the end of substep 4.

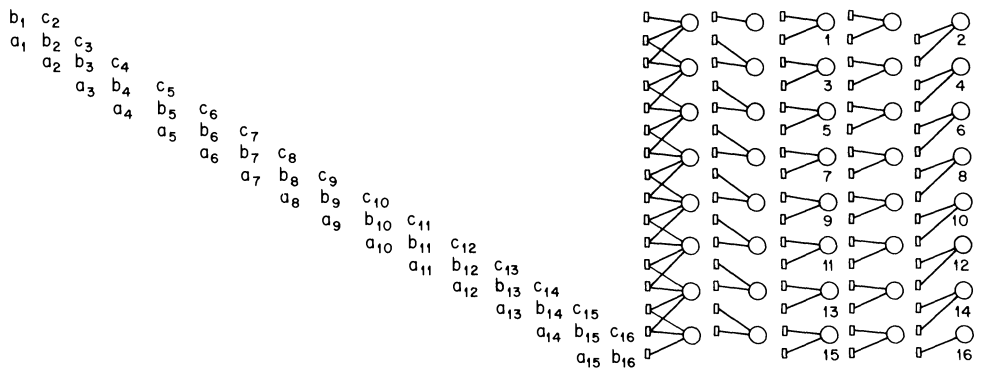


FIG. 4.3a. *Interconnection for Stage* 1, *Step* 2 *of* 16×16 *tridiagonal system.*

The crucial observation here is that while the data sets are shared across processors, the sharing is conflict free within a substep. The connection pattern cycles through the states of a 2-pole, 3-position switch.

H. 2 is a pentadiagonal matrix—the application of an inverse perfect shuffle [Sto71] partitions this matrix into two tridiagonal matrices, one consisting of the odd-numbered coefficient blocks and the other of the even-numbered ones (Fig. 4.3b).

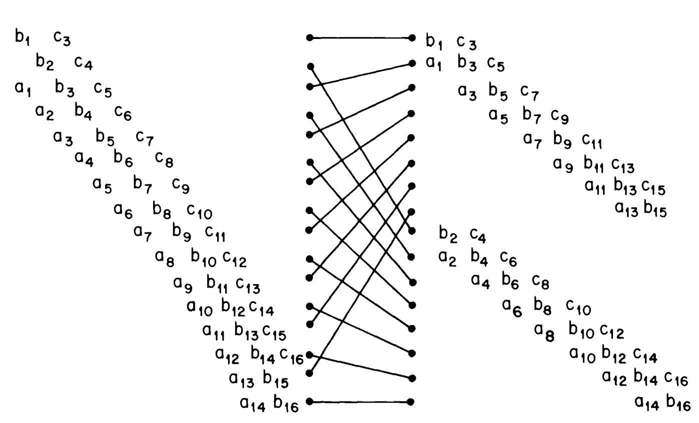


FIG. 4.3b. *Inverse perfect shuffle to form tridiagonal system.*

If the matrix A contains $N = 2{**}\,m$ blocks, then the data flow geometry for the next step 2 is represented by a graph that is a proper subset of the graph used in the earlier step 2 (Fig. 4.4). This inclusion property is seen in every step 2 until the block diagonal is computed.

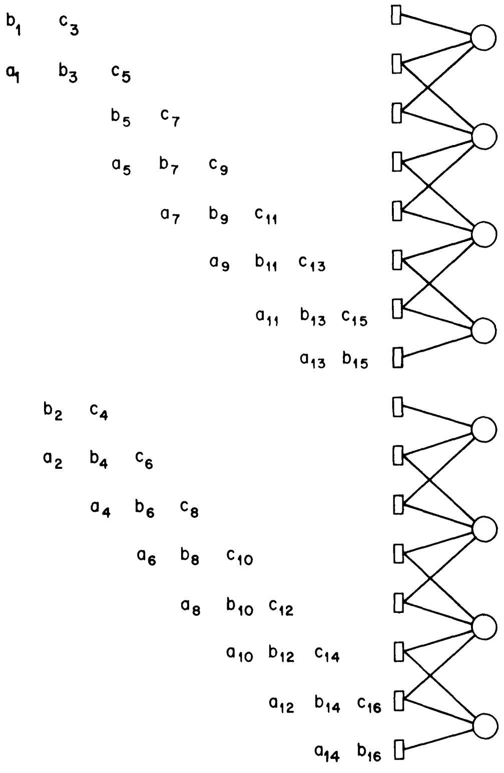


FIG. 4.4. *Interconnection for stage* 2 *of Step* 2 *of* OEE *of* 16×16 *system.*

Thus we use precisely the same data flow template in the generation of every $H.i+1$ from $H.i$; three steps with different connection geometries are needed—a direct connection, a 2-pole 3-position switch based connection, followed by an inverse perfect shuffle.

Two modes of parallel operation are now possible—the newly instantiated substreams consisting of uncoupled tridiagonal systems can be permitted to proceed independently; alternately, all substreams can be resynchronized at the end of a stage. The first mode is more efficient, while the second is easier to control. This is discussed further, below.

Most of the reconfigurable machines mentioned earlier can implement these and other communication behaviors quite efficiently. Note that if we were to hard-wire the interconnection pattern, we would be using a special purpose machine of limited applicability to other problems (e.g., the shuffle exchange network in high performance FFT boxes). Instead, reconfigurable computers provide a general framework for implementing a large class of problems with widely varying communication behavior [Sej80].

Two implementations of odd-even elimination on TRAC are now sketched briefly. The first one uses switched memory and the second is based entirely on the use of packets.

The processors are scheduled as partitions with width commensurate with the block size under consideration. The shared datasets are stored in switched memory modules—each rectangle of Fig. 4.3a is realized as an array of switched memories of width conforming with the processor width. The time to throw the 2-pole and 3-position based switch is a critical parameter in the performance of the algorithm. In the current implementation of TRAC, this hardware operation requires between 1 and 8 microseconds. An assembly language instruction for executing this operation requires almost 50 microseconds because of the overhead associated with invoking the operation.

In the packet based implementation, data sets are shared by having a holding partition physically transmit its data to a buffer in a receiving partition. When more than one partition transmits at the same time, the best throughput is obtained if all packet movements occur without collisions. In general, however, the transmissions are affected both by the nature of the traffic (in terms of the physical locations of the sources and targets of data) and by the intensity of the traffic (in terms of number of simultaneous sources).

4.2.1.2. *Intratask data flow.* The use of an SISD partition for each block avoids the problem of alternate row/column addressing. Alternate row and column access is necessary because the block matrices, $a(k)$ and $c(k)$, are involved alternatively in premultiplication and postmultiplication. The use of SIMD partitions would introduce efficiencies in the computational part of the formulation. Data distribution would, however, become more complex. Packet movement would be necessary to realign data between pre- and postmultiplication stages. This aspect is not considered any further.

4.2.2. Performance estimation. The mode of operation described in the previous subsection consists of asynchronously executing processes which synchronize periodically to transfer data. Operations on different blocks may require different computation times. There may be, for example, different search times for the choice of pivot rows. Additionally, in the case where packets are used, a further dimension of nondeterminacy is introduced. Thus for a performance model to accurately represent this kind of behavior, it must be based on nondeterministic time parameters.

Two modes of operation based on different synchronization requirements were described in the previous subsection. The mode with explicit synchronization at the end of every stage is easier to control: here all streams of control must wait until the longest stream in the stage is done. The entire elimination, then, is the sum of the lengths of the longest stream in every stage. Figure 4.5a is an estimation of such a scenario. The second approach allows streams to proceed independently; synchronization is required only at points of data dependency. Here the total execution time is the length of the longest stream chain; this is usually less than the time in the first case (Fig. 4.5b).

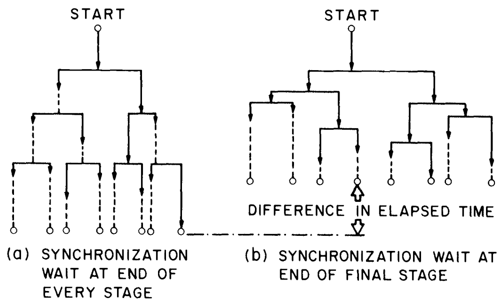

START
START
DIFFERENCE IN ELAPSED TIME
(a) SYNCHRONIZATION WAIT AT END OF EVERY STAGE
(b) SYNCHRONIZATION WAIT AT END OF FINAL STAGE

FIG. 4.5. *Synchronization waits.*

We will now present a naive analysis of the first mode as a step towards developing a performance model of reconfigurable computer operation. This analysis will first be given for the switched memory implementation and then for the packet implementation.

The time for data movement depends upon the precision of the processors and the precision of the arithmetic. A definite configuration is chosen to illustrate the magnitude of the communication costs. A 16-bit-wide SISD partition will be assigned to each block. (A 16-bit-wide SISD partition is equivalent to a conventional computer with a 16-bit word.) A partition or logical computer will be denoted by Pa, Pb, Pc, etc. If blocks are of uneven size a wider partition can be assigned to larger blocks. Arithmetic will be on floating point numbers with 64-bit mantissas and 8-bit exponents.

Let the system be 16×16 blocks and each block be 8×8 (total matrix dimension 128×128). Each 8×8 block requires about 600 bytes ($8\times 8\times$(8 byte mantissa+1 byte exponent)) of storage. A block row consisting of three 8×8 nonzero blocks and an 8×1 vector requires about 2000 words of storage.

The following notation is used for representing operation times:

T.fpa: floating point addition,
T.fpm: floating point multiply/divide,
T.xfre: memory to memory transfer time for one word,
T.swi: acquisition and setup time to obtain shared memory,
T.pkt: minimum byte transfer time using packets.

The evaluation of *H.*5 from *H.*1 proceeds in four stages with each stage evaluating *H.i*+1 from *H.i.* The first step of a stage consists of the *LU* factorization of b which is used to calculate $\underset{\sim}{a}$, $\underset{\sim}{c}$ and $\underset{\sim}{v}$. The second step consists of three substeps that correspond to the three positions of the 2-pole 3-position switch. The final step performs the inverse perfect shuffle to position data for the next stage. From the discussion of the previous subsection it is evident that the first and last steps display 16-way parallelism and the second step, 8-way parallelism.

*H.*5 is finally solved as 16 uncoupled linear systems to obtain the required solution.

We will use the notation $Pi(k, l, m, \dots)$ to represent the state where partition Pi is connected to datasets $k, l, m, \dots$ and dataset k contains $a(k)$, $b(k)$, $c(k)$ and $v(k)$. For example, $Pa(1)$ states that partition Pa is executing on data set 1.

The timing for the implementation using switched memories is now presented.

STAGE 1: Compute *H.*2 from *H.*1
Step 1:
Configuration: $Pa(1), Pb(2), \dots, Pp(16)$
Setup time ~ *T.swi*
Substep 1.1:

```
Computation:  Pa:  b(1)  (LU decomposition)
              Pb:  b(2)
               ⋮
              Pp:  b(16)
```

Compute time ~ $200 * T.fpm + 200 * T.fpa$
Substep 1.2:

```
Computation:  Pa:   a(1), c(1), v(1)
              Pb:   a(2), c(2), v(2)
               ⋮
              Pp:   a(16), c(16), v(16)
```

Compute time ~ $1200 * T.fpm + 1200 * T.fpa$

```
Step 2:
Substep 2.1:
Configuration:  Pa(1), Pc(2, 3), Pe(4, 5),
                ..., Po(14, 15)
Setup time ~ 2 * T.swi
Substep 2.2:
Configuration:  Pa(1, 2), Pc(3, 4), Pe(5, 6),
                ..., Po(15, 16)
Setup time ~ 2 * T.swi
Computation:    Pa:    b(1).2, a(1).2,
                       c(1).2, v(1).2
                Pc:    b(3).2, a(3).2,
                       c(3).2, v(3).2
                 .
                 .
                 .
Compute time ~ 2000 * T.fpm + 2000 * T.fpa
Substep 2.3:
Configuration:  Pa(1, 2), Pc(3, 4), Pe(5, 6)
                ..., Po(15, 16)
Setup time ~ 0 * T.swi
Substep 2.4:
Configuration:  Pa(2, 3), Pc(4, 5), Pe(6, 7),
                ..., Po(16)
Setup time ~ 2 * T.swi
Computation:    Pa:    b(2).2, a(2).2,
                       c(2).2, v(2).2
                Pc:    b(4).2, a(4).2,
                       c(4).2, v(4).2
                 .
                 .
                 .
Compute time ~ 2000 * T.fpm + 2000 * T.fpa
Step 3:
Substep 3.1:
Configuration:  Pa(1),   Pb(2),   Pc(3),   Pd(4),
                Pe(5),   Pf(6),   Pg(7),   Ph(8),
                Pi(9),   Pj(10),  Pk(11),  Pl(12),
                Pm(13),  Pn(14),  Po(15),  Pp(16)
Setup time ~ T.swi
Computation:  Transfer source data set contents to local buffer.
Compute time ~ 2000 * T.xfer
Substep 3.2:
Configuration:  Pa(1),   Pb(9),   Pc(2),   Pd(10),
                Pe(3),   Pf(11),  Pg(4),   Ph(12),
                Pi(5),   Pj(13),  Pk(6),   Pl(14),
                Pm(7),   Pn(15),  Po(8),   Pp(16)
Setup time ~ T.swi
Computation:  Copy local buffer contents to target data set.
Compute time ~ 2000 * T.xfer
```

The implementation using packets avoids a large fraction of the configuration setup time; instead, large volumes of data must be moved within steps (specifically in step 2), between steps and between stages. The following timing calculations are based on the assumption that the average packet transfer time is $2*T.pkt$ ($T.pkt$ is the minimum packet transfer time—this is seen when no collisions occur. Actual transfer times are related to parameters such as the nature of the background traffic, the actual source, destination traffic pattern in the program and the physical placement of the source memories and the target processors). The computation times remain precisely the same as before.

The following are the setup and data transfer times for the packet implementation:

Step 1:
Setup time ~ $T.swi$
Transfer ~ $4000*T.pkt$
Step 2:
Setup time ~ $2*T.swi$
Transfer ~ $8000*T.pkt$
Step 2.1:
Transfer ~ $4000*T.pkt$
Step 2.2:
Transfer ~ $4000*T.pkt$
Step 2.3:
Transfer ~ $0*T.pkt$
Step 2.4:
Transfer ~ $4000*T.pkt$
Step 3:
Setup time ~ $T.swi$
Transfer ~ $8000*T.pkt$

The important concern is the ratio of direct computation time to the sum of the total noncomputation time (this consists of the various $T.swi$, $T.xfer$, $T.pkt$, synchronization times, etc.). Let us make a reasonable assumption that the ratio of execution time for $T.xfer|T.pkt|T.fpa|T.fpm|T.swi$ are 1|3|10|100|1000 and let $T.xfer$ be one microsecond. Estimate the setup time for $T.fpa$, $T.fpm$ and $T.xfer$ to be equal to the arithmetic execution time.

For the shared memory implementation, the per-stage direct computation time is 1188 milliseconds (ms) per stage, reconfiguration time is 9 ms and data transfer time is 8 ms. The total runtime for OEE is then comprised of approximately 5000 ms of direct computation time and 70 ms of data transfer time. Thus, if synchronization time is zero, then the overhead associated with mapping the odd-even elimination to a parallel basis is about 70/5000 or less than 2%. Synchronization delays result solely from the differences in processing time for each block. For uniform size blocks, processing time differences between blocks will result from differing efforts for pivot selection. This should not exceed 1%. Reconfigurable architectures can assign processing partitions with power proportional to block size. (SISD partitions with a factor of 8 variation in power for 64-bit floating point numbers can be constructed on TRAC.) Thus synchronization delays should not be more than 10% of direct execution time. Using this as an upper bound, the total overhead cost in this formulation is approximately 12%.

For the packet implementation, the per-stage compute time remains 1188 ms. The reconfiguration time is 4 ms and the data transfer time is 100 ms. In this case the data transfer and synchronization delays amount to approximately 20% of the direct computation costs. The speedup over a comparable uniprocessor implementation of

OEE is approximately $N/2$ for the circuit switching, as well as the packet switching, implementations.

4.3. Odd-even reduction. OER can now be derived from the groundwork established in the previous section. Consider an elimination step in the previous section. Suppose we collect even-numbered equations into a linear system $(A.2)(x.2)=(w.2)$, where

$$\begin{aligned}
A.2 = (&-a(2j)*b^{-1}(2j-1)*a(2j-1),\\
&b(2j)-a(2j)*b^{-1}(2j-1)*c(2j-1)-c(2j)*b^{-1}(2j+1)*a(2j+1),\\
&-c(2j)*b(2j+1)*c(2j+1))_{N.2},\\
x.2 = {}&x(2j)_{N.2},\\
w.2 = {}&(v(2j))-a(2j)*b^{-1}(2j-1)*v(2j-1)-c(2j)*b^{-1}(2j+1)*v(2j+1))_{N.2}\\
= {}&v.2(2j)_{N.2},\\
N.2 = {}&\text{floor}\,(N/2).
\end{aligned}$$

This is a reduction step: it is a reduction step because $A.2$ is half the size of the original coefficient matrix A. Once this new system is solved, we can compute the remaining components of $x.1=x$ by back substitution:

$$x(2j-1)=b^{-1}(2j-1)*(v(2j-1)-a(2j-1)*x(2j-2)-c(2j-1)*x(2j)),$$

where $j=1,2,\cdots,\text{ceiling}\,(N/2)$.

Since $A.2$ itself is block tridiagonal, we can apply this procedure recursively to obtain a sequence of systems $(A.i)(x.i)=w.i$, where $A.1=A$ and $w.1=v$. It is convenient to restrict $N=2**m-1$ because the reduction then stops with $(A.m)(x.m)=w.m$ where $m=\text{ceiling}(\log(N+1))$ when the original $A.1$ is reduced to a single block in $A.m$. The single block is solved and the back substitution begins, culminating in $x.1=x$.

4.3.1. Elimination versus reduction. It will be fruitful at this point to contrast OEE with OER in a parallel environment. In a sequential computing environment, OER is regarded as a compact form of OEE [Hel77]. The main differences between OER and OEE are that the natural degree of parallelism at different steps is not the same, and that OER requires a back substitution sequence that is not needed in OEE.

In this comparison we will consider a linear system with $N=2**m$ for OEE and $N=2**m-1$ for OER. In the elimination sequence for OEE (Fig. 4.6), a doubling of the number of tasks occurs at the end of every step; each task, however, is half as large as the tasks in the previous step, and so the net parallelism remains constant. At the end of the last elimination step, N linear systems are solved independently (where N is the block dimensionality of the original coefficient matrix).

In the case of OER (Fig. 4.7), the number of tasks remains constant from reduction step to step; furthermore, the task size in a given step is half the task size of the previous step. Under the assumption that the reduction step times are equal, the average degree of parallelism is $\log N$.

After the last reduction is performed, exactly one block matrix is left—the solution of this matrix forms the basis for the back substitution sequence. This sequence grows from a parallelism of 2 to $N/4$ (Fig. 4.8). This back substitution sequence also shows an average degree of parallelism of $\log N$.

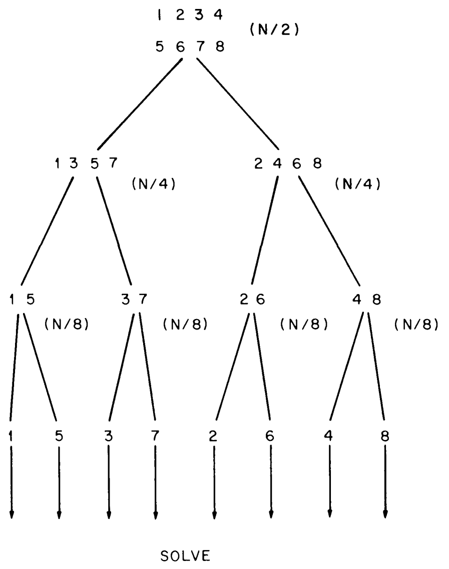


FIG. 4.6. *Elimination steps and solution for* OEE, $N = 8$.

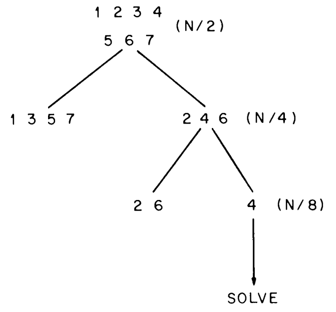


FIG. 4.7. *Reduction steps for* OER, $N = 7$.

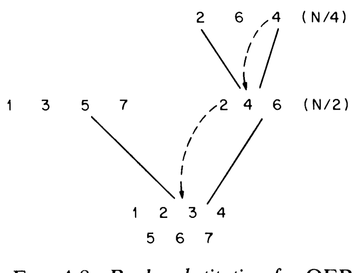


FIG. 4.8. *Back substitution for* OER.

The elimination sequence in OEE (for $N = 2{*}{*}m$, Fig. 4.6) requires one more step than the reduction sequence in OER (for $N = 2{*}{*}m - 1$, Fig. 4.7) as indicated by the tree heights in the two figures. OEE does require extraneous operations within each step, but these are performed in parallel and so do not contribute to the execution time. After the solution step, OER requires an additional back substitution sequence (Fig. 4.8) that is not needed in OEE. If the back substitution sequence runtime is less than one elimination/reduction stage runtime, OEE is superior to OER. In the example shown later in this section, this is the case for reasonably sized coefficient matrices.

4.3.2. OER back substitution data flow. From the back substitution equation of § 3.4, it is evident that each solved block is used in the solution of two unsolved blocks in each stage of the back substitution sequence. Figure 4.9 shows a configuration suitable for implementing one such stage. Each stage then requires a step with no switched memory access for computing block diagonal inverses; thereafter, switched memory access is necessary in two steps—once for each of the unsolved blocks where the solved blocks are needed.

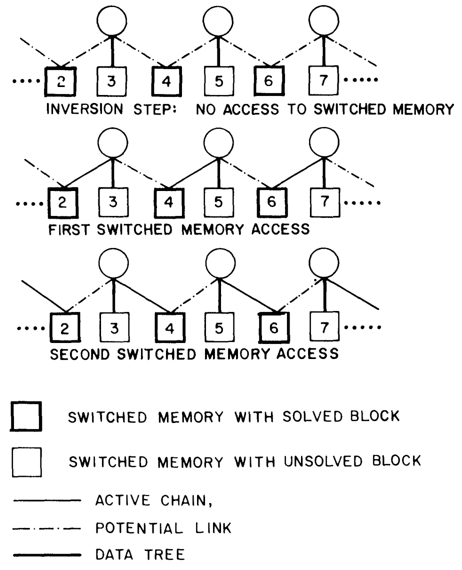

2
3
4
5
6
7
INVERSION STEP: NO ACCESS TO SWITCHED MEMORY
FIRST SWITCHED MEMORY ACCESS
SECOND SWITCHED MEMORY ACCESS
SWITCHED MEMORY WITH SOLVED BLOCK
SWITCHED MEMORY WITH UNSOLVED BLOCK
ACTIVE CHAIN,
POTENTIAL LINK
DATA TREE

FIG. 4.9. *Step from back substitution sequence of* OER.

4.3.3. Performance estimation for OER. The computation and data movement costs for OER can be estimated by a procedure similar to that used in § 4.3.2 for OEE. We consider only switched memory data movement for OER.

Continuing with the example of § 4.3.2, for the reduction sequence the per-stage computation time is 1188 ms and the data transfer time is 17 ms (this is identical to the elimination sequence perstage times). In the back substitution sequence, the per-stage computation time is 240 ms and the per-stage data transfer time is 2 ms. The total computation time for OER is approximately 4500 ms and the data transfer time is 60 ms, as opposed to 5000 ms and 70 ms for OEE. With 64×64 block and larger systems, the back substitution sequence requires more time than an elimination/reduction stage. From this point on, OEE requires less time than OER. The speedup for OER is approximately log N over a comparable uniprocessor implementation. Figure 4.10 shows the relative performance of OEE versus OER in the parallel formulations described in this paper.

5. Summary and conclusions. This paper has defined, described and illustrated the concept of multitype, multiphase parallel structuring on reconfigurable array computers. The illustration using odd-even elimination and odd-even reduction has established the following characteristics for multitype, multiphase parallel execution of these algorithms:

1. Speed-up linear in the number of processors can be obtained.

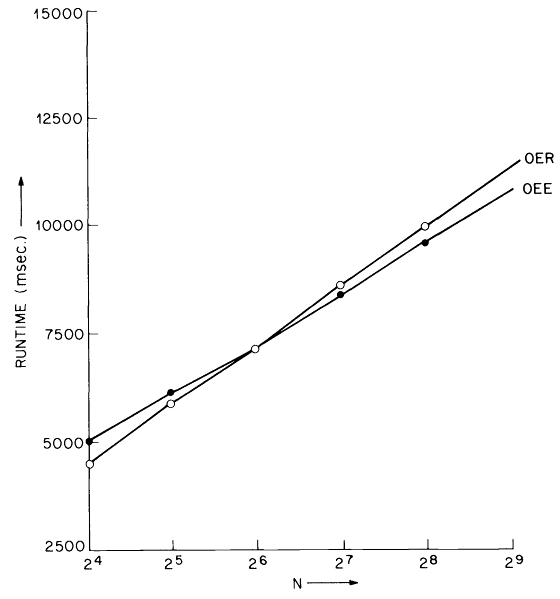


FIG. 4.10. OEE *and* OER *runtime for* $N \times N$ *block system* (8×8 *blocks*).

2. The overhead cost of computation (data movement, synchronization delay and reconfiguration time) can be kept to approximately 10% of instruction execution time on such architectures.

3. Odd-even elimination becomes the algorithm of choice on reconfigurable array architectures for linear system sizes above approximately 512×512. The relative speed-up over comparable uniprocessor implementations is approximately $n/2$ for OEE and $\log n$ for OER. Thus we have an instance of support for the conjecture that good uniprocessor algorithms may be outperformed by relatively poor uniprocessor algorithms on multiprocessor computers.

There are two cautions which must be observed. This study assumes that the configurations required can always be set up to completely meet the resource requirements of an executing phase. We have not considered the problem of restructuring the algorithm in the presence of insufficient resources. It is clear, however, that linear speed-up can be retained, since the problem partitions with respect to the number of blocks. The issue of the utility of packet switching versus circuit switching, memory switching has not been addressed in detail. From a strict performance point of view, circuit switching fits in naturally with algorithms that repeatedly require block transfers over fixed geometries, as is the case for block OEE and block OER. The packet switching implementation is only marginally inferior to the circuit switching implementation of OEE. In cases where circuits cannot be established, packet switching may offer a viable alternative with only a minor degree of performance degradation.

Acknowledgments. We wish to thank the two referees and the editor of this journal for their suggestions for making this paper more readable.

REFERENCES

[1] [Age81] T. K. M. AGERWALA AND B. J. LINT, *Communication issues in the design and analysis of parallel algorithms*, IEEE Trans. Software Engrg., SE-7/2 (1981), pp. 174–188.

[2] [Bro82] J. C. Browne and G. J. Lipovski, *Reconfigurable network architectured computer systems*, in Proc. Internat. Workshop on High-Level Language Computer Architectures, Dec. 1982, pp. 40–49.

[3] [Bur77] BSP: *Overview, Perspective, Architecture*, Burroughs Corp., 1977.

[4] [Bus62] B. Bussel, *Properties of a variable structure computer system in the solution of parabolic partial differential equations*, Ph.D dissertation, Electrical Engineering Dept., Univ. California, Los Angeles, 1962.

[5] [DeG80] R. D. DeGroot and M. M. Malek, *Resource allocation for macro-pipelines*, in Proc. of Distributed Data Acquisition, Computation and Control Symposium, Dec. 1980, pp. 23–27.

[6] [DeG81] R. D. DeGroot, *Mapping computation structures to regular* SW *banyans*, Ph.D dissertation, Dept. Computer Science, Univ. Texas at Austin, Austin, TX, Dec. 1981.

[7] [Fly66] M. J. Flynn, *Very high speed computing systems*, Proc. IEEE, 54 (1966), pp. 1901–1909.

[8] [Gan81] D. Gannon, *On mapping non uniform P.D.E. structures and algorithms onto uniform array architectures*, in Proc. 10th Intl. Conference on Parallel Processing, Bellaire, MI, Aug. 1981, IEEE Computer Society, Long Beach, CA, pp. 100–105.

[9] [Gen78] W. M. Gentleman, *Some complexity results for matrix computations on parallel processors*, J. Assoc. Comput. Mach., 25 (1978), pp. 112–115.

[10] [Gro80] C. Grosch, *The effect of data transfer pattern of an array computer on the efficiency of some algorithms for the tridiagonal and Poisson problem*, in Array Architectures for Computing in the 80's and the 90's, ICASE Workshop, Hampton, Virginia, April, 1980.

[11] [Hel77] D. Heller, *Direct and iterative methods for block tridiagonal linear systems*, Ph.D dissertation, Computer Science Dept. Carnegie–Mellon Univ., Pittsburgh, PA, 1977.

[12] [Hin72] R. G. Hintz and D. P. Tate, *Control data* STAR-100 *design*, in 6th Annual IEEE Computer Society Conference, COMPCON, Institute of Electrical and Electronics Engineers, New York, 1972, pp. 1–4.

[13] [Kap80] R. N. Kapur, U. V. Premkumar and G. J. Lipovski, *Organization of the* TRAC *processor-memory subsystem*, Proc. NCC 80, AFIPS Press, Arlington, VA, 1980, pp. 623–629.

[14] [Kap81] R. N. Kapur and J. C. Browne, *Block tridiagonal system solution on reconfigurable array computers*, Proc. 10th International Conference on Parallel Processing, Bellaire, MI, Aug. 1981, IEEE Computer Society, Long Beach, CA, pp. 92–99.

[15] [Kap82] R. N. Kapur, *On the synthesis and analysis of reconfigurable computer programs*, Ph.D Dissertation, Dept. of Electrical Engineering, Univ. Texas, Austin, May, 1982.

[16] [Kar82] S. P. Kartashev and S. I. Kartashev, *Distribution of programs for a system with dynamic architecture*, IEEE Trans. Comput., C-31 (1982), pp. 488–514.

[17] [Law75] D. H. Lawrie, *Access and alignment of data in array processors*, IEEE Trans. Comput., C-24 (1975), pp. 1145–1155.

[18] [Lin79] B. J. Lint, *A study of communication issues in parallel algorithms*, Ph.D. Dissertation, Dept. of Electrical Engineering, Univ. Texas, TX, 1979.

[19] [Lip73] G. J. Lipovski and L. R. Goke, *Banyan networks for partitioning of multi-processor systems*, in Proc. 1st Symposium on Computer Architecture, 1973, IEEE Computer Society, Long Beach, CA, pp. 21–30.

[20] [Lip77] G. J. Lipovski and A. Tripathi, *A reconfigurable varistructure array processor*, in Proc. 1977 International Conference on Parallel Processing, Bellaire, MI, August 1977, IEEE Computer Society, Long Beach, CA, pp. 165–174.

[21] [Mad75] N. Madsen and G. Rodrigue, *A comparison of direct methods for tridiagonal system solution on the* STAR-100, Lawrence Livermore Laboratory, Livermore, CA, 1975.

[22] [Ole78] P. Oleinick, *The implementation and evaluation of parallel algorithms on the* C.mmp, Ph.D Dissertation, Dept. of Computer Science, Carnegie–Mellon Univ., Pittsburgh, PA, No. 1978.

[23] [Pre79] U. V. Premkumar, R. N. Kapur and G. S. Lipovski, *Interprocessor communication in* TRAC, 1st International Conference on Distributed Computers and Systems, Aug. 1979, pp. 51–62.

[24] [Pre80] U. V. Premkumar, R. Kapur, M. Malek, G. J. Lipovski and P. Horne, *Design and implementation of the banyan interconnection network in* TRAC, Proc. of NCC 80, AFIPS Press, pp. 633–642.

[25] [Sej80] M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D. P. S. Charlu and G. J. Lipovski, *An overview of the Texas reconfigurable array computer*, Proceedings of NCC 80, AFIPS Press, 1980, pp. 631–641.

[26] [Sie79] H. J. Siegel, et al., PASM, TR-EE 79.40, School of Electrical Engineering, Purdue Univ., Lafayette, IN, August 1979.

[27] [Sny81] L. Snyder, *Introduction to the configurable highly parallel computer*, Report CSD-TR-351, Purdue Univ., Lafayette, IN, May, 1981.

[28] [Sto71] H. S. STONE, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C-20 (1971), pp. 153–161.

[29] [Tra76] J. F. TRAUB, D. K. STEVENSON AND D. F. HELLER, *Accelerated iterative methods for the solution of tridiagonal systems on parallel computers*, J. Assoc. Comput. Mach., 23 (1976), pp. 636–654.

[30] [Vic79] C. R. VICK, *A dynamically reconfigurable distributed computing system*, Ph.D. Dissertation, Auburn Univ., Auburn, Alabama, 1979.

[31] [Wul72] W. WULF AND G. C. BELL, C.MMP—*A multi-mini processor*, Proc. AFIPS FJCC, 42 (1972), pp. 765–777.

[32] [You72] D. M. YOUNG AND R. T. GREGORY, *A Survey of Numerical Mathematics*, 2 vol., Addison-Wesley, Reading, MA, 1972.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

A VARIANT OF HUBER ROBUST REGRESSION[*]

CHARLES G. BONCELET, JR.[†] AND BRADLEY W. DICKINSON[†]

Abstract. In this paper, we develop a variant of Huber robust regression. Our approach is inspired by Huber's M estimates; however, we deal with the scale estimate differently. The class of estimators we develop includes least squares and least absolute residuals as limiting cases and can be considered as a generalization of the trimmed mean to regression. We exhibit an algorithm to compute these estimates and prove its correctness. Also, we show how to extend our estimators to include weighting, equality and inequality constraints, and the addition or deletion of data points.

Key words. linear regression, constrained regression, l_1 estimation, robust regression, trimmed mean

1. Introduction. Linear regression is one of the most studied and important problems in statistics. The following model is assumed:

$$y = X\beta + \varepsilon, \tag{1}$$

where y is an $n \times 1$ vector of observations, X is an $n \times m$ known matrix assumed to be of rank m, and ε is an $n \times 1$ vector of random errors. Classically, this problem is solved by the method of least squares (LS). Letting $x_i' = i$th row of X, one obtains

$$\sum_{i=1}^{n} (y_i - x_i'\beta)^2 = \min!. \tag{2}$$

The solution to this problem, denoted by β_{LS}, is easily found.

$$\beta_{\text{LS}} = (X'X)^{-1}X'y. \tag{3}$$

As is well known, this estimate is the best linear unbiased estimate (BLUE) and is the maximum-likelihood estimate (MLE) if the errors are Gaussian.

Unfortunately, the LS estimator is not robust and can be quite poor for heavy-tailed noise distributions. (Throughout this paper, the word *robust* will be used loosely and will represent the notions that the estimator's efficiency remains acceptable even if the noise is heavier-tailed than Gaussian and that the estimator is *resistant* to a suitably small number of *outliers*, points whose error is comparatively large. These notions are discussed in considerable detail and in full rigor in the monograph by Huber [1].) Most robust procedures are based on Huber's M-estimates.

$$\sum_{i=1}^{n} \rho(y_i - x_i'\beta) = \min!, \tag{4}$$

where $\rho(x)$ is a robustifying loss function. Probably the most widely used of these is the least absolute residuals (LAR) estimate where $\rho(x) = |x|$. Algorithms to compute the LAR estimate are discussed in Barrodale and Roberts [2] and Bloomfield and Steiger [3].

[*] Received by the editors March 15, 1983, and in revised form June 20, 1983. This research was supported by the Air Force Office of Scientific Research, under grant AFOSR-81-0186, and by the National Science Foundation under grant ECS82-05772.

[†] Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey 08544.

The robust estimator we are most interested in was suggested by Huber [4] in analogy to the location problem for which he was able to prove a min-max result [5].

$$\rho_H = \begin{cases} kx - k^2/2, & x \geqq k, \\ x^2/2, & -k \leqq x \leqq k, \\ -kx - k^2/2, & x \leqq -k. \end{cases}$$

k is chosen so that the estimator has suitable efficiency at the normal model. Substituting this loss function into (4) and differentiating with respect to β enables one to characterize the solution, denoted β_H, as

$$(5) \qquad 0 = \sum_{i=1}^{n} x_{ij}\psi_H(y_i - x_i'\beta_H), \qquad j = 1, m,$$

where

$$\psi_H(x) = \max(\min(x, k), -k).$$

Unfortunately, this estimator is not scale invariant and the scale is usually not known in advance. To alleviate this problem, the residuals $r_i = y_i - x_i'\beta$ are replaced by suitably scaled versions r_i/s where s is a robust estimate of scale. Equations (5) are usually solved in an iterative manner. See Huber [1], Dutter [6], and Huber and Dutter [7] for details. For the location problem, Huber [5] made three heuristic proposals to deal with the computation of the estimate when the scale is unknown. He suggests that k be picked beforehand (by past experience or by the statistician's intuition) and that two simultaneous equations be solved as exactly as possible for the scale and location estimates. He argues that this approach is asymptotically equivalent to Winsorizing.

We propose that *k be chosen so that the number of data points whose residual is larger in magnitude than k be equal to a prespecified fraction of the data.* We need the following definitions. Let

$$\mathbf{A} = \{i: r_i \leqq -k\}, \quad \mathbf{B} = \{i: -k \leqq r_i \leqq k\}, \quad \mathbf{C} = \{i: r_i \geqq k\}.$$

Also, let n_A = number of elements in $\mathbf{A}$, n_B = number in $\mathbf{B}$, and n_C = number in $\mathbf{C}$. Then choose k so that $(n_A + n_C) = \min(n - m, \lceil 2\alpha n \rceil)$ where $0 \leqq \alpha \leqq \frac{1}{2}$. In general, $n_A + n_C \leqq n - m$ and at the LAR estimate equals $n - m$. The second clause, $\lceil 2\alpha n \rceil$ is necessary because of the granularity of the integers. (This is precisely the same problem faced in defining the trimmed mean for location. Usually some linear combination of the nearest two order statistics is used. However, this reduces the robustness of the estimator.) The LS estimate corresponds to $\alpha = 0$; the LAR estimate to $\alpha = \frac{1}{2}$.

If we define $a = n \times 1$ vector with $a_i = 1$ if $i \in \mathbf{A}$ and $=0$ if not, $B = n \times n$ diagonal matrix with $b_{ii} = 1$ if $i \in \mathbf{B}$ and $=0$ if not, and, similarly, $c = n \times 1$ vector with $c_i = 1$ if $i \in \mathbf{C}$ and $=0$ if not, then we can rewrite (5) as

$$\begin{aligned} 0 &= \sum_{i=1}^{n} x_{ij}\psi_H(y_i - x_i'\beta) && j = 1, m, \\ (6) \qquad &= \sum_{i \in \mathbf{A}} -x_{ij}k + \sum_{i \in \mathbf{B}} x_{ij}(y_i - x_i'\beta) + \sum_{i \in \mathbf{C}} x_{ij}k && j = 1, m, \\ &= -(X'BX)\beta + X'By + kX'(c - \alpha). \end{aligned}$$

The solution to (6) for fixed k is obvious. (By assumption, X has rank m; it is shown in Theorem 2 that this insures that $X'BX$ has rank m and is therefore invertible.)

$$(7) \qquad \beta = (X'BX)^{-1}X'[By + k(c - a)].$$

For location problems ($m=1, x_{i1}=1$), (7) reduces to

$$\beta=\frac{\sum_{i\in\mathbf{B}} y_i}{n_B}+k\frac{(n_C-n_A)}{n_B}. \tag{8}$$

If $n_A=n_C=\alpha n$ (which will frequently occur when the noise is symmetric), then (8) reduces to the α-trimmed mean. Therefore this estimate can be considered to be a generalization of the α-trimmed mean and will be denoted here as β_α.

Unfortunately, k, as it is presently defined, is almost always not unique. This nonuniqueness will be dealt with shortly. First, some definitions and background are needed. Let $\hat{\beta}=(X'BX)^{-1}X'By=$ least squares estimate based on those data points in $\mathbf{B}$ and let $\delta\beta=(X'BX)^{-1}X'(c-a)=$ correction based on "large" deviation points. Also let $\hat{r}_i=y_i-x_i'\hat{\beta}$ and $\delta r_i=x_i'\delta\beta$. Then $\beta=\hat{\beta}+k\delta\beta$ and $r=\hat{r}-k\delta r$. Consider a data point in $\mathbf{A}$. We have

$$r_i\leqq -k$$

or, substituting,

$$\hat{r}_i-k\delta r_i\leqq -k$$

or, trivially,

$$\hat{r}_i\leqq -k(1-\delta r_i). \tag{9}$$

If $1-\delta r_i>0$ then $k\leqq -\hat{r}_i/(1-\delta r_i)$; else, if $1-\delta r_i<0$ then $k\geqq -\hat{r}_i/(1-\delta r_i)$. If $1-\delta r_i=0$ then $\hat{r}_i\leqq 0$. Limiting relations for points in $\mathbf{B}$ and $\mathbf{C}$ are derived in the same manner and are summarized in Table 1.

This suggests an algorithm to compute the estimate.

1. Guess a partition with $n_A+n_C=2\alpha n$.
2. Compute $\hat{\beta}$ and $\delta\beta$ and from them $\hat{r}$ and δr.
3. Compute k_L, k_U. If $k_L\leqq k_U$ accept $\beta=\hat{\beta}+k\delta\beta$, $k\in[k_L,k_U]$ as the answer. If not, return to step 1.

The problem with this algorithm is the absence of guidance for guessing the correct partition given that the previous one was incorrect. Furthermore, there is an exponential number of partitions with $n_A+n_C=2\alpha n$. A better algorithm will be developed below, but first some definitions are needed. If a point has $r_i=\pm k$, then that point is at a *corner*. If some partition $(\mathbf{A},\mathbf{B},\mathbf{C})$ yields $k_L\leqq k_U$, then that partition is *valid*. Let us assume we have a valid partition for which $n_A+n_C<2\alpha n$. Let $k=k_L$, then some point, say l, is at a corner. For definiteness, let $r_l=k$ and $l\in\mathbf{B}$. Further assume that l is the only point at a corner. Then

$$\begin{aligned}0&=(X'BX)\beta-X'[k(c-a)+By]\\&=(X'BX)\beta-X'[k(c-a)+By]+x_l(y_l-x_l'\beta-k)\\&=(X'BX-x_lx_l')\beta-X'[k(c-a+e_l)+By-e_ly_l].\end{aligned} \tag{10}$$

If $\bar{a}=a$, $\bar{\mathbf{B}}=\mathbf{B}-e_le_l'$, and, $\bar{c}=c+e_l$ and the corresponding changes are made to $\mathbf{A}$, $\mathbf{B}$, and, $\mathbf{C}$, then this is a new partition which is valid and which yields the same value of β as the previous one, but one for which $\hat{\beta}$ and $\delta\beta$ are different. We will say that the point has undergone a *transition* and will use the symbolic notation $\mathbf{B}\to\mathbf{C}$. From Table 1, it is easy to see that initially (for $l\in\mathbf{B}$) $\delta r_l>-1$. We prove in Theorem 3 that after the transition $\overline{\delta r_l}>-1$. Hence from Table 1, it can be seen that $k_L=\overline{k_U}$ i.e., that the

TABLE 1

Partition	$\hat{r}_i$	δr_i	k
A	>0	>1	$k \geqq -\hat{r}_i/(1-\delta r_i)$
		$\leqq 1$	n.a.*
	<0	$\geqq 1$	$k \geqq 0$
		<1	$k \leqq -\hat{r}_i/(1-\delta r_i)$
	$=0$	<1	n.a.
		$\geqq 1$	$k \geqq 0$
B	>0	$\leqq -1$	n.a.
		>-1	$k \geqq \hat{r}_i/(1+\delta r_i)$
		>1	$k \leqq -\hat{r}_i/(1-\delta r_i)$
	<0	<-1	$k \leqq \hat{r}_i/(1+\delta r_i)$
		<1	$k \geqq -\hat{r}_i/(1-\delta r_i)$
		$\geqq 1$	n.a.
	$=0$	$\lvert\cdot\rvert \leqq 1$	$k \geqq 0$
		$\lvert\cdot\rvert > 1$	n.a.
C	>0	>-1	$k \leqq \hat{r}_i/(1+\delta r_i)$
		$\leqq -1$	$k \geqq 0$
	<0	<-1	$k \geqq \hat{r}_i/(1+\delta r_i)$
		$\geqq -1$	n.a.
	$=0$	$\leqq -1$	$k \geqq 0$
		>-1	n.a.

* Not allowed.

previous lower limit for k is now its upper limit. Furthermore, since point l is the only point at a corner, the new lower limit, $\overline{k_L}$, is strictly less than k_L. Using the rules given in Table 1, $\overline{k_L}$ can be found and a new point, say $\bar{l}$, is at a corner. This process can continue until termination. Equation (10) and the argument following easily carry over to the other transitions, $\mathbf{B} \rightarrow \mathbf{A}$, $\mathbf{C} \rightarrow \mathbf{B}$ and $\mathbf{A} \rightarrow \mathbf{B}$. For brevity we omit these cases.

In general, there could be more than one partition with $n_A + n_C = 2\alpha n$, each corresponding to a different interval, $[k_L, k_U]$. Let superscript j denote the different partitions. Then we suggest that *k be chosen equal to* $\max_j k_L^j$. This corresponds to the first such partition obtained by our algorithm.[1] The choice $k = k_L$ affords the maximum insensitivity to the outlying points.

[1] Our experience suggests that this partition is almost always unique.

The only remaining detail is to show how to start the algorithm. Let $\mathbf{A}=\mathbf{C}=\{\varphi\}$, $\mathbf{B}=\{1,2,\cdots,n\}$. Then $\hat{\beta}=(X'BX)^{-1}X'By=(X'X)^{-1}X'y=\beta_{LS}$. Further, $\delta\beta=(X'BX)^{-1}X'(c-a)=0$. Let $k=k_L=\max|r_i|$. This is a valid partition ($k_U=+\infty$). The algorithm, which we call the ABC algorithm, is summarized below.

```
THE ABC ALGORITHM
Input: $X$ an $n\times m$ matrix of rank $m$, $y$ an $n\times 1$ vector, and $\alpha$
Output: $\beta_\alpha$
Step 1: Set $\mathbf{A}=\mathbf{C}=\{\varphi\}$ and $\mathbf{B}=\{1,2,\cdots,n\}$. Compute Least Squares estimate, $\beta_{\mathrm{LS}}$. Set $\hat{\beta}=\beta_{\mathrm{LS}}$, $\delta\beta=0$. Compute $r=\hat{r}=y-X\hat{\beta}$ and set $\delta r=0$. Set $k=k_L=\max|r_i|$. Requires $O(nm^2)$ operations.
Step 2: At each stage do:
    a) Make transition (bookkeeping). $O(1)$ operations.
    b) Compute $(X'BX\pm x_lx_l')^{-1}=(X'BX)^{-1}\mp\frac{(X'BX)^{-1}x_lx_l'(X'BX)^{-1}}{1\pm x_l'(X'BX)^{-1}x_l}$. $O(m^2)$ operations.
    c) Compute $\delta\beta=(X'BX\pm x_lx_l')^{-1}X'(c-a)$, $\hat{\beta}=\beta-k\delta\beta$. $O(m^2)$ operations.
    d) Compute $\delta r=X\delta\beta$, $\hat{r}=r+k\delta r$. $O(nm)$ operations.
    e) Find new $k_L$ using the rules in Table 1. $O(n)$ operations.
    f) If $n_A+n_C=\min(n-m,\lceil 2\alpha n\rceil)$, stop and accept $\beta=\hat{\beta}+k_L\delta\beta$ as the estimate. If not, go to step 2a.
```

Some comments on this algorithm are due.

1. In practice, $X'BX$ will be kept in factored form, e.g. as LDL'. Step 2b will be replaced by a modification of the factors. See Gill et al. [8] for details on updating factorizations following rank 1 changes. At any rate, these updates can be done in $O(m^2)$ operations.

2. For numerical reasons, especially in large problems, it may be necessary to periodically recompute the solution for a given partition "from scratch" using the definitions for $\hat{\beta}$ and $\delta\beta$ rather than the update formula. This is easily done. This requires as much work as does computing the LS solution initially.

3. The principal open question regarding this algorithm is "Is it polynomial?"[2] Since the initial least squares computation is polynomial and each stage can be done in polynomial time, the question rests on the number of stages required. In general, the LAR solution corresponds to $n_B=m$. Therefore, at least $n-m$ stages may be required to compute it. The largest problem (and the worst case) we have solved to date with the ABC algorithm had $n=224$ and $m=29$ and required 203 stages as compared with $n-m=195$. However, we have not tested this algorithm extensively and cannot assert that the behavior is never worse than this or even that this behavior is, in some sense, average. Nor can we prove that $O(n)$ stages are sufficient.

4. If one assumes that $O(n)$ stages are sufficient, then, since each stage is $O(nm)$, the whole algorithm is $O(n^2m)$. For large n, this growth may be unacceptable. However, many large problems are sparse and for many sparse problems computing $\delta r=X\delta\beta$ can be done in $O(n)$ operations. In this case, $O(n^2+nm^2)$ will be sufficient. Of course, β_a can be computed in approximately $2\alpha n$ stages.

[2] This question is of some theoretical interest because an easy variant of the ABC algorithm solves the linear inequalities (LI) problem: Does there exist a β such that $X\beta\geqq y$? LI is equivalent to linear programming (LP) for which there is a polynomial time algorithm, the Soviet ellipsoid algorithm (Khachian [9]).

5. This algorithm can be used to compute the Huber estimate for fixed scale. Replace Step 2f by: If $k_H \in [k_L, k_U]$, accept $\beta_H = \hat{\beta} + k_H \delta\beta$ as the answer. If not, go to 2a.

6. It is possible to envision a scheme where the stopping fraction is chosen dynamically. More work needs to be done in this area.

2. Correctness of the ABC algorithm. In this section, we will show that the ABC algorithm correctly finds the LAR estimate and that the algorithm works correctly.

The step in going from the minimization problem (4) to the "normal equations" (5) is not valid when $k=0$ since $\psi_H(x) = \operatorname{sign}(x)$ and is not continuous at $x=0$. In the following theorem, it is shown that the ABC algorithm does converge to the correct solution.

THEOREM 1. *Let $\beta = \hat{\beta} + k\delta\beta$ be the minimizing solution of* (4) (*i.e. solves* (5)) *with $k \in (0, k_U]$ and $k_U > 0$. Assume further that X and y are bounded. Then $\beta_{\text{LAR}} = \hat{\beta}$.*

Comment. Such a solution with $k_U > 0$ exists since originally $k_U = +\infty$ and always $k_U \geqq k_L$. Take the first solution with $k_L = 0$.

Proof. By contradiction. Assume that there exists some $\tilde{\beta} \neq \hat{\beta}$ with

$$\sum_{i=1}^{n} |y_i - x_i'\tilde{\beta}| < \sum_{i=1}^{n} |y_i - x_i'\hat{\beta}|;$$

then, for k sufficiently small, $\beta = \hat{\beta} + k\delta\beta$ will not be optimal in (4) which contradicts the assumption. Let

$$\varepsilon = \sum_{i=1}^{n} |y_i - x_i'\hat{\beta}| - \sum_{i=1}^{n} |y_i - x_i'\tilde{\beta}| > 0.$$

Let $Q(\beta, k)$ be defined as below.

$$Q(\beta, k) = \sum_{i \notin \mathbf{B}} \left\{ |y_i - x_i'\beta| - \frac{k}{2} \right\} + \frac{1}{2k} \sum_{i \in \mathbf{B}} (y_i - x_i'\beta)^2.$$

Then, for k sufficiently small (so that $|y_i - x_i'\tilde{\beta}| \leqq k \Rightarrow y_i - x_i'\tilde{\beta} = 0$ and $|y_i - x_i'\hat{\beta}| \leqq k \Rightarrow y_i - x_i'\hat{\beta} = 0$),

$$Q(\tilde{\beta}, k) = \sum_{i \notin \mathbf{B}} |y_i - x_i'\tilde{\beta}| - k\frac{(\tilde{n}_A + \tilde{n}_C)}{2},$$

$$\begin{aligned} Q(\hat{\beta} + k\delta\beta, k) &= \sum_{i \notin \mathbf{B}} |y_i - x_i'(\hat{\beta} + k\delta\beta)| - k\frac{\hat{n}_A + \hat{n}_C}{2} + \frac{1}{2k} \sum_{i \in \mathbf{B}} (y_i - x_i'(\hat{\beta} + k\delta\beta))^2 \\ &= \sum_{i \notin \mathbf{B}} |y_i - x_i'\hat{\beta} - kx_i'\delta\beta| - k\frac{\hat{n}_A + \hat{n}_C}{2} + \frac{1}{2k} \sum_{i \in \mathbf{B}} (kx_i'\delta\beta)^2. \end{aligned}$$

Using the relation $|a-b| \geqq |a| - |b|$ gives

$$\geqq \sum_{i \notin \mathbf{B}} |y_i - x_i'\hat{\beta}| + k\left[-\sum_{i \notin \mathbf{B}} |x_i'\delta\beta| - \frac{\hat{n}_A + \hat{n}_C}{2} + \frac{1}{2} \sum_{i \in \mathbf{B}} (x_i'\delta\beta)^2 \right].$$

Letting η = term in brackets [ ] and using $0 = y_i - x_i'\hat{\beta}$ for $i \in \mathbf{B}$ gives

$$= \sum_{i=1}^{n} |y_i - x_i'\hat{\beta}| + k\eta.$$

Therefore,

$$\begin{aligned}Q(\hat{\beta}+k\delta\beta, k)-Q(\tilde{\beta}, k) &\geqq \sum_{i=1}^{n} |y_i - x_i'\hat{\beta}| + k\eta - \sum_{i=1}^{n} |y_i - x_i'\tilde{\beta}| - k\frac{(\tilde{n}_A+\tilde{n}_C)}{2} \\ &= \varepsilon + k\left\{\eta - \frac{\tilde{n}_A+\tilde{n}_C}{2}\right\} \\ &> 0\end{aligned}$$

for k sufficiently small by the boundedness of η which follows from the boundedness assumption on X and y, a contradiction. □

Since X is assumed to have rank m, the initial LS estimate is well-defined. However, it is not necessary that $X'DX$ will have rank m for all choices of the diagonal matrix D. In particular, it is not clear that $X'BX$ will have full rank. In the next theorem, we prove that it does.

THEOREM 2. *The matrix $X'BX$ has rank m.*

Proof. It is clear that initially $X'BX = X'X$ has rank m. It is also clear that the transitions, $\mathbf{A}\to\mathbf{B}$ and $\mathbf{C}\to\mathbf{B}$ cannot reduce the rank of $X'BX$. Therefore, it is sufficient to consider only the transitions $\mathbf{B}\to\mathbf{A}$ and $\mathbf{B}\to\mathbf{C}$. If $X'BX$ were to go singular, then there must be a first time it does. Without loss of generality, the rows of X can be permuted and a linear change of variables made in β so that X has the following form:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & G \\ u & V \end{bmatrix}$$

with the first row corresponding to the transition element, the next $n_B - 1$ rows corresponding to the other elements in $\mathbf{B}$, and the last $n - n_B$ rows corresponding to those elements in $\mathbf{A}$ and $\mathbf{C}$. G has rank $m-1$. Then

$$BX(X'BX)^{-1}X'B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & G(G'G)^{-1}G' & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

which implies that $\hat{r}_1 = 0$ and from Table 1 it can be seen that point 1 cannot be the limiting point for k_L and, hence that it is not a transition element. □

It is conceivable that the ABC algorithm could get "stuck" if a point moved from, say, $\mathbf{B}$ to $\mathbf{A}$, then back again immediately. In the next theorem, we prove that this cannot happen.

THEOREM 3. *When a point undergoes a transition, it will remain in its new partition for at least one stage.*

Proof. Consider the $\mathbf{B}\to\mathbf{C}(\mathbf{C}\to\mathbf{B})$ transition, the other two being very similar. Let l be the index of the point undergoing the transition. Then $\delta r_l > -1$ $(\delta r_l < -1)$. Let $\overline{\delta r_l}$ denote the new value of δr_l after the transition. Then, it will be shown below that $\overline{\delta r_l} > -1$ (< -1) which from Table 1 implies that the point cannot be a transition element. Let the upper sign correspond to the $\mathbf{B}\to\mathbf{C}$ transition and the lower to the $\mathbf{C}\to\mathbf{B}$ transition.

$$\begin{aligned}\overline{\delta r_l} &= x_l'(X'BX \mp x_l x_l')^{-1}X'(c-a\pm e_l) \\ &= x_l'\left\{(X'BX)^{-1} \pm \frac{(X'BX)^{-1}x_l x_l'(X'BX)^{-1}}{1 \mp x_l'(X'BX)^{-1}x_l}\right\}X'(c-a\pm e_l).\end{aligned}$$

Let $h_l = x_l'(X'BX)^{-1}x_l$ and $1 > h_l > 0$ since it is a diagonal element of the rank m idempotent matrix, $BX(X'BX)^{-1}X'B$, and because the matrix $\bar{B}X(X'\bar{B}X)^{-1}X'\bar{B}$ also has rank m by Theorems 2 and 3. Then

$$= \delta r_l \pm h_l \pm \frac{h_l}{1 \mp h_l}(\delta r_l \pm h_l)$$

$$= \frac{\delta r_l \mp \delta r_l h_l \pm h_l - h_l^2 \pm h_l \delta r_l + h_l^2}{1 \mp h_l} = \frac{\delta r_l \pm h_l}{1 \mp h_l}.$$

So

$$\overline{\delta r_l} \gtrless -1 \Leftrightarrow \frac{\delta r_l \pm h_l}{1 \mp h_l} \gtrless -1$$

$$\Leftrightarrow \delta r_l \pm h_l \gtrless -1 \pm h_l$$

$$\Leftrightarrow \delta r_l \gtrless -1. \qquad \square$$

In the absence of *ties*—times when two or more points reach a corner at the same time—Theorem 3 is sufficient to guarantee that the ABC algorithm reaches the LAR estimate in finite time. For then at each stage, k_L decreases monotonically. This implies that a partition is never repeated. Since there are only a finite number of partitions, the algorithm is then finite. Ties, which in the "real world" will occur with zero probability, can be dealt with by insuring that a partition is never repeated. For instance, this can be done by using Breadth First Search (BFS) to trace out the partition "tree". (See Aho et al. [10] for details on BFS.) In our experience, however, we have found that ties can be ignored. They occur infrequently and when they do, they seldom cause problems.[3]

3. Location problems. Since location problems are a special case of regression, the ABC algorithm can be applied directly to them. However, since the location case is so much simpler than regression, it is possible to prove much more. For instance, we will prove that the location estimate can be found in $O(n \log n)$ time and that ties are no problem. In Theorem 4, it is proved that a data point, once it leaves **B**, never returns. Since there are only n data points, at most $n-1$ stages can occur. Furthermore, if the data is presorted (which requires $O(n \log n)$ time in general), then each stage can be done in $O(1)$ time. The location estimate is

$$\beta = \frac{\sum_{i\in \mathbf{B}} y_i}{n_B} + k\frac{(n_C - n_A)}{n_B}. \tag{8}$$

Furthermore, for each data point, $\delta r_i = x_i'\delta\beta = (n_C - n_A)/n_B$ independent of i.

THEOREM 4. *Let* $\delta r_i = (n_C - n_A)/n_B$. *Let* $k > 0$. *Then* $|\delta r_i| \leqq 1$.

Comment. From Table 1, it can be seen that if $|\delta r| \leqq 1$, then the transitions, $\mathbf{A} \to \mathbf{B}$ and $\mathbf{C} \to \mathbf{B}$ are forbidden.

Proof. Consider those points, i, contained in **B**.

$$-k \leqq y_i - \frac{\sum_{i\in\mathbf{B}} y_i}{n_B} - k\frac{n_C - n_A}{n_B} \leqq k.$$

Sum over $i \in \mathbf{B}$.

$$-kn_B \leqq \sum_{i\in\mathbf{B}} y_i - \sum_{i\in\mathbf{B}} y_i - k(n_C - n_A) \leqq kn_B.$$

[3] None of our implementations of this algorithm deals with ties in a sophisticated way and never have ties caused us any problems.

Dividing by kn_B yields

$$-1 \leqq \frac{n_C - n_A}{n_B} \leqq 1. \qquad \square$$

The improved algorithm results from the observation that only two of the points in **B** at any stage are candidates for transitions—the largest and the smallest. For all points in **B**

$$(11) \qquad -k \leqq y_i - \hat{\beta} - k\delta\beta \leqq k,$$

or, rewriting (11),

$$(12) \qquad -k(1-\delta\beta) + \hat{\beta} \leqq y_i \leqq k(1+\delta\beta) + \hat{\beta}.$$

In this form it is easy to see that only the maximum and minimum values of y_i can determine k_L. To adapt the ABC algorithm for location, the following steps are needed:

1. Sort the data. Requires $O(n \log n)$ operations.
2. Delete Step 2d. (There is no need to calculate the residuals at each stage.)
3. Replace Step 2e with: Find new k_L using (12) and the rules of Table 1 using only the maximum and minimum values of y_i for $i \in B$.

4. Adding and deleting data points. In many applications it is desirable to have a recursive implementation of the ABC algorithm. For instance, in "on-line" problems, the data arrives sequentially. Given β^{n-1}, the estimate with $n-1$ data points, can β^n, the updated estimate, be found easily? Conversely, in some problems, deletion of data points is necessary. In "moving average" problems both needs arise.

Consider the addition of a data point, $(\bar{x}, \bar{y})$, first. Assume that (**A**, **B**, **C**) is a valid partition for $k \in [k_L, k_U]$. Then our algorithm is a two-step procedure:

1. With k fixed, find a new valid partition (**A**, **B**, **C**).
2. Increase or decrease k until the desired fraction of data points in **A** and **C** is achieved.

Step 2 can be done using the standard ABC algorithm if k needs to be decreased and with obvious changes if k needs to be increased. Hence we will concentrate only on Step 1.

Consider the following:

$$(7) \qquad (X'BX)\beta = X'[k(c-a) + By].$$

It is clear that if $\bar{y} - \bar{x}'\beta = 0$, then that point can be added or deleted trivially without affecting β. Put the new point in **B** and write

$$(13) \qquad (X'BX + \bar{x}\bar{x}')(\beta + \Delta\beta) = X'[k(c-a) + By] + \bar{x}[\bar{x}'\beta + d(\bar{y} - \bar{x}'\beta)].$$

For $d = 0$ and $\Delta\beta = 0$, (13) reduces to (7). Solving for $\Delta\beta$ gives

$$(14) \qquad \Delta\beta = d(X'BX + \bar{x}\bar{x}')^{-1}(\bar{y} - \bar{x}'\beta).$$

Now increase d from 0 to 1. If a point reaches a corner for $d = \bar{d} < 1$, do:

1. Let $\hat{\beta} = \beta + \bar{d}(X'BX + \bar{x}\bar{x}')^{-1}(\bar{y} - \bar{x}'\beta)$.
2. Make appropriate transition. If the new point is moving from **B** to **A** or from **B** to **C**, then stop. Else, let $\hat{B}$ denote the new "B" matrix. Then

$$(15) \qquad (X'\hat{B}X + \bar{x}\bar{x}')\hat{\beta} = X'[k(c-a) + By] + \bar{x}[\bar{x}'\beta + \bar{d}(\bar{y} - \bar{x}'\beta)]$$

as before, with

$$(16)\qquad \begin{aligned}&(X'\hat{B}X+\bar{x}\bar{x}')(\hat{\beta}+\Delta\beta)\\ &\quad =X'[k(c-a)+By]+\bar{x}[\bar{x}'\beta+\bar{d}(\bar{y}-\bar{x}'\beta)+(d-\bar{d})(\bar{y}-\bar{x}'\beta)].\end{aligned}$$

Initially $d=\bar{d}$ and $\Delta\beta=0$. Solving for $\Delta\beta$ yields

$$(17)\qquad \Delta\beta=(X'\hat{B}X+\bar{x}\bar{x}')^{-1}(\bar{y}-\bar{x}'\beta)(d-\bar{d}).$$

Continue in this manner until $d=1$.

Deletion of a point is essentially the reverse of adding a point. (We assume that deleting the point does not reduce the rank of $X'BX$. By Theorem 2, the rank will not decrease unless $\hat{r}=0$.) Let the point to be deleted be denoted $(\bar{x},\bar{y})$. Then the approach is to modify $\bar{y}$ continuously until $\bar{r}=0$. If $(\bar{x},\bar{y})$ is in **A** or **C**, change $\bar{y}$ so that the point is at its respective corner. Move the point into **B** (while it is still at the corner). This process leaves β unchanged. Change the definition of B so that it excludes the point in question, i.e., the appropriate diagonal element is 0, not 1. Then

$$(18)\qquad (X'BX+\bar{x}\bar{x}')\beta=X'[k(c-a)+By]+\bar{x}\bar{y},$$

with

$$(19)\qquad \bar{r}=\bar{y}-\bar{x}'\beta.$$

Let $\bar{y}\to\bar{y}+\Delta\bar{y}$ and make the corresponding changes to β and $\bar{r}$. Then

$$(20)\qquad \Delta\beta=(X'BX+\bar{x}\bar{x}')^{-1}\bar{x}\Delta\bar{y},$$

$$(21)\qquad \Delta\bar{r}=(1-\bar{x}'(X'BX+\bar{x}\bar{x}')^{-1}\bar{x})\Delta\bar{y}.$$

The term in parentheses $(1-\bar{x}'(X'BX+\bar{x}\bar{x}')^{-1}\bar{x})$ is between 0 and 1 since it is a diagonal element of an idempotent matrix. Therefore if $\bar{r}>0$ make $\Delta\bar{y}$ negative and make it positive if $\bar{r}>0$, $\bar{y}$ can be changed until either of two things happens: (1) $\bar{r}$ reaches 0, in which case remove it, or (2) a data point reaches a corner. Then make the appropriate transition and continue.

Both the addition and deletion algorithms adapt straightforwardly to the situation where p data points are added or deleted at a time. The only facet to be resolved is how to calculate $(X'BX+\bar{X}'\bar{X})^{-1}$. There are two possible solutions.

1. The well-known formula for the inverse of a sum of matrices (Henderson and Searle [11]) gives

$$(22)\qquad \begin{aligned}&(X'BX+\bar{X}'\bar{X})^{-1}\\ &\quad =(X'BX)^{-1}-(X'BX)^{-1}\bar{X}'(I+\bar{X}(X'BX)^{-1}\bar{X}')^{-1}\bar{X}(X'BX)^{-1}.\end{aligned}$$

2. Since $(X'BX+\bar{X}'\bar{X})=(X'BX+\sum_{i=1}^{p}\bar{x}_i\bar{x}_i')$, Step 2b of the ABC algorithm can be iterated p times.

The first approach requires $O(pm^2+p^2m+p^3)$ operations while the second requires $O(pm^2)$ operations.

5. Constraints. In some applications, it is necessary to consider the following constrained regression problem

$$(23)\qquad X\beta=y+\varepsilon$$

subject to

$$(24)\qquad G\beta\geqq f,$$

$$(25)\qquad E\beta=h.$$

For instance, if the β_j have the interpretation of being multinomial probabilities, then the following constraints would be appropriate:

$$\sum_{j=1}^{m} \beta_j = 1, \tag{26}$$

$$\beta_j \geqq 0, \qquad j = 1, m. \tag{27}$$

Then adding Lagrange multipliers, λ for the inequality constraints and η for the equality ones, (6) becomes

$$0 = (X'BX)\beta - X'[k(c-a) + By] - G'\lambda - E'\eta, \tag{28}$$

with

$$\lambda'(f - G\beta) = 0, \tag{29}$$

$$E\beta = h, \tag{25}$$

$$\lambda \geqq 0. \tag{30}$$

Let g_j' represent the jth row of G. Let $\mathbf{I} = \{j: g_j'\beta - f_j > 0\}$, the *Inactive* inequality constraints, and $\mathbf{Q} = \{j: g_j'\beta - f_j = 0\}$, the *Active* inequality constraints. Reorder the constraints so that λ, G, and f can be partitioned as below:

$$\lambda = \begin{bmatrix} \lambda_I \\ \lambda_Q \end{bmatrix}, \quad G = \begin{bmatrix} G_I \\ G_Q \end{bmatrix}, \quad f = \begin{bmatrix} f_I \\ f_Q \end{bmatrix}.$$

Define $\tilde{E}$, $\tilde{h}$, and $\tilde{\lambda}$ as

$$\tilde{E} = \begin{bmatrix} G_Q \\ E \end{bmatrix}, \quad \tilde{h} = \begin{bmatrix} f_Q \\ h \end{bmatrix}, \quad \tilde{\lambda} = \begin{bmatrix} \lambda_Q \\ \eta \end{bmatrix}.$$

Then (28)–(30) become

$$0 = (X'BX)\beta - X'[k(c-a) + By] - \tilde{E}'\tilde{\lambda}. \tag{31}$$

$$\tilde{E}\beta = \tilde{h}, \tag{32}$$

$$\lambda_I = 0, \tag{33a}$$

$$\lambda_Q \geqq 0. \tag{33b}$$

Multiplying (31) by $\tilde{E}(X'BX)^{-1}$ and substituting (32) yields

$$0 = \tilde{h} - \tilde{E}(X'BX)^{-1}X'[k(c-a) + By] - \tilde{E}(X'BX)^{-1}\tilde{E}'\tilde{\lambda}. \tag{34}$$

Let $H = \tilde{E}(X'BX)^{-1}\tilde{E}'$. Therefore for H invertible

$$\tilde{\lambda} = H^{-1}\tilde{h} - H^{-1}\tilde{E}(X'BX)^{-1}X'[k(c-a) + By]. \tag{35}$$

Substituting (35) into (31) and rearranging yields

$$\beta = (X'BX)^{-1}(I - \tilde{E}'H^{-1}\tilde{E}(X'BX)^{-1})X'[k(c-a) + By] + (X'BX)^{-1}\tilde{E}'H^{-1}\tilde{h}. \tag{36}$$

The question remains, "How does one determine the initial partition?" Two approaches are available:

1. Any of several well-known algorithms to compute the constrained LS solution can be used in the first stage of the ABC algorithm. (See Lawson and Hanson [12, Chaps. 20–23] for a discussion of this problem.) Then an obvious adaptation of the ABC algorithm can be used to reduce k until the desired fraction of data points in $\mathbf{A}$

and **C** is achieved. However, the constraint partition must also be checked and changed appropriately as the constraints reach their "corners".

2. Below we describe an algorithm which can impose the constraints at any time. This algorithm can impose the constraints on the LS problem initially, but can also be used to study the effects of adding the constraints to an otherwise solved problem. For instance, one might be interested in how much the constrained β differs from the unconstrained one.

First, it is sufficient to consider only the imposition of inequality constraints. Replace each equality constraint, $e_i'\beta - h_i = 0$ by two inequality constraints, $e_i'\beta - h_i \geqq 0$ and $-e_i'\beta + h_i \geqq 0$. Clearly if both are satisfied, then so is the equality constraint. At the final step in the algorithm below all the inequality constraints will be satisfied (if possible). Then the two inequality constraints can be replaced by the one equality constraint.

Consider the imposition of one inequality constraint to an already solved problem. (The generalization to more than one constraint is straightforward and complicates only the notation.) Let the solution to the solved problem be denoted by $\tilde{\beta}$, and the Lagrange multipliers associated with the active constraints by $\tilde{\lambda}_Q$. Let the new constraint be denoted as $\bar{g}'\beta - \bar{f} \geqq 0$. The idea is to introduce a new variable q and a nonnegative weight w and change the objective function as follows:

$$\min \sum_{i=1}^{n} \rho(y_i - x_i'\beta) + (f - G\beta)'\lambda + (-\bar{g}'\beta + \bar{f} - q)\xi - q\zeta + wq,$$

where ζ and ξ are Lagrange multipliers. By the Karush–Kuhn–Tucker theorem (Pourciau [13]), the solution to this constrained minimization problem is characterized by

(37) $$0 = (X'BX)\beta - X'[k(c-a) + By] - G'\lambda - \xi\bar{g},$$

(38) $$0 = -\xi - \zeta + w,$$

(39) $$0 = (G\beta - f)'\lambda,$$

(40) $$0 = \xi(\bar{g}'\beta - \bar{f} + q),$$

(41) $$0 = \zeta q,$$

(42) $$\zeta \geqq 0,$$

(43) $$\xi \geqq 0,$$

(44) $$G\beta - f \geqq 0,$$

(45) $$\bar{g}'\beta - \bar{f} + q \geqq 0,$$

(46) $$q \geqq 0.$$

If $q = 0$, the problem is trivial. Else, let $\tilde{q} = \bar{f} - \bar{g}'\tilde{\beta}$. For $q > 0$, $(41) \Rightarrow \zeta = 0$, and in turn, with $(38) \Rightarrow \xi = w$, and further, $\Rightarrow q = -\bar{g}'\beta + \bar{f}$. One can assume without loss of generality that $X'BX = I$. Furthermore for $w = 0$, this problem reduces to the former. Make the same partition of the constraints into **Q** and **I**. Then (37), (39), and (40) become

$$\begin{bmatrix} I & -G_Q' & 0 \\ G_Q & 0 & 0 \\ \bar{g}' & 0 & 1 \end{bmatrix} \begin{bmatrix} \beta \\ \lambda_Q \\ q \end{bmatrix} = \begin{bmatrix} X'[k(c-a) + By] \\ f_Q \\ \bar{f} \end{bmatrix} + w \begin{bmatrix} \bar{g} \\ 0 \\ 0 \end{bmatrix}.$$

Assuming the matrix on the left has an inverse (in general, it will and, if not, it can be modified to have one by taking one or more of the constraints and moving them into **I**) and multiplying by it yields:

$$\begin{bmatrix}\beta\\ \lambda_Q\\ q\end{bmatrix}=\begin{bmatrix} I-G_Q'(G_QG_Q')^{-1}G_Q & G_Q'(G_QG_Q')^{-1} & 0\\ -(G_QG_Q')^{-1}G_Q & (G_QG_Q')^{-1} & 0\\ -\bar{g}'(I-G_Q'(G_QG_Q')^{-1}G_Q) & -\bar{g}'G_Q'(G_QG_Q')^{-1} & 1\end{bmatrix}$$

$$\times\left\{\begin{bmatrix}X'[k(c-a)+By]\\ f_Q\\ \bar{f}\end{bmatrix}-w\begin{bmatrix}\bar{g}\\ 0\\ 0\end{bmatrix}\right\}$$

$$=\begin{bmatrix}(I-G_Q'(G_QG_Q')^{-1}G_Q)X'[k(c-a)+By]+G_Q'(G_QG_Q')^{-1}f_Q\\ -(G_QG_Q')^{-1}G_QX'[k(c-a)+By]+(G_QG_Q')^{-1}f_Q\\ \bar{f}-\bar{g}'\tilde{\beta}\end{bmatrix}$$

$$+w\begin{bmatrix}(I-G_Q'(G_QG_Q')^{-1}G_Q)\bar{g}\\ -(G_QG_Q')^{-1}G_Q\\ -\bar{g}'(I-G_Q'(G_QG_Q')^{-1}G_Q)\bar{g}\end{bmatrix}$$

$$=\begin{bmatrix}\tilde{\beta}\\ \tilde{\lambda}_Q\\ \tilde{q}\end{bmatrix}+w\begin{bmatrix}\delta\beta_j/\delta w\\ \delta\lambda_{Qj}/\delta w\\ \delta q/\delta w\end{bmatrix}.$$

For $w=0$, these equations reduce to (31), (32), and (45). Now increase w. Any of five things can happen:

1. $q=0$. Stop. The problem is solved.
2. Some constraint in **I** reaches equality. Then move it into **Q** making the appropriate changes in the matrices and continue.
3. For some constraint in **Q**, $\lambda=0$. Move it into **I** and continue.
4. Some data point reaches a corner. Make the appropriate transition and continue.
5. β and q become independent of w and λ_Q is nondecreasing in w i.e. for all $j\in\mathbf{Q}$, $\delta\lambda_{Qj}/\delta w\geqq 0$. Then the constraints are infeasible.

This algorithm can also be used to determine if the constraints are feasible. Let $n=m$, $X'BX=I$, and $y=0$. Then one finds the feasible point of minimum square norm.

6. Weighting. There are at least three different types of weighting which can be applied. Let W be an $n\times n$ diagonal matrix with each $w_{ii}>0$.

1. "Inside" weighting: Replace $\rho(y_i-x_i'\beta)$ by $\rho(w_i(y_i-x_i'\beta))$ in the objective function. Equation (6) becomes

$$(X'W^2BX)\beta=X'W[k(c-a)+WBy].$$

This type of weighting could be used to "correct" for differences in scale in the error associated with each data point.

2. "Outside" weighting: Replace $\rho(\cdot)$ by $w_{ii}\rho(\cdot)$. Then (6) becomes

$$(X'WBX)\beta=X'W[k(c-a)+By].$$

This type of weighting could be used to more closely fit certain data points than others. Such a need might arise in time series analysis. Another reason might be to provide some robustness against outliers in X. (See Maronna et al. [14], for details.)

3. "Robust" weighting: Replace $\rho(\cdot)$ by $\rho_i(\cdot)$ where $\rho_i(\cdot)$ differs from $\rho(\cdot)$ in that k is replaced by $w_{ii}k$. Equation (6) becomes

$$(X'BX)\beta = X'[Wk(c-a)+By].$$

This type of weighting could reflect that idea that certain data points are less likely to be outliers than others. If some w_{ii} goes to $+\infty$, then that data point cannot be an outlier.

7. Conclusion. In this paper, we have presented a new variant of Huber's classic M-estimator. Perhaps the most important facet of our estimator is its intuitive appeal. Intuitively, least squares is good for small errors and least absolute residuals is good for large errors. Our estimator should perform acceptably well in either environment. Various tradeoffs between efficiency at the Gaussian and robustness can be made by the choice of α. We made the point that our estimator can be considered to be a generalization of the trimmed mean. Although it was not stressed in this paper, for asymmetric contamination our estimator should perform better than the trimmed mean.

We also presented a series of algorithms to compute our estimator. These algorithms all have a unifying structure:

We compute the solution to a problem that we know how to solve. Then we modify that problem continuously via a one-dimensional parameter. Within a restricted range, the solution varies linearly with the parameter. When the limit is reached, a partition is changed and the process repeats.

This type of algorithm allows for exact calculations as opposed to most iterative algorithms which terminate when the difference in the solution from one iteration to the next is sufficiently small. More work needs to be done on the time requirements of these algorithms, principally the ABC algorithm and the algorithm to include constraints on an otherwise unconstrained problem. It is possible that the recursive form of the ABC algorithm is faster than the nonrecursive form. Also, it is not known whether the best time to impose the constraints is at the beginning, sometime in the middle, or at the end. It is also not known how fast our algorithm for determining the feasibility of constraints is compared with our algorithms, e.g. Simplex.

REFERENCES

[1] P. J. HUBER, *Robust Statistics*, John Wiley, New York, 1981.

[2] I. BARRODALE AND F. D. K. ROBERTS, *Solution of an overdetermined system of equations in the l_1 norm*, Comm. ACM, 17 (1974), pp. 319–320.

[3] P. BLOOMFIELD AND W. STEIGER, *Least absolute deviations curve-fitting*, this Journal, 2 (1980), pp. 290–301.

[4] P. J. HUBER, *Robust regression: asymptotics, conjectures and Monte Carlo*, Ann. Statist., 1 (1973), pp. 799–821.

[5] ———, *Robust estimation of a location parameter*, Ann. Math. Statist., 35 (1964), pp. 73–101.

[6] R. DUTTER, *Algorithms for the Huber estimator in multiple regression*, Computing, 18 (1977), pp. 167–176.

[7] P. J. HUBER AND R. DUTTER, *Numerical solutions of robust regression problems*, in COMPSTAT 1974, Proc. in Computational Statistics, G. Bruckmann, ed., Physika Verlag, Vienna, 1974.

[8] P. E. GILL, G. H. GOLUB, W. MURRAY AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.

[9] L. G. KHACHIAN, *A polynomial time algorithm for linear programming*, Doklady Akad. Nauk USSR, 244 (1979), pp. 1093–1096; Soviet Math. Doklady, 20, pp. 191–194.

[10] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[11] H. V. Henderson and S. R. Searle, *On deriving the inverse of a sum of matrices*, SIAM Rev., 23 (1981), pp. 53–60.

[12] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[13] B. H. Pourciau, *Modern multiplier rules*, Amer. Math. Monthly (June–July, 1980), pp. 433–452.

[14] R. A. Maronna, O. Bustos and V. Yohai, *Bias and efficiency robustness of general M-estimates with random carriers*, in Proc. Heidelberg, 1979, M. Rosenblatt, ed., Springer-Verlag, New York, 1979.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 3, September 1984

THE COLLINEARITY PROBLEM IN LINEAR REGRESSION. THE PARTIAL LEAST SQUARES (PLS) APPROACH TO GENERALIZED INVERSES*

S. WOLD†, A. RUHE‡, H. WOLD§ AND W. J. DUNN III¶

Abstract. The use of partial least squares (PLS) for handling collinearities among the independent variables X in multiple regression is discussed. Consecutive estimates (rank $1, 2, \cdots$) are obtained using the residuals from previous rank as a new dependent variable y. The PLS method is equivalent to the conjugate gradient method used in Numerical Analysis for related problems.

To estimate the "optimal" rank, cross validation is used. Jackknife estimates of the standard errors are thereby obtained with no extra computation.

The PLS method is compared with ridge regression and principal components regression on a chemical example of modelling the relation between the measured biological activity and variables describing the chemical structure of a set of substituted phenethylamines.

Key words. collinearity, linear regression, conjugate gradients, principal components, cross validation, chemometrics

1. Introduction.

In multiple regression, linear or nonlinear, collinearities among the independent variables x_j sometimes cause severe problems. (For notation, see below equation (1)). The estimated coefficients $\hat{\beta}_j$, can be very unstable, and thereby far from their target values. In particular, this makes predictions by the regression model to be poor.

In many chemical applications of multiple regression, like the present example of relationships between chemical structure and biological activity, the predictive properties of the models are of prime importance and the regression estimates therefore often need to be stabilized. The present example can be seen as a special case of response surface modelling, an area where the collinearity problem has been recognized as a serious problem (Box and Draper (1971), Gorman and Toman (1966), Draper and Smith (1966)).

In applied work, the collinearity problem is often handled by selecting a subset of variables by a stepwise procedure. See Hocking (1976) for a review. We shall not consider this subset strategy here, but shall limit ourselves to the data analysis with all variables included in the model.

2. Existing methods.

Three principal ways are described in the statistical literature to accomplish a stabilization of the regression estimates $\hat{\beta}_j$ in a given regression model. Adopting the usual notation (see e.g. Draper and Smith (1981, p. 72)), the linear model is:

$$y = Xb + e. \tag{1}$$

Here X is a $n \times p$ matrix, containing the values of the p predictor variables at the n data points, $b = (\beta_1, \cdots, \beta_p)'$ is a p-dimensional column vector containing the regression coefficients, and $e = (\varepsilon_1, \cdots, \varepsilon_n)'$ is an n vector containing the errors which

* Received by the editors April 19, 1982, and in revised form April 28, 1983.

† Research Group for Chemometrics, Institute of Chemistry, Umeå University, S-901 87 Umeå, Sweden.

‡ Present address: Department of Computer Science, Chalmers University of Technology, S-41296 Göteborg, Sweden.

§ Department of Statistics, Uppsala University, S-751 21 Uppsala, Sweden.

¶ Department of Medicinal Chemistry, College of Pharmacy, University of Illinois at the Medical Center, Chicago, Illinois 60680.

are assumed to be uncorrelated, to be normal and to have the same variance. The prime $'$ denotes the transpose of a vector or matrix.

The first way, usually called *ridge regression* (Hoerl and Kennard (1970)) is based on adding a small number, k, to all elements in the diagonal of the moment matrix $X'X$. Thus, instead of inverting $X'X$, one inverts $(X'X+k\cdot I)$, which gives the regression estimates

$$b_{\text{ridge}}=(X'X+k\cdot I)^{-1}\cdot X'y. \tag{2}$$

The "ridge parameter," k, is usually chosen between 0 and 1. The specific value of k is often based on an inspection of a plot of the regression estimates against k (see Fig. 1 for an example). Golub, Heath and Wahba (1979) estimate k directly from the data using generalized cross validation. Ridge regression has recently been reviewed by Draper and Van Nostrand (1979) and by Hocking (1976) (see also Smith and Campbell (1980)). The solution of nonlinear regression using ridge estimates in the iterative updating is a well established numerical practice (see Marquardt (1970) where the correspondence between the two situations is discussed).

The second approach to the stabilization of the regression estimates is based on the contraction of $X'X$ to a matrix of smaller rank. This is, for instance, accomplished by expanding $X'X$ in terms of its eigenvectors (principal components) and then retaining only the first r of these p eigenvectors to represent $X'X$ $(r<p)$. This gives the solution the form of a generalized inverse (Marquardt, 1970). A more elegant and more numerically stable formulation of that approach is based on the singular value decomposition (SVD, see Golub and Kahan (1965)) of X itself. The r first singular vectors can then be regarded as new independent predictor variables, *principal components regression* (see Hocking (1976) for a review).

Marquardt (1970) and others (Hawkins (1975), Mayer and Willke (1973)), showed the close similarity (but nonequivalence) between ridge and generalized inverse estimates. A large literature on closely related problems in numerical analysis, so-called ill-posed problems, addresses the collinearity problem in linear models along essentially the same two lines (see Varah (1979), Björck and Elden (1979), Wahba (1977)).

A third, less often used approach, is the so called *James–Stein estimates.* These consist of the ordinary least squares estimates multiplied by a shrinking factor φ $(0<\varphi<1)$. The use of these shrunk estimates in multiple regression has recently been reviewed by Draper and van Nostrand (1979) and Hocking (1976).

3. The PLS method. In the present article we investigate the properties of so called PLS estimates (partial least squares) adapted to the multiple regression problem. PLS was recently developed for modelling information-scarce situations in Social Sciences (H. Wold, 1975, 1982). We show that the present PLS estimation is a variant of the conjugate gradient method developed in numerical analysis for the calculation of generalized inverses (Hestenes and Stiefel (1952), Golub and Kahan (1965), Paige and Saunders (1982)). Each step of the conjugate gradient algorithm gives the PLS estimate of the corresponding rank.

The predictive significance of each sequential estimate can be tested by cross validation with only a small amount of additional computation. One simultaneously obtains jackknife estimates of the regression coefficient standard errors. Thus an efficient stopping rule is obtained; the rank is used which gives the model the best predictive properties in the cross-validatory sense.

The method is particularly attractive because (1) only two vector-matrix multiplications are needed for each successive rank estimate; and (2) the calculations are performed on the raw data y and X and the moment matrix needs not be calculated. This makes the method suitable for large problems where the amount of computation with standard methods of matrix inversion or diagonalization becomes prohibitive. Examples of such large problems are found in protein X-ray crystallography (Konnert (1976)) and geodesy (Kolata (1978)). With the proliferation of microcomputers, fast and simple-to-program methods are of interest also for small and moderate problems.

4. Details of the estimation. The PLS algorithm, in its general, mode A formulation, deals with variables blocked in q blocks, and forms a sequence of rank one approximations to the combined data matrix. In this paper, we consider only the case with two blocks, one of them furthermore restricted to consisting of only one variable. Let the data matrices for the two blocks be X and y, and denote the combined matrix by

$$Z=[X|y].$$

We then successively form a sequence of residual matrices Z_s, using the following algorithm:

```
ALGORITHM PLS.
1. Start $Z_1=[X|y]$, $b_0=0$.
2. For $s=1,2,\cdots$, until $\|Z_s\|$ is small
    1. $u_s=X_sX_s'y_s/\|X_s'y_s\|$.
    2. $c_s=Z_s'u_s/u_s'u_s$, $c_s=(a_s', \rho_s)'$.
    3. $Z_{s+1}=Z_s-u_sc_s'$.
    4. Solve $A_s'b_s=r_s$, $r_s=(\rho_1,\cdots,\rho_s)'$ for $b_s$.
```

We first note that

$$U_r=[u_1,\cdots,u_r]$$

builds up an orthogonal basis of the range of X, since X_{s+1} is the projection of X orthogonal to U_s, (see step 2.3):

$$X_{s+1}=X_s-\frac{u_su_s'X_s}{u_s'u_s}=\left(I-\frac{u_su_s'}{u_s'u_s}\right)X_s=\left(I-\frac{u_su_s'}{u_s'u_s}\right)\cdots\left(I-\frac{u_1u_1'}{u_1'u_1}\right)X$$

making $u_{s+1}'U_s=0$. Noting that step 2.1 is a gradient step, we see then that the PLS algorithm is actually equivalent to a conjugate gradient algorithm applied to the normal equations of the system (1). The right vectors denoted by a_s are not orthogonal, so in order to solve for the regression coefficients, we have to update the solution of an underdetermined system in step (2.4).

The formulation in algorithm PLS is of interest since it shows the close connection to the principal components regression and total least squares (Golub, van Loan (1980)) algorithms, in that they all build up Z as a sum of rank one matrices. However for actual computation, especially for large problems where the updating step (2.3) becomes time-consuming, we suggest that a reliable implementation of conjugate gradients be used. The algorithm LSQR (Paige and Saunders (1982)) is the preferred choice.

```
ALGORITHM LSQR. (see Paige and Saunders (1982) for details).
1. Start $\theta_1v_1=X'y$, $\rho_1p_1=Xv_1$.
2. For $s=1,2,\cdots$
    1. $\theta_{s+1}v_{s+1}=X'p_s-\rho_sv_s$.
    2. $\rho_{s+1}p_{s+1}=Xv_{s+1}-\theta_{s+1}p_s$.
```

The positive numbers θ_s and ρ_s are normalization coefficients chosen to give the vectors v_s and p_s unit length. We have given the algorithm bidiag 2 of Paige and Saunders (1982), which is a simple transformation of the algorithm bidiag 1 that they have actually implemented.

We see that the vectors p_s of algorithm LSQR are proportional to the vectors u_s of algorithm PLS. In fact, the two algorithms are equivalent, and give the same sequence of solutions. Convergence occurs when θ_{s+1} or ρ_{s+1} become negligible, and since the vectors are orthogonal, this occurs for $s=r$, the rank of X, if not earlier. Often sufficiently good numerical results are obtained much earlier.

5. Data scaling. The PLS estimates depend on the scaling of the variables x_j as do ridge and principal components regression estimates. Thus, a variable x_j with large variance will get a larger weight v_j, and hence give a larger influence on the latent variable u than a variable x_j with a small variance.

When no prior information about the relative importance of the independent variables is available, centering the variables x_j to mean zero and scaling to unit length is probably the best alternative. As noted by Draper and van Nostrand (1979): "This at least forces everyone to do the same calculations in circumstances where compelling prior information is lacking."

6. Cross validation and jackknife. Cross validation (Stone (1974) and Geisser (1974)) is a technique which is very useful in estimating the optimal complexity of a model for a given data set. The data set is divided into a number of groups. The model, with a given complexity, is fitted to the data set reduced by one of the groups. Predictions are calculated by the fitted model for the deleted data and the sum of squares of predicted minus observed values for the deleted data is formed. Then, in a second round, the same procedure is repeated but with the second group held out. Then a third round is performed, etc., until each data point has been held out once and only once. The total sum of squares of predictions minus observations then contains one term from each point. This sum, abbreviated PRESS, is a measure of the predictive power of the model with the given complexity for the given data set.

Cross validation has attractive theoretical properties (Wahba (1977)). Golub, Heath and Wahba (1979) use it to estimate the optimal ridge factor in ridge regression. It has been used as a criterion to select variables in multiple regression (Allen, 1971), for selecting the smoothing factor in spline fitting (Wahba and Wold (1974), Craven and Wahba (1979)), to select the best number of components in principal components analysis (S. Wold, 1978), and for hypothesis testing in PLS modelling (H. Wold (1982)). In the PLS estimation discussed here, we wish to estimate the optimal rank of the estimate, i.e. when to stop the algorithm. We divide the cases into G groups. With one of these groups deleted one still gets estimates for v_{s+1} in step (2.1) of algorithm LSQR, which allows the "latent" variable Xv_{s+1} in step (2.2) to be estimated for all n data points, including those deleted. The residual p_{s+1} is then estimated based on the retained points, and prediction errors computed for those deleted. The PRESS is then calculated as the sum of squares of the predicted residuals for the deleted points. A second part of the data set is then held out, squared prediction errors are added to the PRESS and so on.

Note that in algorithm LSQR we have scaled p_s to unit length. When comparing prediction errors in different steps, we use the unscaled residuals

$$p_{s_{\mathrm{UNSC}}}=(\rho_1\rho_2\cdots\rho_s)/(\theta_1\cdots\theta_{s-1})p_s.$$

An alternative way of performing cross validation, which is to be used when the successive PRESS · es are to be compared with ridge-PRESS and ordinary least squares PRESS, is to make the total calculation up to the last significant PLS rank separately for each subgroup deletion.

In large problems, repeated calculations with deletion of one group after another may be too time-consuming. In such cases, however, there are usually so many data points that one with little loss can make a small part, say 5 or 10%, of the data a "test set." The estimation is then based on the remaining 90 or 95% of the data and the validation is made by letting the model predict the values in the test set. Thus the test set never enters the estimation; it is used only to estimate the "optimal rank" of the model.

The jackknife method (see Duncan (1978) and Miller (1974) for reviews) is closely related to cross validation in that it makes use of the data several times, each time with a subset of the data deleted from the calculation. The scope of the jackknife is to use the variation in the resulting parameter estimates $\hat{\beta}$ to calculate standard errors of these estimates. It is a "soft," data-oriented approach, in contrast to methods based on "hard" models such as maximum-likelihood methods (see H. Wold (1982) for a discussion). Denoting the value of $\hat{\beta}_j$ obtained when subgroup i $(i=1,2,\cdots,I)$ is deleted by $\hat{\beta}_{ji}$, one first forms the pseudo-values P_{ji} as:

$$P_{ji}=I\cdot\hat{\beta}_j-(I-1)\hat{\beta}_{ji}\qquad(i=1,2,\cdots,I).$$

These values have the averages

$$\bar{P}_j=\frac{1}{I}\sum_i P_{ji}$$

and the estimated standard errors:

$$s_j=\left[\frac{1}{I(I-1)}\sum(P_{ji}-\bar{P}_j)^2\right]^{1/2}.$$

7. A chemical example. Dunn, Wold and Martin (1978) tried to relate the biological activity, y (the stimulation of the β-receptor measured by Lefkowitz et al., 1976), of $n=16$ similar chemical compounds to a set of 8 variables x_j. These variables describe various morphological and physicochemical properties of these compounds, such as electronic and steric properties of the substituent in a certain position and the lipophilic character and receptor-binding strength of the whole molecule. The data are shown in Table 1.

In their analysis, Dunn, Wold and Martin (1978) use principal components regression, contracting the matrix X to its first two singular vectors.

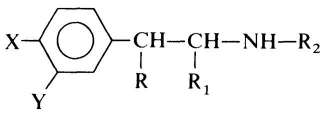
Here we reanalyze the same data using partial least squares, PLS. For comparison we have used ordinary least squares (OLS), principal component regression (PCR), James Stein shrunk estimate (JS), ridge regression (RR), and finally total least squares (TLS).

To evaluate the goodness of fit of the various models, we use the predictive sums of squares (PRESS) as described above in § 6. The data were divided into 3 groups (no. 1 = points 1, 4, 7, · · · , no. 2 = points 2, 5, 8, · · · , no. 3 = points 3, 6, 9, · · ·). Thus, the jackknife standard errors were also obtained for the PLS estimates.

Figure 1 shows the variation of the ridge estimates, PRESS and the residual sum of squares with the ridge parameter k. PRESS has a minimum around $k=2.0$, which is larger than the maximal recommended value of 1.0 (Hoerl and Kennard (1970)).

TABLE 1

Observed β-receptor agonist activity (y) for 15 compounds of the formulae



The structural descriptors x_1 to x_8 are, in order: $x_1 = \log$ equilibrium constant for the binding of the molecule to a receptor model, $x_2 = \log$ acidity constant, $x_3 =$ lipophilicity (tendency to prefer fatty tissues compared to aqueous phase, e.g. blood) of the phenyl part (X, Y—C_6H_4 in the formulae above) as measured on the model system octanol–water (see Hansch et al. (1973)), x_4 and x_5 are lipophilicities of groups R_1 and R_2, respectively, defined in the same way as x_3, x_6 and x_7 are Taft's σ^* and E_s of the group R_2 which measure the electronic and steric properties of the group (see Hansch et al. (1973)) and x_8 is an indicator variable equal to 1 when R=OH and 0 when R=H.

t	y	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	4.39	4.55	8.93	1.14	0.70	0.19	0.49	1.24	1
2	4.42	4.74	8.93	1.14	1.23	0.19	0.49	1.24	1
3	5.00	5.07	9.29	1.14	0.19	0.70	0.00	0.00	1
4	5.85	5.77	9.90	1.14	0.19	1.64	−0.10	−0.47	1
5	4.35	4.62	9.90	1.14	1.23	1.64	−0.10	−0.47	1
6	4.51	4.41	9.93	1.14	1.23	2.35	−0.20	−0.51	1
7	6.33	6.17	9.19	1.14	0.19	2.83	−0.13	−0.93	1
8	6.37	6.17	9.19	1.14	0.19	2.56	−0.13	−0.93	1
9	4.68	4.33	10.03	1.14	0.19	2.42	−0.08	−0.38	0
10	5.04	4.62	10.29	1.14	0.19	3.36	−0.13	−0.93	0
11	7.10	7.22	9.29	1.14	0.19	2.43	−0.30	−1.60	1
12	5.04	4.64	10.22	1.14	0.19	2.95	−0.08	−0.38	0
13	6.00	5.62	9.94	−0.07	0.19	1.64	−0.19	−0.47	1
14	5.48	6.19	9.77	−0.07	0.19	1.64	−0.19	−0.47	1
15	7.10	7.85	9.29	−0.07	0.19	3.80	−0.30	−1.60	1

The OLS PRESS is 3.16 which is considerably higher than the best ridge value of 2.11. The PLS PRESS is 2.06, very close to the ridge optimum. The PCR value of 2.56 (with two components) is somewhat larger. The James–Stein estimates give a minimal PRESS = 2.57 for the shrinking factor of 0.82. The PCR PRESS for three components equals 2.12. TLS (minimum norm) gave PRESS = 2.02 for rank = 3, and behaved very similarly to PCR for low ranks. When full rank is approached, it behaved differently. Note that in order to get a minimum norm TLS solution, the complete singular value decomposition is needed (see Golub and Van Loan (1980, formula (2.7)).

We also tested cross validation with 5 and 15 groups, i.e. only one point deleted. The results are not significantly different; in some cases we got a sharper minimum for PRESS with 5 groups. Generally, few groups are expected to give a conservative estimate of complexity, since the risk of overfitting is smaller.

Table 2 shows the estimated values of the regression coefficients $\hat{\beta}_j$ for the various methods. It is seen that the Ridge, PLS and PCR values are shrunk compared with the OLS values, the PLS and PCR values being the most shrunk. The ridge, PCR (2) and the PLS (2) agree within about 2 standard errors of the latter.

8. Discussion. The PLS method gives a solution to the multiple regression problem which is stabilized in comparison with the OLS solution and which has, at least in the examples investigated, a comparable prediction error to ridge regression. The very simple computations involved makes the method suitable for large problems and

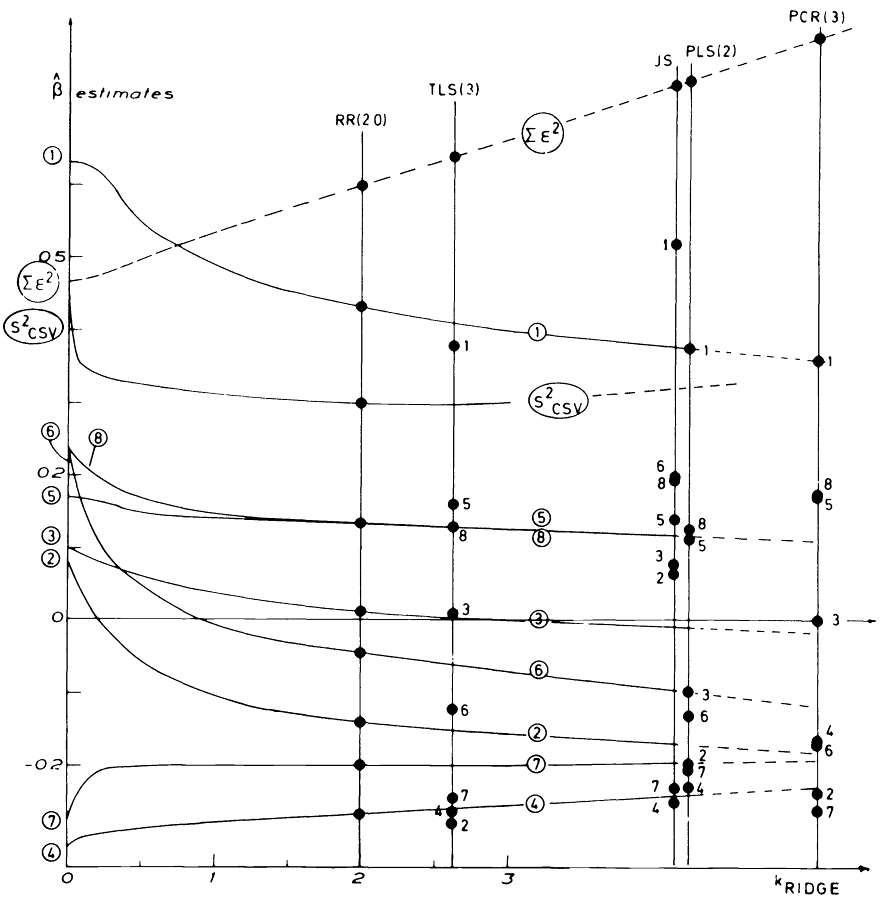


FIG. 1. *Estimated values for regression coefficients* 1 *through* 8 *as functions of the ridge parameter k. Also plotted are the estimates corresponding to the* TLS, JS, PLS *and* PCR *solutions* (*see Table* 2). *The sum of squared residuals and the cross validation variance are plotted in relative scales.*

TABLE 2

Resulting estimates of the regression coefficients β_j (*data in Table* 1*) *for the different methods of estimation.* SS *is the residual sum of squares obtained when the model is fitted to all data.* PRESS *is the corresponding predictive sum of squares obtained by cross validation with the data divided into three groups.* OLS *is ordinary least squares,* JS *is James–Stein estimate with shrinking factor* 0.82, RR *ridge regression with* $k = 2.0$, PCR *principal components regression with* 2 *and* 3 *components,* TLS *is total least squares and* PLS *is partial least squares rank* 1 *and* 2, *respectively. The* PLS(2) *standard errors* (s.err) *were calculated by the jackknife method.*

Var. no	1	2	3	4	5	6	7	8	SS	PRESS
OLS	.636	.080	.095	−.308	.169	.241	−.278	.238	0.61	3.156
JS (.82)	.522	.066	.078	−.252	.138	.197	−.228	.195	1.04	2.569
RR (2.0)	.432	−.143	.011	−.267	.133	−.049	−.200	.133	0.82	2.114
PCR (2)	.325	−.121	−.210	−.144	.111	−.184	−.210	.194	1.99	2.557
PCR (3)	.354	−.235	.018	−.162	.168	−.164	−.262	.172	1.15	2.117
TLS (3)	.379	−.274	.030	−.260	.161	−.125	−.244	.132	0.88	2.015
PLS (1)	.284	−.069	−.122	−.192	.161	−.184	−.223	.085	2.15	3.030
PLS (2)	.372	−.205	−.090	−.226	.119	−.128	−.211	.166	1.06	2.062
s.err	.058	.081	.103	.046	.009	.047	.016	.045		

* All variables including y scaled to mean zero and unit variance.

for the implementation on micro-computers and desk calculators. Cross validation provides a simple and straightforward stopping rule; it also makes it simple to compare the estimates of different methods and to calculate jackknife standard errors.

The situation treated in the present article is a simple special case of the general PLS modelling and estimation. The favorable predictive properties of the PLS estimates found in this case is encouraging for the utility of more extended PLS models (cf. H. Wold, 1982).

We have been using the principal components of X as a means of discriminating between different classes of "objects" (cases, compounds), each object being characterized by values of the variables x_j (Albano et al. (1978), Dunn et al. (1978), (1979), S. Wold et al. (1976), (1977), (1978)). The extra information provided by the variable y in the present case might make the corresponding vectors b_s superior to the ordinary principal components in the context of discriminant analysis and pattern recognition, a subject which we are currently investigating.

In a forthcoming report we will also investigate statistical and numerical aspects on the PLS method, and show how the 8 theorems given by Marquardt (1970) for ridge and principal component regression translate into this situation. We will also investigate computational aspects, especially for large problems. For large sparse problems, e.g., the geodesy problem, a reformulation as a multiple block PLS problem might prove advantageous.

Acknowledgments. The present project is supported by grants from the Swedish Natural Science Research Council (NFR), the Swedish Board of Research Councils (FRN) and the Stiftung Volkswagenwerk. Helpful discussions with Carl-Ivar Brändén and Åke Björck are greatly appreciated.

REFERENCES

[1] C. Albano, W. J. Dunn, U. Edlund, E. Johansson, B. Norden, M. Sjöström and S. Wold, (1978), *Four levels of pattern recognition*, Anal. Chim. Acta. Comput. Techn. Optim., 2, pp. 429–443.

[2] D. M. Allen, (1971), *Mean square error of prediction as criterion for selecting variables*, Technometrics, 13, pp. 469–475.

[3] Å. Björck and L. Elden, (1979), *Methods in numerical algebra for ill-posed problems*, Tech. Rep. MAT-R-33-1979, Linköping University, Sweden.

[4] G. E. P. Box and N. R. Draper, (1971), *Factorial designs, the $|X'X|$ criterion and some related matters*, Technometrics, 13, pp. 731–742.

[5] P. Craven and G. Wahba, (1979), *Smoothing noisy data with spline functions: estimating the correct degree of smoothing by the method of generalized cross validation*, Numer. Math., 31, pp. 377–403.

[6] N. R. Draper and H. Smith, (1981), *Applied Regression Analysis*, 2nd ed., John Wiley, New York.

[7] N. R. Draper and R. C. van Nostrand, (1979), *Ridge regression and James–Stein estimation. Review and comments.*, Technometrics, 21, pp. 451–466.

[8] G. T. Duncan, (1978), *An empirical study of jackknife constructed confidence regions in non-linear regression*, Technometrics, 20, pp. 123–129.

[9] W. J. Dunn III, S. Wold and Y. Martin, (1978), *Structure activity study of beta adrenergic agents with the* SIMCA *method*, J. Med. Chem., 21, pp. 922–930.

[10] W. J. Dunn III and S. Wold, (1980), *Relationships between chemical structure and biological activity modelled by* SIMCA *pattern recognition*, Bioorg. Chem., 9, pp. 505–523.

[11] S. Geisser, (1974), *A predictive approach to the random effect model*, Biometrica, 61, pp. 101–107.

[12] G. H. Golub and W. Kahan, (1965), *Calculating the singular values and pseudo-inverses of a matrix*, SIAM J. Numer. Anal., 2, pp. 205–224.

[13] G. H. Golub, M. Heath and H. Wahba, (1979), *Generalized cross validation as a method for choosing a good ridge parameter*, Technometrics, 21, pp. 215–224.

[14] G. H. Golub and C. F. Van Loan, (1980), *An analysis of the total least squares problem*, SIAM J. Numer. Anal., 17, pp. 883–893.

[15] J. W. GORMAN AND R. J. TOMAN, (1966), *Selection of variables for fitting equations to data*, Technometrics, 8, pp. 27–51.

[16] C. HANSCH, S. UNGER, K. H. KIM, D. NIKAITANI AND E. J. LIEN, (1973), *Aromatic substituent constants for structure-activity correlations*, J. Med. Chem., 16, pp. 120–126.

[17] D. M. HAWKINS, (1975), *Relations between ridge regression and eigenanalysis of the augmented correlation matrix*, Technometrics, 17, pp. 477–480.

[18] M. R. HESTENES AND E. STIEFEL, (1952), *Methods of conjugate gradients for solving linear systems*, J. Res. NBS, 49, pp. 409–436.

[19] R. R. HOCKING, (1976), *The analysis and selection of variables in linear regression*, Biometrics, 32, pp. 1–49.

[20] A. E. HOERL AND R. W. KENNARD, (1970), *Ridge regression: biased estimates for non-orthogonal problems*, Technometrics, 12, pp. 55–67.

[21] G. B. KOLATA, (1978), *Geodesy: Dealing with an enormous computer task*, Science 200, April 28, pp. 421–422.

[22] J. H. KONNERT, (1976), *A restrained parameter structure sector least squares procedure for large asymmetric units*, Acta Cryst., A32, pp. 614–617.

[23] D. W. MARQUARDT, (1970), *Generalized inverses, ridge regression, biased linear and non-linear estimation*, Technometrics, 12, pp. 591–612.

[24] L. S. MAYER AND T. A. WILLKE, (1973), *On biased estimation in linear models*, Technometrics, 15, pp. 497–508.

[25] R. G. MILLER, (1974), *The jackknife, a review*, Biometrika, 61, pp. 1–15.

[26] C. MUKHERJEE, M. C. CARON, D. MULLIKEN AND R. J. LEFKOWITZ, (1976), *Structure-activity relationships of adenyl cyclase coupled beta-adrenergic receptors: determination by direct binding studies*, Molecular Pharmacology, 12, pp. 16–31.

[27] C. C. PAIGE AND M. A. SAUNDERS, (1982), *A bidiagonalization algorithm for sparse linear equations and least squares problems*, ACM Trans. Math. Software, 8, pp. 43–71.

[28] G. SMITH AND F. CAMPBELL, (1980), *A critique of some ridge regression methods (with comments)*, J. Amer. Statist. Assoc., 75, pp. 74–103.

[29] M. STONE, (1974), *Cross-validitory choice and assessment of statistical predictions*, J. Roy. Statist. Soc., 36, pp. 111–133.

[30] J. M. VARAH, (1979), *A practical examination of some numerical methods for linear discrete ill-posed problems*, SIAM Rev., 21, pp. 100–111.

[31] G. WAHBA, (1977), *Practical approximate solution to linear operator equations when the data are noisy*, SIAM J. Numer. Anal., 14, pp. 651–657.

[32] G. WAHBA AND S. WOLD, (1975), *A completely automatic French curve: Fitting spline functions by cross-validation*, Comm. Statist., 4, pp. 1–17.

[33] H. WOLD, (1975), *Soft modelling by latent variables; the non-linear iterative partial least squares approach*, in Perspectives in Probability and Statistics, Papers in Honour of M. S. Bartlett, J. Gani, ed., Academic Press, London.

[34] ———, (1977), *On the transition from pattern recognition to model building*, in Mathematical Economics and Game Theory, Essays in Honour of Oscar Morgenstern, R. Hern and O. Moeschlin, eds., Springer-Verlag, Berlin.

[35] ———, (1982), *Soft modeling, The basic design and some extensions*, In Systems Under Indirect Observation, I-II, K. G. Jöreskog and H. Wold, eds., North-Holland, Amsterdam.

[36] S. WOLD, (1976), *Pattern recognition by means of disjoint principal components models*, Pattern Recogn., 8, pp. 127–139.

[37] ———, (1978), *Cross validatory estimation of the number of components in factor and principal components models*, Technometrics, 20, pp. 397–405.

[38] S. WOLD AND M. SJÖSTRÖM, (1977), SIMCA, *a method for analyzing chemical data in terms of similarity and analogy*, in Chemometrics, Theory and Practice, B. R. Kowalski, ed., ACS Publ. Ser. 52, American Chemical Society, pp. 243–282.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

AN EFFICIENT ALGORITHM FOR SOLVING GENERAL LINEAR TWO-POINT BVP*

R. M. M. MATTHEIJ† AND G. W. M. STAARINK‡

Abstract. A new method is described to compute the solutions of linear BVP in an efficient and stable way. The stability is achieved by decoupling the multiple shooting recursion; this means that the choice of output points can be made virtually without regard to restrictions. By fixing the number of integration steps per "shooting" interval and assembling as many of them as is needed to fit the user's requirements, high efficiency is gained. Apart from a mathematical description, we also give a stability analysis of the method. A large number of numerical examples confirm this analysis and illustrate the possibilities of the algorithm.

Key words. linear boundary value problem, multiple shooting, decoupling, adaptivity

AMS (MOS) classification. 65L10

1. Introduction.

There exists an ever growing literature on two-point-boundary-value problems that has produced a large number of methods for solving them, see e.g. [1], [2], [4], [5], [6], [8], [9], [17], [18]. Still, even for linear problems there is room for improvements. These are related to questions regarding the efficiency of the method, in particular, for more general boundary conditions (BC), the flexibility with respect to the choice of output points (whether user requested or code determined) and last but not least the problem of how to control the stability. In this paper we shall describe a multiple shooting algorithm which grew out of a number of ideas, developed in part in previous work cf. [11], [12], [13], [15] and matured while attempting to write a general purpose FORTRAN code MUTS. We consider the linear ODE

$$\frac{dx}{dt}=L(t)x+f(t), \qquad \alpha \leqq x \leqq \beta, \tag{1.1}$$

where L is an $n\times n$ matrix function and f an n vector function, and assume that the solution x satisfies the BC

$$M_\alpha x(\alpha)+M_\beta x(\beta)=b, \tag{1.2}$$

where M_α and M_β are $n\times n$ matrices and b is an n vector. We consider mildly stiff ODE only (i.e. $|L|$ has no very large eigenvalues). Since our method employs orthogonalization at the shooting points it is a remote cousin of the Godunov–Conte algorithm [3] (see also the implementation in [18]). We like to emphasize, however, that our motivation is different. While orthogonalization in [3], [18] is used to maintain (or rather "restore") independence of a number of basic solutions, we feel that decoupling of solution spaces into increasing and decreasing modes is the key to get around the inherent (initial value) instability. Therefore, in our opinion, it is the *triangularization* (at least block triangularization) of the incremental matrices, that is crucial for this technique and we shall give some remarkable examples that underline this. Triangularization of the multiple shooting recursion makes the use of (special) sparse matrix solvers, like in [2], or (perhaps not always stable) initial value recursion

* Received by the editors June 15, 1982, and in revised form May 20, 1983.

† Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York 12181. On leave from Mathematisch Instituut, Katholieke Universiteit, Toernooiveld, Nijmegen, the Netherlands.

‡ Mathematisch Instituut, Katholieke Universiteit, 6525 ED Toernooiveld, Nijmegen, the Netherlands.

techniques, like in [5], [20], superfluous. Like most codes we employ an adaptive integrator. Another novelty is that we use efficiency arguments to select the so-called minor shooting intervals (cf. [15]). The so-called major shooting points (where the solution will be given as output) are a subset of the previous set and can be specified by the user, either directly or indirectly. Using the decoupling it can be shown that assembly of minor into major intervals does not affect the global errors, thus insuring stability and reasonable flexibility in the choice of output points.

In § 2 we first give a description of the triangularization technique. The actual algorithm is given in § 3. The stability of the method is considered in § 4 and we conclude with a number of numerical examples in § 5.

2. Triangularization of multiple shooting recursions. In a multiple shooting algorithm, the interval $[\alpha, \beta]$ is divided into a number, say N, of subintervals, $[t_i, t_{i+1}]$, $i=0,\cdots,N-1$. On each interval $[t_i, t_{i+1}]$ a fundamental solution, say F_i, and a particular solution, say w_i, is computed. Hence for each i there exists a vector y_i such that

$$x(t)=F_i(t)y_i+w_i(t), \qquad i=0,\cdots,N. \tag{2.1}$$

(N.B. the relation for $i=N$ is added to make the formulae later on look nicer.) By matching the relations (2.1) at t_{i+1} for $i=0,\cdots,N-1$ we obtain the recurrence relation

$$F_{i+1}(t_{i+1})y_{i+1}=F_i(t_{i+1})y_i+w_i(t_{i+1})-w_{i+1}(t_{i+1}). \tag{2.2}$$

Suppose we choose the fundamental solutions F_i such that $\forall_i\ F_i(t_i)=I$, then

$$A_i:=F_i(t_{i+1}) \tag{2.3}$$

is the incremental matrix (Wronskian) on $[t_i, t_{i+1}]$. If we let

$$g_i:=w_i(t_{i+1})-w_{i+1}(t_{i+1}) \tag{2.4}$$

then (2.2) reads

$$y_{i+1}=A_iy_i+g_i, \qquad i=0,\cdots,N-1. \tag{2.5}$$

From (1.2) and (2.1) we therefore see that the sequence $\{y_i\}_{i=0}^N$ must satisfy the BC

$$M_\alpha y_0+M_\beta y_N=c:=b-M_\alpha w_0(t_0)-M_\beta w_N(t_N). \tag{2.6}$$

The relations (2.5) and (2.6) together constitute the multiple shooting equations, cf. [5], [8], [12], [16], [17], [20]. Rather than solving them by some linear system solver, cf. [2], [8], [21], we use the recursion (2.5), in a way different from the approaches in [5], [20].

In order to understand the basic idea of our algorithm it is useful to assume that the homogeneous solution space of (1.1) is *dichotomic*, i.e. such that for some k there exists a k dimensional subspace of increasing solutions and an $(n-k)$ dimensional subspace of nonincreasing solutions. This implies that forward recursion of (2.5) is unstable (recall that A_i was the incremental matrix). We therefore apply the decoupling method of [10], [11] in order to compute growing and nongrowing components separately. This goes as follows: let Q_0 be a given orthogonal matrix (see § 3). Then compute recursively a sequence of orthogonal matrices $\{Q_i\}_{i=1}^N$ and upper triangular matrices $\{U_i\}_{i=0}^{N-1}$ such that

$$A_iQ_i=Q_{i+1}U_i, \qquad i=0,\cdots,N-1. \tag{2.7}$$

Writing

$$\tilde{y}_i := Q_i^{-1} y_i, \qquad \tilde{g}_i := Q_{i+1}^{-1} g_i, \tag{2.8}$$

we obtain the triangular recursion

$$\tilde{y}_{i+1} = U_i \tilde{y}_i + \tilde{g}_i, \qquad i = 0, \cdots, N-1. \tag{2.9}$$

It has been shown in [10], [11] that this decoupling gives U_i of which the $k \times k$ left upper block represents the increments of the (transformed) increasing solutions and the $(n-k) \times (n-k)$ right lower block the increments of the nonincreasing solutions, if Q_0 is chosen appropriately. (Observe that the rounding error in U_i is $O(\|A_i\| \varepsilon_M)$, where ε_M denotes the *machine epsilon.*) We now partition matrices and vectors as

$$U_i = \begin{pmatrix} B_i & C_i \\ \varnothing & E_i \end{pmatrix}, \qquad \tilde{y}_i = \begin{pmatrix} \tilde{y}_i^1 \\ \tilde{y}_i^2 \end{pmatrix}, \tag{2.10}$$

(where B_i is a $k \times k$ matrix and y_i^1 is a k vector) and employ the following decoupled recursions

$$\begin{aligned} &\text{(a)} \quad \tilde{y}_{i+1}^2 = E_i \tilde{y}_i^2 + \tilde{g}_i^2, \qquad i = 0, \cdots, N-1, \\ &\text{(b)} \quad B_i \tilde{y}_i^1 = \tilde{y}_{i+1}^1 - C_i \tilde{y}_i^2 - \tilde{g}_i^1, \qquad i = N-1, \cdots, 0. \end{aligned} \tag{2.11}$$

On account of the growth behaviour of the E_i and B_i, we expect $\|\prod_0^{i-1} E_j\|$, $\|[\prod_i^{N-1} B_j]^{-1}\| = O(1)$,[1] i.e. the recursions (2.11) are expected to be stable. This decoupled form is now employed to compute some particular solution of (2.9) and also a fundamental solution of the homogeneous part of (2.9). The desired particular solution $\{\tilde{y}_i\}$ then follows by superposition using the BC. The computation of these solutions goes as follows: Let the particular solution $\{\tilde{z}_i\}_{i=0}^N$ satisfy

$$\tilde{z}_0^2 = 0, \qquad \tilde{z}_N^1 = 0. \tag{2.12}$$

Then $\{\tilde{z}_i^2\}_{i=0}^N$ can be found in a stable way using (2.11)(a), and $\{\tilde{z}_i^1\}_{i=N}^0$ using (2.11)(b) (note that the $C_i \tilde{z}_i^2$ terms are known now). Let the fundamental solution $\{\tilde{\Phi}_i\}_{i=0}^N$ satisfy

$$\tilde{\Phi}_0^2 = [\varnothing \quad I_{n-k}], \qquad \tilde{\Phi}_N^1 = [I_k \quad \varnothing]. \tag{2.13}$$

Then $\{\tilde{\Phi}_i^2\}_{i=0}^N$ can be computed from the homogeneous part of (2.11)(a) (i.e. with $\forall_i \tilde{g}_i^2 = 0$) and $\{\tilde{\Phi}_i^1\}_{i=N}^0$ from the homogeneous part of 2.11(b) (i.e. with $\forall_i \tilde{g}_i^1 = 0$).

Clearly, for some fixed vector a we must have

$$\tilde{y}_i = \tilde{z}_i + \tilde{\Phi}_i a, \qquad i = 0, \cdots, N. \tag{2.14}$$

If we substitute this for $i = 0$, N in the BC, we obtain a simple equation for a, viz

$$Ra = c - M_\alpha Q_0 \tilde{z}_0 - M_\beta Q_N \tilde{z}_N, \tag{2.15}$$

where

$$R := M_\alpha Q_0 \tilde{\Phi}_0 + M_\beta Q_N \tilde{\Phi}_N. \tag{2.16}$$

Hence the solution to the recursion (2.5), satisfying (2.6) is given by

$$y_i = Q_i [\tilde{z}_i + \tilde{\Phi}_i a], \qquad i = 0, \cdots, N. \tag{2.17}$$

The stability of this method will be discussed in § 4.

[1] $\prod_{j=p}^q E_j$ is defined as $E_q E_{q-1} \cdots E_p$ if $q \geqq p$ and as I otherwise.

3. The multiple shooting algorithm. In this section we describe a multiple shooting algorithm that is designed to obtain flexibility with respect to output points and reliability with respect to the stability of all computations involved. It employs the ideas described in [15] and § 2. In § 3.1 we show how the orthogonalization and triangularization is actually implemented. In § 3.2 we indicate how the integration of the ODEs is performed and how this is related to selecting the shooting points. In § 3.3 we consider the question how the output points may be chosen. In § 3.4 we show how an appropriate matrix Q_0 and the proper partitioning of the matrices U_i (which was an important prerequisite for the stability of the recursions (2.11)) can be found. Finally, in § 3.5 we give a special strategy for choosing the particular solutions w_i in order to obtain higher efficiency, in cases where the desired solution is very smooth.

3.1. Computation of the upper triangular recursion. In an actual implementation, the matrices A_i (cf. (2.3), (2.5)) do not appear explicitly as we directly use the orthogonal matrices to define initial values of suitable fundamental solutions. This goes as follows: On $[t_0, t_1]$ we compute a fundamental solution $\hat{F}_0$ say with

$$\hat{F}_0(t_0) := Q_0. \tag{3.1}$$

At t_1, decompose $\hat{F}_0(t_1)$ as

$$\hat{F}_0(t_1) = Q_1 U_0, \quad Q_1 \text{ orthogonal}, \quad U_0 \text{ upper triangular}. \tag{3.2}$$

On $[t_1, t_2]$ we proceed with the fundamental solution $\hat{F}_1$, satisfying

$$\hat{F}_1(t_1) := Q_1. \tag{3.3}$$

In general, we compute on $[t_i, t_{i+1}]$ the fundamental solution $\hat{F}_i$ with

$$\hat{F}_i(t_i) := Q_i, \tag{3.4}$$

and decompose

$$\hat{F}_i(t_{i+1}) = Q_{i+1} U_i. \tag{3.5}$$

Obviously the $\hat{F}_i$ are nothing but the transformed F_i of § 2, i.e. there holds

$$\hat{F}_i(t) = Q_i F_i(t). \tag{3.6}$$

As for the particular solution, we choose

$$w_i(t_i) := 0, \qquad i = 0, \cdots, N. \tag{3.7}$$

This means that

$$\tilde{y}_i = Q_i^{-1} x(t_i). \tag{3.8}$$

The transformed matching recursion then reads

$$\tilde{y}_{i+1} = U_i \tilde{y}_i + Q_{i+1}^{-1} w_i(t_{i+1}). \tag{3.9}$$

3.2. Adaptive integration and optimal complexity. A basic part of the algorithm is the integration of the particular and the fundamental solutions by some adaptive method. At present there are no codes available that are specially designed for integration both of increasing solutions and of nonincreasing solutions. Nevertheless, if the problem is not too stiff (in forward and backward direction) most currently available methods perform quite well. On account of its simplicity a fourth-fifth-order Runge–Kutta–Fehlberg method [7] is used in MUTS. A more detailed discussion of the use of this method for a nearly optimal shooting strategy can be found in [15]; we use the results obtained there. First, the adaptivity feature is only employed to compute the particular solutions w_i on some grid on which the *number of points* is fixed (say 5). Then the fundamental solution is integrated by the fourth order method on the same grid. The matrix at the last grid point is decomposed into an orthogonal and an upper triangular matrix as described in § 2.

In order to understand why this strategy makes sense, we first remark that each solution (homogeneous or inhomogeneous) is likely to contain a multiple of the most dominant mode. Since this mode essentially dictates the steplength, there is no need to use the adaptivity feature for more than one solution. Second, the QU-decomposition is relatively cheap compared to an integration step (about a factor 1/5) and therefore the complexity is mainly governed by the costs of integration. On account of the presence of increasing modes, larger intervals tend to make the integration less efficient; small intervals can make the overhead due to initialization etc. too high. Therefore a small (fixed) number of integration steps per shooting interval is preferred. This has two additional advantages. In the first place the shooting points are equidistributed (regarding the growth of the dominant mode) and in the second place we can choose our output points from a fairly dense grid. We remark that usually this strategy leads to a larger number of points than one may be interested in. In the next subsection we return to the latter question.

3.3. Assembly of minor shooting intervals. The strategy of § 3.2 not only gives us more points than desired, usually, but also more than desirable from a storage point of view. Indeed, in principle we have to store the matrices Q_i, U_i and the vectors $\tilde{g}_i$ at each shooting point. We propose two criteria for picking a subset suitable as output points. The first one is to choose points that correspond to an interval on which the most dominant mode does not grow more than a preset value; this results in a global equidistribution of these (major) shooting points. The second criterion is to take just a prescribed number of points. Both criteria can be used while marching from t_0 to t_N by checking either the increments or the interval length. In composing such a *major shooting interval* we compute an updated incremental matrix and an inhomogeneous term at each step. Hence the number of incremental matrices to be stored is equal to the (smaller) number of major shooting intervals.

Suppose we want the incremental growth to be bounded by M. This assembly of (*minor*) shooting intervals then goes as follows. Let t_{i_j} be the initial point of a major shooting interval. Define

$$(3.10)\qquad W_0 := U_{i_j}, \qquad \tilde{G}_j := \tilde{g}_j.$$

If $\|W_0\| \geqq M$, then $t_{i_{j+1}} := t_{i_j+1}$, i.e. the minor interval is a major interval. Suppose this is not the case. Then for $s = 1, 2, \cdots$ we compute

$$(3.11)\qquad W_s := U_{i_j+s}W_{s-1}, \qquad \tilde{G}_s := U_{i_j+s}\tilde{G}_{s-1} + \tilde{g}_{i_j+s},$$

until $\|W_s\| \geqq M$. As a major interval we take (t_{i_j}, t_{i_j+s+1}) and define

$$(3.12)\qquad V_j := W_s, \qquad \tilde{H}_j := \tilde{G}_s \quad (\text{also if } s = 0).$$

The global (major) recursion for the sequence $\{\tilde{y}_{i_j}\}$, then reads

$$(3.13)\qquad \tilde{y}_{i_{j+1}} = V_j\tilde{y}_{i_j} + \tilde{H}_j.$$

At this point one might be suspicious whether we destroyed the advantages of our algorithm, as the updating in (3.11) is nothing but forward recursion! However, there is not a real threat for two reasons. First, we may choose M such that $M\varepsilon_M$ (ε_M

the machine accuracy) is still smaller than the required tolerance (cf. [15]). Second, since the recursions are decoupled the error propagation is very special and we show in § 4.3 that no significant error build-up, due to this assembly, is felt in the final solution approximant, however large M may be chosen.

3.4. Choosing an appropriate Q_0 and the proper partitioning. As we remarked in § 2 we need to choose Q_0 properly in order to make sure that the induced upper triangular matrices U_i have the desired ordering, i.e. such that the left upper blocks represent the incremental values of the increasing modes. Before we show how this is achieved in our algorithm it is useful to investigate how different choices of the initial value matrix Q_0 induce different sequences of orthogonal and upper triangular matrices and how these are related. So let $\hat{Q}_0$ be another initial value matrix and $\{\hat{Q}_i\}_{i=0}^{N}$ and $\{\hat{U}_i\}_{i=0}^{N-1}$ be the induced orthogonal and upper triangular matrices (cf. (2.7)), i.e.

$$A_i\hat{Q}_i = \hat{Q}_{i+1}\hat{U}_i. \tag{3.14}$$

Write (cf. (2.8))

$$\hat{y}_i := \hat{Q}_i^{-1}y_i, \qquad \hat{g}_i := \hat{Q}_{i+1}^{-1}g_i. \tag{3.15}$$

Then $\{\hat{y}_i\}$ satisfies

$$\hat{y}_{i+1} = \hat{U}_i\hat{y}_i + \hat{g}_i. \tag{3.16}$$

The recursions (2.9) and (3.16) may be called *equivalent* since we can define an equivalence relation by introducing

$$S_i := Q_i^{-1}\hat{Q}_i. \tag{3.17}$$

It is easy to see that the following equalities hold

$$\begin{aligned} &\text{(a)}\quad U_iS_i = S_{i+1}\hat{U}_i,\\ &\text{(b)}\quad \hat{g}_i = S_{i+1}^{-1}\tilde{g}_i,\\ &\text{(c)}\quad \hat{y}_i = S_i^{-1}\tilde{y}_i. \end{aligned} \tag{3.18}$$

In our algorithm, we like Q_0 to be chosen such that the B_i reflect the growth of the increasing solutions. This may be measured by inspecting the diagonal elements (= eigenvalues); in particular if the (major) shooting interval is not too small, this seems a suitable strategy. Assuming there is a global dichotomy (i.e. on the entire interval $[t_0, t_N]$) it suffices to make sure that the first assembled matrix V_0 (see § 3.3) has a properly ordered diagonal. This is done as follows. We start at $t = t_0$ with $Q_0 = I$. Due to the tendency of the triangularization to give ordered diagonals in the upper triangular matrices (cf. power method arguments, see also [10], [11]), this will quite often be a satisfactory choice to achieve our goal. Anyway, at t_1 we check whether diag (U_0) is ordered. If this is not the case, the columns of U_0 are reordered according to the absolute magnitude of the diagonal elements. This can be described by a permutation matrix $P_0^1(1)$ say (the one between parentheses indicates that the checking has been performed at $t = t_1$ and the superscript one that it is the first permutation trial). The permuted matrix is again decomposed into an orthogonal and an upper triangular matrix, say

$$U_0(0)P_0^1(1) = P_1^1(1)U_0^1(1), \tag{3.19}$$

where $U_0(0) := U_0$.

If the matrix $U_0^1(1)$ is not yet ordered, this procedure has to be repeated. One should realize, however, that disordering is caused by the fact that the subspace spanned by the first k column vectors of Q_0 makes an extremely small angle with an initial value of a nonincreasing mode (cf. [11, § 5.1]); therefore in the (*quite unlikely*!) case of a disordered U_0 we expect such a permutation process to give a desired result after a few steps only, in which we have performed, for $j = 1, \cdots, p$ say,

$$U_0^{j-1}(1)P_0^j(1) = P_1^j(1)U_0^j(1). \tag{3.20}$$

Writing

$$S_0(1) := P_0^1(1) \cdots P_0^p(1);\ S_1(1) := P_1^p(1) \cdots P_1^1(1);\ U_0(1) := U_0^p(1), \tag{3.21}$$

we thus have

$$U_0(0)S_0(1) = S_1(1)U_0(1). \tag{3.22}$$

If we indicate the original sequences $\{Q_i\}$ and $\{U_i\}$ by $\{Q_i(0)\}$ and $\{U_i(0)\}$, then using $Q_0(1) := S_0(1)$ as an initial matrix induces sequences of orthogonal matrices $\{Q_i(1)\}$ and upper triangular matrices $\{U_i(1)\}$. Obviously we thereby obtain an equivalent upper triangular recursion of which the transformation matrices are related by (cf. (3.17))

$$S_i(1) := [Q_i(0)]^{-1}Q_i(1). \tag{3.23}$$

(Note that $Q_0(0) = I$.) Since we are building a major shooting step (cf. § 3.3), we next check whether

$$W_1(1) := U_1(1)U_0(1) \tag{3.24}$$

is ordered. If this is not the case we permute columns of $W_1(1)$ and decompose, as we did before with $U_0(1)$,

$$W_1(1)S_0(2) = S_2(2)W_1(2) \tag{3.25}$$

where $S_2(2)$ is orthogonal and $W_1(2)$ is upper triangular. Now $S_0(2)$ induces another equivalent recursion which would follow from the transformations $\{Q_i(2)\}$ with $Q_0(2) := S_0(1)S_0(2)$ and for which

$$S_i(2) := [Q_i(1)]^{-1}Q_i(2). \tag{3.26}$$

At this step the inhomogeneous term is updated like

$$\tilde{G}_1(2) := [S_2(2)]^{-1}\{U_1(1)\tilde{G}_0(1) + \tilde{g}_1(1)\}, \tag{3.27}$$

where $\tilde{g}_j(1) := Q_j^{-1}(1)\tilde{g}_j$ (cf. also (3.18)(b)). Typically at this point we have to store the most recently found initial matrix $Q_0(2)$, and also $Q_2(2)$, $W_1(2)$ and $\tilde{G}_1(2)$. These four arrays are overwritten at each consecutive step by the new initial transformation matrix $Q_0(\cdot)$ (only if reordering of diagonal elements is needed), the most recent orthogonal factorization matrix $Q_{i+1}(\cdot)$, the most recent assembled upper triangular matrix $W_i(\cdot)$ and the most recent assembled forcing $\tilde{G}_i(\cdot)$ respectively. This updating is continued until the endpoint of the first major shooting interval is reached. At subsequent points no reordering is performed; if the solution space is dichotomic this seems quite reasonable. However, if there is no global dichotomy the ordering found at the beginning may not be the appropriate one globally. Therefore, the algorithm finally checks the product of the diagonals of the upper triangular matrices. If this turns out not to be ordered, a new round of permutations is performed giving new transformation matrices at the major shooting points and new upper triangular matrices describing increments between them (carried out as described in (2.7)).

Two remarks must be made. First, due to the tendency of larger increments to be in the left upper block if not by rounding errors then at least by (usually larger) discretization errors, there is a "natural" self-restoring effect (though not always appropriate, see Example 5.2). Second, if the problem is not dichotomic, it cannot be expected to be a very well conditioned one, as follows from [14, Thm. 3.17]; in other words, it is the problem, rather than the algorithm that is to be blamed for possible instabilities! In § 4.1 we indicate how we may monitor this instability. After the global ordering has been found to be satisfactory, a value of k, the dimension of the dominant subspace, is determined from inspection of the diagonal elements.

3.5. Special choice of the w_i, when the solution is very smooth. One of the main problems in multiple shooting is that the efficiency of the integration is almost always dictated by the most rapidly increasing modes. This is unsatisfactory if the solution x is very smooth (and most multiple shooting codes suffer from this problem). In looking for ways to make our code as efficient as possible, we have experimented with a version that chooses particular solutions w_i in which the dominant mode component is much less influential. The idea is conceptually very simple and is another evidence of the usefulness of decoupling the recursion while marching from α to β. First we define some $\bar{w}_i(t_i)$, equal to $w_{i-1}(t_i)$ except for components that may belong to the dominant modes, by projection of $w_{i-1}(t_i)$ on the dominant solution space. Specifically,

$$\bar{w}_i(t_i) := w_{i-1}(t_i) - Q_i\begin{pmatrix}[Q_i^{-1}w_{i-1}(t_i)]^1 \\ \varnothing\end{pmatrix} = Q_i\begin{pmatrix}\varnothing \\ [Q_i^{-1}w_{i-1}(t_i)]^2\end{pmatrix}. \tag{3.28}$$

Because of the smoothness we now expect

$$\tilde{y}_i \approx \tilde{y}_{i-1}, \tag{3.29}$$

so

$$\tilde{y}_i^1 \approx (I - B_{i-1})^{-1}(C_{i-1}\tilde{y}_i^2 + \tilde{g}_{i-1}^2). \tag{3.30}$$

If we are optimistic enough, we may hope that $\tilde{y}_i \approx \tilde{w}_{i-1}(t_i)$, apart from dominated modes. Therefore we propose to use as initial value of w_i at t_i:

$$w_i(t_i) := Q_i\begin{pmatrix}(I - B_{i-1})^{-1}(C_{i-1}[\tilde{w}_{i-1}(t_i)]^2 + \tilde{g}_{i-1}^1) \\ [\tilde{w}_{i-1}(t_i)]^2\end{pmatrix}. \tag{3.31}$$

For a numerical justification, see Example 5.6.

4. The stability of the algorithm. In this section we consider the numerical stability of the different parts of the algorithm. Although the solution of the linear system (2.15) describes the final stage of the method, we analyze this first in § 4.1 which deals with the important notion of well conditioning. Then in § 4.2 we investigate the stability of the recursions (2.11)(a) and (b). Finally in § 4.3 we analyze the effect of assembling the recursion as was described in § 3.3.

4.1. Well conditioning and stability. As was shown in [12], [13] a useful quantity for studying the inherent stability of a BVP with respect to the BC is given by the condition number

$$\mathcal{CN} := \max_t \|F(t)[M_\alpha F(\alpha) + M_\beta F(\beta)]^{-1}\|, \tag{4.1}$$

where F is any fundamental solution. In particular, if we neglect discretization and rounding errors, we have for the fundamental solution Φ of the recursion (2.5), where

$\forall_j \Phi_j := Q_j \tilde{\Phi}_j$:

$$\max_j \|\Phi_j R^{-1}\| \leqq \mathscr{C}\mathscr{N}. \tag{4.2}$$

As was shown in [13], the condition number enables us to give fairly straightforward estimates for the global error if the solution space is dichotomic. We can also prove that this condition number is a good measure for the stability of the equation (2.15), see the following theorem.

THEOREM 4.3. *Let* $\|\cdot\| = \|\cdot\|_2$. *Then* $\|R^{-1}\| \leqq 2\mathscr{C}\mathscr{N}$.

Proof. Denote $\kappa := \max \|\Phi_j R^{-1}\|$. Then it follows that $\|\tilde{\Phi}_0^2 R^{-1}\| \leqq \|\tilde{\Phi}_0 R^{-1}\| = \|\Phi_0 R^{-1}\| \leqq \kappa$, and similarly $\|\tilde{\Phi}_N^1 R^{-1}\| \leqq \kappa$. Hence

$$\|R^{-1}\| = \left\| \begin{pmatrix} \tilde{\Phi}_N^1 \\ \tilde{\Phi}_0^2 \end{pmatrix}^{-1} \begin{pmatrix} \tilde{\Phi}_N^1 \\ \tilde{\Phi}_0^2 \end{pmatrix} R^{-1} \right\| \leqq \|\tilde{\Phi}_N^1 R^{-1}\| + \|\tilde{\Phi}_0^2 R^{-1}\| \leqq 2\kappa,$$

i.e.

$$\|R^{-1}\| \leqq 2\mathscr{C}\mathscr{N}. \qquad \square$$

If we can compute $\tilde{\Phi}$ and $\tilde{z}$ stably, then the problem of computing the vector a from (2.15) is as well conditioned as the BVP itself!

4.2. The stability of the recurrence relation (2.11). In order to analyze the stability of (2.11), we examine the effects of additive perturbations $\{z_i^2\}$ and $\{z_i^1\}$ in (2.11)(a) and (b) respectively, like was done in [11, § 4]. So let $\{g_i\}$ satisfy the perturbed recursion

$$\text{(4.4)} \quad \begin{aligned} &\text{(a)}\ g_{i+1}^2 = E_i g_i^2 + z_{i+1}^2, \qquad g_0^2 = z_0^2, \\ &\text{(b)}\ g_i^1 = B_i^{-1}\{g_{i+1}^1 - C_i g_i^2\} + z_i^1, \qquad g_N^1 = z_N^1, \end{aligned}$$

where one may think of $\|z_i^1\|$, $\|z_i^2\|$ to be of the order of $\|A_i\|\varepsilon_M$ (cf. § 2). We then obtain

$$\text{(4.5)} \quad \begin{aligned} &\text{(a)}\ g_i^2 = \sum_{l=0}^{i} \left(\prod_{j=l}^{i-1} E_j\right) z_l^2, \\ &\text{(b)}\ g_i^1 = \sum_{l=0}^{i} \left\{\Omega_{i,N}\left(\prod_{j=l}^{i-1} E_j\right) z_l^2\right\} + \sum_{l=i+1}^{N-1} \left\{\left(\prod_{j=i}^{l-1} B_j\right)^{-1} \Omega_{l,N} z_l^2\right\} + \sum_{l=i}^{N} \left\{\left(\prod_{j=i}^{l-1} B_j\right)^{-1} z_l^1\right\}, \end{aligned}$$

where $\Omega_{p,q}$ is a shorter notation for

$$\Omega_{p,q} = -\sum_{l=p}^{q-1} \left\{\left(\prod_{j=p}^{l} B_j\right)^{-1} C_l \left(\prod_{j=p}^{l-1} E_j\right)\right\}. \tag{4.6}$$

Now if the partitioning is chosen correctly and if there is a dichotomic solution space, then it follows that $\|\prod_{j=l}^{i} E_j\|$ and $\|(\prod_{j=l}^{i} B_j)^{-1}\|$ are of order one. Moreover, if the solutions are directionally well separated, $\|\Omega_{p,q}\|$ will not be large (cf. [10, Rem. 6.13]. Hence contamination of errors will not produce large errors.

It was shown in [11] that a wrong choice for Q_0 (i.e. the span of the first k column vectors being close to an initial value of a decreasing mode) produces large $\|E_j\|$ and large $\|B_j^{-1}\|$ initially and therefore usually large $\Omega_{p,q}$, for p small. This led us to the permutation updating of § 3.4; in Example 5.2 we shall show how important this "optimal" choice for Q_0 may be. If there is no dichotomic solution space, one should hope that $\max_{i,l} \|\prod_{j=l}^{i} E_j\|$ and $\max_{i,l} \|(\prod_{j=l}^{i} B_j)^{-1}\|$ are not so large as to produce global rounding errors that threaten the desired discretization error. However because of a result given in [14, Thm. 3.17] we are inclined to believe that this can only happen for problems that are inherently unstable.

4.3. The effect of assembling on the global error. The last and perhaps most intriguing part of this stability analysis is to show that the assembly does not influence (at least not significantly) the global rounding error. This can be proven using an adapted version of the error analysis of [13], which we shall give below.

As before, let ε_M denote the machine epsilon. Then a realistic description of the rounding error in a major shooting incremental matrix V_j, assembled going from $t_{i_{j-1}}$ to t_{i_j}, is given by $\varepsilon_M \tilde{\Phi}_{i_j}[\tilde{\Phi}_{i_{j-1}}]^{-1}D_j$ where D_j is some matrix with $\|D_j\| = O(1)$. Similarly the inhomogeneous terms at t_i are perturbed by something like $\varepsilon_M \tilde{\Phi}_{i_j}[\tilde{\Phi}_{i_{j-1}}]^{-1}d_j$, where $\|d_j\| = O(1)$.

For simplicity we only investigate the global error caused by the latter perturbations. Therefore we introduce discrete Green's functions $Z_j(s)$ defined by

$$\text{(4.7)}\qquad \begin{aligned} &\text{(a)}\ Z_j(s) = V_j Z_{j-1}(s) + \Delta_{js},\\ &\text{(b)}\ M_\alpha Q_0 Z_0(s) + M_\beta Q_N Z_N(s) = 0, \end{aligned}$$

and where

$$\text{(4.8)}\qquad \Delta_{js} = \begin{cases} \tilde{\Phi}_{i_j}[\tilde{\Phi}_{i_{j-1}}]^{-1} & \text{if } j = s,\\ 0 & \text{if } j \neq s. \end{cases}$$

Similarly to [13, (3.5)(a)] we get

$$\text{(4.9)}\qquad Z_j(s) = -\tilde{\Phi}_{i_j}(R^{-1}M_\beta Q_N \tilde{\Phi}_N)\tilde{\Phi}_{i_s}^{-1}, \qquad j \leqq s,$$

the only distinction with the analogous formulae in [13] being that the Green's functions here are a "factor" $\tilde{\Phi}_{i_j}[\tilde{\Phi}_{i_{j-1}}]^{-1}$ bigger. Using these $Z_j(s)$ we obtain for the global error due to the above indicated perturbations

$$\text{(4.10)}\qquad e_j = \varepsilon_M \sum_s Z_j(s) d_s,$$

which is analogous to [13, Prop. 4.5].

Property 4.11. Assume that the solution space satisfies similar splitting properties as in [13], viz. there exist positive ν and μ, such that the increasing solutions grow at least like $\exp[(\mu(t_{i_j} - t_{i_{j-1}})]$ and the decreasing solutions at most like $\exp[-\nu(t_{i_j} - t_{i_{j-1}})]$. Then the global error (4.10) is estimated by

$$\|e_j\| \leqq \bar{\chi}(\mathscr{CN}+1)\left(\frac{\bar{\gamma}_1}{1-e^{-\nu h}} + \frac{\bar{\gamma}_2}{1-e^{-\mu h}}\right)\varepsilon_M \max_s \|d_s\|.$$

(N.B. $h = \min_s (t_{i_s} - t_{i_{s-1}})$, for $\mathscr{CN}$ see (4.1); $\bar{\gamma}_1$ and $\bar{\gamma}_2$ are just constants appearing in estimating the basis solutions.)

Remark 4.12. Using the notation $\Omega_{p,q}$ (see (4.6)) we can write

$$\tilde{\Phi}_i = \begin{pmatrix} I & \Omega_{i,N} \\ \varnothing & I \end{pmatrix} \begin{pmatrix} \prod_{j=0}^{i-1} B_j & \varnothing \\ \varnothing & \prod_{j=0}^{i-1} E_j \end{pmatrix}.$$

Hence, by definition of T_i in [13], the so-called "direction matrix", we see that

$$T_i=\begin{pmatrix} I & \Omega_{i,N}K_i \\ \varnothing & K_i \end{pmatrix}$$

for some K_i with $\|K_i^{-1}\| \leqq 1$. Hence we see that

$$\bar{\chi}=\max_i \|T_i^{-1}\| \leqq \max_i \left\| \begin{pmatrix} I & \Omega_{i,N} \\ \varnothing & D_i^{-1} \end{pmatrix} \right\| \leqq 1+\max_i \|\Omega_{i,N}\|.$$

Therefore, if the solutions are directionally well separated, implying that $\|\Omega_{i,N}\|$ is not large (cf. § 4.2), we have $\bar{\chi}=O(1)$.

The importance of the result in (4.11) is that it shows that the global effect of assembling is more or less independent of the length of the major shooting interval. Of course, one should realize that $\|d_s\|$ is of order $\sum_{l=i_{s-1}}^{i_s-1} \|A_l\|$ and hence about linear in the number of minor shooting intervals; however, this is not at all the exponential error growth that would occur in assembling coupled recursions! The crucial point is that, though the absolute errors in the computed $\|V_j\|$ may be large, they get smaller the lower the row index, i.e. relative to the increment of a certain mode as is given by a sequence of a certain column of the $\tilde{\Phi}_j$; hence, we have stability (thanks to our special computation by forward and backward recursion).

The problem that remains is: What happens when there is no dichotomy? This is a difficult question and certainly needs a more extensive study than we have made. Although one may be able to show that there exist a solution that does not increase and another that changes behaviour somewhere (like in turning point problems), a suitable linear combination of these may still exhibit dichotomy. In any case, assuming well conditioning, the dichotomy is assured, cf. [14]. Moreover, it is not difficult to give examples where lack of dichotomy leads to severe error growth, see Example 5.3.

In MUTS we have implemented a safety check in order to warn that global (rounding) errors may threaten the required accuracy. The best check would be to determine

$$\max_{p,q} \|\Omega_{p,q}\|, \max_{j,i} \left\| \prod_{l=j}^{i} E_l \right\| \quad \text{and} \quad \max_{j,i} \left\| \left(\prod_{l=j}^{i} B_j \right)^{-1} \right\|$$

and use this to get hold of the expression in (4.5). The next best check (as this is at least practically performable and easy to implement) is to compute

$$\rho=\max_{j,i} \left\| \prod_{l=j}^{i} \operatorname{diag}(E_l) \right\| \cdot \max_{j,i} \left\| \prod_{l=j}^{i} \operatorname{diag}(B_l^{-1}) \right\|.$$

Indeed, the maximal diagonal elements of the matrices $|B_l^{-1}|$ and $|E_l|$ are often a good estimate of their respective norms (cf. [11, see 6]). Hence if ρ is not large we can expect that both $\|\prod E_l\|$ and $\|(\prod B_l)^{-1}\|$ are $O(1)$. Since $\|B_l^{-1}C_l\|$ usually, is $O(1)$, we therefore may also use ρ as an estimate for $\max_{p,q} \|\Omega_{p,q}\|$. This estimate ρ is now used as follows. Suppose the user wants an accuracy tol. If the code detects that $\rho\varepsilon_M >$ tol, a warning error is given indicating that rounding errors may be blurring the result. However, in all test problems we noted that ρ also gave a fairly good estimate for the global discretization error amplification, see Examples 5.3 and 5.4.

5. Examples. In this section we give a number of numerical examples in order to illustrate the remarks and analyses above. All solutions of the following problems have been computed using the double precision code for the inhomogeneous problems, DMUTSG, as was developed at Nijmegen cf. [19]. The computations were performed on an IBM 4341/MVS computer.

We did not make explicit comparisons with other existing codes, like PASVAR [9], COLSYS [1] or SUPORT [18]. The main reason is that a simple comparison of cpu time is not very meaningful, as the number of examples is too small to draw significant conclusions and/or the way each code is implemented may blur any relative theoretical efficiency predictions. For SUPORT there was the additional argument that this code is designed for separated BC only. As far as memory requirements are concerned it might be obvious that any reasonable multiple shooting code is superior to methods that necessarily need to store information at all grid points. However, if a user wants a lot of output points, the storage requirements of our code and so-called global methods become comparable again. We finally note that the triangularization of the recursion and the solution of resulting BVP has a similar complexity as solving a (sparse) multiple shooting system by some decomposition method (cf. [15]); an LU-decomposition method might, however, require a complicated pivoting strategy and additional memory space, whereas our method only needs $\approx N(n^2+n)$ numbers (N—the number of major shooting intervals) to store and is straightforward.

Example 5.1. Consider the ODE

$$\frac{dx}{dt}=\begin{pmatrix} 1-19\cos 2t & 0 & 1+19\sin 2t \\ 0 & 19 & 0 \\ -1+19\sin 2t & 0 & 1+19\cos 2t \end{pmatrix} x+f(t), \tag{5.1}$$

where $f(t)=e^t(-1+19(\cos 2t-\sin 2t),\ -18,\ 1-19(\cos 2t+\sin 2t))^T$ and the BC

$$x(0)+x(\pi)=(1+e^\pi, 1+e^\pi, 1+e^\pi)^T. \tag{5.2}$$

The exact solution to this problem is $x=(e^t, e^t, e^t)^T$. The homogeneous part has solutions growing like $\sim e^{20t}$, $\sim e^{19t}$, $\sim e^{-18t}$, cf. [12, Example 6.2]. As requirements to have our solution approximated, we asked for an absolute tolerance of 1.0 −6 ($=10^{-6}$) and a maximal increment of homogeneous solutions $M=1.0+3$. (N.B. the code considers values between $.5M$ and $2.0M$ to be acceptable.) In Table 5.1 we give the result (up to two decimals). Note that the last major shooting interval is smaller than the rest. In Table 5.2 we give the result with tolerance 1.0 −7 (as before) but now with a maximal increment $M=1.0+30$. Note that this increment means that the *recursion* is solved by single shooting! The fact that the errors are so much smaller than was asked for is a typical vice of multiple shooting; indeed, as we let the integrator determine a fairly general solution (which most likely contains a component of increasing modes) within the required tolerance, the resulting grid usually gives a significantly smaller error for a smooth solution. Finally we remark that the cpu time for both cases was almost the same, as we noted in [15].

Example 5.2. Consider the same BVP as in Example 5.1. We used DMUTSG, now asking for output on 15 equally spaced points (hence the increment M per interval equals ≈ 100). Moreover we deliberately skipped that part where an optimal Q_0 is to be determined, so Q_0 was taken to be I. Since a fundamental solution of (5.1) is given by (cf. [12, Ex. 6.2])

$$F_0(t)=\begin{bmatrix} \sin t & 0 & -\cos t \\ 0 & 1 & 0 \\ \cos t & 0 & \sin t \end{bmatrix} \operatorname{diag}(e^{20t}, e^{19t}, e^{-18t}), \tag{5.3}$$

TABLE 5.1

i	t	U_{i-1}			$\|\text{error}\|_\infty$
0	0.00				1.9 −9
1	0.34	1.1 +3	0	2.5 −3	
		0	8.2 +2	0	
		0	0	1.7 −3	1.2 −9
2	0.70	7.0 +2	0	8.5 −4	
		0	5.0 +2	0	
		0	0	2.8 −3	1.8 −9
3	1.05	5.7 +2	0	5.0 −4	
		0	4.2 +2	0	
		0	0	3.2 −3	1.7 −9
⋮	⋮	⋮	⋮	⋮	⋮
9	2.83	1.7 +3	0	6.2 −5	
		0	1.2 +3	0	
		0	0	1.2 −3	1.9 −9
10	3.14	1.6 +1	0	3.8 −7	
		0	1.4 +1	0	
		0	0	8.1 −2	1.9 −9

TABLE 5.2

i	t	U_{i-1}			$\|\text{error}\|_\infty$
0	0				1.9 −9
1	3.14	1.9 +27	0	4.2 +21	
		0	8.4 +25	0	
		0	0	2.8 −25	1.9 −9

the 1st column of Q_0 generates a decreasing mode, if the computations were exact. In particular we can no longer hope that the B_i represent the increments of the increasing modes. However, due to discretization errors the *discrete* fundamental solution will most likely deviate from "the" exact solution; specifically, we can expect a discrete dominant solution to exist which has as initial value $(\varepsilon_1, \varepsilon_2, 1)^T$, where ε_1, ε_2 are of the order of the discretization error ε. Therefore, in practice this first column of Q_0 still contains a small component of the most unstable mode. After some time this mode has grown by a factor $1/\varepsilon$ and will become dominant; this can be seen from the U_i which will then have left upper blocks representing the increments of the dominant modes again. The consequence of this temporary "disorder" is that for smaller l both $\|[\prod_{j=0}^{l} B_j]^{-1}\|$ and $\|\prod_{j=0}^{l} E_j\|$ may be $O(1/\varepsilon)$ thus making the $\|\Omega_{0,q}\|$ (cf. (4.6)) larger. It can be shown that $\max_{p,q} \|\Omega_{p,q}\| = O(1/\varepsilon)$ is achievable (cf. [11, Ex. 5.2]). From the error analysis in § 4.2, cf. (4.5), it follows that we therefore may expect global errors of the order $\varepsilon_M/\varepsilon$. This is a funny result, since it means that for

ε smaller than $(\varepsilon_M)^{1/2}$, a smaller tolerance will give a larger global error (being a result of rounding errors!). Since we are working with a $\varepsilon_M \approx 1.0\,-16$ we expect this to happen if $\varepsilon \approx 1.0\,-8$. (Bearing in mind the "over-killing" effect, noted in Example 5.1, the threshold value of tol is $\approx 1.0\,-6$.) This is nicely demonstrated in Table 5.3. It is also instructive to see how the "errors" restore the proper ordering in the diagonal of the U_i after a few steps and how this has its impact on the error (see Table 5.4), where we only give the $(1,1)$, $(1,3)$ and $(3,3)$ element of the U_i, for a tolerance tol $= 1.0\,-10$.

TABLE 5.3

tol	max $\|\text{error}\|_\infty$	min $\|\text{error}\|_\infty$	"ε"
1.0 −4	2.2 −7	1.5 −7	1.0 −6
1.0 −6	5.0 −9	1.5 −9	1.0 −8
1.0 −8	6.6 −7	1.5 −11	1.0 −10
1.0 −10	1.1 −4	2.0 −13	1.0 −12

TABLE 5.4

i	U_{i-1}		$\|\text{error}\|_\infty$
0			1.1 −4
1	1.5 −2 0	1.9 −6 1.1 +2	2.3 −6
2	1.5 −2 0	1.4 −2 1.1 +2	3.9 −8
3	2.1 −2 0	1.2 +2 7.7 +1	7.0 −10
4	1.1 +2 0	6.5 +1 1.6 −2	9.5 −12
5	1.1 +2 0	1.1 −6 1.5 −2	2.3 −13
⋮	⋮	⋮	⋮

Example 5.3. Consider the following ODE

$$\frac{dx}{dt} = \begin{pmatrix} \psi(t) & 0 \\ 2\psi(t) & -\psi(t) \end{pmatrix} x + \begin{pmatrix} (1-\psi(t))\, e^t \\ 2\, e^t \end{pmatrix} \tag{5.4}$$

where $\psi(t) = 20 \sin t + 20t \cos t$. Let the BC be given by

$$x(0) + x(T) = \begin{pmatrix} 1 + e^T \\ 2(1+e^T) \end{pmatrix}, \qquad T > 0. \tag{5.5}$$

As one may check, a fundamental solution for (5.7) is given by

$$F_0(t)=\begin{pmatrix}1 & 0\\ 1 & 1\end{pmatrix}\operatorname{diag}(e^{\phi(t)}, e^{-\phi(t)}), \tag{5.6}$$

where $\phi(t)=20t\sin t$. Apparently $x=\binom{1}{2}e^t$ satisfies (5.5) and (5.4). For larger values of T it can be seen that there does not exist a dichotomic solution space. In fact we have a kind of turning point problem. In Table 5.5 we show the dramatic effect of this lack of dichotomy on the global error by giving some results for $T=2$, 2.5 and 3. We gave a tolerance of 1.0 −6 and asked for output at equally spaced points with distance 0.1. The quantity κ is an estimate for $\mathscr{C}\mathscr{N}$ (cf. [12], [13]); ρ is defined in § 4.3, being an estimate for the skewness Ω. In order to understand this result, one should realize

TABLE 5.5

T	max $\|\text{error}\|_\infty$	min $\|\text{error}\|_\infty$	ρ	κ
2	4.2 −8	4.0 −10	1.5	2.4
2.5	2.4 −3	5.2 −10	4.0 +5	2.4
3	5.0 +4	1.8 −2	2.8 +11	3.9 +9

that for $t=0$ and $t\approx 2.029$ we have a turning point, at which the increasing mode in F_0 becomes decreasing and the decreasing one increasing. Hence, even though there may be a globally dominant mode, we can no longer expect the diagonal of the upper triangular matrices U_i to be ordered for all i; the effect of this disordering is shown by the large ρ for $T=2.5$ and the very large ρ for $T=3$. For $T=2.5$ this instability has a limited effect on the accuracy of the solutions (although we lose 3 digits compared to $T=2$); also the conditioning of the system (2.15) (cf. also (4.1)) is still reasonable for $T=2.5$. However, for $T=3$, we may expect error amplification of the order of 1.0 +11 which nicely agrees with the result in the second column (note that we may expect local errors of the order of 1.0 −8 in this example, cf. the result for $T=2$). It was interesting to see that direct use of a Crout routine to solve the multiple shooting system either gave slightly worse results or no result at all (e.g. for $T=3$ the system was found to be numerically singular).

Example 5.4. Consider the ODE

$$\frac{dx}{dt}=\begin{pmatrix}t(1-\cos 2t) & 1+t\sin 2t\\ -1+t\sin 2t & t(1+\cos 2t)\end{pmatrix}x+f(t) \tag{5.7}$$

and a BC where $M_\alpha=M_\beta=I$. The function $f(t)$ and the vector b (in (1.2)) are chosen such that

$$x=\begin{pmatrix}1+\cos t\\ 1-\sin t\end{pmatrix}.$$

A fundamental solution of (5.7) is given by

$$F_0(t)=\begin{pmatrix}\cos t & \sin t\\ -\sin t & \cos t\end{pmatrix}\operatorname{diag}(1, e^{t^2}). \tag{5.8}$$

Hence, we see that the second basis solution changes at $t=0$ from a decreasing solution to an increasing solution. Therefore we have a kind of dichotomy (i.e. a splitting between nondecreasing and nonincreasing solutions) if either $[\alpha, \beta] \subset [0, \infty]$ or $[\alpha, \beta] \subset [-\infty, 0]$. In Table 5.6 below we have given errors and condition numbers for computations on the intervals $[0, 4]$, $[-2, 2]$ and $[-4, 4]$ respectively. The tolerance was set to $1.0 -8$ and we asked for output on equally spaced points (distance $= 0.4$); ρ and κ are defined as in Example 5.3. Once again we see that ρ quite accurately predicts the loss

TABLE 5.6

$[\alpha, \beta]$	max $\|\text{error}\|_\infty$	min $\|\text{error}\|_\infty$	ρ	κ
$[0, 4]$	5.8 −9	2.0 −9	1.1 +1	4.1
$[-2, 2]$	3.9 −7	2.5 −7	2.2 +2	1.6
$[-4, 4]$	4.2 −2	2.2 −2	2.5 +7	1.1

of significant digits. Like in the previous example, linear algebraic methods to compute solutions of the multiple shooting system did not perform any better.

Example 5.5. As the last of a series of "turning point" problems consider the scalar problem

$$\frac{d^2\xi}{dt^2} + 40t\frac{d\xi}{dt} = (1+40t)\, e^t, \tag{5.9}$$

with BC $\xi(\alpha) = e^\alpha$, $\xi(\beta) = e^\beta$. Hence $\xi(t) = e^t$. Written in vector form this corresponds to the BVP

$$\text{(5.10)}\quad \begin{aligned} &\text{(a)}\ \frac{dx}{dt} = \begin{pmatrix} 0 & 1 \\ 0 & -40t \end{pmatrix} x + \begin{pmatrix} 0 \\ (1+40t)\, e^t \end{pmatrix}, \qquad x = \begin{pmatrix} \xi \\ \dfrac{d\xi}{dt} \end{pmatrix}, \\ &\text{(b)}\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} x(\alpha) + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} x(\beta) = \begin{pmatrix} e^\alpha \\ e^\beta \end{pmatrix}. \end{aligned}$$

A fundamental solution is given by

$$F_0(t) = \begin{pmatrix} \int_\alpha^t e^{-20s^2}\, ds & 1 \\ e^{-20t^2} & 0 \end{pmatrix}. \tag{5.11}$$

For $t \ll 0$ we see that the first column is $\approx \binom{0}{0}$ whereas for $t \gg 0$ this column almost has the same direction as $\binom{1}{0}$. This indicates that some appropriate linear combination of the columns in $F_0(t)$ might give a better conditioned representation (in the sense of directional independence). Indeed computations reveal that we have a basis solution that does not increase and one that does not decrease, at least not significantly, and which are almost orthogonal at each point. We have computed the solutions on $[-1, 1]$ for several tolerances. First we give in Table 5.7 the infinity norms of the first and the second basis solution as they are found from using the decoupled recursion (cf. § 2). It is no surprise that we obtain stable results for the desired solution x. In Table 5.8 we give the maximal error in the computed result on $[-1, 1]$ at equally spaced output points for several tolerances. We remark that in this example the use of incremental

TABLE 5.7

t	$\|\text{first column of } \tilde{\Phi}(t)\|_\infty$	$\|\text{second column of } \tilde{\Phi}(t)\|_\infty$
−1.0	5.2 −9	1.0
− .8	7.0 −6	1.0
− .6	1.9 −3	1.0
− .4	1.0 −1	1.0
− .2	1.1	1.0
0	2.6	9.8 −1
.2	1.4	7.8 −1
.4	1.0	1.0 −1
.6	1.0	1.9 −3
.8	1.0	7.0 −6
1.0	1.0	5.2 −9

TABLE 5.8

tolerance	$\|\text{error}\|_\infty$	ρ	κ
1.0 −4	2.0 −6	1.6 +1	1.0
1.0 −6	2.0 −8	1.6 +1	1.0
1.0 −8	4.7 −10	1.6 +1	1.0

values M (the more "standard implementation") would not give a satisfactory distribution of output points for negative t (both basis solutions have low activity for large negative t!) and in this way one certainly would not "detect the turning point".

Example 5.6. Finally we would like to illustrate our remark in § 3.5 about smooth problems. Consider the ODE

$$\frac{dx}{dt} = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 19 & 0 \\ 0 & 0 & -18 \end{pmatrix} x + \begin{pmatrix} -20 \\ -19 \\ 18 \end{pmatrix} \tag{5.12}$$

and $x(0) + x(\pi) = (2, 2, 2)^T$. Apparently $x(t) = (1, 1, 1)^T$. We computed this solution in four ways. First we asked for output at 10 equally spaced nodes, both with the choices $w_i(t_i) = 0$ (cf. (3.7)) at each of the minor shooting points and with the choice of $w_i(t_i)$ as in (3.31). Then we did the same computations now with a prescribed amplification $M = 1.0 + 3$. As a tolerance we had $1.0 - 3$. It is not surprising that the obtained accuracy was essentially on the order of the machine precision. The results are given in Table 5.9 (the cpu time is in milliseconds). Since we have such a low tolerance, our efficiency strategy could not work optimally (note the doubling of grid points and shooting points in the second experiment as compared to the last one). On the other hand we see that in the last row the number of minor shooting points is equal to the number of major shooting points, indicating that for larger values of M there even may be more room for efficiency. Indeed, if we take $M = 1.0 +6$ the cpu time drops to 20 milliseconds.

TABLE 5.9

OUTPUT required	$w(t_i)$	# grid pts.	# minor s.p.	# major s.p.	cpu time
10 equal pts.	(3.7)	130	30	10	108
10 equal pts.	(3.31)	76	21	10	70
$M = 1.0\ \ +3$	(3.7)	131	27	8	116
$M = 1.0\ \ +3$	(3.31)	38	8	8	33

REFERENCES

[1] U. ASCHER, J. CHRISTIANSEN AND R. D. RUSSELL, COLSYS—*a collocation code for boundary value problems*, in Lecture Notes in Computer Science 76, Springer-Verlag, Berlin, 1979, pp. 164–165.

[2] C. DEBOOR AND R. WEISS, SOLVEBLOCK: *A package for solving almost block diagonal linear systems*, ACM Trans. Math. Software, 6 (1980), pp. 80–87.

[3] S. D. CONTE, *The numerical solution of linear boundary value problems*, SIAM Rev., 8 (1966), pp. 309–321.

[4] P. DEUFLHARD, *A modified Newton method for the solution of ill-conditioned systems of nonlinear equations, with application to multiple shooting*, Numer. Math., 22 (1974), pp. 289–315.

[5] ———, *Recent advances in multiple shooting techniques*, in Computational Techniques for Ordinary Differential Equations, Academic Press, New York, London, 1980, Section 10, pp. 217–272.

[6] R. ENGLAND, *A program for the solution of boundary value problems for systems of ordinary differential equations*, Culham Lab., Abingdon, Techn. Rep. CLM-PDN 3/73, 1976.

[7] G. E. FORSYTHE, M. A. MALCOLM AND C. B. MOLER, *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[8] H. B. KELLER, *Numerical solution of two point boundary value problems*, CBMS Regional Conference Series in Applied Mathematics 24, Society for Industrial and Applied Mathematics, Philadelphia, 1976.

[9] M. LENTINI AND V. PEREYRA, *An adaptive finite difference solver for nonlinear two-point boundary problems with mild boundary layers*, SIAM J. Numer. Anal., 14 (1977), pp. 91–111.

[10] R. M. M. MATTHEIJ, *Characterization of dominant and dominated solutions of linear recursions*, Numer. Math., 35 (1980), pp. 421–442.

[11] ———, *Stable computation of solutions of unstable linear initial value recursions*, BIT, 22 (1982), pp. 79–93. See also Report 8108, Mathematisch Instituut, Nijmegen, which is somewhat more elaborate.

[12] ———, *The conditioning of linear boundary value problems*, SIAM J. Numer. Anal., 19 (1982), pp. 963–978.

[13] ———, *Estimates for the errors in the solution of linear boundary value problems, due to perturbations*, Computing, 27 (1981), pp. 299–318.

[14] ———, *The stability of LU-decompositions of block tridiagonal matrices*, Rep. Dept. Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY, 12181, 1982. Math. Comp. to appear.

[15] R. M. M. MATTHEIJ AND G. W. M. STAARINK, *On optimal shooting intervals*, Report 8123, Mathematisch Instituut, Katholieke Universiteit, Nijmegen, the Netherlands, 1981.

[16] M. R. Osborne, *The stabilized march is stable,* SIAM J. Numer. Anal., 16 (1979), pp. 923–933.

[17] S. M. Roberts and J. S. Shipman, *Two Point Boundary Value Problems: Shooting Methods,* Elsevier, New York, 1972.

[18] M. R. Scott and H. A. Watts, *Computational solution of linear two point boundary value problems via orthonormalization,* SIAM J. Numer. Anal., 14 (1977), pp. 40–70.

[19] G. W. M. Staarink and R. M. M. Mattheij, BOUNDPACK: *A package for solving boundary value problems,* Mathematisch Instituut, Katholieke Universiteit, Toernooiveld, Nijmegen, the Netherlands, 1982.

[20] J. R. Stoer and R. Bulirsch, *Einführung in die Numerische Mathematik* II, HTB 114, Springer, Berlin, 1973.

[21] 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. 5, No. 4, December 1984

OBLIQUE PROCRUSTES ROTATIONS IN FACTOR ANALYSIS*

FRANKLIN T. LUK†

Abstract. This paper concerns the oblique rotation of a factor matrix so as to be a least squares fit to a target matrix. An iterative computing procedure is presented.

Key words. oblique rotations, Procrustes problem, factor analysis, least squares

1. Introduction. An important problem in factor analysis is the so-called Procrustes problem (cf. Harman [7, § 15.5]). It addresses the extent to which a given body of data can be described in terms of a prescribed factor pattern. Let A be the given $p \times m$ factor matrix and B the prescribed $p \times m$ factor pattern. Suppose that X is a nonsingular $m \times m$ transformation matrix and that $Z = AX$. We want to find X so as to minimize the least squares criterion

$$\sum_{j=1}^{m} \left\{ \sum_{i=1}^{p} w_{ij}(z_{ij} - b_{ij})^2 \right\}, \tag{1.1}$$

where $Z = (z_{ij})$, $B = (b_{ij})$ and w_{ij} are some fixed arbitrary nonnegative weights (usually equal to one or zero). The case where X is orthogonal and all weights equal unity, i.e., when the target B is fully specified, has been solved by Schönemann [10]. His solution involves the computation of the singular value decomposition of the matrix $A^T B$. The Procrustes problem is more difficult when only some of the weights are one and the rest are zero, i.e., when the target B is only partially specified. Browne [2] shows how one may approximate X by a sequence of plane rotations, and Luk [9] presents an efficient numerical procedure for computing these rotations. The case of an oblique transformation, i.e., X must satisfy the condition (cf. [8, p. 315])

$$\operatorname{diag}\{(X^T X)^{-1}\} = I, \tag{1.2}$$

remains unsolved for both a fully and a partially specified target B. There are in the literature procedures for finding an approximate solution. Gruvaeus [6] adopts a penalty function approach, using a series of Fletcher and Powell [4] minimizations. His method is very expensive because it involves an inversion of an $m \times m$ matrix at each minimization step. Browne [3] suggests that one approximate X by a sequence of elementary oblique rotations (see [8]). He reports numerical results [3, p. 210] that compare favorably with those given by Gruvaeus' method. Due to its simplicity (hence lower execution cost) and geometrical elegance, Browne's procedure is usually preferred over Gruvaeus' approach.

Browne [3] proposes that one apply Newton's method to compute the elementary oblique rotations. The purpose of this paper is to show how these rotations can be determined in an easier manner using Lagrange multipliers.

2. The algorithm. Let $I(j)$ represent the set of row indices corresponding to specified elements of column j of B, for $j = 1, 2, \cdots, m$. The minimization criterion (1.1) simplifies to

$$\sum_{j=1}^{m} \left\{ \sum_{i \in I(j)} (z_{ij} - b_{ij})^2 \right\}. \tag{2.1}$$

* Received by the editors February 1, 1982, and in revised form February 15, 1983. This research was supported in part by the U.S. Army Research Office under grant DAAG 29-79-C0124.

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

Browne [3] approximates X by a product of elementary oblique rotations. An elementary oblique rotation of the ith factor (primary factor) over the plane defined by the ith and jth (say $i<j$) factors is of the form [8, p. 316]:

(2.2)
$$\begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & 0 & \\ & & 1 & & & & & \\ & & & \gamma & & -\delta & & \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \\ & 0 & & & & & & \ddots \\ & & & & & & & & 1 \end{pmatrix}\begin{array}{l} \\ \\ \\ i \\ \\ \\ j \\ \\ \\ \end{array},$$

with i and j marking the columns of γ and $-\delta$ respectively.

i.e., the identity matrix with the (i, i)th and (i, j)th elements altered. The parameters satisfy the relation [8, p. 317]:

(2.3)
$$\gamma^2=\delta^2+2c_{ij}\delta+1,$$

where c_{ij} $(-1<c_{ij}<1)$ represents the correlation coefficient between the ith and jth factors before rotation.

Designate by F the current rotated factor matrix at an intermediate stage, and let $F=(f_{ij})=(f_1, f_2, \cdots, f_m)$. Let us consider a rotation of the ith factor over the plane defined by the ith and jth factors. Such a rotation will affect only the ith and jth columns of F and may be represented as

(2.4)
$$(f_i, f_j)\begin{pmatrix} \gamma & -\delta \\ 0 & 1 \end{pmatrix}.$$

The criterion (2.1) thus simplifies to [3, p. 208]

(2.5)
$$g(\delta)\equiv a_2\delta^2+a_1\delta+a_*\gamma+a_0,$$

where

$$a_2=\sum_{r\in I(i)} f_{ri}^2+\sum_{s\in I(j)} f_{si}^2,$$

$$a_1=2c_{ij}\sum_{r\in I(i)} f_{ri}^2-2\sum_{s\in I(j)}\{f_{si}(f_{sj}-b_{sj})\},$$

$$a_*=-2\sum_{r\in I(i)} f_{ri}b_{ri},$$

and a_0 is an unimportant constant. Browne forces the coefficient a_* to be nonpositive (by reflecting f_i if necessary) so that γ always requires a positive square root (cf. (2.3) and (2.5)). He proposes that one use Newton's method to solve the equation

(2.6)
$$g'(\delta)\equiv 2a_2\delta+a_1+a_*(\delta+c_{ij})\gamma^{-1}=0,$$

and describes how one may determine an interval containing one minimum (the global one) of $g(\delta)$.

Browne carries out successive rotations of a single factor over each of the $(m-1)$ planes defined by the factor and every remaining factor, provided that the angle of rotation is sufficiently large. Iteration is continued until no rotations are carried out during one complete cycle of all $m(m-1)$ possible rotations. He also suggests that one interchange the factors before each rotation in accordance with the criterion in [2, p. 119]. The following example shows that Browne's method does not always minimize the criterion (2.1). Let

$$A=\begin{pmatrix} 3 & -2 & -2 \\ -2 & 3 & -2 \\ -2 & -2 & 3 \\ 3 & 2 & 2 \\ 0 & \sqrt{2} & \sqrt{2} \end{pmatrix}$$

and

$$B=\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Browne's procedure terminates because only null rotations are generated. But the transformation

$$X=\frac{1}{3}\begin{pmatrix} 1 & -2 & -2 \\ -2 & 1 & -2 \\ -2 & -2 & 1 \end{pmatrix}$$

will decrease the value of (2.1).

We propose here an alternate scheme for the subproblem of finding an oblique transformation. Suppose that $\alpha=|I(i)|$ and $\beta=|I(j)|$. Let $i(1), i(2), \cdots, i(\alpha)$ and $j(1), j(2), \cdots, j(\beta)$ denote the row indices of the specified elements of columns i and j, respectively, of the target B. Our goal is therefore the determination of the parameters γ and δ so as to minimize the Euclidean length of the vector

$$(2.7)\qquad \begin{pmatrix} f_{i(1),i} & 0 \\ f_{i(2),i} & 0 \\ \vdots & \vdots \\ f_{i(\alpha),i} & 0 \\ 0 & -f_{j(1),i} \\ 0 & -f_{j(2),i} \\ \vdots & \vdots \\ 0 & -f_{j(\beta),i} \end{pmatrix}\begin{pmatrix} \gamma \\ \delta \end{pmatrix}-\begin{pmatrix} b_{i(1),i} \\ b_{i(2),i} \\ \vdots \\ b_{i(\alpha),i} \\ b_{j(1),j}-f_{j(1),j} \\ b_{j(2),j}-f_{j(2),j} \\ \vdots \\ b_{j(\beta),j}-f_{j(\beta),j} \end{pmatrix}.$$

In other words, we want to solve the constrained least squares problem:

$$(2.8)\qquad \|K\mathbf{x}-\mathbf{q}\|=\text{minimum},$$

where $\|\cdot\|$ denotes the Euclidean vector norm, subject to the constraint:

$$(2.9)\qquad x_1^2=(x_2+c)^2+(1-c^2),$$

with $-1<c<1$. It follows that

$$(2.10)\qquad x_1\neq 0.$$

Using Lagrange multipliers, we obtain the matrix equation

$$(K^TK+\lambda E)\mathbf{x}=K^T\mathbf{q}+c\lambda\mathbf{e}_2, \tag{2.11}$$

where

$$E=\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$$

and λ is a Lagrange parameter. But

$$K^TK=\begin{pmatrix}d_1 & 0\\ 0 & d_2\end{pmatrix}, \tag{2.12}$$

with

$$d_1=\sum_{k=1}^{\alpha} f_{i(k),i}^2 \quad\text{and}\quad d_2=\sum_{k=1}^{\beta} f_{j(k),i}^2.$$

We may assume that d_1+d_2 is positive, else the solution is $\mathbf{x}=(\pm1,0)^T$. Letting

$$\mathbf{h}\equiv\begin{pmatrix}h_1\\ h_2\end{pmatrix}\equiv K^T\mathbf{q}, \tag{2.13}$$

we can rewrite (2.11) as

$$\begin{pmatrix}d_1+\lambda & 0\\ 0 & d_2-\lambda\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}=\begin{pmatrix}h_1\\ h_2+c\lambda\end{pmatrix} \quad\text{or}\quad \begin{pmatrix}h_1\\ (h_2+cd_2)-c(d_2-\lambda)\end{pmatrix}. \tag{2.14}$$

We now show how the proper value of λ can be determined.

LEMMA 1. *Let both vectors* $\mathbf{x}$ *and* $\mathbf{y}$ *satisfy* (2.9). *Then*

$$|x_1y_1|\geqq|(x_2+c)(y_2+c)+(1-c^2)|, \tag{2.15}$$

where the inequality is strict if $x_2\neq y_2$, *and*

$$(x_2+c)(y_2+c)+(1-c^2)-x_1y_1=\tfrac{1}{2}[(x_1-y_1)^2-(x_2-y_2)^2]. \tag{2.16}$$

Proof. Using (2.9) to substitute for x_1 and y_1, we get

$$\begin{aligned}x_1^2y_1^2&=[(x_2+c)(y_2+c)+(1-c^2)]^2+(1-c^2)(x_2-y_2)^2\\ &\geqq[(x_2+c)(y_2+c)+(1-c^2)]^2.\end{aligned}$$

We prove (2.16) by expanding its right-hand side and applying (2.9). □

LEMMA 2. *If* $(\mathbf{x},\lambda_x)$ *and* $(\mathbf{y},\lambda_y)$ *are solutions to the equations* (2.9) *and* (2.14), *then*

$$\|K\mathbf{y}-\mathbf{q}\|^2-\|K\mathbf{x}-\mathbf{q}\|^2=(\lambda_x-\lambda_y)[(x_2+c)(y_2+c)+(1-c^2)-x_1y_1]. \tag{2.17}$$

Proof. Applying the same technique as Gander [5, Thm. 1], we can show that

$$\|K\mathbf{y}-\mathbf{q}\|^2-\|K\mathbf{x}-\mathbf{q}\|^2=\tfrac{1}{2}(\lambda_x-\lambda_y)[(x_1-y_1)^2-(x_2-y_2)^2].$$

Equation (2.16) thus completes the proof. □

THEOREM 1. *Let* $(\mathbf{x}, \lambda_x)$ *and* $(\mathbf{y}, \lambda_y)$ *be solutions to the equations* (2.9) *and* (2.14). *Suppose that either* (i) $\lambda_x > -d_1$ *and* $\lambda_y < -d_1$, *or* (ii) $\lambda_y > \lambda_x > -d_1$. *Then*

$$\|K\mathbf{y}-\mathbf{q}\|^2 - \|K\mathbf{x}-\mathbf{q}\|^2 > 0.$$

Proof. Since $\lambda_x \neq \lambda_y$, we get $x_2 \neq y_2$ and thus a strict inequality in (2.15). The proof follows from considering the sign of $x_1 y_1$ in (2.17). □

Let us first assume that $h_1(h_2 + cd_2) \neq 0$. From (2.9) and (2.14) we obtain

$$\left(\frac{h_1}{d_1+\lambda}\right)^2 = \left(\frac{h_2+cd_2}{d_2-\lambda}\right)^2 + (1-c^2), \tag{2.18}$$

which can be transformed into a quartic equation in λ. It is easy to see graphically that the equation has one real root in the open interval $(-\infty, -d_1)$, one real root in $(-d_1, d_2)$, and two distinct real or complex conjugate roots in (d_2, ∞). Theorem 1 says that we need the smallest root of λ that is greater than $-d_1$, i.e., that unique root in the open interval $(-d_1, d_2)$. To find the root we may apply either the standard technique for solving quartic equations, or any zero-finding procedure with assured convergence, e.g., Brent [1, Chap. 4].

Now we give explicit solutions to two simpler special cases: (i) $h_1 = 0$, and (ii) $h_2 + cd_2 = 0$. For case (i) we get from $x_1 \neq 0$ that $\lambda = -d_1$. We then use (2.14) to compute x_2 and (2.9) to determine x_1 (either sign is acceptable). Case (ii) implies that either $x_2 = -c$ or $\lambda = d_2$. From Theorem 1 we get

$$\lambda = \min\left\{-d_1 + \left(\frac{h_1^2}{1-c^2}\right)^{1/2}, d_2\right\}.$$

We then use equations (2.9) and (2.14) to determine the solution vector $\mathbf{x}$. This special case is not observed by Browne [3]. Two other advantages of our procedure over Browne's approach are that (2.18) is easier to solve than (2.6), and that no column reflections are required.

After the parameters δ and γ for the elementary oblique rotation have been found, the factor matrix is updated in accordance to (2.4). The angle θ of rotation is given by

$$\theta = \cos^{-1}[(1+\delta c_{ij})/\gamma]. \tag{2.19}$$

Details of the complete algorithm are given in [3].

3. Numerical example. We were able to find only one example (Gruvaeus [6, p. 500]) for which the target matrix B is partially prescribed and some specified elements are nonzero. The given factor matrix is

$$A = \begin{pmatrix} .90 & -.09 & -.03 \\ .83 & .09 & -.04 \\ .87 & -.01 & .07 \\ .55 & .79 & -.07 \\ .56 & .65 & .04 \\ .63 & .60 & .03 \\ .28 & .27 & .45 \\ .38 & .20 & .63 \\ .38 & .19 & .77 \end{pmatrix}$$

and the partially prescribed matrix is

$$B=\begin{pmatrix} .98 & 0 & 0 \\ .76 & 0 & 0 \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ 0 & 0 & .76 \\ 0 & 0 & .93 \end{pmatrix}.$$

If the rotation angle θ of (2.19) is so small that $\cos\theta \geqq .9999995$, we regard the rotation as null and forgo the transformation. The fourth cycle consists wholly of null rotations, resulting in the final rotated matrix:

$$AX=\begin{pmatrix} .94 & -.09 & -.02 \\ .79 & .10 & .02 \\ .83 & -.04 & .11 \\ .17 & .84 & .15 \\ .20 & .66 & .24 \\ .30 & .61 & .22 \\ -.04 & .15 & .57 \\ .02 & .02 & .75 \\ -.03 & -.03 & .90 \end{pmatrix}.$$

The criterion of (2.1) decreases as follows:

Cycle	Criterion
0	.437400
1	.055413
2	.024703
3	.024691
4	.024691

Acknowledgments. The author is grateful to the three referees for their valuable suggestions.

REFERENCES

[1] R. P. BRENT, *Algorithms for Minimization Without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[2] M. W. BROWNE, *Orthogonal rotation to a partially specified target*, Br. J. Math. Statist. Psychol., 25 (1972), pp. 115–120.

[3] ———, *Oblique rotation to a partially specified target*, Br. J. Math. Statist. Psychol., 25 (1972), pp. 207–212.

[4] R. FLETCHER AND M. J. D. POWELL, *A rapidly convergent descent method for minimization*, Comput. J., 2 (1963), pp. 163–168.

[5] W. GANDER, *Least squares with a quadratic constraint*, Numer. Math., 36 (1981), pp. 291–307.

[6] G. T. GRUVAEUS, *A general approach to Procrustes pattern rotation*, Psychometrika, 35 (1970), pp. 493–505.

[7] H. H. HARMAN, *Modern Factor Analysis*, 3rd ed., University of Chicago Press, Chicago, 1976.
[8] R. I. JENNRICH AND P. F. SAMPSON, *Rotation for simple loadings*, Psychometrika, 31 (1966), pp. 313–323.
[9] F. T. LUK, *Orthogonal rotation to a partially specified target*, this Journal, 4 (1983), pp. 223–228.
[10] P. H. SCHÖNEMANN, *A generalized solution of the orthogonal Procrustes problem*, Psychometrika, 31 (1966), pp. 1–10.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

A NUMERICAL METHOD FOR CONFORMAL MAPPING OF DOUBLY CONNECTED REGIONS*

BENGT FORNBERG†

Abstract. A linearly converging iteration scheme has been found which yields the conformal mapping from the annulus between two concentric circles to any given doubly connected region with smooth boundaries. Each iteration costs $O(N \log N)$ operations where N is the number of points used to discretize the boundaries. Failure of convergence was never observed when the initial guess was sufficiently accurate.

Key words. conformal mapping, doubly connected regions, fast Fourier transform

1. Introduction. The mapping theorem by Riemann states that there are three degrees of freedom in a conformal mapping between any simply connected region and the unit circle. For example we can specify the positions of three boundary points (in the same order on the two peripheries) or of one interior and one boundary point. In the case of doubly connected regions, a mapping is always possible to the annulus between the unit circle and a concentric smaller circle with radius ρ. In this case, a rotation is the only available freedom. The radius ρ is uniquely determined.

Table 1 gives a very brief list of some major methods which we hope can serve as a first guide to the literature in the field. The cost per iteration of the methods (measured in number of arithmetic operations where N is the number of free parameters to be determined) has been given but we should note that this alone is a very incomplete measure of the usefulness of a method in a specific application. An extensive (but by now somewhat outdated) survey of methods was given by Gaier in 1964 [2].

Several of the numerical methods for simply connected regions are possible to generalize to the doubly connected case. So far, the best demonstrated procedure seems to be a generalization of Theodorsen's method (Garrick [3], Gaier [2], Ives [9], Henrici [7]). However, linear convergence will occur only for regions which have "near-circular" inner and outer boundaries. Since this method is the fastest one known for such regions and the operation count per iteration is nearly the same as that for the method we introduce here, these two methods will be compared extensively in the last section of this work. Another especially interesting method is proposed by Reichel [15]. His version of Symm's method converges to a final answer in a total of $O(N^2 \log N)$ operations. There is no need for the regions to be of simple shape or for a close initial guess to be available. In cases where the methods costing $O(N \log N)$ operations per iteration converge slowly, Reichel's method may prove faster.

Reference [1] describes a Newton–conjugate gradient scheme for simply connected regions which converges quadratically. Each iteration costs $O(N \log N)$ operations. The present work is a first attempt to exploit the formulation used in [1] for doubly connected cases. We have not been able to find an equally powerful numerical algorithm this time and the convergence rate is reduced to linear. However, the ability to converge for all smooth regions is maintained. This is a major advantage in comparison with Theodorsen's method.

Conformal mappings provide a powerful approach to many problems governed by Laplace's equation. This includes electrical and magnetical fields, potential fluid

* Received by the editors July 20, 1981, and in final revised form July 21, 1983. This research was performed at California Institute of Technology and supported by Control Data Corporation, the U.S. Department of Energy (Office of Basic Energy Sciences) and the John Simon Guggenheim Memorial Foundation.

† Exxon Research and Engineering Company, Clinton Township, Route 22 East, Annandale, New Jersey 08801.

TABLE 1
Brief survey of numerical methods for conformal mappings.

Name	Method Comments	Cost per iteration, Different connectivities			Convergence rate	
		Simply C	Doubly C	General C		
Theodorsen 1931 [7]	Nonlinear integral equation in polar coordinates	$N \log N$	$N \log N$	—	Linear	Near circular regions only
Gutknecht 1981 [5], [8]	Functional iteration SOR type relaxation	$N \log N$			Linear	
	Linear singular integral equation, solved by:					
Symm 1966 [7], [17], [18]	Direct iteration	$N^2 \log N$	$N^2 \log N$	—	Linear	
Los Alamos groups 1972, 1980 [4], [12]	Gaussian elimination	N^3	—	—	Direct	Regions of general shape
Reichel 1981 [15]	Part Cholesky factorized, rest iterated	$N^2 \log N$	$N^2 \log N$	$N^2 \log N$	Very fast linear	
Meiron, Orszag, Israeli 1980 [11]	ODE for initial value problem. Requires very small continuation step from correct mapping of nearby region	$N \log N$	—	—	Direct	
Fornberg 1980 [1]	Newton–conjugate gradient	$N \log N$	—	—	Quadratic	
Papamichael et al. 1981 [13], [14]	Variational method. Allows corners, but ill conditioned. Needs $N < 30$	N^3	N^3	—	Direct	
Fornberg 1984 (present work)	Laurent expansion. FFT	—	$N \log N$	—	Linear	

flow past a body etc. A calculation of time dependent surface waves on deep water was described in [1] as an application for simply connected mappings. The present method can similarly be applied to waves over a bottom of any shape or to waves on deep water passing over a submerged object.

Another area of application is computer generated computational grids. A drawback with simply connected conformal mappings is their tendency to give a very nonuniform resolution. A local boundary fitting grid partially overlapping with a rectangular grid has been found to be effective in many cases of finite difference approximations of partial differential equations (G. Starius [16], B. Kreiss [10]). As Figs. 1 and 2 illustrate (the outer boundary is the same in the two cases) there need not be any resolution problem for doubly connected mappings.

2. Formulation of the mapping problem. Different analytical properties of the conformal mapping function can be exploited in the construction of numerical methods. We will now very briefly outline these for a few of the major methods (in case of simply connected regions).

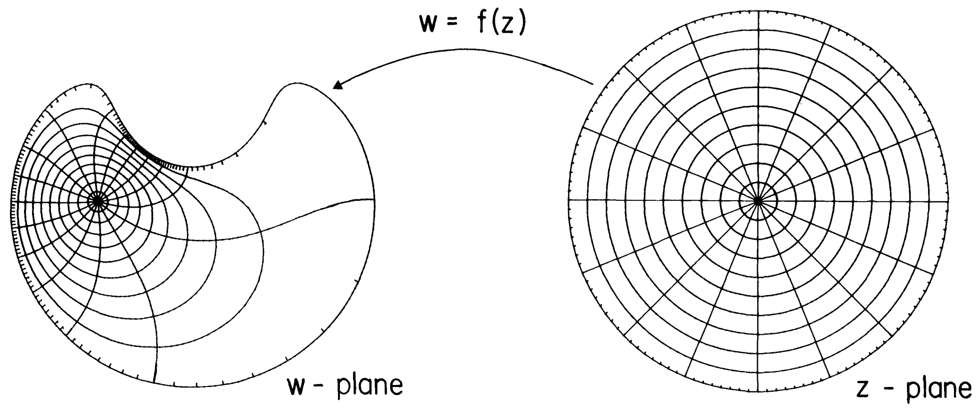


FIG. 1. *Example of a simply connected mapping.*

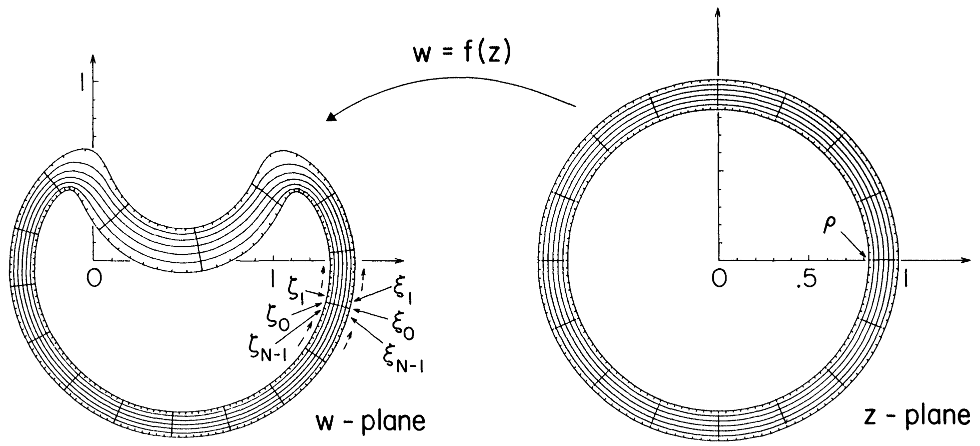


FIG. 2. *Example of a doubly connected mapping.*

Theodorsen's method is based on a Hilbert transform relation between the real and imaginary parts of $w = \log(f(z)/z)$ where $f(z)$ is the mapping function from the unit circle. If the given region is represented in polar coordinates, the relation takes a particularly simple form.

In Symm's method, the inverse mapping (from the given region to the unit circle) is considered. Through the (unknown) mapping function, the polar coordinate angle θ on the unit circle is a function of t, some parameter describing positions along the given boundary. The function $\theta'(t)$ can be shown to satisfy a linear Fredholm integral equation of the first kind.

The methods used by Papamichael are closely related to the Ritz method: the derivative of the mapping function from the given domain onto the unit circle can be shown to minimize

$$\iint_{\Omega} |u(z)|^2 \, dx \, dy \tag{1}$$

over a certain class of functions.

In paper [1] by the present author, we considered a Fast Fourier Transform applied to a set of points on the given boundary. This gives Laurent coefficients for an analytic function which maps roots of unity to these points. The mapping function has to be regular at the origin. The requirement that all negative Laurent terms vanish provides a formulation.

The methods of Theodorsen, Symm and Papamichael all generalize almost directly to doubly connected domains [7], [14], [18]. This work is a first attempt to make use of a formulation similar to the one in [1]. In case of a doubly connected region, a full Laurent expansion with both positive and negative powers is allowed. However, N 'boundary correspondence points' on either boundary is sufficient to determine N Laurent coefficients. Since the Laurent expansion is unique, this imposes constraints on the point positions, sufficient to determine their locations. Although we are not aware of this formulation having been used earlier, we do not believe it to be new. The original contribution of the present work lies in the construction of the numerical method. In the rest of this chapter, we present our formulation in more detail.

Figure 2 illustrates a mapping of a doubly connected region. With a region given in a complex w-plane, the inner radius as well as the mapping function $w=f(z)$ are uniquely determined up to a rotation in the z-plane. A Laurent expansion of $f(z)$ in the annulus $\rho \leqq |z| \leqq 1$ takes the form

$$w=f(z)=\sum_{\nu=-\infty}^{\infty} a_\nu z^\nu. \tag{2}$$

Let N, a power of two, be large enough that a_ν and $\rho^\nu a_\nu$ can be ignored for $|\nu|>N/2-1$. We consider (2) for $z=\omega^k$ and $z=\rho\omega^k$, $k=0,1,\cdots,N-1$, $\omega=e^{2\pi i/N}$. These equidistant points on the outer and inner circles in the z-plane are mapped by (2) into points ξ_k and ζ_k on the boundaries of the given region. Relation (2) for the points on the outer boundary can be written in matrix form

(3)

$$\begin{bmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \xi_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & \omega & \omega^2 & & \omega^{N/2-1} & -1 & \omega^{-N/2+1} & & \omega^{-2} & \omega^{-1} \\ 1 & \omega^2 & \omega^4 & & \cdot & 1 & \cdot & & \omega^{-4} & \omega^{-2} \\ \cdot & & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ \cdot & & & & \cdot & \cdot & \cdot & & \cdot & \cdot \\ 1 & \omega^{N-1} & \omega^{2N-2} & \cdot & \cdot & -1 & \cdot & \cdot & \omega^{-2N+2} & \omega^{-N+1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ a_{N/2-1} \\ a_{\pm N/2} \\ a_{-N/2+1} \\ \cdot \\ \cdot \\ a_{-2} \\ a_{-1} \end{bmatrix}.$$

For later algebraic convenience, an extra 'reflection frequency' denoted $a_{\pm N/2}$ is included. Under the assumptions on the decay of $|a_\nu|$, the size of $a_{\pm N/2}$ is negligible. Noting that $\omega^k=\omega^{N+k}$, (3) takes the simpler form

$$\begin{bmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \xi_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2N-2} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ 1 & \omega^{N-1} & \omega^{2N-2} & \cdots & \omega^{(N-1)^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ a_{N/2-1} \\ a_{\pm N/2} \\ a_{-N/2+1} \\ \cdot \\ a_{-1} \end{bmatrix}. \tag{4}$$

For the inner boundary, we similarly obtain

$$
(5)\quad
\begin{bmatrix} \zeta_0 \\ \zeta_1 \\ \zeta_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \zeta_{N-1} \end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & \dots & 1 \\
1 & \omega & \omega^2 & \dots & \omega^{N-1} \\
1 & \omega^2 & \omega^4 & \dots & \omega^{2N-2} \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
1 & \omega^{N-1} & \omega^{2N-2} & \dots & \omega^{(N-1)^2}
\end{bmatrix}
\begin{bmatrix}
1 & & & & & & & & \\
 & \rho & & & & & & & \\
 & & \rho^2 & & & & & & \\
 & & & \ddots & & & & & \\
 & & & & \rho^{N/2-1} & & & & \\
 & & & & & 1 & & & \\
 & & & & & & \rho^{-N/2+1} & & \\
 & & & & & & & \ddots & \\
 & & & & & & & & \rho^{-2} \\
 & & & & & & & & & \rho^{-1}
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ a_{N/2-1} \\ a_{\pm N/2} \\ a_{-N/2+1} \\ \cdot \\ \cdot \\ a_{-1} \end{bmatrix}.
$$

Algebraic elimination of the a-vector in (4) and (5) gives a relation between the vectors $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$. After multiplying the different rows by suitable powers of ρ to make the result more symmetric, we get

$$
(6)\quad
\begin{bmatrix}
1 & & & & & & & & \\
 & \rho^{-1/2} & & & & & & & \\
 & & \rho^{-2/2} & & & & & & \\
 & & & \ddots\, \rho^{-(N/2-1)/2} & & & & & \\
 & & & & 1 & & & & \\
 & & & & & \rho^{(n/2-1)/2} & & & \\
 & & & & & & \ddots\, \rho^{2/2} & & \\
 & & & & & & & \rho^{1/2}
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 & \dots & 1 \\
1 & \omega^{-1} & \omega^{-2} & & \cdot \\
1 & \omega^{-2} & \omega^{-4} & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
1 & \omega^{-N+1} & \cdot & \cdot & \cdot
\end{bmatrix}
\begin{bmatrix} \zeta_0 \\ \zeta_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \zeta_{N-1} \end{bmatrix}
$$

$$
-
\begin{bmatrix}
1 & & & & & & & & \\
 & \rho^{1/2} & & & & & & & \\
 & & \rho^{2/2} & & & & & & \\
 & & & \ddots\, \rho^{(N/2-1)/2} & & & & & \\
 & & & & 1 & & & & \\
 & & & & & \rho^{-(N/2-1)/2} & & & \\
 & & & & & & \ddots\, \rho^{-2/2} & & \\
 & & & & & & & \rho^{-1/2}
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 & \dots & 1 \\
1 & \omega^{-1} & \omega^{-2} & & \cdot \\
1 & \omega^{-2} & \omega^{-4} & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
\cdot & & & & \cdot \\
1 & \omega^{-N+1} & \cdot & \cdot & \cdot
\end{bmatrix}
\begin{bmatrix} \xi_0 \\ \xi_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \xi_{N-1} \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}.
$$

3. Numerical method. Our aim is to construct an iteration where approximations $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$ for the true point positions on the boundaries, gradually improve in accuracy. This is achieved by repeatedly performing the following three steps:

1. From $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$ we find an approximate value ρ for the radius of the inner circle.
2. Substituting $\boldsymbol{\xi}$, $\boldsymbol{\zeta}$ and ρ into (6) gives a residual vector $\mathbf{r}$ instead of zero as a right hand side. From this vector, changes to $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$ are found which move the approximate mapping points ξ_k and ζ_k away from the boundaries but still leaves the L_2-norm of the errors in the point positions invariant.

3. The points ξ_k and ζ_k are returned to respective boundary. This step reduces the error norm.

Repetition of these three steps will in general give convergence to the desired mapping. One way to implement these three steps will now be described in detail.

1. Ideally, the vector $\mathbf{r}$ should be zero. If this were the case, any row in (6) (apart from the two not including ρ) could be used to find ρ. Let $\mathbf{b}=W^{-1}\boldsymbol{\xi}$ and $\mathbf{c}=W^{-1}\boldsymbol{\zeta}$ denote the products of the Fourier matrix with $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$ in (6). The approximation

$$\rho=\frac{|c_1|+|b_{N-1}|}{|b_1|+|c_{N-1}|} \tag{7}$$

uses a combination of the second and the last rows.

2. Evaluate $\mathbf{r}$ by substituting $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$ into the left-hand side of (6) and form the following two vectors of length $N/2$:

$$[\mathbf{r}_\xi]=\begin{bmatrix}1 &&&& \\ & \rho^{1/2} &&& \\ && \rho^{2/2} && \\ &&& \ddots & \\ &&&& \rho^{(N/2-1)/2}\end{bmatrix}\begin{bmatrix}r_0\\ r_{N-1}\\ r_{N-2}\\ \vdots \\ r_{N/2+2}\\ r_{N/2+1}\end{bmatrix} \tag{8}$$

and

$$[\mathbf{r}_\zeta]=\begin{bmatrix}1 &&&& \\ & \rho^{(N/2-1)/2} &&& \\ && \ddots && \\ &&& \rho^{2/2} & \\ &&&& \rho^{1/2}\end{bmatrix}\begin{bmatrix}r_{N/2}\\ r_{N/2-1}\\ r_{N/2-2}\\ \vdots \\ r_2\\ r_1\end{bmatrix}. \tag{9}$$

Denote by $W_{N/2}$ the discrete Fourier transform matrix of size $N/2$ with elements $w_{kl}=\omega^{2(k-1)(l-1)}$ and by D a diagonal matrix with elements $d_{kk}=\omega^{-(k-1)}$. Further, let $\Delta\boldsymbol{\xi}_{\text{even}}$ be a vector of length $N/2$ containing changes to $\xi_0, \xi_2, \xi_4, \cdots, \xi_{N-2}$ and use similar notation for $\Delta\boldsymbol{\xi}_{\text{odd}}$ etc. We use

$$\begin{aligned}\Delta\boldsymbol{\xi}_{\text{even}}&=W_{N/2}^{-1}\mathbf{r}_\xi, & \Delta\boldsymbol{\zeta}_{\text{even}}&=-W_{N/2}^{-1}\mathbf{r}_\zeta,\\ \Delta\boldsymbol{\xi}_{\text{odd}}&=W_{N/2}^{-1}D\mathbf{r}_\xi, & \Delta\boldsymbol{\zeta}_{\text{odd}}&=W_{N/2}^{-1}D\mathbf{r}_\zeta,\end{aligned} \tag{10}$$

The justification for these formulas will be presented in the following chapter on convergence properties.

3. The changed points are moved back to the boundary curves. If a boundary curve is given in the form $f(x, y)=0$, a point x_0, y_0 off the curve is approximately brought back by the steps

$$d=\frac{f}{(f_x^2+f_y^2)}, \quad x_1=x_0-df_x, \quad y_1=y_0-df_y. \tag{11}$$

Repetition of this correction procedure gives quadratic convergence to a point on $f(x, y)=0$. However, we apply it only once.

Figure 3 illustrates graphically the steps 2 and 3 in this iteration.

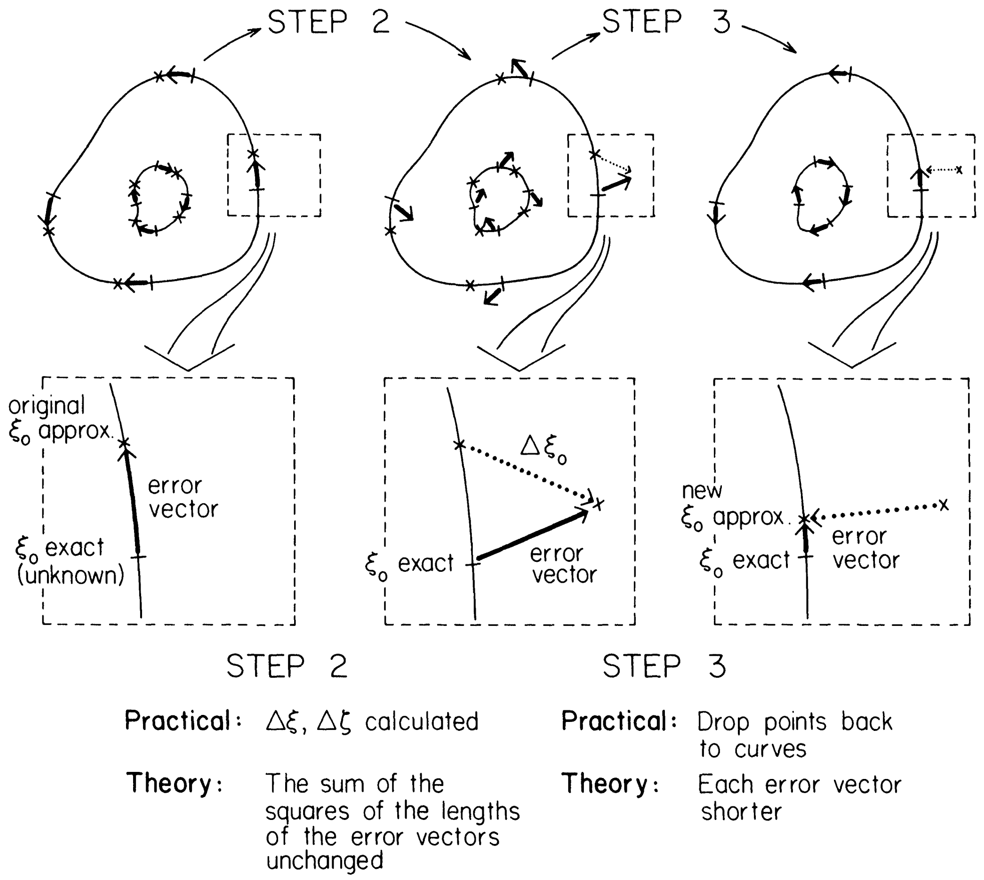

STEP 2
STEP 3
original
ξ_0 approx.
error
vector
ξ_0 exact
(unknown)
$\Delta\xi_0$
ξ_0 exact
error
vector
new
ξ_0 approx.
ξ_0 exact
error
vector
STEP 2
STEP 3
Practical: $\Delta\xi$, $\Delta\zeta$ calculated
Theory: The sum of the squares of the lengths of the error vectors unchanged
Practical: Drop points back to curves
Theory: Each error vector shorter

FIG. 3. *Graphical illustration of steps* 2 *and* 3 *in the iteration procedure.*

4. Convergence properties. Assume that the boundary curves are smooth and that a sufficient number of points are used in order to well resolve the boundaries. Assume further that all points lie in correct mapping positions apart from one point ξ_0 which is perturbed tangentially to $\xi_0+\varepsilon_0$. Substitution in the right hand side of (6) gives a residual vector $\mathbf{r}$. From (8) and (9) follow

$$(12)\qquad [\mathbf{r}_\xi]=-\varepsilon_0\begin{bmatrix}1\\1\\1\\\vdots\\1\end{bmatrix}$$

and

$$(13)\qquad [\mathbf{r}_\zeta]=-\varepsilon_0\begin{bmatrix}1\\\rho^{N/2-1}\\\vdots\\\rho^2\\\rho^1\end{bmatrix}.$$

Straightforward algebra based on (10) gives

$$\Delta \boldsymbol{\xi}_{\text{even}} = -\varepsilon_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{14}$$

and

$$\Delta \boldsymbol{\xi}_{\text{odd}} = \frac{2\varepsilon_0}{N} \left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix} - i \begin{bmatrix} \cot(\pi/N) \\ \cot(3\pi/N) \\ \cot(5\pi/N) \\ \vdots \\ -\cot(5\pi/N) \\ -\cot(3\pi/N) \\ -\cot(\pi/N) \end{bmatrix} \right\}. \tag{15}$$

Since

$$\|\Delta \boldsymbol{\xi}_{\text{even}}\|^2 = \|\Delta \boldsymbol{\xi}_{\text{odd}}\|^2 = \varepsilon_0^2 \tag{16}$$

while

$$\|\Delta \boldsymbol{\zeta}_{\text{even}}\|^2 = \|\Delta \boldsymbol{\zeta}_{\text{odd}}\|^2 < \frac{2\varepsilon_0^2}{N(1-\rho^2)} = \varepsilon_0^2 \cdot O\left(\frac{1}{N}\right), \tag{17}$$

we assume N to be sufficiently large that we are justified in ignoring $\Delta \boldsymbol{\zeta}_{\text{even}}$ and $\Delta \boldsymbol{\zeta}_{\text{odd}}$ in comparison with $\Delta \boldsymbol{\xi}_{\text{even}}$ and $\Delta \boldsymbol{\xi}_{\text{odd}}$. This simplification is necessary to obtain the invariance of the error norm during step 2 of the iteration procedure.

The changes in the point positions are negligible for the inner points, provide an exact correction to the original perturbation in ξ_0 and introduce new errors according to (15) at all the odd numbered outer points. We note that these errors are largest at points near ξ_0. Their directions there are almost normal to the curve.

If not just ξ_0 was perturbed by ε_0 but all even numbered points $\boldsymbol{\xi}_{\text{even}}$ were perturbed by $\boldsymbol{\varepsilon}_{\text{even}}$, we are similarly left with errors only in $\boldsymbol{\xi}_{\text{odd}}$ and they satisfy

$$\boldsymbol{\varepsilon}_{\text{odd}} = Q \boldsymbol{\varepsilon}_{\text{even}} \tag{18}$$

where Q is a unitary matrix with elements

$$Q_{kl} = \frac{2}{N}[1 - i \cot \pi(2(k-l)+1)/N]. \tag{19}$$

The situation is similar for initial perturbations in $\boldsymbol{\xi}_{\text{odd}}$, $\boldsymbol{\zeta}_{\text{even}}$ and $\boldsymbol{\zeta}_{\text{odd}}$. Since equations (6) and (8) to (10) are linear in $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$, any arbitrary initial guess for these vectors can be treated as a superposition of the four simpler cases with only one of the vectors perturbed from the correct values. From the fact that Q is unitary follows the key property of step 2, i.e. the L_2-norm of all the positional errors has been left unchanged. When the points are dropped back to the curves in step 3, this norm is decreased.

There are only two ways in which convergence has been observed to fail (for close initial guesses):

(i) Every one of the new errors after step 2 happens to fall along the boundary. The step of bringing the points back to the curve, which normally reduces the error norm, will have no effect.

(ii) There may exist errors which do not affect the residual vector $\mathbf{r}$.

The simplest possible mapping, the identity mapping from an annulus to itself, illustrates both these possibilities. Consider a perturbation on the outer boundary of ξ_{even} to $\xi_{\text{even}} \cdot (1+i\delta)$ and ξ_{odd} to $\xi_{\text{odd}} \cdot (1-i\delta)$ and similarly on the inner boundary with factors $(1+i\varepsilon)$ and $(1-i\varepsilon)$ resp. Equation (6) is affected only in line $N/2+2$ in that $\mathbf{r}_{N/2+1}$ becomes (approximately) $-N\rho^{(-N/2+1)/2} \cdot i\delta$ instead of 0. The last component of the vector $\mathbf{r}_\xi$ is changed from 0 to $-N\delta i$ and $\mathbf{r}_\zeta$ is left unaffected. The corrections, according to (10), reverse the direction of the perturbation on the outer boundary and leave the inner boundary points unchanged. In the present case of the identity mapping, a single application of the following two corrective procedures will remove these errors completely. We have found no case where repeated use has failed to give convergence.

(i) Oscillatory error patterns on the outer boundary change sign every iteration. Without this problem, the approximations for ρ at successive iterations change in general monotonically. When the mapping has converged so far that the oscillatory errors dominate, the approximations for ρ start to oscillate. Whenever this is observed, we start the following iteration not with ξ, ζ from the last iteration but with the average of ξ, ζ from the last two iterations. In typical test cases, this step is invoked about once every 5–10 iterations. The cost of this averaging is negligible.

(ii) Oscillatory error patterns on the inner boundary may remain invariant. When $W^{-1}\zeta$ is formed in (6) to calculate the residual, the sum of the magnitudes of components with indices $N/2+1$ and $N/2+2$ (which may have remained large) is compared against components $N/2-1$ and $N/2-2$ (for which the method works fine). If the ratio of the sums exceeds a given threshold, say 30, the following step is inserted: Form

$$(20) \qquad [W]\begin{bmatrix} w^{1/2} & & & & & & \\ & w^{2/2} & & & & & \\ & & w^{3/2} & & & & \\ & & & \cdot & & & \\ & & & & \cdot & & \\ & & & & & w^{-2/2} & \\ & & & & & & w^{-1/2} \end{bmatrix}[W^{-1}\zeta]$$

which corresponds to a rotation of the mapping half the angle that separates adjacent gridpoints. Bringing the points back to the curve cancels the oscillatory errors that were present. This procedure is repeated once more to rotate the points back again. In typical test cases, this step got invoked once about every 10–20 iterations. The added cost of 4 FFTs of size N is therefore negligible.

5. Test results.

5. Test results. Two test cases for the mapping method are described below. In each of them, a one parameter family of successively altered regions is mapped. Theodorsen's method converges for some parameter values in the first test case. The convergence rates were then compared. The operation count per iteration for the two methods is very similar. With N points on each boundary, one iteration of the present method requires

2 complex FFT over N points.
4 complex FFT over $N/2$ points.
overhead dominated by bringing the points back to the curves and occasional rotations of the inner boundary to remove oscillations.

Theodorsen's method requires

6 real FFT over N points.

overhead dominated by evaluating polar coordinate formulas for the points and N logarithms for trapezoidal rule approximation of a new radius.

It is possible that the linear convergence rate of both of these methods can be improved upon by suitable acceleration techniques. In the case of Theodorsen's method for simply connected regions, this has been investigated by Gutknecht [5], [6] who proposes a relaxation procedure. However, Hübner [8] notes that this acceleration fails for regions which are not symmetric. We have not considered any acceleration possibilities in this work.

Convergence progressed in all cases until one of the following two limits were reached.

(i) Machine (roundoff) accuracy.

(ii) Truncation accuracy, i.e., size of first ignored Laurent coefficients.

The numerical results in this section were obtained using 64 bit floating point (48 bit mantissa) on the Control Data Cyber 205 computer at the Cybernet Service Center in Arden Hills, Minnesota. The algorithm can be completely vectorized. One iteration, including all overhead, takes approximately 1.1 ms with $N=128$ and 46 ms with $N=4096$ on a 2-pipe version of the Cyber 205.

Test case 1. The outer curve is a unit circle, the inner an ellipse with focii at 0 and a where a satisfies $0 \leqq a < 1$. The size is such that the sum of the two radii equals one for any boundary point. The equations for the boundaries are:

$$\text{OUTER:}\quad x^2+y^2-1=0, \qquad \text{INNER:}\quad 4(x^2+y^2)-(a^2-2ax-1)^2=0. \tag{21}$$

Figure 4 shows the mappings for some different values of a.

Both the present method and Theodorsen's method converge linearly with a rate that is independent of N. Table 2 gives these rates. Tables 3 and 4 show in more detail how the obtained accuracy varies as a function of N for the two methods. We consider the cases $a=.5$ and $a=.7$ and iterate both methods until convergence with $N=16$, 32 and 64. The positional error of every individual boundary point is measured. The largest of these is tabulated and for the present method is seen to be very closely linked to the size of the first omitted Fourier coefficients. Theodorsen's method proves to be less accurate for same values of N. The last columns of Table 2 are based on this maximal boundary position test. Both methods give values for ρ which are far more accurate than the boundary points.

One further comparison between the methods has been performed. The issue is how accurate an initial guess has to be to give convergence. The solutions for $a=0.,.5,.7,.9$ and $.99$ were taken as initial guesses and the point positions were transferred to new boundaries by keeping their polar coordinate angles unchanged. (Since the outer boundary was not changed, these points were not moved.) The question we studied was how far up from $a=0.,.5$ etc. one could go in one single continuation step. Table 5 shows the result. For Theodorsen's method, the maximal $a=.71$ could be reached directly from any of the initial guesses. The present method performed almost equally well. The solution at $a=0.$ was accurate enough for $a=.64$ while $a=.5$ allowed a single step to $a=.83$, well beyond the range of Theodorsen's method. As might be expected, the steps had to be taken smaller for the extremely deformed final cases.

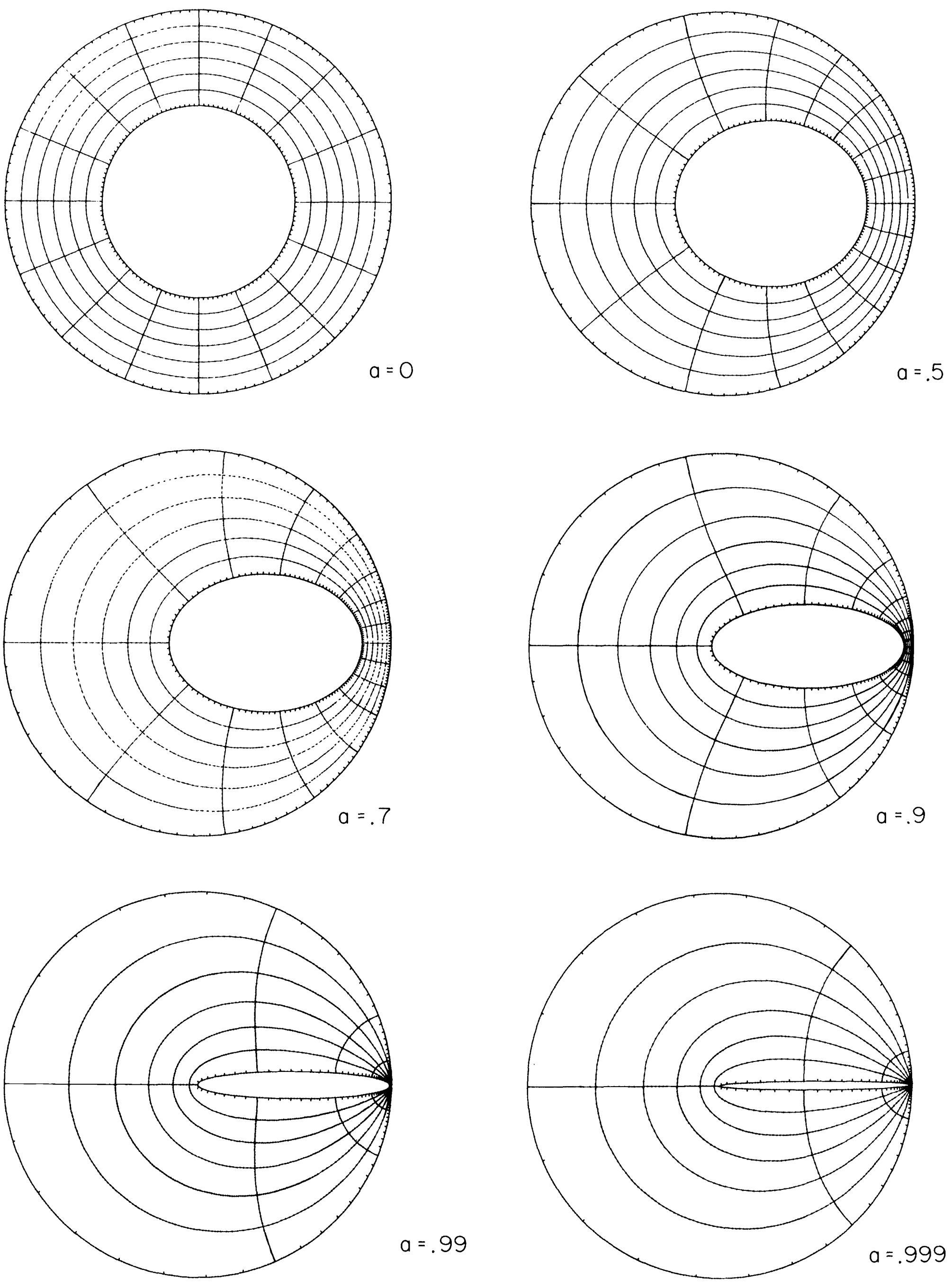


FIG. 4. *The mappings for different values of a in test case* 1.

TABLE 2
Summary of test case 1.

a	ρ	Geometric convergence rate		Number of points N required for accuracy	
		Present	Theodorsen	10^{-4}	10^{-8}
0.	.5000	.50	0.0	—	—
.5	.5118	.55	.56	32	32
.7	.5273	.67	.94	32	64
.9	.5641	.87	Diverges	64	64
.99	.6334	.98	for	64	128
.999	.6859	.997	$a > .71$	128	256

TABLE 3
Accuracy of mapping as function of N in test case 1, $a = 0.5$.

Number of points on each boundary N	First omitted Fourier coefficient		Largest error in any point position (measured in arc length)		Error in calculated value of radius	
	component number	size of coefficient	Present	Theodorsen	Present	Theodorsen
16	8	$.1\times10^{-3}$	$.2\times10^{-3}$	$.2\times10^{-3}$	$.1\times10^{-7}$	$.2\times10^{-9}$
32	16	$.8\times10^{-9}$	$.4\times10^{-9}$	$.8\times10^{-7}$	$<10^{-10}$	$<10^{-10}$
64	32	$<10^{-10}$	$<10^{-10}$	$<10^{-10}$	$<10^{-10}$	$<10^{-10}$

TABLE 4
Accuracy of mapping as function of N in test case 1, $a = 0.7$.

Number of points on each boundary N	First omitted Fourier coefficient		Largest error in any point position (measured in arc length)		Error in calculated value of radius	
	component number	size of coefficient	Present	Theodorsen	Present	Theodorsen
16	8	$.1\times10^{-2}$	$.2\times10^{-2}$	$.6\times10^{-2}$	$.3\times10^{-5}$	$.6\times10^{-5}$
32	16	$.2\times10^{-6}$	$.1\times10^{-6}$	$.5\times10^{-4}$	$<10^{-10}$	$.5\times10^{-9}$
64	32	$<10^{-10}$	$<10^{-10}$	$.7\times10^{-8}$	$<10^{-10}$	$<10^{-10}$

TABLE 5
Longest single continuation steps that can be taken from different values of a in test case 1.

a	Present method	Theodorsen's method
0.	.64	.71
.5	.83	.71
.7	.89	.71
.9	.954	—
.99	.992	—

Test case 2. We map a series of strips of increasing width. With d small, an approximate equation for a curve at a distance d from $f(x, y) = 0$ is $g(x, y) = 0$ where

$$g(x, y) = f(x, y) - d^*\sqrt{f_x^2 + f_y^2}. \tag{22}$$

The outer boundary is $f(x, y) = 0$ with

$$f(x, y) = (x^2 + (y - .5)^2)*(1 - x^2 - y^2) - 0.1. \tag{23}$$

The inner boundaries correspond to $d = .05$, $.1$ and $.2$. In the initial guess for $d = .05$ the points ξ_k were equidistantly distributed along the boundary. Matching ζ_k were obtained from (11) and (23). The case $d = .2$ was illustrated in Fig. 2. Table 6 summarizes this test case. This test case, which is typical for the application of finding boundary fitting local grids, is well outside the range of Theodorsen's method. We note that significantly more points N are required to resolve this problem than was called for in test case 1.

TABLE 6
Summary of test case 2.

d	ρ	Geometric convergence rate	Number of points N required for accuracy 10^{-4}	10^{-8}
.05	.9521	.97	128	512
.1	.9096	.93	256	1024
.2	.8356	.89	512	2048

REFERENCES

[1] B. FORNBERG, *A numerical method for conformal mappings*, this Journal, 1 (1980), pp. 386–400.

[2] D. GAIER, *Konstruktive Methoden der Konformen Abbildung*, Springer Tracts in Natural Philosophy, 3, Springer, Berlin, 1964.

[3] I. E. GARRICK, *Potential flow about arbitrary biplane wing sections*, Rept. 542, NACA, 1936.

[4] J. K. HAYES, D. K. KAHANER AND R. KELLNER, *An improved method for numerical conformal mapping*, Math. Comp. 26 (1972), pp. 327–334.

[5] M. GUTKNECHT, *Solving Theodorsen's integral equation for conformal maps with the fast Fourier transform and various nonlinear iterative methods*, Numer. Math., 36 (1981), pp. 405–429.

[6] ———, *Numerical experiments on solving Theodorsen's integral equation for conformal maps with the fast Fourier transform and various nonlinear iterative methods*, this Journal, 4 (1983), pp. 1–30.

[7] P. HENRICI, *Fast Fourier methods in computational complex analysis*, SIAM Rev., 21 (1979), pp. 481–527.

[8] O. HÜBNER, *Über die Anzahl der Lösungen der diskreten Theodorsen–Gleichung*, Numer. Math., 39 (1982), pp. 195–204.

[9] D. C. IVES, *A modern look at conformal mapping including multiply connected regions*, AIAA J., 14 (1976), pp. 1006–1011.

[10] B. KREISS, *Construction of a curvilinear grid*, this Journal, 4 (1983), pp. 270–279.

[11] D. I. MEIRON, S. A. ORSZAG AND M. ISRAELI, *Applications of numerical conformal mapping*, J. Comput. Phys., 40 (1981), pp. 345–360.

[12] R. MENIKOFF AND C. ZEMACH, *An integral equation method in conformal mapping*, J. Comput. Phys., 36 (1980), pp. 366–410.

[13] N. PAPAMICHAEL AND C. A. KOKKINOS, *Two numerical methods for the conformal mapping of simply-connected domains*, Computer Meth. Appl. Mech. and Eng., 28 (1981), pp. 285–307.

[14] ———, *The use of singular functions for the approximative conformal mapping of doubly-connected domains*, Brunel University Technical Report TR/01/82, 1982.

[15] L. REICHEL, *A fast method for solving certain integral equations of the first kind with application to conformal mappings*, Technical Report, Royal Institute of Technology, Stockholm, TRITA-NA-8117, 1981.

[16] G. STARIUS, *Composite mesh difference methods for elliptic boundary value problems*, Numer. Math., 28 (1977), pp. 243–258.

[17] G. T. SYMM, *An integral equation method in conformal mapping*, Numer. Math., 9 (1966), pp. 250–258.

[18] ———, *Conformal mapping of doubly-connected domains*, Numer. Math., 13 (1969), pp. 448–457.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

A MATHEMATICAL PROGRAMMING APPROACH TO EDITING OF CONTINUOUS SURVEY DATA*

PATRICK G. McKEOWN†

Abstract. Since data from surveys often contain errors, it is desirable to attempt to detect these errors. Clearly, not all errors can be detected: What can be attempted is to detect those errors that result in inconsistencies. The detection process is referred to as *editing* of the answer fields of the survey record. In this paper, we present a mathematical programming approach to data editing whose objective is to minimize the weighted number of answer fields that are suggested to be in error relative to a set of internal consistency conditions. This procedure is then applied to test data from the Annual Survey of Manufacturers and computational results are presented.

Key words. mathematical programming, data editing

1. Introduction. A key step in the processing of large-scale data is the editing and imputation of the data. By *editing*, we refer to the process of checking answer fields to detect inconsistencies. The fields suggested to be in error may then be changed to an acceptable value. The process of changing responses is referred to as *imputation*. Survey data can be broadly classified as either coded or continuous. In coded or qualitative data there are only a few possible pre-specified responses to a given question. Continuous data, on the other hand, may have almost any values as responses. Due to the difference in the type of data considered, it is necessary to approach the editing and imputation of the coded and continuous problems differently. In this paper we will consider only continuous data.

Fellegi and Holt [1] presented a procedure for editing coded data. In that paper they based their work on the assumption that one would wish to alter the original observations to the least possible extent. (They used a procedure to find all edit conditions which are implied by specific edit conditions. All of the resulting edit conditions are then used to solve the edit problem. The work has been refined recently by Liepins [7].) In an earlier paper on economic data, Freund and Hartley [2] used a least squares approach to minimize a combination of the weighted sum of squared differences between original and corrected data and the weighted sum of squares of deviations from the edit conditions. In that paper, the edits were made up of internal consistency conditions in the form of linear inequalities that would have to be satisfied for an edit condition to be met.

In this paper our objective will be to minimize the weighted number of fields that would have to be changed through imputation in order for the resulting answers to meet a set of internal consistency conditions which are expressed as linear inequalities. The weights represent confidence in the value in each field. In this way, we have attempted to combine key portions of the Fellegi–Holt and Freund–Hartley approaches. However, there are important differences in each case. In the first case, we are working with continuous data while Fellegi and Holt were editing coded data. In the second case, we require that *all* consistency conditions be satisfied while Freund and Hartley allow for consistency conditions to go unsatisfied so long as the sums of squares are minimized. In all cases, we are considering *deterministic errors*, that is, errors which can be determined with certainty.

* Received by the editors November 19, 1981, and in final revised form July 8, 1983. This research was partially sponsored by Bureau of the Census, U.S. Department of Commerce.

† College of Business Administration, University of Georgia, Athens, Georgia 30602.

Before continuing, two comments should be made about our objective function. First, given certain assumptions (stated below), minimizing the weighted number of fields changed can be thought of as a surrogate for the objective: Maximize the product of the probabilities that a changed field is in error and that an unchanged field is correct [7]. Since such an objective would be extremely difficult to work with, the objective of minimizing the number of fields which are changed is used as a replacement. This substitution has the desired effect since it tends to reduce the probability of changing a correct field.

If a field has a high probability of being correct, we would assign it a relatively large weight. If a field has a low probability of being correct, we would assign it a relatively low weight. Since we are minimizing the sum of the weights, an optimal solution to the problem would seek to avoid changing fields with high weights in favor of changing fields with low weights. In so doing, fields with a high probability of being correct have less chance of being changed.

Second, by using the minimization of the weighted number of fields that are changed as our objective in an editing procedure, we are making an important assumption. We are assuming that the occurrence of an error in a field is unrelated to the magnitude of the value in the field. In the context of surveys, these assumptions may not be unrealistic when one considers the many transitions that the data must go through between the respondent and final tabulation. Errors may occur at any stage, i.e., respondent calculation in computing values for responses, respondent error in entering response, typographical error by respondent, keypunching error or error in transfer of data from computer output. Only in the initial case would the assumption be obviously unrealistic. For this reason, the objective of retaining as much of the original data as possible appears to be a reasonable approach to the editing problem.

It is important here to state that our objective will be to present a procedure to *edit* continuous survey data. While this procedure will demonstrate that imputations do *exist* for the fields suggested to be in error, this is only a side effect of the editing procedure. It is our contention that editing and imputation are separate and distinct problems, each requiring a different solution methodology. If we can solve the editing problem, then other methods can be used to impute values for the fields suggested to be in error.

In the next section we will present a mathematical programming formulation of the editing problem for continuous data and discuss a solution method for that problem. In § 3 we present an example, while § 4 discusses the application of our approach to sample data of the form generated by the Annual Survey of Manufacturers (ASM). Finally, we will state some conclusions and discuss refinements of this work.

2. A mathematical programming approach. In a recent paper, Sande [11] suggested the following formulation of the editing problem for continuous data:

$$\text{(1)} \qquad \text{Minimize:} \quad \sum_{j=1}^{k} w_j\delta(y_j)+\sum_{j=1}^{k} w_j\delta(z_j),$$

$$\text{(2)} \qquad \text{subject to:} \quad A(x^0+y-z)\leqq b,$$

$$\text{(3)} \qquad x^0+y-z\geqq 0,$$

$$\text{(4)} \qquad y\geqq 0,$$

$$\text{(5)} \qquad z\geqq 0,$$

$$\text{(6)} \qquad y^Tz=0,$$

where

(7) $$\delta(y_j)=\begin{cases}1 & \text{if } y_j>0,\\ 0 & \text{otherwise,}\end{cases} \qquad j=1,\cdots,k,$$

(8) $$\delta(z_j)=\begin{cases}1 & \text{if } z_j>0,\\ 0 & \text{otherwise,}\end{cases} \qquad j=1,\cdots,k,$$

and

k = number of fields to be edited,

x^0 = a given record of continuous survey data,

w_j = weights on each field of the record denoting the degree of confidence in the value in that field.

A and b are $m\times k$ and $m\times 1$ matrices, respectively, which represent the m consistency conditions. If $Ax^0 \leqq b$, the record x^0 will satisfy the consistency conditions. This model assumes that the consistency conditions or edits can be written as linear inequalities. The values of y_j and z_j indicate one possible set of minimum changes required to produce a consistent record. In practice, $\bar{x}=x^0+y-z$ will be a consistent record on the boundary of the acceptance region with y and z being possible changes. If y_j is positive, this indicates that an increase of an amount greater than or equal to y_j in the jth field of x^0 is needed for consistency. Similarly, if z_j is positive, the jth field of x^0 must decrease by an amount greater than or equal to z_j in order for the record to become consistent. The weights, w_j, serve to make changes more costly in those fields which the user believes to be more likely to be correct. The weights may be determined subjectively or through past data on the survey being edited.

The above formulation ((1)–(8)) defines a problem with an objective of minimizing the weighted cardinality of the fields in which changes are required in order to make the record x^0 consistent. This is done by use of the deviational variables, y and z. If a given deviational variable must be made positive in order for the sum x^0+y-z to be consistent, i.e., $A(x^0+y-z)\leqq b$, then the cost w_j will be incurred because the corresponding $\delta(y_j)$ or $\delta(z_j)$ will be equal to 1. This formulation is a form of mathematical programming known as a *fixed charge problem* [5].

The vector x^0 is the initial record of data to be edited, so to actually solve the problem (1)–(8), we must transform the inequalities as follows to form problem (P_0):

(9) Minimize: $$\sum_{j=1}^{k} w_j\delta(y_j)+\sum_{j=1}^{k} w_j\delta(z_j)$$

(10) subject to: $$A(y-z)\leqq b-Ax^0$$

(11) (P_0) $$y_j-z_j\geqq x_j^0, \quad j=1,\cdots,k,$$

(12) $$y_j\geqq 0, \quad j=1,\cdots,k,$$

(13) $$z_j\geqq 0, \quad j=1,\cdots,k,$$

where

(14) $$\delta(y_j)=\begin{cases}1 & \text{if } y_j>0,\\ 0 & \text{otherwise,}\end{cases} \qquad j=1,\cdots,k,$$

(15) $$\delta(z_j)=\begin{cases}1 & \text{if } z_j>0,\\ 0 & \text{otherwise,}\end{cases} \qquad j=1,\cdots,k.$$

Note that we have left the complementarity condition, $y^T z = 0$, out of this final formulation. This condition is unnecessary in a fixed charge formulation because both y_j and z_j will not be nonzero at the same time for a given value of j in a minimum value solution. This is true because if both y_j and z_j were nonzero, the objective value could be reduced while still satisfying the constraints by forcing either y_j or z_j to be zero and making a corresponding increase in the other. Note that when either y_j or z_j is nonzero, increasing its value does *not* change the value of the objective function.

Another important result about (P_0) is that constraint (11), i.e., $y_j - z_j \geqq -x_j^0$, can be handled implicitly as an upper or lower bound. To see this, note that if $x_j^0 \geqq 0$, then (11) can be replaced by

(11a) $$z_j \leqq x_j^0.$$

Similarly, if $x_j^0 < 0$, then (11) can be replaced by

(11b) $$y_j \geqq x_j^0.$$

This latter situation can occur when a nonresponse is replaced by -1 [13], to insure that missing data is recognized as an error.

Because our solution procedure uses linear programming to search for an optimal solution, replacing (11) by (11a) or (11b) is very useful since upper or lower bounds can be easily handled in the solution procedure without explicitly carrying along the constraints. This has a dramatic effect on the efficiency of the procedure, since the number of constraints to be explicitly satisfied is reduced by the number of fields. The number of constraints tends to be a key variable in solution speed of linear programming-based solution procedures such as the one used here.

The solution to P_0 provides a solution to the editing problem, that is, the fields in the record x^0 that could be causing the inconsistency are determined. A side effect is that the determination of the y_j and z_j values for each field demonstrate that imputations that will yield a consistent record do exist.

Even though the formulation of the fixed charge probem (9)–(15) is very similar in form to that of the well-known linear programming problem, the approach to finding a solution is very different. For linear programming problems, one can use the simplex algorithm. Unfortunately, no simplex-like procedure has been developed for fixed charge problems because the objective function for fixed charge problems is concave rather than linear. It is a well known result that the optimal solution to the minimization problem with a concave objective function and linear constraints will occur at an extreme point [5]. However, it is also true that it is not easy to find *which* extreme point is the global minimum due to the existence of local optima [10].

Given that the fixed charge problem is not easy to solve, it becomes necessary to use a search procedure to determine an optimal solution. We have chosen the branch and bound procedure for this. Branch and bound methods work by subdividing the feasible region into subproblems (branching) and deleting subproblems from consideration if it can be shown that an optimal solution will not be found in that subproblem (bounding). For an excellent discussion of branch and bound, see Geoffrion and Marsten [3].

A branch and bound procedure for fixed charge problems, which has been previously developed by the author [8], was extended to solve the fixed charge problem (9)–(15) with implicit upper and lower bounds. This procedure is well suited to solve this type of problem which has only fixed charges, in that a bound on the fixed charge component is computed separately. Most solution procedures for fixed charge problems depend upon the continuous variable cost to be efficient (for example, see [9]).

This branch and bound procedure solves (9)–(15) with implicit upper and lower bounds, by partitioning the problem into a linear programming problem and a set covering problem. The linear programming problem (P_{LP}) is:

Find a feasible solution to:

$$(P_{LP})\qquad \begin{aligned} &A(y-z)\leqq b-Ax^0,\\ &y-z\geqq -x^0,\\ &y\geqq 0,\\ &z\geqq 0,\end{aligned}$$

and the set covering problem (P_{SC}) is:

$$(P_{SC})\qquad \begin{aligned} &\text{Minimize:} && \sum_{j=1}^{k} w_j u_j,\\ &\text{subject to:} && \sum_{j=1}^{k} \Delta_{ij} u_j \geqq 1 \quad \text{for } i\in\Gamma,\\ & && u_j \geqq 1 \quad \text{for } i\in\phi,\\ & && u_j\in\{0,1\},\end{aligned}$$

where

$$\Delta_{ij}=\begin{cases}1 & \text{if } a_{ij}\neq 0,\\ 0 & \text{if } a_{ij}=0,\end{cases}$$

and

$$\Gamma=\left\{i\,\middle|\, b_i-\sum_{j=1}^{k} a_{ij}x_j^0<0\right\},\qquad \phi=\{i\,|\,x_i^0<0\}.$$

The set covering problem is equivalent to the failed edit matrix used by Liepins [7] in solving discrete coded data editing problems. It is made up of the constraints or bounds of the edit problem that are causing the input record, x^0, to fail the edits.

In the branch and bound procedure, the branching is determined by choosing any y_j or z_j that is basic and nondegenerate to branch upon. If a y_j or z_j is positive, then the corresponding u_j must also be basic in (P_{SC}). Similarly, if y_j or z_j is forced to be zero, then the corresponding u_j must also be zero. The bounding is accomplished in two ways. If forcing a y_j or z_j to zero results in (P_{LP}) being infeasible, then the resulting subproblem is terminated. Or, if the objective value of (P_{SC}) becomes greater than the current incumbent solution when u_j is forced into or out of the basis, the resulting subproblem is terminated. The choice of a y_j or z_j to branch upon is done through the use of penalties as described in [3].

A computer code based upon the procedure described in [8] has been developed and used to solve the editing problem for data supplied by the U.S. Census Bureau. Another extension of the branch and bound procedure allows for the determination of equivalent solutions to the editing problem. These equivalent solutions will be presented in the section on computational results.

3. Example. As an example[1] of the fixed charge formulation for the fields-to-impute problem, consider the following set of consistency or edit conditions:

$$(16)\qquad x_1\geqq 1,$$

[1] This example was first suggested by Sande [12].

(17) $$x_1 \leqq 4,$$

(18) $$x_2 \geqq 1,$$

(19) $$x_2 \leqq 4,$$

(20) $$x_1 + x_2 \leqq 7,$$

(21) $$x_1 + x_2 \geqq 3,$$

(22) $$-x_1 + x_2 \leqq 2,$$

(23) $$x_1 - x_2 \leqq 2.$$

Geometrically, these edit conditions specify that a record which satisfies the edit must lie on or within an octagon in R^2. Figure 1 shows the octagon with a record which fails the edit conditions being marked by an asterisk.

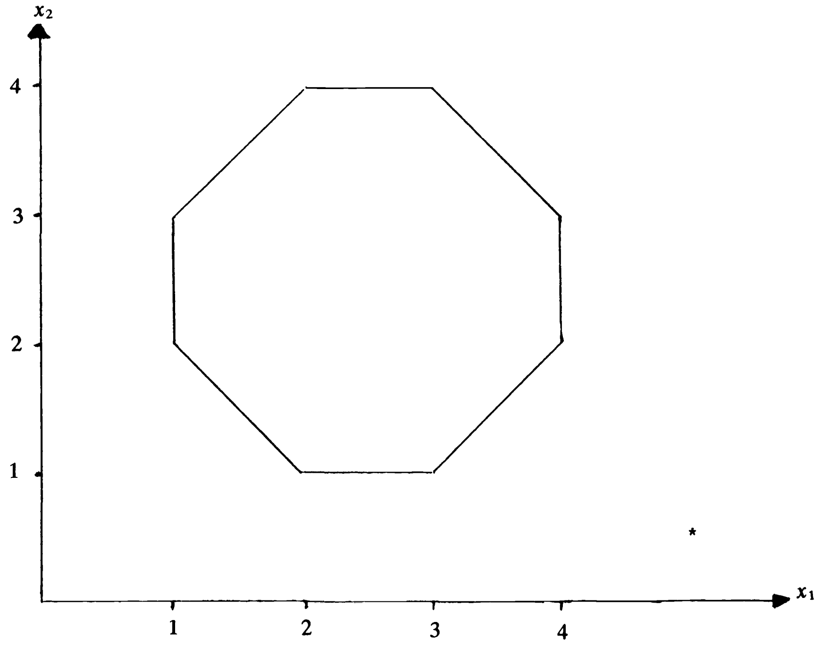


FIG. 1

If we now consider a record, $x^0 = (5, 0.5)$, we find that the product of x^0 and A, Ax^0, is $(5, 5, 0.5, 0.5, 5.5, 5.5, 4.5, 4.5)$. If we then compare these values to the edit conditions (16)–(23), we find that x^0 fails edit conditions (17), (18) and (23). Since the record x^0 fails the edit conditions, we will now set up the fixed charge formulation to find the field(s) that are suggested to be in error and a value(s) that can be imputed to yield a consistent record. When this is done, we obtain the following equally-weighted problem (where the Ax^0 values have been moved to the right-hand side of the inequalities):

(24) Minimize: $$\delta(y_1) + \delta(y_2) + \delta(z_1) + \delta(z_2)$$

(25) subject to: $$y_1 \quad - z_1 \quad \geqq -4,$$

(26) $$y_1 \quad - z_1 \quad \leqq -1^*,$$

$$y_2 - z_2 \geqq 0.5^*, \tag{27}$$

$$y_2 - z_2 \leqq 3.5, \tag{28}$$

$$y_1 + y_2 - z_1 - z_2 \leqq 1.5, \tag{29}$$

$$y_1 + y_2 - z_1 - z_2 \geqq -2.5, \tag{30}$$

$$-y_1 + y_2 + z_1 - z_2 \leqq 6.5, \tag{31}$$

$$y_1 - y_2 - z_1 + z_2 \leqq -2.5^*, \tag{32}$$

$$y_1, \quad y_2, \quad z_1, \quad z_2 \geqq 0. \tag{33}$$

In this formulation we have marked with an asterisk (*) those inequalities which cannot be satisfied by setting $y_1 = y_2 = z_1 = z_2 = 0$. Note that these inequalities correspond to failed edits.

If we solve (24)–(33) as a fixed charge problem, an extremal solution is given by $z_1 = 2$, $z_2 = 0$, $y_1 = 0$, $y_2 = 0.5$ with a cardinality of 2. This implies that both coordinates must be changed to yield a consistent record. In terms of Fig. 1, if we use the values found in the solution process, we move the record x^0 to the boundary of the consistency octagon at (3, 1) by changing the fewest number of fields or coordinates. In this case, *any* point on or in the octagon is an acceptable imputation with the same number of changes.

4. Application to Census Bureau edits. To apply the fixed charge formulation to the editing of economic survey data, it is necessary to derive consistency conditions, $Ax^0 \leqq b$. The Industry Division of the Bureau of the Census provided test data for the Annual Survey of Manufacturers (ASM) for testing purposes. These data are currently edited by the ASM General Statistics Edit [6] which uses ratios between various fields, year-to-year ratios and the requirement that the sum of the parts must equal the whole to determine consistency. This is a very special set of edits since, if the edits requiring that the sum of the parts equal the whole are ignored, the ratio edits plus all implied edits can be treated as a network. The edit problem then becomes one of solving the associated node disconnect problem. Greenberg [4] has discussed this approach and presented a heuristic to solve this problem. However, he presents no results on solving the problem optimally or on solving heuristically the complete editing problem (including the sum of the parts edits).

If x_k is the value in the kth field and x_j is the value in the jth field, then a ratio edit involving these two fields would be written as

$$l_i \leqq \frac{x_k}{x_j} \leqq u_i, \tag{34}$$

where l_i and u_i are lower and upper bounds, respectively, for ith field-to-field consistency condition and $x_j \neq 0$. The single ratio edit may be transformed into linear inequalities as follows:

$$x_k - l_i x_j \geqq 0, \tag{35}$$

and

$$x_k - u_i x_j \leqq 0. \tag{36}$$

For the same two field values, x_j and x_k, a year-to-year ratio edit would be written as

$$(37)\qquad \bar{l}_i \leqq \frac{x_k/x_j}{\bar{x}_k/\bar{x}_j} \leqq \bar{u}_i,$$

where $\bar{x}_k$ = prior year value for x_k (assumed to be known),

$\bar{x}_j$ = prior year value for x_j (assumed to be known),

$\bar{l}_i$ = ith year-to-year lower bound,

$\bar{u}_i$ = ith year-to-year upper bound.

This ratio edit can also be converted into two linear inequalities but this is not necessary. This ratio converts into another ratio involving only x_j and x_k:

$$(38)\qquad (\bar{l}_i)\left(\frac{\bar{x}_k}{\bar{x}_j}\right) \leqq \frac{x_k}{x_j} \leqq \left(\frac{\bar{x}_k}{\bar{x}_j}\right)(\bar{u}_i).$$

Now referring to the ith field-to-field ratio in (36), note that if $u_i > (\bar{u}_i)(\bar{x}_k/\bar{x}_j)$, then we may replace u_i by $(\bar{u}_i)(\bar{x}_k/\bar{x}_j)$ in (36). Similarly, if $l_i > \bar{l}_i(\bar{x}_k/\bar{x}_j)$, then l_i may be replaced by $\bar{l}_i(\bar{x}_k/\bar{x}_j)$ in (35). This combination of field-to-field and year-to-year ratios serves two purposes. First, if there are n field-to-field and n year-to-year ratios, we would need only $2n$ linear inequalities rather than the $4n$ linear inequalities that would be needed without the combination. Secondly, the linear inequalities which result are the $2n$ most restrictive inequalities, so there are no redundant or unnecessary inequalities.

The final set of consistency conditions, those which require the sum of the parts to be equal to the whole, may be stated as:

$$(39)\qquad \sum_{j=1}^{r} x_j = B_i,$$

where r = number of fields which make up the whole,

B_i = the value for the field corresponding to the whole,

x_j = the values of the fields corresponding to the parts.

These edits do not require a conversion since they are already linear inequalities.

The ASM forms come in two main categories, MA-100 and MA-100(s). Both forms request information about industries with the MA-100(s) being used for smaller companies and requiring less detailed information. A single set of ratios is used for both forms. There are 51 field-to-field ratios and 42 year-to-year ratios, which are used in the editing process. We will present results for the MA-100 form since it us used for many more companies than the MA-100(s).

The MA-100 form used in the ASM requests information in twenty *major* fields. A given field is *not* a major field if the only ratio edit in which it is involved also involves a field for which the given field is a part. For example, if $A+B=C$ and the only ratio involving field A is A/C, then A is not a major field and is termed a *minor field.* The objective becomes to change the minimal number of major fields using mathematical programming and then to use other procedures to edit the minor fields since they depend on the major fields. For the MA-100 edit, the twenty major fields and their mnemonics are given in Table 1. We will only consider the editing of the major fields here.

TABLE 1
Major field for MA-100 *edit.*

Field	Quantity edited	Mnemonic
1	production workers	PW
2	other employees	OE
3	total employees	TE
4	production workers' salaries	WW
5	other workers' salaries	OW
6	total wages and salaries	SW
7	legally required labor cost	LE
8	payments to voluntary programs	VP
9	total supplemental labor cost	LC
10	plant hours worked	MH
11	cost of products purchased	CR
12	cost of materials	CM
13	ending inventories	ET
14	value of assets at beginning of year plus capital expenditures during year	TCE
15	value of assets at end of year	TAE
16	rental payments—building	BR
17	rental payments—equipment	MR
18	total rental payments	TR
19	value of resales	VR
20	value of shipments	VS

There are 23 ratio edits and four sum-of-the-parts or balance edits which involve the twenty major fields. The ratio edits, with their upper and lower bounds, are shown in Table 2. The balance edits are:

$$\mathrm{PW}+\mathrm{OE}=\mathrm{TE}, \tag{40}$$

$$\mathrm{BR}+\mathrm{MR}=\mathrm{TR}, \tag{41}$$

$$\mathrm{OW}+\mathrm{WW}=\mathrm{SW}, \tag{42}$$

$$\mathrm{LE}+\mathrm{VP}=\mathrm{LC}. \tag{43}$$

For the MA-100 edits, there are 50 inequality conditions which result from the 23 ratio edits and four balance edits. The conditions plus the bounds of the form $X^0+Y-Z\geqq 0$, result in a fixed charge problem with 40 X and Y variables, 40 0–1 ($\delta(X)$ or $\delta(Y)$) variables, 50 constraints and 20 upper or lower bound conditions.

To test the computer code after it had been developed to handle the problem presented by the MA-100 form, the Industry Division supplied seven MA-100 forms of test data. These data are shown in Table 3. For each record, the prior data and current year data are shown. A dash is shown in those fields where there was no response. These records were then run on the branch-and-bound solution procedure after it was combined with a heuristic procedure developed by Walker [14] for finding good solutions to fixed charge problems.

The results of these computer runs are shown in Tables 4 and 5. Table 4 shows the current record before and after editing. The minimal fields which the procedure determined need to be changed for each record are denoted with two asterisks (**) in Table 4. The statistics for the solution procedure for each record are presented in Table 5. These statistics include: The CPU time to generate the matrix for the fixed charge constraint set from the ratio edits, the balance edits and the current record; the CPU time to solve the two parts of the lower bound via linear programming; the

TABLE 2
Ratio edits for ASM *form* MA-100.

Ratio	Current year (field-to-field)		Year-to-year	
	Lower limit	Upper limit	Lower limit	Upper limit
SW/VS	0.010	0.510	0.251	0.749
SW/TE	3.000	30.000	0.525	1.575
WW/SW	0.250	1.000	0.250	4.000
OW/SW	0.100	1.000	0.100	10.000
CM/VS	0.120	0.850	0.300	1.700
ET/VS	0.010	0.357	0.100	10.000
TAE/VS	0.010	6.000	0.100	10.000
BR/VS	0	0.200	0.050	20.000
MR/VS	0	0.200	0.050	20.000
VR/VS	0	0.750	0	20.000
PW/TE	0.25	1.000	0.506	2.000
OE/TE	0.100	1.000	0.100	10.000
WW/PW	3.000	25.000	0.546	2.000
WW/MH	2.000	15.000	0.574	2.000
OW/OE	3.000	30.000	0.250	4.000
CR/CM	0	1.000	0	20.000
CR/VR	0.250	1.480	0.237	3.000
MH/PW	1.372	2.710	0.475	1.525
TCE/TAE	0.010	0.505	—	—
(SW+CM)/VS	0.100	1.182	—	—
TCE/VS	0	1.000	—	—
LE/SW	0.040	0.700	—	—
VP/SW	0.005	0.400	—	—

CPU time to search for and prove a solution to be optimal via branch and bound; the total CPU time for each record; and the number of branch and bound subproblems encountered in the search procedure. This last statistic is an indicator of the difficulty that the solution procedure encountered in finding and proving an optimal solution.

In looking at Table 4, we observe several interesting characteristics of the fixed charge editing procedure for the test problem. First, the fact that records 1 and 3 have only missing data is confirmed by the fixed charge editing procedure. Secondly, record 2 demonstrates an important capability of the editing procedure. In this record, the edits for field MH are with WW and DW, both of which are missing but the editing procedure still suggested that MH was incorrect. This capability of the fixed charge editing procedure to edit records with both missing and erroneous data is also demonstrated by records 4 and 6. Note that some fields in records 2 and 6 appear to have scaling errors which might be corrected in a pre-editing procedure that checks for scaling. In any case, the fixed charge editing procedure suggests that these fields are in error.

In Table 5, we see that all records except number 6 were solved in less than two seconds and with five or less subproblems. Problem 6 turned out to be somewhat more difficult in terms of CPU time and number of subproblems. One can only conjecture why this record was more difficult than the others but it may have to do with a seeming reversal of the numbers of production workers (PW) and other workers (OW) plus the missing value for the production workers' salaries (WW). These are subtle errors which tend to make difficult the process of proving a given solution to be optimal.

TABLE 3
Prior year (P) and current year (C) test records.

Record number		PW	OE	TE	WW	OW	SW	LE	VP	LC	MH	CR	CM	ET	TCE	TAE	BR	MR	TR	VR	VS
1	P	114	27	141	1,650	429	2,079	162	91	253	212	25	20,912	705	3,450	19,421	50	340	390	38	36,497
	C	—	—	151	—	—	2,705	—	—	—	224	—	24,801	2,133	3,751	—	—	—	450	139	37,761
2	P	114	27	141	1,650	429	2,079	162	91	253	212	25	20,912	705	3,450	19,421	50	340	390	38	36,497
	C	—	—	151	—	—	2,705	—	—	—	224,500	—	24,801	2,133	3,751	25,172	—	—	—	139,000	37,761
3	P	114	27	141	1,650	429	2,079	162	91	253	212	25	20,912	705	3,450	19,421	50	340	390	38	36,497
	C	—	—	—	2,090	615	2,705	—	—	—	204	30	24,801	2,133	—	23,172	—	—	450	60	37,761
4	P	114	27	141	1,650	429	2,079	162	91	253	212	25	20,912	705	3,450	19,421	50	340	390	38	36,497
	C	—	—	151	2,705	—	2,705	—	—	—	—	—	24,801	—	—	23,172	—	—	—	139	37,761
5	P	34	7	41	400	137	537	39	54	93	64	0	3,908	181	25	2,229	1	1	2	0	4,524
	C	36	7	43	446	136	582	44	63	107	61	0	4,340	213	45	2,274	0	1	1	0	4,696
6	P	9	5	14	88	134	222	12	36	48	23	8,407	12,010	887	23	207	34	145	179	9,817	14,025
	C	7	27	34	—	421	474	27	47	74	10	5,965	7,296	904	23	230	43	0	43	7,895	8,982
7	P	42	20	62	705	390	1,095	74	170	244	96	1,741	7,040	624	555	5,656	70	395	465	3,121	12,715
	C	42	20	62	791	414	1,205	83,000	191,000	274,000	101	1,552	6,745	806	697	6,453	55	289	344	2,871	10,247

TABLE 4
Current year (C) *and edited records* (S).

Record number		PW	OE	TE	WW	OW	SW	LE	VP	LC	MH	CR	CM	ET	TCE	TAE	BR	MR	TR	VR	VS
1	C	—	—	151	—	—	2,705	—	—	—	224	—	24,801	2,133	3,751	—	—	—	450	139	37,761
	S	**	**	151	**	**	2,705	**	**	**	224	**	24,801	2,133	3,751	**	**	**	450	139	37,761
2	C	—	—	151	—	—	2,705	—	—	—	224,500	—	24,801	2,133	3,751	25,172	—	—	—	139,000	37,761
	S	**	**	151	**	**	2,705	**	**	**	**	**	24,801	2,133	3,751	25,172	**	**	**	**	37,761
3	C	—	—	—	2,090	615	2,705	—	—	—	204	30	24,801	2,133	—	23,172	—	—	450	60	37,761
	S	**	**	**	2,090	615	2,705	**	**	**	204	30	24,801	2,133	**	23,172	**	**	450	60	37,761
4	C	—	—	151	2,705	—	2,705	—	—	—	—	—	24,801	—	—	23,172	—	—	—	139	37,761
	S	**	**	151	2,705	**	**	**	**	**	**	**	24,801	**	**	23,172	**	**	**	139	37,761
5	C	36	7	43	446	136	582	44	63	107	61	0	4,340	213	45	2,274	0	1	1	0	4,696
	S	36	7	43	446	136	582	44	63	107	61	0	4,340	213	45	2,274	**	1	**	0	**
6	C	7	27	34	—	421	474	27	47	74	10	5,965	7,296	904	23	230	43	0	43	7,895	8,982
	S	7	**	**	**	**	**	27	47	74	10	5,965	7,296	904	23	230	43	**	**	**	8,982
7	C	42	20	62	791	414	1,205	83,000	191,000	274,000	101	1,552	6,745	806	697	6,453	55	289	344	2,871	10,247
	S	42	20	62	791	414	1,205	**	**	**	101	1,552	6,745	806	697	6,453	55	289	344	2,871	10,247

TABLE 5
Solution statistics for FCEDIT.

Record	Matrix generation time	LP solution time	B and B search time	Total time	B and B subproblems
1	0.067	0.491	0.013	0.571	0
2	0.064	0.563	0.588	1.215	3
3	0.065	0.450	0.010	0.525	0
4	0.070	0.552	0.228	0.850	2
5	0.063	0.324	0.564	0.951	2
6	0.081	0.665	5.312	6.058	48
7	0.064	0.343	1.372	1.779	5

In addition to the seven test records discussed above, 163 test records were formed by perturbing one or more fields of an acceptable record to a very large, incorrect value in such a way that records having up to ten incorrect fields were generated. These records were then run on the fixed charge editing procedure to determine if it would pick out the perturbed values. In this testing, the fixed charged editing procedure was 100% successful in finding the perturbed values.

The final testing involved a modification to the branch and bound procedure to allow for determination of alternate solutions for records. As an example of the result of finding alternate solutions, consider record 6. From Table 4, we can see that the number of fields suggested to be in error was 8. In Table 6, we see the original fields found in error along with four alternate solutions for record 6. In all cases, the cardinality is 8.

TABLE 6
Equivalent edits.

Alternative	Fields in error	Cardinality
1	2 3 4 5 6 17 18 19	8
2	1 3 4 6 10 17 18 20	8
3	1 2 4 6 10 17 18 20	8
4	1 3 4 6 10 16 17 20	8
5	1 2 4 6 10 16 17 20	8

Since the ASM editing problem is of such a special nature, it may be possible to derive a special purpose code to find the optimal number of fields to change that will be more efficient than the general purpose branch and bound procedure discussed here. However, we have shown that our procedure does work in a reasonable time for this set of edits even though no modifications were made in the procedure to take advantage of the special structure of the ASM edits. The strength of this algorithm is that it will work on any set of linear edits regardless of the structure (if any) of the problem.

5. Summary and suggestions for future work. We have described a fixed charge formulation of the editing of economic data for deterministic edits and discussed the use of a branch and bound procedure to solve this formulation for a minimal number of fields to be imputed. This fixed charge approach was applied to Annual Survey of Manufacturers (ASM) test data. Results from using a computer code based on the fixed charge editing procedure to edit the ASM test data were presented and discussed. The results appear to be encouraging for several reasons. First, it has been shown that it is possible to edit economic surveys with either missing or erroneous data or a combination of both. As long as the edit conditions are consistent, a side effect of this editing is a demonstration that an imputation exists. Secondly, this editing procedure is easy to understand and to use. The errors found involve a minimum number of fields to be imputed in order to yield a consistent record, and it is possible to vary the weights on fields to yield different imputations. Finally, a capability to find alternate patterns of fields suggested to be in error was demonstrated.

It should be noted that the approach to data editing described here could be used to edit *any* type of economic survey data *so long* as the edit conditions can be written as linear inequalities. The ASM edit conditions shown here were given as an example and the procedure is not specific to this set of edits. It would only be necessary to modify the data and edit conditions to fit the form described here before a fixed charge solution procedure could be used.

In terms of future work, the usefulness of implied edits as discussed by Fellegi and Holt [1] as they might be incorporated into the fixed charge editing procedure should be tested. If such implied edits could tighten up the lower bound calculations, then they might be useful in reducing the branch and bound search time.

REFERENCES

[1] I. P. Fellegi and D. Holt, *A systematic approach to automatic edit and imputation*, J. Amer. Statist. Assoc., 71 (1976), pp. 17–35.

[2] R. J. Freund and H. O. Hartley, *A procedure for automatic data editing*, J. Amer. Statist. Assoc., 64 (1969), pp. 1256–75.

[3] A. M. Geoffrion and R. E. Marsten, *Integer programming algorithms: a framework and state-of-the-art survey*, Management Sci., 18 (1972), pp. 465–491.

[4] B. Greenberg, *Developing an edit system for industry statistics*, Computer Science and Statistics: Proc. of the 13th Symposium on the Interface, 1981, pp. 11–16.

[5] W. M. Hirsch and G. B. Dantzig, *The fixed charge problem*, Naval Res. Logist. Quart., 15 (1968), pp. 413–424.

[6] G. L. Kusch and D. F. Clark, *Annual survey of manufacturers general statistics edit*, presented at the ASA meeting, August 1979.

[7] G. E. Liepins, *A rigorous, systematic approach to automatic data editing and its statistical basis*, ORNL/TM-7126, 1980.

[8] P. G. McKeown, *A branch and bound approach to solving fixed charge problems*, Naval Res. Logist. Quart., 28 (1981), pp. 607–617.

[9] ———, *A vertex ranking algorithm for the linear fixed charge problem*, Oper. Res., 23 (1975), pp. 1183–1191.

[10] ———, *An extreme point ranking algorithm for solving the linear fixed charge problem*, Ph.D. Dissertation, Univ. North Carolina, Chapel Hill, 1973.

[11] G. Sande, *An algorithm for fields to impute problems of numerical and coded data*, Working Paper, Statistics Canada.

[12] ———, *Diagnostic capabilities for a numerical edit specification analysis*, Working Paper, Statistics Canada, November 1976.

[13] ———, private communication, 1978.

[14] Warren E. Walker, *A heuristic adjacent extreme point algorithm for the fixed charge problem*, Management Sci., 22 (1976), pp. 587–596.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

THE PERFORMANCE OF k-STEP PSEUDORANDOM NUMBER GENERATORS UNDER THE UNIFORMITY TEST*

HARALD NIEDERREITER†

Abstract. Sequences of integers generated by k-step linear recursions can be transformed into sequences of uniform pseudorandom numbers by the digital method. We show by a priori bounds that if the least period of such a sequence is sufficiently large, then the pseudorandom numbers perform well under the uniformity test.

Key words. uniform pseudorandom numbers, digital method, uniformity test

1. Introduction. The most commonly used method for the generation of uniform pseudorandom numbers is the multiplicative congruential method of Lehmer [4], which is based on the one-step recursion

$$y_{n+1} \equiv ay_n \bmod M \quad \text{for } n = 0, 1, \cdots,$$

where M is a large integer, a is a suitably chosen integral multiplier, and the y_n are integers with $0 < y_n < M$. A sequence of uniform pseudorandom numbers in the interval $[0, 1]$ is derived by setting $x_n = y_n/M$. The statistical properties of multiplicative congruential pseudorandom numbers have been studied extensively both from the theoretical and the empirical point of view (see [3], [9] for surveys). The idea of using general k-step recursions for pseudorandom number generation was brought up shortly after the appearance of Lehmer's work (see e.g. van Wijngaarden [19]), but it received wider attention only through a later article of Tausworthe [16]. Since then, a considerable amount of literature has been devoted to k-step pseudorandom number generators; see e.g. [1], [2], [5], [7], [8], [10], [11], [12], [14], [15], [17], [18], [21], [22].

Pseudorandom number generation by k-step recursions proceeds as follows. For a prime number p a sequence $y_0, y_1, \cdots$ of integers satisfying $0 \leqq y_n < p$ is generated by the recursion

$$y_{n+k} \equiv a_{k-1}y_{n+k-1} + \cdots + a_0 y_n \bmod p \quad \text{for } n = 0, 1, \cdots, \tag{1}$$

where the a_i are constant integral coefficients with $a_0 \not\equiv 0 \bmod p$. The sequence of y_n is determined by the initial values $y_0, \cdots, y_{k-1}$. For obvious reasons it is assumed that not all of these initial values are 0. There are two different methods of obtaining uniform pseudorandom numbers from the y_n:

(a) *Normalization method.* Here one chooses p to be a large prime and normalizes the numbers y_n by setting $x_n = y_n/p$ for all n. Then $x_0, x_1, \cdots$ is taken as a sequence of uniform pseudorandom numbers in $[0, 1]$.

(b) *Digital method.* Let p be a small prime (often $p = 2$) and choose an integer $m \geqq 2$. Then set

$$x_n = \sum_{i=1}^{m} y_{n+i-1}p^{-i} \quad \text{for all } n, \tag{2}$$

that is, blocks of consecutive y_n of length m are interpreted as digits of x_n in the base p. In (2) these blocks are *overlapping*. One can also consider *nonoverlapping blocks* by

* Received by the editors March 15, 1983, and in revised form July 12, 1983.

† Mathematical Institute, Austrian Academy of Sciences, Dr. Ignaz-Seipel-Platz 2, A-1010 Vienna, Austria.

setting

$$(3) \qquad x_n=\sum_{i=1}^{m} y_{mn+i-1}p^{-i} \quad \text{for all } n.$$

In either case, $x_0, x_1, \cdots$ is taken as a sequence of uniform pseudorandom numbers in $[0, 1]$.

An important statistical test for the suitability of uniform pseudorandom numbers for simulation purposes is the uniformity (or equidistribution) test (see [3, Chap. 3]). The performance of uniform pseudorandom numbers obtained by the normalization method under univariate and multivariate uniformity tests was already investigated by the author [8], [10], [11], [12]. In the present paper we apply the (univariate) uniformity test to uniform pseudorandom numbers obtained by the digital method. The theoretical analysis of this method is more complicated than that of the normalization method.

If $x_0, x_1, \cdots$ is a sequence of uniform pseudorandom numbers in $[0, 1]$, then the *uniformity test* amounts to measuring the maximum deviation between the empirical distribution function of the sequence and the uniform distribution function on $[0, 1]$. In detail, let N be a positive integer and $E_N(t)$ the empirical distribution function, i.e., $E_N(t)$ is N^{-1} times the number of $n<N$ with $x_n \leqq t$, and set

$$(4) \qquad D_N=\sup_{0\leqq t\leqq 1} |E_N(t)-t|.$$

We establish bounds for D_N with x_n obtained from either (2) or (3).

It should be noted that a sequence $y_0, y_1, \cdots$ obtained from (1) is always periodic. Let τ denote the least period of the sequence. Then τ is also a period of the sequence of x_n obtained from either (2) or (3). It is easily seen that in the first case (i.e., in the case of overlapping blocks) τ is again the least period, whereas in the second case (i.e., in the case of nonoverlapping blocks) the least period of the sequence of x_n is given by $\tau/\gcd(m, \tau)$. In order to achieve a large value of τ, we assume that the *characteristic polynomial* of (1), i.e., the polynomial

$$(5) \qquad f(x)=x^k-a_{k-1}x^{k-1}-\cdots-a_0,$$

considered as a polynomial over the finite field $F_p=\mathbb{Z}/p\mathbb{Z}$, is irreducible over F_p. Then τ is always a divisor of p^k-1, and the value $\tau=p^k-1$ can be achieved (see § 2). In the case of nonoverlapping blocks we assume further that $\gcd(m, \tau)=1$. Then for both overlapping and nonoverlapping blocks the least period of the sequence of x_n is equal to τ. We also assume $m\leqq k$ so as to prevent obvious correlations among the digits of any single x_n.

Because of the periodic nature of the x_n it is only of interest to study D_N for $1\leqq N\leqq \tau$. We can now state the bounds for D_N that are valid under the assumptions above. For the sake of convenience we list again all the hypotheses: (i) $m\leqq k$; (ii) the polynomial f in (5) is irreducible over F_p with $f(0)\neq 0\in F_p$; (iii) the initial values $y_0, \cdots, y_{k-1}$ are not all 0; (iv) $\gcd(m, \tau)=1$ in the case of nonoverlapping blocks.

THEOREM 1. *For a sequence $x_0, x_1, \cdots$ generated by the digital method we have*

$$D_\tau<\frac{1}{p^m}+\left(\frac{p^{k/2}}{\tau}-\frac{1}{1+p^{k/2}}\right)\left(\left(\frac{2}{\pi}\log p+\frac{2}{5}\right)m+\frac{(m-1)(p-1)}{p}\right).$$

In the case $p=2$ we have

$$D_\tau\leqq\frac{1}{2^m}+\left(\frac{2^{k/2}}{\tau}-\frac{1}{1+2^{k/2}}\right)(m-\tfrac{1}{2}).$$

THEOREM 2. *For a sequence $x_0, x_1, \cdots$ generated by the digital method we have*

$$D_N < \frac{1}{p^m} + \left(\frac{p^{k/2}}{N}\left(\frac{2}{\pi}\log \tau + \frac{2}{5}\right) + \frac{p^{k/2}}{\tau} - \frac{1}{1+p^{k/2}}\right)$$
$$\cdot\left(\left(\frac{2}{\pi}\log p + \frac{2}{5}\right)m + \frac{(m-1)(p-1)}{p}\right) \quad \text{for } 1 \leqq N < \tau.$$

In the case $p=2$ we have

$$D_N < \frac{1}{2^m} + \left(\frac{2^{k/2}}{N}\left(\frac{2}{\pi}\log \tau + \frac{2}{5}\right) + \frac{2^{k/2}}{\tau} - \frac{1}{1+2^{k/2}}\right)\left(m - \frac{1}{2}\right) \quad \text{for } 1 \leqq N < \tau.$$

These results demonstrate that uniform pseudorandom numbers generated by the digital method perform well under the uniformity test if the period τ is sufficiently large. Particularly good results are obtained for the maximum period $\tau = p^k - 1$.

In § 2 we present some preparatory results. Sections 3 and 4 contain the proof of Theorem 1 and Theorem 2, respectively. In § 5 we show lower bounds for D_N that indicate to what extent Theorems 1 and 2 are best possible. A brief discussion of the results follows in § 6.

2. Preparatory results. For an arbitrary sequence $w_0, w_1, \cdots$ in $[0, 1)$ whose terms are given by finite digit expansions, we establish a bound for D_N by means of exponential sums. Let $b \geqq 2$ be an integer that serves as the base for the digit expansions, and let

$$w_n = \sum_{i=1}^{m} w_n^{(i)} b^{-i} \quad \text{for all } n,$$

where the digits $w_n^{(i)}$ are integers in the interval $[0, b-1]$. For $r \geqq 1$ we define $C_r(b)$ to be the set of $(h_1, \cdots, h_r) \in \mathbb{Z}^r$ with $-b/2 < h_j \leqq b/2$ for $1 \leqq j \leqq r$, and we let $C_r^*(b)$ be the set $C_r(b)$ with the r-tuple $(0, \cdots, 0)$ deleted. We set $C(b) = C_1(b)$ and $C^*(b) = C_1^*(b)$. For real u we write $e(u) = e^{2\pi i u}$.

LEMMA 1. *For the sequence $w_0, w_1, \cdots$ we have for any $N \geqq 1$,*

$$D_N \leqq \frac{1}{b^m} + \sum_{r=1}^{m} \frac{1}{b^r} \sum_{(h_1,\cdots,h_{r-1}) \in C_{r-1}(b)} \sum_{h_r \in C^*(b)} \frac{1}{\sin \pi |h_r/b|} \left| \frac{1}{N} \sum_{n=0}^{N-1} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_r w_n^{(r)}}{b}\right) \right|$$
$$+ \sum_{r=1}^{m-2} \frac{b-1}{b^{r+1}} \sum_{(h_1,\cdots,h_r) \in C_r^*(b)} \left| \frac{1}{N} \sum_{n=0}^{N-1} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_r w_n^{(r)}}{b}\right) \right|$$
$$+ \frac{1}{b^{m-1}} \sum_{(h_1,\cdots,h_{m-1}) \in C_{m-1}^*(b)} \left| \frac{1}{N} \sum_{n=0}^{N-1} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_{m-1} w_n^{(m-1)}}{b}\right) \right|.$$

Proof. For $0 \leqq t \leqq 1$ let

$$t = \sum_{i=1}^{\infty} t_i b^{-i}$$

be the expansion of t to the base b, with the usual condition that not almost all digits $t_i = b-1$, except in the case $t=1$ where we take all $t_i = b-1$. Then we have $w_n \leqq t$ if and only if $w_n^{(1)} = t_1, \cdots, w_n^{(r-1)} = t_{r-1}, w_n^{(r)} < t_r$ for some r with $1 \leqq r \leqq m-1$ (for $r=1$ the condition reduces to $w_n^{(1)} < t_1$) or $w_n^{(1)} = t_1, \cdots, w_n^{(m-1)} = t_{m-1}, w_n^{(m)} \leqq t_m$. Thus, if

$A(N, t)$ denotes the number of $n < N$ with $w_n \leqq t$, then

$$A(N,t) = \sum_{r=1}^{m-1} \sum_{j=0}^{t_r-1} \sum_{n=0}^{N-1} c_{t_1}(w_n^{(1)}) \cdots c_{t_{r-1}}(w_n^{(r-1)}) c_j(w_n^{(r)})$$
$$+ \sum_{j=0}^{t_m} \sum_{n=0}^{N-1} c_{t_1}(w_n^{(1)}) \cdots c_{t_{m-1}}(w_n^{(m-1)}) c_j(w_n^{(m)}),$$

where $c_j(w) = 1$ for $j = w$ and $c_j(w) = 0$ for $j \neq w$ with integers j, $w \in [0, b-1]$. Now

$$c_j(w) = \frac{1}{b} \sum_{h \in C(b)} e\left(\frac{h(w-j)}{b}\right),$$

and so

$$A(N,t) = \sum_{r=1}^{m-1} \sum_{j=0}^{t_r-1} \sum_{n=0}^{N-1} \frac{1}{b^r} \sum_{(h_1,\cdots,h_r) \in C_r(b)} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_r w_n^{(r)} - h_1 t_1 - \cdots - h_{r-1} t_{r-1} - h_r j}{b}\right)$$
$$+ \sum_{j=0}^{t_m} \sum_{n=0}^{N-1} \frac{1}{b^m} \sum_{(h_1,\cdots,h_m) \in C_m(b)} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_m w_n^{(m)} - h_1 t_1 - \cdots - h_{m-1} t_{m-1} - h_m j}{b}\right).$$

Separating the contributions from $(h_1, \cdots, h_r) = (0, \cdots, 0)$ in the sums above, we get

$$A(N,t) = N \sum_{r=1}^{m-1} \frac{t_r}{b^r} + \sum_{r=1}^{m-1} \sum_{j=0}^{t_r-1} \sum_{n=0}^{N-1} \frac{1}{b^r} \sum_{(h_1,\cdots,h_r) \in C_r^*(b)} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_r w_n^{(r)} - h_1 t_1 - \cdots - h_{r-1} t_{r-1} - h_r j}{b}\right)$$
$$+ \frac{N(t_m+1)}{b^m} + \sum_{j=0}^{t_m} \sum_{n=0}^{N-1} \frac{1}{b^m} \sum_{(h_1,\cdots,h_m) \in C_m^*(b)} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_m w_n^{(m)} - h_1 t_1 - \cdots - h_{m-1} t_{m-1} - h_m j}{b}\right).$$

It follows that

$$\left|\frac{A(N,t)}{N} - t\right|$$
$$\leqq \frac{1}{b^m} + \sum_{r=1}^{m-1} \frac{1}{b^r} \sum_{(h_1,\cdots,h_r) \in C_r^*(b)} \left|\frac{1}{N} \sum_{n=0}^{N-1} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_r w_n^{(r)}}{b}\right)\right| \left|\sum_{j=0}^{t_r-1} e\left(\frac{-h_r j}{b}\right)\right|$$
$$+ \frac{1}{b^m} \sum_{(h_1,\cdots,h_m) \in C_m^*(b)} \left|\frac{1}{N} \sum_{n=0}^{N-1} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_m w_n^{(m)}}{b}\right)\right| \left|\sum_{j=0}^{t_m} e\left(\frac{-h_m j}{b}\right)\right|.$$

For $h \in C(b)$ and an integer $L \geqq -1$ we have

$$\sum_{j=0}^{L} e\left(\frac{-hj}{b}\right) = L + 1 \quad \text{for } h = 0$$

and

$$\left|\sum_{j=0}^{L} e\left(\frac{-hj}{b}\right)\right| \leqq \frac{2}{|e(h/b)-1|} = \frac{1}{\sin \pi|h/b|} \quad \text{for } h \neq 0.$$

Using these results and the fact that $E_N(t) = A(N, t)/N$, we obtain Lemma 1 from (4). □

COROLLARY 1. *Let $w_0, w_1, \cdots$ be as above and suppose that for all $(h_1, \cdots, h_r) \in C_r^*(b)$, $1 \leqq r \leqq m$, we have*

$$\left|\frac{1}{N}\sum_{n=0}^{N-1} e\left(\frac{h_1 w_n^{(1)} + \cdots + h_r w_n^{(r)}}{b}\right)\right| \leqq B \tag{6}$$

for some constant B. Then

$$D_N \leqq \frac{1}{b^m} + B\left(\frac{m}{b}\sum_{h \in C^*(b)} \frac{1}{\sin \pi|h/b|} + \frac{(m-1)(b-1)}{b}\right).$$

Proof. Using (6), we get from Lemma 1

$$D_N \leqq \frac{1}{b^m} + B\left(\frac{m}{b}\sum_{h \in C^*(b)} \frac{1}{\sin \pi|h/b|} + \sum_{r=1}^{m-2} \frac{(b-1)(b^r-1)}{b^{r+1}} + \frac{b^{m-1}-1}{b^{m-1}}\right),$$

and the desired result follows by a straightforward calculation. □

The integers $y_0, y_1, \cdots$ as well as the coefficients a_i in (1) may be interpreted as elements of the finite field F_p. The recursion (1) becomes then the linear recurrence relation

$$y_{n+k} = a_{k-1}y_{n+k-1} + \cdots + a_0 y_n \quad \text{for } n = 0, 1, \cdots \tag{7}$$

over F_p. We use the theory of finite fields to get an explicit formula for the y_n. We refer to [6] for the necessary background on finite fields. Set $q = p^k$ and let F_q be the finite field of order q. Let Tr denote the trace function from F_q to F_p; it is a surjective F_p-linear mapping. Furthermore, the irreducible polynomial f over F_p in (5) splits in F_q; let $\alpha \in F_q$ be a fixed root of f. We denote by F_q^* the multiplicative group of nonzero elements of F_q.

LEMMA 2. *Let the sequence $y_0, y_1, \cdots$ of elements of F_p satisfy* (7), *and suppose that not all initial values $y_0, \cdots, y_{k-1}$ are* 0. *Then there exists $\beta \in F_q^*$ such that*

$$y_n = \mathrm{Tr}\,(\beta\alpha^n) \quad \textit{for all } n.$$

Proof. Since f is irreducible over F_p, $\{1, \alpha, \alpha^2, \cdots, \alpha^{k-1}\}$ is a basis of F_q over F_p. Then [20, Corollary 3-7-7] implies that the $k \times k$ matrix with (i, j) entry $\mathrm{Tr}\,(\alpha^{i-1}\alpha^{j-1})$ is nonsingular. It follows that there exist $b_1, \cdots, b_k \in F_p$ with

$$\sum_{i=1}^{k} b_i \,\mathrm{Tr}\,(\alpha^{i-1}\alpha^{j-1}) = y_{j-1} \quad \text{for } 1 \leqq j \leqq k.$$

Putting $\beta = b_1 + b_2\alpha + \cdots + b_k\alpha^{k-1} \in F_q$, we get

$$\mathrm{Tr}\,(\beta\alpha^n) = y_n \quad \text{for } n = 0, 1, \cdots, k-1.$$

We have $\beta \neq 0$ because not all initial values are 0. Since the y_n are uniquely determined by $y_0, \cdots, y_{k-1}$ and (7), it suffices to show that $\mathrm{Tr}\,(\beta\alpha^n)$ satisfies the relation (7).

Using properties of the trace function, we get

$$\begin{aligned}&\operatorname{Tr}(\beta\alpha^{n+k})-a_{k-1}\operatorname{Tr}(\beta\alpha^{n+k-1})-\cdots-a_0\operatorname{Tr}(\beta\alpha^n)\\&\quad=\operatorname{Tr}(\beta(\alpha^{n+k}-a_{k-1}\alpha^{n+k-1}-\cdots-a_0\alpha^n))=\operatorname{Tr}(\beta\alpha^n f(\alpha))=0\end{aligned}$$

since $f(\alpha)=0$. □

Since $f(0)\neq 0$ by hypothesis, we have $\alpha\neq 0$, so that α is an element of the group F_q^*. Let s be the order of α in that group. Then it follows from Lemma 2 that $y_{n+s}=y_n$ for all n, i.e., that s is a period of the sequence $y_0, y_1, \cdots$. Thus, the least period τ of this sequence satisfies $\tau\leqq s$. On the other hand, Lemma 2 implies $\operatorname{Tr}(\beta\alpha^{n+\tau})=\operatorname{Tr}(\beta\alpha^n)$ for all n, hence in particular

$$\operatorname{Tr}(\beta\alpha^n(\alpha^\tau-1))=0 \quad \text{for } n=0,1,\cdots,k-1.$$

Using linear combinations with coefficients in F_p and the fact that $\{1,\alpha,\alpha^2,\cdots,\alpha^{k-1}\}$ is a basis of F_q over F_p, we deduce that

$$\operatorname{Tr}(\gamma(\alpha^\tau-1))=0 \quad \text{for all } \gamma\in F_q.$$

As the trace function is surjective, this is only possible if $\alpha^\tau=1$. Consequently, we have $\tau=s$. It follows that τ is a divisor of $q-1=p^k-1$. The value $\tau=q-1=p^k-1$ can be achieved by letting f be the minimal polynomial over F_p of a generator of the cyclic group F_q^*, i.e., by letting f be a so-called primitive polynomial over F_p (compare with [6, Ch. 3]). From $\tau=s$ and the fact that τ is independent of the choice of a root α of f, we get another proof of the well-known result that any root of f has the same order in F_q^*.

We consider now character sums over the finite field F_q. We note first of all that

$$\chi(\beta)=e\left(\frac{1}{p}\operatorname{Tr}(\beta)\right) \quad \text{for all } \beta\in F_q \tag{8}$$

defines a character of the additive group of F_q. Let ψ be a character of F_q^*. Then

$$G(\psi)=\sum_{\beta\in F_q^*}\psi(\beta)\chi(\beta)$$

is a so-called Gaussian sum (see [6, Ch. 5]). With each ψ there is associated its conjugate character $\bar\psi$, defined by $\bar\psi(\gamma)=\psi(\gamma^{-1})$ for all $\gamma\in F_q^*$.

LEMMA 3. *If χ is given by* (8), *then*

$$\chi(\gamma)=\frac{1}{q-1}\sum_{\psi}G(\bar\psi)\psi(\gamma) \quad \textit{for all } \gamma\in F_q^*,$$

where the sum is extended over all characters ψ of F_q^*.

Proof. For $\gamma\in F_q^*$ we have

$$\sum_{\psi}G(\bar\psi)\psi(\gamma)=\sum_{\psi}\psi(\gamma)\sum_{\beta\in F_q^*}\bar\psi(\beta)\chi(\beta)=\sum_{\beta\in F_q^*}\chi(\beta)\sum_{\psi}\psi(\gamma\beta^{-1})=(q-1)\chi(\gamma)$$

by the orthogonality relations for characters, and the result follows. □

3. Proof of Theorem 1. By Lemma 1, the estimation of D_N reduces to the estimation of certain exponential sums. We consider first the case of *overlapping blocks.* Then $w_n^{(i)}=y_{n+i-1}$ for $1\leqq i\leqq m$ and all n, so that the exponential sums in Lemma 1 are of the form

$$\sum_{n=0}^{N-1} e\left(\frac{h_1y_n+h_2y_{n+1}+\cdots+h_ry_{n+r-1}}{p}\right) \tag{9}$$

with $(h_1, \cdots, h_r) \in C_r^*(p)$, $1 \leqq r \leqq m$. Since the h_i and y_n only matter mod p, we can view them as elements of F_p. Setting

$$z_n = h_1 y_n + h_2 y_{n+1} + \cdots + h_r y_{n+r-1},$$

we get from Lemma 2 with a suitable $\beta \in F_q^*$,

$$z_n = h_1 \operatorname{Tr}(\beta\alpha^n) + h_2 \operatorname{Tr}(\beta\alpha^{n+1}) + \cdots + h_r \operatorname{Tr}(\beta\alpha^{n+r-1})$$
$$= \operatorname{Tr}(\beta\alpha^n(h_1 + h_2\alpha + \cdots + h_r\alpha^{r-1})).$$

The hypothesis $m \leqq k$ implies $r \leqq k$, and since $\{1, \alpha, \alpha^2, \cdots, \alpha^{k-1}\}$ is a basis of F_q over F_p and not all h_i are 0, it follows that $h_1 + h_2\alpha + \cdots + h_r\alpha^{r-1} \neq 0$. Consequently, there exists $\gamma \in F_q^*$ with

(10) $$z_n = \operatorname{Tr}(\gamma\alpha^n) \quad \text{for all } n.$$

It follows from (8) and (10) that the sum in (9) attains the form

(11) $$\sum_{n=0}^{N-1} \chi(\gamma\alpha^n).$$

For the case $N = \tau$ these character sums can be estimated by the following general result.

LEMMA 4. *If λ is an element of F_q^* of order d, then for any $\gamma \in F_q^*$ we have*

$$\left|\sum_{n=0}^{d-1} \chi(\gamma\lambda^n)\right| \leqq q^{1/2} - \frac{d}{1+q^{1/2}}.$$

Proof. From Lemma 3 we get

$$\sum_{n=0}^{d-1} \chi(\gamma\lambda^n) = \frac{1}{q-1}\sum_{n=0}^{d-1}\sum_{\psi} G(\bar{\psi})\psi(\gamma\lambda^n) = \frac{1}{q-1}\sum_{\psi} G(\bar{\psi})\psi(\gamma)\sum_{n=0}^{d-1}\psi(\lambda)^n.$$

The inner sum in the last expression is equal to d if $\psi(\lambda) = 1$ and equal to 0 if $\psi(\lambda) \neq 1$ since $\psi(\lambda)^d = \psi(\lambda^d) = \psi(1) = 1$. Therefore,

(12) $$\sum_{n=0}^{d-1} \chi(\gamma\lambda^n) = \frac{d}{q-1}\sum_{\substack{\psi \\ \psi(\lambda)=1}} G(\bar{\psi})\psi(\gamma).$$

The last sum contains $(q-1)/d$ terms. Furthermore, we have $G(\psi) = -1$ if ψ is the trivial character and $|G(\psi)| = q^{1/2}$ if ψ is nontrivial (see [6, Ch. 5]), so that (12) yields

$$\left|\sum_{n=0}^{d-1} \chi(\gamma\lambda^n)\right| \leqq \frac{d}{q-1}\left(\left(\frac{q-1}{d} - 1\right)q^{1/2} + 1\right) = q^{1/2} - \frac{d}{1+q^{1/2}}. \qquad \square$$

Since α has order τ in F_q^* (see § 2), it follows that for $N = \tau$ any sum of the form (9) is bounded in absolute value by $q^{1/2} - \tau/(1+q^{1/2})$. Therefore, the condition (6) is satisfied with

$$B = \frac{q^{1/2}}{\tau} - \frac{1}{1+q^{1/2}}.$$

Corollary 1 implies

$$D_\tau \leqq \frac{1}{p^m} + \left(\frac{q^{1/2}}{\tau} - \frac{1}{1+q^{1/2}}\right)\left(\frac{m}{p}\sum_{h \in C^*(p)} \frac{1}{\sin \pi|h/p|} + \frac{(m-1)(p-1)}{p}\right).$$

For $p=2$ we have

$$\sum_{h\in C^*(p)} \frac{1}{\sin \pi|h/p|}=1.$$

For general p we have

$$\sum_{h\in C^*(p)} \frac{1}{\sin \pi|h/p|}<\frac{2}{\pi}p\log p+\tfrac{2}{5}p$$

according to [8, p. 574]. Recalling that $q=p^k$, we have shown Theorem 1 in the case of overlapping blocks.

In the case of *nonoverlapping blocks* we have $w_n^{(i)}=y_{mn+i-1}$ for $1\leqq i\leqq m$ and all n. Thus, instead of (9) we get sums of the form

$$\sum_{n=0}^{N-1} e\left(\frac{h_1y_{mn}+h_2y_{mn+1}+\cdots+h_ry_{mn+r-1}}{p}\right) \tag{13}$$

with $(h_1,\cdots,h_r)\in C_r^*(p)$, $1\leqq r\leqq m$. The argument leading to (11) shows then that a sum in (13) attains the form

$$\sum_{n=0}^{N-1} \chi(\gamma\alpha^{mn}) \tag{14}$$

with a suitable $\gamma\in F_q^*$. Again, α has order τ in F_q^*, and since in the case of nonoverlapping blocks we assume gcd $(m,\tau)=1$, it follows that α^m also has order τ in F_q^*. Using Lemma 4 with $\lambda=\alpha^m$, we get

$$\left|\sum_{n=0}^{\tau-1}\chi(\gamma\alpha^{mn})\right|\leqq q^{1/2}-\frac{\tau}{1+q^{1/2}}.$$

Thus, for $N=\tau$ the condition (6) is satisfied with

$$B=\frac{q^{1/2}}{\tau}-\frac{1}{1+q^{1/2}}.$$

The proof of Theorem 1 is completed as in the case of overlapping blocks.

4. Proof of Theorem 2. In the case of *overlapping blocks* we have to consider sums of the form (9), which can be written in the form (11) according to the argument in § 3. These sums are estimated by the following general result.

LEMMA 5. *If λ is an element of F_q^* of order d, then for any $\gamma\in F_q^*$ we have*

$$\left|\sum_{n=0}^{N-1}\chi(\gamma\lambda^n)\right|<q^{1/2}\left(\frac{2}{\pi}\log d+\frac{2}{5}\right)+\frac{Nq^{1/2}}{d}-\frac{N}{1+q^{1/2}}\quad \textit{for all } N\geqq 1.$$

Proof. As in the proof of Lemma 4 we get

$$\sum_{n=0}^{N-1}\chi(\gamma\lambda^n)=\frac{1}{q-1}\sum_{\psi}G(\bar{\psi})\psi(\gamma)\sum_{n=0}^{N-1}\psi(\lambda)^n.$$

Distinguishing between the cases $\psi(\lambda)=1$ and $\psi(\lambda)\neq 1$, we obtain

$$\begin{aligned}\sum_{n=0}^{N-1}\chi(\gamma\lambda^n)&=\frac{N}{q-1}\sum_{\substack{\psi\\ \psi(\lambda)=1}}G(\bar{\psi})\psi(\gamma)+\frac{1}{q-1}\sum_{\substack{\psi\\ \psi(\lambda)\neq 1}}G(\bar{\psi})\psi(\gamma)\frac{\psi(\lambda^N)-1}{\psi(\lambda)-1}\\ &=\frac{N}{d}\sum_{n=0}^{d-1}\chi(\gamma\lambda^n)+\frac{1}{q-1}\sum_{\substack{\psi\\ \psi(\lambda)\neq 1}}G(\bar{\psi})\psi(\gamma)\frac{\psi(\lambda^N)-1}{\psi(\lambda)-1},\end{aligned}$$

where we used (12) in the last step. By Lemma 4 and the fact that $|G(\psi)| = q^{1/2}$ for nontrivial ψ, it follows that

$$\left|\sum_{n=0}^{N-1} \chi(\gamma\lambda^n)\right| \leqq \frac{N}{d}\left(q^{1/2} - \frac{d}{1+q^{1/2}}\right) + \frac{2q^{1/2}}{q-1} \sum_{\substack{\psi \\ \psi(\lambda)\neq 1}} \frac{1}{|\psi(\lambda)-1|}.$$

Each dth root of unity $e(j/d)$ occurs exactly $(q-1)/d$ times as a value of $\psi(\lambda)$, hence

$$\left|\sum_{n=0}^{N-1} \chi(\gamma\lambda^n)\right| \leqq \frac{N}{d}\left(q^{1/2} - \frac{d}{1+q^{1/2}}\right) + \frac{2q^{1/2}}{q-1} \cdot \frac{q-1}{d} \sum_{j=1}^{d-1} \frac{1}{|e(j/d)-1|}$$

$$= \frac{N}{d}\left(q^{1/2} - \frac{d}{1+q^{1/2}}\right) + \frac{q^{1/2}}{d} \sum_{j=1}^{d-1} \frac{1}{\sin(\pi j/d)}.$$

By [8, p. 574] we have

$$\sum_{j=1}^{d-1} \frac{1}{\sin(\pi j/d)} < \frac{2}{\pi} d \log d + \tfrac{2}{5}d,$$

and the desired result follows. □

As in § 3 we see then that condition (6) is satisfied with

$$B = \frac{q^{1/2}}{N}\left(\frac{2}{\pi}\log \tau + \frac{2}{5}\right) + \frac{q^{1/2}}{\tau} - \frac{1}{1+q^{1/2}}.$$

The proof of Theorem 2 for the case of overlapping blocks is completed by the arguments of § 3.

In the case of *nonoverlapping blocks* we have to consider sums of the form (13), which can be written in the form (14). We use Lemma 5 with $\lambda = \alpha^m$, but otherwise proceed as in the earlier case.

5. Lower bounds. The term p^{-m} appearing in the bounds for D_N in Theorems 1 and 2 can be viewed as the "discretization error," resulting from the fact that all the pseudorandom numbers x_n are rationals with denominator p^m. Indeed, it can be shown easily that p^{-m} is a lower bound for D_N. We note that any x_n generated by (2) or (3) satisfies

$$x_n \leqq \sum_{i=1}^{m} (p-1)p^{-i} = 1 - p^{-m},$$

so that (4) implies

$$D_N \geqq |E_N(1-p^{-m}) - (1-p^{-m})| = |1-(1-p^{-m})| = p^{-m} \tag{15}$$

for all $N \geqq 1$.

By more elaborate arguments, we can establish lower bounds for D_N that relate to the second terms in the bounds given in Theorems 1 and 2. We use the notation introduced in § 2.

For a prime p define the constant

$$A_p = \sup_{0\leqq\theta<1} \sum_{j=0}^{p-2} \left|\cos 2\pi\left(\frac{j}{p}+\theta\right) - \cos 2\pi\left(\frac{j+1}{p}+\theta\right)\right|.$$

It is easily seen that $A_2=2$. For general p and any real θ we have

$$\sum_{j=0}^{p-2}\left|\cos 2\pi\left(\frac{j}{p}+\theta\right)-\cos 2\pi\left(\frac{j+1}{p}+\theta\right)\right| = 2\pi\sum_{j=0}^{p-2}\left|\int_{j/p}^{(j+1)/p}\sin 2\pi(x+\theta)\,dx\right|$$
$$<2\pi\int_0^1 |\sin 2\pi(x+\theta)|\,dx=4,$$

so that $A_p \leqq 4$.

THEOREM 3. *Let f be an irreducible polynomial over F_p with $f(0)\neq 0$ as in* (5), *and let τ be the order of any root of f in $F^*_{p^k}$. Then for any N, $1\leqq N\leqq \tau$, there exists a sequence $x_0, x_1, \cdots$ generated by the digital method* (*with overlapping blocks*) *by means of* (1) *such that*

$$D_N \geqq \frac{1}{A_p}\left(\frac{p^k-N}{(p^k-1)N}\right)^{1/2}.$$

The same result holds for the case of nonoverlapping blocks under the usual assumption $\gcd(m,\tau)=1$.

For the proof of this theorem we need two lemmas.

LEMMA 6. *Let f be an irreducible polynomial over F_p with $f(0)\neq 0$ as in* (5), *and let τ be the order of any root of f in F^*_q, $q=p^k$. Then for any N, $1\leqq N\leqq\tau$, there exists a sequence $y_0, y_1, \cdots$ of integers satisfying $0\leqq y_n<p$ and the recursion* (1) *such that not all initial values $y_0,\cdots,y_{k-1}$ are* 0 *and*

$$\left|\sum_{n=0}^{N-1} e\left(\frac{y_n}{p}\right)\right| \geqq \left(\frac{N(q-N)}{q-1}\right)^{1/2}.$$

Proof. Let $\alpha\in F^*_q$ be a root of f and fix N, $1\leqq N\leqq\tau$. Then

$$\sum_{\beta\in F^*_q}\left|\sum_{n=0}^{N-1}\chi(\beta\alpha^n)\right|^2 = \sum_{\beta\in F^*_q}\sum_{j,n=0}^{N-1}\chi(\beta\alpha^j)\chi(-\beta\alpha^n)$$
$$=\sum_{j,n=0}^{N-1}\sum_{\beta\in F^*_q}\chi(\beta(\alpha^j-\alpha^n))=(q-1)N+\sum_{\substack{j,n=0\\ j\neq n}}^{N-1}\sum_{\beta\in F^*_q}\chi(\beta(\alpha^j-\alpha^n)).$$

Since α has order τ in F^*_q, we have $\alpha^j\neq\alpha^n$ for $j\neq n$. Therefore, each inner sum in the last expression has the value

$$\sum_{\gamma\in F^*_q}\chi(\gamma)=\sum_{\gamma\in F_q}\chi(\gamma)-\chi(0)=-1,$$

and so

$$\sum_{\beta\in F^*_q}\left|\sum_{n=0}^{N-1}\chi(\beta\alpha^n)\right|^2=(q-1)N-(N^2-N)=N(q-N).$$

It follows that there exists $\beta\in F^*_q$ with

$$(16)\qquad \left|\sum_{n=0}^{N-1}\chi(\beta\alpha^n)\right|\geqq\left(\frac{N(q-N)}{q-1}\right)^{1/2}.$$

With this β we put $y_n=\mathrm{Tr}\,(\beta\alpha^n)$ for all n. Then one shows as in the proof of Lemma 2 that the y_n satisfy (7). If $y_0,\cdots,y_{k-1}$ were all 0, then using linear combinations with coefficients in F_p and the fact that $\{1,\alpha,\alpha^2,\cdots,\alpha^{k-1}\}$ is a basis of F_q over F_p, it would follow that $\mathrm{Tr}\,(\beta\gamma)=0$ for all $\gamma\in F_q$, hence $\beta=0$, a contradiction. We interpret now the y_n as integers with $0 \leqq y_n < p$. Then the y_n satisfy (1), and the proof is complete by (16) and the observation that $\chi(\beta\alpha^n) = e(y_n/p)$ for all n according to (8). □

LEMMA 7. *Let f and τ be as in Lemma 6, and let* $\gcd(m, \tau) = 1$. *Then for any N, $1 \leqq N \leqq \tau$, there exists a sequence $y_0, y_1, \cdots$ of integers satisfying $0 \leqq y_n < p$ and the recursion* (1) *such that not all initial values $y_0, \cdots, y_{k-1}$ are* 0 *and*

$$\left|\sum_{n=0}^{N-1} e\left(\frac{y_{mn}}{p}\right)\right| \geqq \left(\frac{N(q-N)}{q-1}\right)^{1/2}.$$

Proof. The argument leading to (16) can be carried out with α replaced by α^m, since α^m also has order τ in F_q^*. Hence there exists $\beta \in F_q^*$ with

$$\left|\sum_{n=0}^{N-1} \chi(\beta\alpha^{mn})\right| \geqq \left(\frac{N(q-N)}{q-1}\right)^{1/2}.$$

We put $y_n = \mathrm{Tr}\,(\beta\alpha^n)$ for all n and complete the proof as in Lemma 6. □

Proof of Theorem 3. Let N be fixed. For the case of overlapping blocks we choose a sequence $y_0, y_1, \cdots$ satisfying the properties in Lemma 6. For $j = 0, 1, \cdots, p-1$ let $Z(j)$ be the number of $n < N$ with $y_n = j$, and put

$$S(j) = \sum_{i=0}^{j} \left(Z(i) - \frac{N}{p}\right).$$

Then

$$\sum_{n=0}^{N-1} e\left(\frac{y_n}{p}\right) = \sum_{j=0}^{p-1} Z(j)\, e\left(\frac{j}{p}\right) = \sum_{j=0}^{p-1} \left(Z(j) - \frac{N}{p}\right) e\left(\frac{j}{p}\right),$$

and summation by parts yields

$$\sum_{n=0}^{N-1} e\left(\frac{y_n}{p}\right) = \sum_{j=0}^{p-2} S(j)\left(e\left(\frac{j}{p}\right) - e\left(\frac{j+1}{p}\right)\right),$$

where we used $S(p-1) = 0$. For some θ, $0 \leqq \theta < 1$, we get

$$\begin{aligned}\left|\sum_{n=0}^{N-1} e\left(\frac{y_n}{p}\right)\right| &= e(\theta) \sum_{j=0}^{p-2} S(j)\left(e\left(\frac{j}{p}\right) - e\left(\frac{j+1}{p}\right)\right) \\ &= \sum_{j=0}^{p-2} S(j)\left(e\left(\frac{j}{p} + \theta\right) - e\left(\frac{j+1}{p} + \theta\right)\right),\end{aligned}$$

and taking real parts,

$$\begin{aligned}\left|\sum_{n=0}^{N-1} e\left(\frac{y_n}{p}\right)\right| &= \sum_{j=0}^{p-2} S(j)\left(\cos 2\pi\left(\frac{j}{p} + \theta\right) - \cos 2\pi\left(\frac{j+1}{p} + \theta\right)\right) \\ &\leqq \sum_{j=0}^{p-2} |S(j)| \left|\cos 2\pi\left(\frac{j}{p} + \theta\right) - \cos 2\pi\left(\frac{j+1}{p} + \theta\right)\right|.\end{aligned}$$

Let the integer J, $0 \leqq J \leqq p-2$, be such that

$$|S(J)| = \max_{0 \leqq j \leqq p-2} |S(j)|.$$

Then

$$\left|\sum_{n=0}^{N-1} e\left(\frac{y_n}{p}\right)\right| \leqq |S(J)| \sum_{j=0}^{p-2} \left|\cos 2\pi\left(\frac{j}{p} + \theta\right) - \cos 2\pi\left(\frac{j+1}{p} + \theta\right)\right| \leqq |S(J)| A_p;$$

hence

$$\left|\sum_{i=0}^{J} Z(i)-\frac{(J+1)N}{p}\right| \geqq \frac{1}{A_p}\left(\frac{N(q-N)}{q-1}\right)^{1/2} \tag{17}$$

by Lemma 6 and the definition of $S(J)$.

The sequence $y_0, y_1, \cdots$ yields a sequence $x_0, x_1, \cdots$ generated by the digital method with overlapping blocks of length m. Choose $0 \leqq \varepsilon < p^{-m}$ and put $t_0 = (J+1)p^{-1} - p^{-m} + \varepsilon$. Then $x_n \leqq t_0$ if and only if $y_n \leqq J$. It follows from (4) and (17) that

$$D_N \geqq |E_N(t_0) - t_0| = \left|\frac{1}{N}\sum_{i=0}^{J} Z(i) - \frac{J+1}{p} + p^{-m} - \varepsilon\right|$$

$$\geqq \left|\frac{1}{N}\sum_{i=0}^{J} Z(i) - \frac{J+1}{p}\right| - |p^{-m} - \varepsilon| \geqq \frac{1}{A_p}\left(\frac{q-N}{(q-1)N}\right)^{1/2} - p^{-m} + \varepsilon.$$

Letting ε tend to p^{-m} and recalling that $q = p^k$, we obtain the desired result for the case of overlapping blocks.

For the case of nonoverlapping blocks we proceed as above, but we use y_{mn} instead of y_n and Lemma 7 instead of Lemma 6. □

6. Discussion. As an illustration of the lower bounds in § 5 we consider the case $N = \tau$. If τ has the maximum value $p^k - 1$, then, dropping absolute constants, we see that the upper bound for D_τ in Theorem 1 is of the order $p^{-m} + p^{-k} \log p^m$, and since $m \leqq k$ in that theorem, D_τ has an upper bound of the order $p^{-m} \log p^m$. On the other hand, D_τ is at least p^{-m} by (15). In other cases of interest, τ will be of the form $(p^k - 1)/d$ with a small divisor $d > 1$ of $p^k - 1$. Then, dropping absolute constants, the upper bound for D_τ in Theorem 1 is of the order $p^{-m} + (d-1)p^{-k/2} \log p^m$. On the other hand, D_τ is at least p^{-m} by (15) and, apart from absolute constants, at least of the order $(d-1)^{1/2} p^{-k/2}$ by Theorem 3. It should be noted that the number τ in Theorem 3 is also the least period of the sequence $x_0, x_1, \cdots$, according to the information gathered in § 1 and 2.

These results demonstrate that digital k-step pseudorandom numbers perform better under the uniformity test if the value τ of the least period is increased. For the maximum period $\tau = p^k - 1$ the discussion in the preceding paragraph shows that the order of magnitude of D_τ is, up to a logarithmic factor, given by the discretization error p^{-m}. Since this is independent of the choice of $k \geqq m$, the following procedure is suggested. Choose a small prime p, say $p = 2$, and select m large enough so as to get a tolerable discretization error p^{-m}. Then take $k = m$ and choose the recursion (1) in such a way that its characteristic polynomial (5) is a primitive polynomial of degree m over F_p (compare with § 2).

Apart from the requirement that the a_i in (1) be selected in such a way that the sequence of y_n has a large least period (e.g., such that the characteristic polynomial is primitive), no further condition on the a_i is needed to guarantee good behavior under the univariate uniformity test. This is similar to the situation arising for uniform pseudorandom numbers generated by the normalization method (see [12]). The specific choice of the a_i has a bearing on the performance of digital k-step pseudorandom numbers as soon as multivariate uniformity tests of sufficiently high dimension s are considered, namely for $s > k/m$. This will be discussed in greater detail in [13].

REFERENCES

[1] B. M. FELLEN, *An implementation of the Tausworthe generator*, Comm. ACM, 12 (1969), p. 413.

[2] A. GRUBE, *Mehrfach rekursiv-erzeugte Pseudo-Zufallszahlen*, Z. Angew. Math. Mech., 53 (1973), pp. T223–T225.

[3] D. E. KNUTH, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[4] D. H. LEHMER, *Mathematical methods in large-scale computing units*, Proc. 2nd Symposium on Large-Scale Digital Calculating Machinery, Cambridge, MA, 1949, Harvard Univ. Press, Cambridge, MA, 1951, pp. 141–146.

[5] T. G. LEWIS AND W. H. PAYNE, *Generalized feedback shift register pseudorandom number algorithm*, J. Assoc. Comput. Mach., 20 (1973), pp. 456–468.

[6] R. LIDL AND H. NIEDERREITER, *Finite Fields*, Encyclopedia of Mathematics and Its Applications, Vol. 20, Addison-Wesley, Reading, MA, 1983.

[7] F. NEUMAN AND C. F. MARTIN, *The autocorrelation structure of Tausworthe pseudorandom number generators*, IEEE Trans. Comput., 25 (1976), pp. 460–464.

[8] H. NIEDERREITER, *On the distribution of pseudo-random numbers generated by the linear congruential method.* III, Math. Comp., 30 (1976), pp. 571–597.

[9] ———, *Quasi-Monte Carlo methods and pseudo-random numbers*, Bull. Amer. Math. Soc., 84 (1978), pp. 957–1041.

[10] ———, *Statistical tests for linear congruential pseudo-random numbers*, COMPSTAT 1978: Proceedings in Computational Statistics, Leiden, 1978, Physica-Verlag, Vienna, 1978, pp. 398–404.

[11] ———, *Statistical independence properties of Tausworthe pseudo-random numbers*, Proc. 3rd Caribbean Conf. on Combinatorics and Computing, Cave Hill, Barbados, 1981, Univ. of the West Indies, Cave Hill, Barbados, 1981, pp. 163–168.

[12] ———, *Statistical tests for Tausworthe pseudo-random numbers*, Probability and Statistical Inference, W. Grossmann et al., eds., Reidel, Dordrecht, 1982, pp. 265–274.

[13] ———, *Applications des corps finis aux nombres pseudo-aléatoires*, Sém. Théorie des Nombres 1982–1983, Univ. de Bordeaux I, to appear.

[14] A. I. PAVLOV AND B. B. POHODZEĬ, *Pseudorandom numbers generated by linear recurrence relations over a finite field*, Ž. Vyčisl. Mat. i Mat. Fiz., 19 (1979), pp. 836–842. (In Russian.)

[15] L. A. ROMERO, *On the generation of uniform pseudorandom numbers*, Investigación Oper., 2-3 (1980), pp. 17–27. (In Spanish.)

[16] R. C. TAUSWORTHE, *Random numbers generated by linear recurrence modulo two*, Math. Comp., 19 (1965), pp. 201–209.

[17] J. P. R. TOOTILL, W. D. ROBINSON AND A. G. ADAMS, *The runs up-and-down performance of Tausworthe pseudo-random number generators*, J. Assoc. Comput. Mach., 18 (1971), pp. 381–399.

[18] J. P. R. TOOTILL, W. D. ROBINSON AND D. J. EAGLE, *An asymptotically random Tausworthe sequence*, J. Assoc. Comput. Mach., 20 (1973), pp. 469–481.

[19] A. VAN WIJNGAARDEN, *Mathematics and computing*, Proc. Symposium on Automatic Digital Computation, London, 1954, H. M. Stationery Office, London, 1954, pp. 125–129.

[20] E. WEISS, *Algebraic Number Theory*, McGraw-Hill, New York, 1963.

[21] J. R. B. WHITTLESEY, *A comparison of the correlational behavior of random number generators for the* IBM 360, Comm. ACM, 11 (1968), pp. 641–644.

[22] ———, *On the multidimensional uniformity of pseudorandom generators*, Comm. ACM, 12 (1969), p. 247.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

COLLOCATION FOR SINGULAR PERTURBATION PROBLEMS III: NONLINEAR PROBLEMS WITHOUT TURNING POINTS*

U. ASCHER† AND R. WEISS‡

Abstract. A class of nonlinear singularly perturbed boundary value problems is considered, with restrictions which allow only well-posed problems with possible boundary layers, but no turning points. For the numerical solution of these problems, a close look is taken at a class of general purpose, symmetric finite difference schemes arising from collocation.

It is shown that if locally refined meshes, whose practical construction is discussed, are employed, then high order uniform convergence of the numerical solution is obtained. Nontrivial examples are used to demonstrate that highly accurate solutions to problems with extremely thin boundary layers can be obtained in this way at a very reasonable cost.

Key words. collocation, singular perturbation, nonlinear problems, symmetric difference schemes

1. Introduction.

The numerical solution of nonlinear singular perturbation problems presents some major computational difficulties. At the same time, such problems are abundant in applications, e.g. in semiconductor theory, diffusion-convection processes with a dominant convection term, fluid dynamic problems with large Reynolds numbers, etc.

The basic computational difficulty arising can be roughly described as follows. Suppose that a grid with maximum spacing h is used to discretize the differential problem. While normally h can be assumed to be small compared to the differential problem parameters, in a singular perturbation problem the parameter ε, which multiplies some of the highest derivatives appearing in the problem formulation, is so small that for practical reasons we must consider the case

$$h \geqq \varepsilon, \quad \text{or even} \quad h \gg \varepsilon.$$

Furthermore, from the solution approximation point of view, one has to deal with transition layers, i.e. regions where the solution profile varies rapidly, with gradients proportional to some negative power of ε.

Consider boundary value problems of this type for ordinary differential equations. To recall, there are two classes of general purpose methods for boundary value ODEs. (See, e.g., Keller [13].) The first is that of initial value techniques like multiple shooting. Such techniques fail to perform adequately for singular perturbation problems, requiring $h = O(\varepsilon)$, essentially for the same reason that causes grief when simple shooting is applied to moderately "stiff" problems.

The other class of general purpose methods is that of centered difference (or collocation) schemes. Such schemes offer more hope for singular perturbation problems, requiring small mesh spacing (of size comparable to ε) only in layer regions, not everywhere. However, if the mesh is not fine enough in a layer region then the numerical solution is polluted *everywhere* by oscillatory error components.

* Received by the editors November 24, 1982, and in revised form June 8, 1983.

† Department of Computer Science, University of British Columbia, Vancouver, Canada V6T 1W5. The research of this author was supported in part by the Natural Sciences and Engineering Research Council under grant A4306.

‡ Institut für Angewandte und Numerische Mathematik, Technische Universität Wien, 1040 Wien, Gusshausstrasse 27–29, Austria.

In recent years, a large number of special purpose methods have been proposed which do not require an accurate representation by the mesh of layer regions, while still producing accurate numerical solutions outside the layers. The upwinded Euler scheme is a popular example of such methods, where the error generated inside the layer regions is quickly damped outside.

The approach highlighted in this paper is that of centered, general purpose schemes with local mesh refinement in layer regions. Since this is currently less popular than special, (explicitly or implicitly) upwinded schemes, we now discuss the relative merits of these approaches.

Firstly, as has been noted by others as well, while centered schemes tend to produce wiggly numerical noise if the mesh is inadequate, special one-sided schemes tend to have too much "artificial viscosity," i.e. to be inaccurate (typically only first order in h) and to smear a layer information over a number of neighboring mesh elements. A smooth solution curve can actually be considered worse than one containing numerical ripples if it is wrong, because its form is more deceptive; cf. Gresho and Lee [11].

A natural idea here is to obtain a first, relatively inaccurate, solution by a special purpose method and then to switch to a centered, more accurate, general purpose method with mesh points distributed according to the obtained first solution profile. However, the general implementation of such a switch is far from being trivial, and a simpler and probably more robust technique is to do continuation in ε, using the centered method all along (i.e., solve a sequence of problems with decreasing ε, the first with $\varepsilon = h$, say, and the last with the desired value of ε, gradually upgrading the mesh, as well as the initial solution profile for the nonlinear iterative process). In some special cases, it is well known that replacing ε by $\alpha = O(h)$ in a centered scheme produces a one-sided scheme (see, e.g., Hemker [23]). Then, the usual continuation process for centered schemes can be viewed as a gradual, flexible generalization of a switch from one-sided to centered schemes.

It should be noted that higher order special purpose methods of Runge–Kutta type exist as well (Ringhofer [18], Ascher and Weiss [3]). These methods do share the drawbacks of special purpose schemes mentioned below.

One-sided special purpose schemes generally require upwinding (explicit or implicit). Thus, unless the problem is already in a form where growing and decaying fundamental matrix components are separated, a costly transformation is required to bring it to such form; see Kreiss and Kreiss [15].

Recently, there has been a significant interest in the phenomenon where the difference problem has spurious solutions, which do not correspond to any solution of the differential problem (see, e.g., Beyn and Doedel [5]). When solving a singular perturbation problem with $\varepsilon \ll h$, using any type of scheme, there is in general room for such concern, because the difference operator is not exactly modeled after the differential operator any more. If a centered scheme is used with an inadequate mesh (say, uniform) then spurious solutions may easily result; see, e.g., Kellogg, Shubin and Stephens [14]. This, however, is less important for practical purposes, because such a numerical procedure is nonsensical anyway. Our experience, and reports by others, indicate that with an appropriate mesh (still with $\varepsilon \ll h$), difficulties with spurious solutions are rather rare in practice when using the schemes advocated here.

Finally, note that a nonlinear singular perturbation problem where the location of a transition layer is not precisely known can be far more difficult to solve numerically than one with only known layer locations. At least in the latter case we can always flood the transition layer region with sufficiently many mesh points so as to remove the singular perturbation effect and obtain a solution, even if not in the most cost-effective way. Now, in case the layer location depends on the solution and is not exactly known, the upwinding of a special scheme may be done in the wrong direction. A backward Euler scheme then becomes a forward Euler scheme, which is not even A-stable. This may result in difficulties in the convergence of the nonlinear iteration, yielding a cause for some pessimism in general. Note, though, that in some special cases easy convergence with upwinded one-sided schemes can be guaranteed. For instance, Abrahamsson and Osher [24] have devised a low order scheme for a class of single, quasilinear second order ODEs with possibly many turning points, where Newton's method is replaced by a slow but sure time-variable imbedding and the resulting process is guaranteed to converge. No such guarantees are known for symmetric schemes. However, under more general circumstances, a symmetric (or centered) scheme as discussed in this paper is A-stable in both directions of integration and hence is less prone to disastrous results. Yet, some idea of the location of layers must be at hand when $\varepsilon \ll h$. Continuation in ε can again be used in principle with centered schemes to methodically refine the mesh appropriately. For an example, see Wan and Ascher [21]. Still, the success of such a process may well depend on a (perhaps vague) a priori knowledge of a desirable initial solution profile.

In summary, the one-sided upwinded schemes seem to be suited mainly for special purposes, i.e. for particular classes of problems, and especially when the solution is needed to be accurately known only away from layer regions. For general purposes, e.g. in order to use in a general singular perturbation software for boundary value ODEs, the symmetric schemes seem more suitable.

The purpose of this paper is to consider the computational implementation and performance of a class of symmetric, or centered, collocation schemes which include the most familiar finite difference schemes as special cases. We consider the application of these schemes to a general, but restricted, class of nonlinear singularly perturbed problems which are well posed and allow for boundary layers only. This class of problems is relatively well understood analytically. The analytical knowledge allows us to take a close look at the numerical schemes, and we believe that this is an essential step towards understanding the performance of these schemes on wider classes of problems. Computational experience with such schemes crudely applied to a number of various problems has already been reported (e.g. Hemker et al. [12], Ascher [1], Wan and Ascher [21]).

In Part I [3] and Part II [4] of this work (hereinafter referred to as "Part I" and "Part II," respectively) we have considered symmetric collocation schemes for the numerical solution of linear singularly perturbed problems. In particular, Lobatto and Gauss collocation points have been considered. The simplest instances of these methods are the well-known trapezoidal and midpoint difference schemes. The ideas in these papers have been put into use in Spudich and Ascher [20].

We have shown that these symmetric schemes produce highly accurate numerical solutions at a very reasonable cost, provided that appropriately fine meshes are used near the boundaries, where the analytic solution may have steep boundary layers.

Here we extend these results to nonlinear problems, where Newton's method of quasilinearization is used and the resulting linearized boundary value problems are solved using the collocation implementation discussed in Part II. We demonstrate the potential of these schemes on three examples which appear in the literature.

It turns out that the convergence results as well as the mesh construction in Part II extend, with slight modifications, to the nonlinear case. However, the extension is not trivial. The differences between the linear and nonlinear problems are highlighted in the next section, which prepares the analytic preliminaries.

In § 3 we describe the numerical schemes used, state the convergence results and outline their proofs. Practical mesh construction is discussed in § 4.

In § 5 we discuss in detail our numerical experience with three examples. The numerical schemes are shown to produce highly accurate solutions to problems with extremely thin boundary layers, at a very reasonable cost.

2. Analytic preliminaries. Consider the problem of order $n+m$ for $x(t, \varepsilon) = (y(t, \varepsilon), z(t, \varepsilon))$, where $0 \leqq t \leqq 1$:

$$\varepsilon y' = f(t, y, z, \varepsilon), \tag{2.1}$$

$$z' = g(t, y, z, \varepsilon), \tag{2.2}$$

$$b(x(0); x(1); \varepsilon) = 0. \tag{2.3}$$

Here $\varepsilon > 0$ is a small parameter, y and f have n components, z and g have m, and b is a boundary vector function of size $n+m$. The nonlinear functions f, g and b have asymptotic expansions in ε, with the coefficients being smooth functions of the other variables.

We assume that the Jacobian matrix $f_y(t, y, z, o)$ of the "fast" solution components has a regular splitting with $n_- \geqq 0$ (strictly) stable and $n_+ := n - n_- \geqq 0$ (strictly) unstable eigenvalues, for $0 \leqq t \leqq 1$ and (y, z) in an appropriate domain. This excludes turning points in the linearized problem, and it is natural then to look for a solution $x^*(t, \varepsilon) = (y^*(t, \varepsilon), z^*(t, \varepsilon))$ which has the representation

$$\left.\begin{aligned} y^*(t, \varepsilon) &= \bar{y}(t) + \mu(\tau) + \nu(\sigma) + O(\varepsilon), && (2.4) \\ z^*(t, \varepsilon) &= \bar{z}(t) + O(\varepsilon), && (2.5) \end{aligned}\right\} \quad 0 \leqq t \leqq 1.$$

Here

$$\tau = \frac{t}{\varepsilon}, \qquad \sigma = \frac{t-1}{\varepsilon}, \tag{2.6}$$

$\bar{y}(t)$ and $\bar{z}(t)$ are solutions of the reduced equations

$$\left.\begin{aligned} 0 &= f(t, \bar{y}, \bar{z}, 0), && (2.7) \\ \bar{z}' &= g(t, \bar{y}, \bar{z}, 0), && (2.8) \end{aligned}\right\} \quad 0 \leqq t \leqq 1,$$

subject to m appropriate boundary conditions, and μ and ν are left end and right end layer correction functions. They satisfy

$$\frac{d}{d\tau}\mu = f(0, \bar{y}(0) + \mu(\tau), \bar{z}(0), 0), \qquad 0 \leqq \tau < \infty, \tag{2.9}$$

$$\frac{d}{d\sigma}\nu = f(1, \bar{y}(1) + \nu(\sigma), \bar{z}(1), 0), \qquad -\infty < \sigma \leqq 0, \tag{2.10}$$

and μ and ν decay exponentially to 0 as $\tau \to \infty$, $\sigma \to -\infty$, respectively. Equations (2.7)–(2.10) arise from the representation (2.4), (2.5) by equating $O(1)$ terms with respect to ε in (2.1), (2.2).

Note that the assumption on the eigenvalues of f_y in itself does not necessarily imply the ansatz (2.4), (2.5) (as it would if (2.1)–(2.3) were a linear problem with a uniformly bounded inverse), especially if there are more than one solution to the reduced problem. However, the representation (2.4), (2.5) is of a major practical and theoretical interest.

Now, to construct the solution as in (2.4), (2.5), we substitute into the boundary conditions to obtain

$$b((\bar{y}(0)+\mu(0), \bar{z}(0)); (\bar{y}(1)+\nu(0), \bar{z}(1)); 0)=0. \tag{2.11}$$

The requirement that μ and ν decay exponentially implies that $\mu(0)$ and $\nu(0)$ must be on the stable manifolds of their corresponding equations, and we write these equations as

$$\begin{aligned} &\phi_-(\bar{y}(0), \bar{z}(0), \mu(0))=0 \qquad (n_+ \text{ eqns.}),\\ &\phi_+(\bar{y}(1), \bar{z}(1), \nu(0))=0 \qquad (n_- \text{ eqns.}). \end{aligned} \tag{2.12}$$

Thus in (2.11), (2.12) we have $2n+m$ constraints for the $4n+2m$ unknowns $\bar{x}(0)$, $\bar{x}(1)$, $\mu(0)$, $\nu(0)$.

Eliminating $\mu(0)$ and $\nu(0)$ from (2.11), (2.12) (in principle) leaves a set of m equations to be satisfied by $\bar{y}(0)$, $\bar{z}(0)$, $\bar{y}(1)$ and $\bar{z}(1)$ alone, and these are the boundary conditions for the reduced equations (2.7), (2.8) (cf. Episova [8], O'Malley [17]). Flaherty and O'Malley [9] construct the reduced boundary conditions numerically in the case where (2.1) and (2.3) are linear in y, which implies that (2.11) and (2.12) are linear in y, μ and ν.

Note that everything is much simpler when f is linear in y. Not only can the manifolds (2.12) be explicitly found, but also, and more importantly, (2.9) and (2.10) imply that μ and ν are simply decaying exponential functions. In the more general case, (2.9) and (2.10) are just general systems of ODEs.

The reduced differential equations (2.7), (2.8) plus the reduced boundary conditions form the reduced problem whose solution(s) $\bar{x}(t)=(\bar{y}(t), \bar{z}(t))$ is referred to as a reduced solution. Different reduced solutions yield different solutions to our problem (2.1)–(2.3) provided they can serve in the ansatz (2.4), (2.5). To enable this we assume that the Jacobian matrix f_y at a reduced solution has a hyperbolic splitting for all $0 \leqq t \leqq 1$.

For simplicity we also assume that at the reduced solution, $f_y(t, \bar{y}(t), \bar{z}(t), 0)$ is nondefective. Thus, there is a nonsingular (smooth) matrix function $E(t)$ such that

$$E^{-1}(t)\, f_y(t, \bar{y}(t), \bar{z}(t), 0)\, E(t)=\Lambda(t)\equiv \operatorname{diag}\{\lambda_1(t), \cdots, \lambda_n(t)\} \tag{2.13}$$

and,

$$\operatorname{re}(\lambda_j(t))\begin{cases} <0, & j=1, \cdots, n_-, \\ >0, & j=n_-+1, \cdots, n, \end{cases} \qquad 0 \leqq t \leqq 1. \tag{2.14}$$

Consider now a linearization of our problem (2.1)–(2.3) about an appropriate function $\hat{x}=(\hat{y}, \hat{z})$, which we write in operator form as

$$\mathscr{L}[\hat{x}]x=s[\hat{x}]. \tag{2.15}$$

In detail, (2.15) is written as

$$\varepsilon y' - A_{11}y - A_{12}z = s_1, \tag{2.16}$$

$$z' - A_{21}y - A_{22}z = s_2, \qquad 0 \leqq t \leqq 1, \tag{2.17}$$

$$B_0 x(0) + B_1 x(1) = \beta, \tag{2.18}$$

where

$$A_{11}(t,\varepsilon)=f_y(t,\hat{y},\hat{z},\varepsilon),\qquad A_{12}(t,\varepsilon)=f_z(t,\hat{y},\hat{z},\varepsilon),$$

(2.19)
$$A_{21}(t,\varepsilon)=g_y(t,\hat{y},\hat{z},\varepsilon),\qquad A_{22}(t,\varepsilon)=g_z(t,\hat{y},\hat{z},\varepsilon),$$

$$B_0=\frac{\partial b(x_1,x_2)}{\partial x_1},\qquad B_1=\frac{\partial b(x_1,x_2)}{\partial x_2}\quad \text{at } x_1=\hat{x}(0),\ x_2=\hat{x}(1).$$

We assume that the linear operator $\mathscr{L}[x^*]$ has a bounded inverse, independent of ε, for $0<\varepsilon\leqq\varepsilon_0$. Lipschitz continuity of $\mathscr{L}$ (which follows, e.g., if f, g, b are twice differentiable with respect to the dependent variables) then implies a similar bound for $\mathscr{L}[\hat{x}]$ at points $\hat{x}$ near x^* and, in particular, at the constructed solution (2.4), (2.5) for ε small enough.

Our problem (2.1)–(2.3) can be written in the form (2.16)–(2.19) with $\hat{x}=x^*$, simply by defining $s[x^*]$ appropriately, e.g.

$$s_1(t,\varepsilon)=f(t,y^*,z^*,\varepsilon)-A_{11}y^*-A_{12}z^*,$$

etc. The problem then looks like the one considered previously in Part II, but there is a difference: Here, the matrices A_{ij} and inhomogeneous terms s_i, $1\leqq i$, $j\leqq 2$, are not slowly varying near the interval ends, since they contain the boundary layer effects of y^*. Thus, while on the "long" interval $O(\varepsilon)<t<1-O(\varepsilon)$ away from the layers not much change is expected, in the boundary layer regions a richer solution behavior is now allowed, compared to the usual linear, variable coefficient case, as described above in connection to (2.9) and (2.10).

In Part I and Part II a layer mesh was constructed which took advantage of the known exponential decay of $\mu(\tau)$. In the nonlinear case, then, this mesh construction is less clear. Fortunately, by the exponential decay of μ and ν, the mesh construction can be applied for values of t which correspond to sufficiently large values of τ and $-\sigma$, if we know enough about the reduced solution $\bar{x}(0)$, $\bar{x}(1)$. The practical construction of a mesh in the layer regions near the boundaries is discussed in §§ 4 and 5.

3. Numerical schemes and their convergence. To solve the problem (2.1)–(2.3) numerically we use k-stage, c^0-collocation as described in § 3 of Part I and in § 3 of Part II. The same notation is adhered to here. Thus, on a given mesh

(3.1)
$$\Delta\colon 0=t_1<t_2<\cdots<t_N<t_{N+1}=1,$$
$$h_i:=t_{i+1}-t_i,\quad 1\leqq i\leqq N,\quad h:=\max_{1\leqq i\leqq N} h_i$$

the solution $x_\Delta(t)=(y_\Delta(t),z_\Delta(t))$ is a continuous piecewise polynomial vector function which satisfies the boundary conditions (2.3) and the differential equations (2.1), (2.2) at the collocation points

(3.2)
$$t_{ij}:=t_i+h_i\rho_j,\qquad 1\leqq i\leqq N,\quad i\leqq j\leqq k.$$

The points $0\leqq\rho_1<\cdots<\rho_k\leqq 1$ are chosen to be the Gauss or Lobatto points. This gives the difference scheme, (for $1\leqq i\leqq N$, $2-r\leqq j\leqq k+r$)

(3.3)
$$\varepsilon h_i^{-1}(y_{ij}-y_i)=\sum_{l=1}^{k}\hat{a}_{jl}f(t_{il},y_{il},z_{il},\varepsilon),$$

(3.4)
$$h_i^{-1}(z_{ij}-z_i)=\sum_{l=1}^{k}\hat{a}_{jl}g(t_{il},y_{il},z_{il},\varepsilon),$$

(3.5)
$$b(x_1;x_{N+1};\varepsilon)=0,$$

for $x_i = x_\Delta(t_i)$, $x_{ij} = x_\Delta(t_{ij})$, where $\hat{a}_{jl}$ are known constants and $x_{i,k+1} = x_{i+1}$. For Gauss points, $r=1$, $\rho_1 > 0$, $\rho_k < 1$ (so mesh points are not collocation points) and $\hat{a}_{k+1,l} = \hat{b}_l$, $l = 1, \cdots, k$. The simplest of these schemes, with $k=1$, is the midpoint rule

$$\begin{aligned} \varepsilon h_i^{-1}(y_{i+1}-y_i) &= f(t_{i+1/2}, y_{i+1/2}, z_{i+1/2}, \varepsilon), \\ h_i^{-1}(z_{i+1}-z_i) &= g(t_{i+1/2}, y_{i+1/2}, z_{i+1/2}, \varepsilon) \end{aligned} \tag{3.6}$$

where $t_{i+1/2} = t_i + \frac{1}{2}h_i$, $y_{i+1/2} = \frac{1}{2}(y_i + y_{i+1})$ and $z_{i+1/2} = \frac{1}{2}(z_i + z_{i+1})$. For Lobatto points, $r=0$, $\rho_1 = 0$ and $\rho_k = 1$. Thus the mesh points t_i are collocation points. The simplest of these schemes, with $k=2$, is the trapezoidal rule

$$\begin{aligned} \varepsilon h_i^{-1}(y_{i+1}-y_i) &= \tfrac{1}{2}(f(t_i, y_i, z_i, \varepsilon) + f(t_{i+1}, y_{i+1}, z_{i+1}, \varepsilon)), \\ h_i^{-1}(z_{i+1}-z_i) &= \tfrac{1}{2}(g(t_i, y_i, z_i, \varepsilon) + g(t_{i+1}, y_{i+1}, z_{i+1}, \varepsilon)). \end{aligned} \tag{3.7}$$

Denote by ψ^c the restriction of a function $\psi(t)$ to $\Delta \cup \{t_{ij};\ 1 \leqq i \leqq N,\ 1 \leqq j \leqq k\}$. Equations (3.3)–(3.5) form a nonlinear algebraic system for x_Δ^c, which we attempt to solve by Newton's method. Equivalently, and more naturally for implementation, the quasilinearization can be done before discretization. Thus, given an initial guess $x_\Delta^0(t)$, a sequence of iterates $x_\Delta^0(t), x_\Delta^1(t), \cdots, x_\Delta^j(t), \cdots$ is generated as follows: With $x_\Delta^j(t)$ known, define

$$x_\Delta^{j+1}(t) := x_\Delta^j(t) + \xi_\Delta(t) \tag{3.8}$$

where $\xi_\Delta(t)$ is the collocation solution of the linear problem

$$\mathscr{L}[x_\Delta^j]\xi = s[x_\Delta^j] \tag{3.9}$$

with

$$s_1 = f(t, y_\Delta^j, z_\Delta^j, \varepsilon) - \varepsilon(y_\Delta^j)', \tag{3.10}$$

$$s_2 = g(t, y_\Delta^j, z_\Delta^j, \varepsilon) - (z_\Delta^j)', \tag{3.11}$$

$$\beta = b(x_\Delta^j(0); x_\Delta^j(1); \varepsilon) \tag{3.12}$$

(cf. (2.16)–(2.19)).

The formulation of (3.9) as a difference scheme is similar to (3.3)–(3.5) except that the resulting algebraic equations for ξ_Δ^c are linear and can be written as

$$\mathscr{L}_\Delta[x_\Delta^j]\xi_\Delta^c = s^c[x_\Delta^j] \tag{3.13}$$

with $\mathscr{L}_\Delta$ a possibly large, sparse matrix (see Part II).

Key questions regarding the use of our schemes are the definition of suitable meshes Δ, the existence of solutions to (3.3)–(3.5), their approximation properties with respect to the exact solution of the boundary value problem and the convergence of iterative methods, in particular of Newton's method. The question of convergence of Newton's method is, of course, closely related to that of the stability of the linearized difference operator. Our results regarding these questions are summarized in the following theorem.

THEOREM. *Assume the following*:

(a) *The boundary value problem satisfies the assumptions of* § 2. *Denote by* $S_r(x) = \{u \in C[0,1];\ \|u - x\| \leqq r\}$ *a sphere with radius* $r > 0$ *around* $x \in C[0,1]$.

(b) *The matrix condition*

$$\det \begin{bmatrix} P_- E^{-1}(0) \\ P_+ E^{-1}(1) \end{bmatrix} \neq 0 \tag{3.14}$$

holds, where $E(t)$ has been defined in (2.13) *and*

(3.15) $$P_- = [I \quad 0] \in \mathcal{R}^{n_- \times n}, \qquad P_+ = [0 \quad I] \in \mathcal{R}^{n_+ \times n},$$

I being appropriate identity matrices.

(c) *The following mesh construction is used: For a given tolerance δ, $0 < c\varepsilon \leqq \delta < 1$, near $t = 0$ define $0 = t_1 < t_2 < \cdots < t_{N_0+1}$ by*

(3.16) $$h_i := \begin{cases} \varepsilon c_u \delta^{1/p}, & t_{i+1} \leqq \gamma\varepsilon \leqq T_0\varepsilon, \\ h_{i-1} \exp\left\{\dfrac{1}{p}\dfrac{\nu}{\varepsilon} h_{i-1}\right\}, & \gamma\varepsilon < t_{i+1} \leqq T_0\varepsilon, \end{cases}$$

where

(3.17) $$p = \begin{cases} 2k, & k\text{-stage Gauss scheme}, \\ 2(k-1), & k\text{-stage Lobatto scheme}, \end{cases}$$

(3.18) $$T_0 = \nu^{-1} \ln \delta^{-1},$$

(3.19) $$\nu = \min\{-\mathrm{re}\,(\lambda_j(0)), j = 1, \cdots, n_-\} > 0$$

and c, c_u and γ are positive constants, independent of ε and δ. A similar formula is used near $t = 1$ to construct $(1 >) t_N > \cdots > t_{N-N_1+1} (\geqq 1 - T_1\varepsilon)$ based on the eigenvalues $\lambda_{n_-+1}(1), \cdots, \lambda_n(1)$. In between, a much sparser mesh is used. Let

(3.20) $$\kappa := \varepsilon \underline{h}^{-1}(\bar{i} - \underline{i})$$

where

(3.21) $$\underline{h} := \min\{h_i, \underline{i} \leqq i < \bar{i}\}, \quad \underline{i} := N_0 + 1, \quad \bar{i} := N - N_1 + 1.$$

Then, for each scheme of the class considered there are positive constants ε_0, δ_0, h_0, r_0, κ_0, c and $K \geqq 0$, independent of ε and Δ, such that there is a unique isolated solution $x_\Delta \in S_{r_0}(x^*)$ provided that $0 < \varepsilon \leqq \varepsilon_0$, $\delta \leqq \delta_0$, $h \leqq h_0$ and $\kappa \leqq \kappa_0$. Further, Newton's method converges quadratically to this solution provided that the starting iterate x_Δ^0 satisfies $x_\Delta^0 \in S_{r_1}(x^*)$, with r_1 sufficiently small. Finally, the numerical solution satisfies

(3.22a) $$\|x_\Delta(t_i) - x^*(t_i)\| \leqq c(e + \delta), \qquad 1 \leqq i \leqq N + 1,$$

where e stands for the following:

(3.23) $$e = \begin{cases} h^{k+q}, & k\text{-stage Gauss scheme}, \\ Kh^p + \varepsilon h^{k-1+q}, & k\text{-stage Lobatto scheme}. \end{cases}$$

In (3.23), *$K = 0$ if $m = 0$, p is defined in* (3.17) *and $q = 1$ if the mesh is locally almost uniform, i.e.,*

(3.24) $$h_{i+1} = h_i(1 + O(h_i)) \quad \text{for all } i \text{ odd } \underline{\text{or}} \text{ all } i \text{ even}, \quad \underline{i} \leqq i < \bar{i},$$

and k is odd For Gauss, even for Lobatto schemes; otherwise, $q = 0$. Further, improved estimates for the slow components z are obtained when, up to $O(\varepsilon)$, the boundary conditions (2.18) *contain a subset of m linearly independent conditions involving z alone. In this case the error at mesh points is bounded by*

(3.22b) $$\|z_\Delta(t_i) - z^*(t_i)\| \leqq c(h^p + \varepsilon \underline{h}^{-1}(e + \delta))$$

for Gauss schemes and

$$\|z_\Delta(t_i)-z^*(t_i)\| \leqq c(h^p+h\delta) \tag{3.22c}$$

for Lobatto schemes.

Before giving an outline to the proof of the Theorem, we wish to remark on some of its details, so that a reader with a primary interest in the algorithm can skip the proof.

The condition (3.14) is a restriction on the differential problem which has nothing to do with its well-posedness. It is a limitation on the applicability of our numerical schemes with full success and is needed to guarantee the stability of the discretization process on the interval $[t_{\underline{i}}, t_{\bar{i}}]$. There, $h_i \gg \varepsilon$ and thus the difference operator does not closely approximate the differential operator any more. Computational difficulties can arise when (3.14) is violated, as discussed in Weiss [22] and in Part II.

The principle underlying the layer mesh definition (3.16) is that of keeping the error at the mesh points below the tolerance δ by approximating functions of the type $\exp\{-\lambda_j(0)t/\varepsilon\}$, $j=1,\cdots,n_-$, which determine the decay in the boundary layer; see Part II. Of course, to be really constructive one needs to pin down the constants in (3.16) and to provide a working estimate of ν of (3.19). This is done in the next section.

Finally, it is clear that the size of the constant r_1 determining how close x_Δ^0 has to be to x_Δ for Newton's method to converge is very important practically. Unfortunately, our result in this respect is somewhat incomplete: While we can show that for Lobatto schemes r_1 can be chosen independently of ε and Δ, for Gauss schemes our analysis leads to the condition that r_1 shrinks like $(\bar{i}-\underline{i})^{-1}$, as the mesh on $[t_{\underline{i}}, t_{\bar{i}}]$ becomes dense. However, in practice we have never experienced a difference in the domain of attraction of the Newton iteration for the two types of schemes.

Since the proof of the Theorem is loaded with technicalities, we proceed to give only an outline of it, in an attempt to keep the paper readable.

Outline of the proof of the theorem. We consider a k-stage Lobatto scheme and remark about Gauss schemes at the end of this section. Let c denote a generic constant. Also, all norms appearing in the sequel are appropriately restricted maximum norms. As a first step, consider the application of the collocation scheme to the linearized problem at the exact solution x^* of (2.1)–(2.3),

$$\mathscr{L}[x^*]x = s[x^*] \tag{3.25}$$

(recall (2.16)–(2.19)), where the right-hand side functions are

$$\begin{aligned} s_1(t) &= f(t, y^*(t), z^*(t), \varepsilon) - A_{11}(t)y^*(t) - A_{12}(t)z^*(t), \\ s_2(t) &= g(t, y^*(t), z^*(t), \varepsilon) - A_{21}(t)y^*(t) - A_{22}(t)z^*(t) \end{aligned} \tag{3.26}$$

with the ε dependence of the functions involved omitted for brevity. The solution of (3.25), (3.26) is of course x^* as well.

The collocation scheme for (3.25), (3.26) is, for $1 \leqq i \leqq N$, $2 \leqq j \leqq k$,

$$\varepsilon h_i^{-1}(y_{ij}-y_i) - \sum_{l=1}^{k} \hat{a}_{jl}(A_{11}(t_{il})y_{il} + A_{12}(t_{il})z_{il}) = \sum_{l=1}^{k} \hat{a}_{jl}s_1(t_{il}) \equiv r_{ij}, \tag{3.27a}$$

$$h_i^{-1}(z_{ij}-z_i) - \sum_{l=1}^{k} \hat{a}_{jl}(A_{21}(t_{il})y_{il} + A_{22}(t_{il})z_{il}) = \sum_{l=1}^{k} \hat{a}_{jl}s_2(t_{il}) \equiv s_{ij}, \tag{3.27b}$$

$$B_0x_1 + B_1x_{N+1} = \beta. \tag{3.27c}$$

We now derive stability and convergence results for (3.27). Systems of this form have been investigated in Part II, with only one essential difference: There the matrices A_{ij}

were assumed to be smooth functions of the slow variable t for $0 \leqq t \leqq 1$, which is not the case here. Still, the key idea of the treatment of Part II can be imported here: To establish unique solvability of (3.27) we consider the discrete system separately at first on the three intervals $[0, t_{\underline{i}}]$, $[t_{\underline{i}}, t_{\bar{i}}]$ and $[t_{\bar{i}}, 1]$, subject to the following special boundary conditions:

I. On $[0, t_{\underline{i}}]$,

$$C_- y_1 = \alpha_{\mathrm{I}}, \quad z_1 = \beta_{\mathrm{I}}, \quad P_+ E^{-1}(t_{\underline{i}}) y_{\underline{i}} = \gamma_{\mathrm{I}};$$

II. On $[t_{\underline{i}}, t_{\bar{i}}]$,

$$P_- E^{-1}(t_{\underline{i}}) y_{\underline{i}} = \alpha_{\mathrm{II}}, \quad z_{\underline{i}} = \beta_{\mathrm{II}}, \quad P_+ E^{-1}(t_{\bar{i}}) y_{\bar{i}} = \gamma_{\mathrm{II}};$$

III. On $[t_{\bar{i}}, 1]$,

$$P_- E^{-1}(t_{\bar{i}}) y_{\bar{i}} = \alpha_{\mathrm{III}}, \quad z_{\bar{i}} = \beta_{\mathrm{III}}, \quad C_+ y_{N+1} = \gamma_{\mathrm{III}}.$$

Here, $\alpha_{\mathrm{I,II,III}} \in \mathscr{R}^{n_-}$, $\gamma_{\mathrm{I,II,III}} \in \mathscr{R}^{n_+}$ and $\beta_{\mathrm{I,II,III}} \in \mathscr{R}^m$ are arbitrary parameters. The matrices C_- and C_+ are chosen so that the problems

$$\frac{d}{d\tau}\xi = f_y(0, \bar{y}(0) + \mu(\tau), \bar{z}(0), 0)\xi, \qquad 0 \leqq \tau < \infty,$$

$$C_- \xi(0) = 0, \qquad \lim_{\tau \to \infty} \xi(\tau) = 0,$$

and

$$\frac{d}{d\sigma}\zeta = f_y(1, \bar{y}(1) + \nu(\sigma), \bar{z}(1), 0)\zeta, \qquad -\infty < \sigma \leqq 0,$$

$$C_+ \zeta(0) = 0, \qquad \lim_{\sigma \to -\infty} \zeta(\sigma) = 0,$$

which result from linearizing the layer equations, have only the trivial solutions.

On the "long" interval $[t_{\underline{i}}, t_{\bar{i}}]$ the procedure of Part II is immediately applicable and Lemma 5.2 there yields, with (3.14), unique solvability of the problem for all parameters α_{II}, β_{II}, γ_{II} and s_Δ, r_Δ. (The latter two are abitrary right-hand sides in (3.27a) and (3.27b), respectively.) Also, it yields the explicit dependence of the solution on the parameters, and since κ of (3.20) is sufficiently small, the bounds

$$\|x^c_\Delta\|_{\mathrm{II}} \leqq c(\|\alpha_{\mathrm{II}}\| + \|\beta_{\mathrm{II}}\| + \|\gamma_{\mathrm{II}}\| + (\bar{i} - \underline{i})\|r^c_\Delta\|_{\mathrm{II}} + \|s^c_\Delta\|_{\mathrm{II}}), \tag{3.28}$$

$$\|x^c_\Delta\|_{\mathrm{II}} \leqq c(\|\alpha_{\mathrm{II}}\| + \|\beta_{\mathrm{II}}\| + \|\gamma_{\mathrm{II}}\| + \|s^c_{1\Delta}\|_{\mathrm{II}} + \|s^c_\Delta\|_{\mathrm{II}}). \tag{3.29}$$

On the layer interval $[0, t_{\underline{i}}]$, where the $A_{ij}(t)$ vary like $\mu(t/\varepsilon)$, we can employ the results of Markowich and Ringhofer [16] who treat the layer equations in the variable $\tau = t/\varepsilon$ with fast components only. Their mesh construction is as in (3.16) (with an insignificant modification in (3.19)), and a contraction argument, based on the fact that $t_{\underline{i}} \ll 1$, allows the inclusion of slow variables as well. Thus we obtain the unique solvability of the problem, a representation of the solution in terms of the parameters α_{I}, β_{I} and γ_{I} and the bound

$$\|x^c_\Delta\|_{\mathrm{I}} \leqq c(\|\alpha_{\mathrm{I}}\| + \|\beta_{\mathrm{I}}\| + \|\gamma_{\mathrm{I}}\| + \|s^c_\Delta\|_{\mathrm{I}} + \|r^c_\Delta\|_{\mathrm{I}}). \tag{3.30}$$

An analogous situation occurs for the boundary layer at the other end.

The next step is to patch the solutions obtained above together, to obtain a solution of (3.27) by requiring that the representations be identical at $t_{\underline{i}}$ and $t_{\bar{i}}$ and that the boundary conditions (3.27c) be satisfied. This results in a linear system of equations for the parameters, of dimension $3(n+m)$. Due to assumption (a) of the Theorem, the matrix involved has a bounded inverse; the details follow closely those of Weiss [22]. Hence we finally obtain for (3.27)

$$\|x_\Delta^c\| \leqq c(\|\beta\| + \|r_\Delta^c\|_{\mathrm{I}} + (\bar{i}-\underline{i})\|r_\Delta^c\|_{\mathrm{II}} + \|r_\Delta^c\|_{\mathrm{III}} + \|s_\Delta^c\|), \tag{3.31}$$

$$\|x_\Delta^c\| \leqq c(\|\beta\| + \|s_{1\Delta}^c\| + \|s_\Delta^c\|). \tag{3.32}$$

Having thus established stability for (3.27), we turn to convergence. The solutions of each of the three discrete problems are approximations to the (general) solutions of the continuous problem (3.25) on the three intervals, subject to the special boundary conditions. The relevant convergence results are described in Markowich and Ringhofer [16] for the layer intervals and in Part II for the "long" interval in between. Using these error estimates in the patching procedure, we obtain the following error estimates relating (3.25)–(3.26) and (3.27). At mesh points,

$$\|x_i - x^*(t_i)\| \leqq c(\delta + e), \qquad i = 1, \cdots, N+1, \tag{3.33}$$

while at collocation points other than mesh points,

$$\|x_{ij} - x^*(t_{ij})\| \leqq c(\delta^{k/p} + Kh^k + \varepsilon h^{k-1}), \qquad i = 1, \cdots, N, \quad j = 2, \cdots, k-1. \tag{3.34}$$

This completes the description for the linearized problem. After this preparation we turn to the analysis of the nonlinear scheme (3.3)–(3.5). We employ the contraction mapping principle, the application of which to a nonlinear problem

$$u = \mathcal{N}(u) \tag{3.35}$$

proceeds in two main steps:

(i) defining an approximate solution $\hat{u}$ of the problem which leads to a small residual $\hat{u} - \mathcal{N}(\hat{u})$, and

(ii) obtaining a sufficiently small bound on the Lipschitz constant of $\mathcal{N}$ in a vicinity of $\hat{u}$.

To put our discrete nonlinear system in the form (3.25), we write it as

$$\begin{aligned} &\varepsilon h_i^{-1}(y_{ij} - y_i) - \sum_{l=1}^{k} \hat{a}_{jl}(A_{11}(t_{il})y_{il} + A_{12}(t_{il})z_{il}) \\ &\quad = \sum_{l=1}^{k} \hat{a}_{jl}(f(t_{il}, y_{il}, z_{il}, \varepsilon) - A_{11}(t_{il})y_{il} - A_{12}(t_{il})z_{il}), \end{aligned} \tag{3.36a}$$

$$1 \leqq i \leqq N, \quad 2 \leqq j \leqq k,$$

$$\begin{aligned} &h_i^{-1}(z_{ij} - z_i) - \sum_{l=1}^{k} \hat{a}_{jl}(A_{21}(t_{il})y_{il} + A_{22}(t_{il})z_{il}) \\ &\quad = \sum_{l=1}^{k} \hat{a}_{jl}(g(t_{il}, y_{il}, z_{il}, \varepsilon) - A_{21}(t_{il})y_{il} - A_{22}(t_{il})z_{il}), \end{aligned} \tag{3.36b}$$

$$B_0x_1 + B_1x_{N+1} = B_0x_1 + B_1x_{N+1} - b(x_1; x_{N+1}; \varepsilon), \tag{3.37}$$

where the matrices A_{rs}, B_r are as above. In concise form, (3.36), (3.37) are written as

$$\mathcal{L}_\Delta[x^*]x_\Delta = F(x_\Delta). \tag{3.38}$$

By (3.31) we know that $\mathscr{L}_\Delta^{-1}[x^*]$ exists, whence we write (3.38) as

(3.39) $$x_\Delta = \mathscr{L}_\Delta[x^*]^{-1}F(x_\Delta) \equiv \mathcal{N}(x_\Delta),$$

which is of the type (3.35). Next we establish a contraction argument as follows.

As an approximate solution $\hat{x}_\Delta$ of (3.36)–(3.37) we choose the solution of (3.27). When substituting $\hat{x}_\Delta$ into (3.36) a residual $\tilde{d}_{ij}$ of the form

$$\tilde{d}_{ij} = \sum_{l=1}^{k} \hat{a}_{jl} d_{il}, \qquad 1 \leqq i \leqq N, \quad 2 \leqq j \leqq k,$$

is obtained, where the vector d_Δ^c formed from the d_{ij} values is bounded in norm by the right-hand term of (3.34). Hence, by (3.32)

(3.40) $$\|\hat{x}_\Delta - \mathcal{N}(\hat{x}_\Delta)\| \leqq c(\delta^{k/p} + Kh^k + \varepsilon h^{k-1}).$$

So, $\hat{x}_\Delta$ produces a small residual of (3.39), as desired.

Further, it is clear that due to the smoothness of f, g and b as functions of x and due to (3.32), the Lipschitz constant of $\mathcal{N}_\Delta$ in a suitably restricted sphere about $\hat{x}_\Delta$ can be made sufficiently small. The contraction mapping principle then yields the existence and uniqueness of a solution x_Δ of (3.39) in this sphere, and the bound

(3.41) $$\|x_\Delta - \hat{x}_\Delta\| \leqq c\|\hat{x}_\Delta - \mathcal{N}(\hat{x}_\Delta)\|.$$

Combining (3.41), (3.40) and (3.33), (3.34) we finally obtain the convergence result at collocation points for (3.3)–(3.5),

(3.42) $$\|x_{ij} - x^*(t_{ij})\| \leqq c(\delta^{k/p} + Kh^k + \varepsilon h^{k-1}), \qquad 1 \leqq i \leqq N, \quad 1 \leqq j \leqq k.$$

It is now easy to see that Newton's method can be applied to (3.39) with quadratic convergence. Further, Newton's method is clearly invariant under the transformation that carries (3.38) to (3.39), so the quadratic convergence result applies to the scheme actually employed in practice.

The result (3.42) corresponds to the global convergence estimate for Lobatto schemes in the usual (not singularly perturbed) case. To obtain the corresponding superconvergence results at mesh points, note that

(3.43) $$\|x^{*c} - x_\Delta^c\| \leqq \|x^{*c} - \hat{x}_\Delta^c\| + \|\hat{x}_\Delta^c - x_\Delta^c\|.$$

To bound the second term in this inequality, we compare (3.36) to (3.27), apply a Taylor expansion to the right-hand side of (3.36) and utilize (3.32) once again to yield

(3.44) $$\|\hat{x}_\Delta^c - x_\Delta^c\| \leqq c\|x^{*c} - x_\Delta^c\|^2.$$

Hence from (3.33), (3.42), (3.43) and (3.44) we finally obtain the desired result (3.22a). The sharper estimates (3.23c) for the slow components follow similarly from corresponding estimates in the linear case; see Theorem 3.3 of Part II. This completes the proof for Lobatto-type schemes.

The analysis for Gauss-type schemes is significantly less pleasant. The basic reason is that special favorable things occur at collocation points which, in the case of a Gauss scheme, do not include the mesh points. This property, which is actually welcome in some applications (because it allows for a slick implementation for problems with discontinuous coefficients or problems with artificial singularities), causes here weaker convergence and stability properties than those enjoyed by Lobatto schemes, and a harder analysis to prove them. In particular, weaker convergence properties are already evident in the desired estimates (3.22a), (3.23) (which are sharp for the fast solution components); a factor of $\bar{i}-\underline{i}$ creeps into the stability estimate (3.29) (hence (3.32)); and the patching procedure at $t_{\underline{i}}$ and $t_{\bar{i}}$ is harder to justify because these are not collocation points any more.

Using a more elaborate analysis for Gauss-type schemes, we were able to show existence of a discrete solution, unique in a sphere about the exact solution, and the convergence estimates (3.22), (3.23). The convergence of Newton's method, however, is guaranteed in our analysis only when the starting approximation is already in a sphere about x_Δ whose radius shrinks like $(\bar{i}-\underline{i})^{-1}$.

4. Mesh construction. In this section we discuss the practical mesh construction in the layer region $[0, T_0\varepsilon]$, where T_0 is given by (3.18), (3.19). An analogous construction holds, of course, for the right end layer region $[1-T_1\varepsilon, 1]$, while in between a sparse mesh, fine enough only to approximate the reduced solution, is used.

The purpose of the mesh selection is to obey a uniform error tolerance δ (which is considered as an estimate, not a bound), and the strategy is to equidistribute the error with respect to $\mu(\tau)$, which is the dominant solution component in this region; see (2.4), (2.5). This is already the strategy behind the definition (3.16) and we wish here to somewhat refine and precisely specify this selection.

The proposed mesh construction is as follows:

$$(4.1)\qquad h_1 := \frac{\varepsilon}{\lambda}\left[\frac{\nu}{\lambda|c_\gamma|}\right]^{1/p}\delta^{1/p}, \qquad \lambda := \max\{|\lambda_j(0)|, j=1,\cdots,n_-\},$$

$$(4.2)\qquad h_i := h_{i-1}\exp\left\{\frac{\nu}{p\varepsilon}h_{i-1}\right\}, \qquad i=2,\cdots,N_0 \quad \text{until } t_{N_0} \leqq T_0\varepsilon \leqq t_{N_0+1}.$$

Here p is defined in (3.17), c_γ is a known constant depending on p and defined in Part I, and ν is a slight modification of (3.19), to be discussed below.

The mesh selection strategy (4.1), (4.2) can be easily seen to be at least as conservative as (3.16) for suitable constants c_u and γ. The number of mesh points N_0 obtained by this construction is independent of ε and is proportional to $\delta^{1/p}$; see Theorem 4.2 of Part II. Comparing (4.1), (4.2) to the mesh (3.46), (3.47) of Part II, we see that they are essentially the same. The difference is that ν here cannot be determined on the basis of the eigenvalues of $A_{11}(0,0)$.

Since we do not really wish to always compute the reduced solution, for the practical evaluation of ν (at "$\tau\to\infty$", which is only $O(\varepsilon|\ln\delta|)$ away from (3.19)) we can calculate the eigenvalues of $A_{11}=f_y$ at $t=T_0\varepsilon$, say. These eigenvalues will, of course, depend on the currently available approximation to the reduced solution. Thus, a strategy blending the nonlinear Newton iterations with mesh refinement suggests itself. Luckily, however, the solution is not very sensitive to the exact location of the layer mesh points, so re-evaluation of the eigenvalues (and the corresponding redefinition of the layer mesh) is usually not needed more than once.

Consider the function $\mu(\tau)$, with respect to which we are choosing the mesh. In (4.2), we are simply capitalizing on its known exponentially decaying behavior for τ large enough. If f of (2.1) is linear in y then (2.9) implies that $\mu(\tau)$ has that known exponential behavior right from $\tau=0$. Thus, in Part II we have used the "exponential mesh" throughout the layer region. In the more general nonlinear case, what one needs, strictly speaking, is a general nonstiff ODE error control for (2.9), of which the first constant steps in (3.16) are a primitive instance. However, a sophisticated error control can hardly be justified here and, in fact, the mesh (4.1), (4.2) can be used as well to obtain an error proportional to δ, with a moderate constant of proportionality.

5. Numerical examples. In order to test the theoretical results numerically and to demonstrate the power of the obtained schemes, a computer program was written. Newton's method of quasilinearization is implemented and the linearized problems are solved by collocation using local parameter elimination, as described in § 3 of Part II. The mesh in possible layer regions is automatically constructed using (4.1)–(4.2), as discussed in the previous section, with the initial solution profile, provided by the user, being used to calculate the eigenvalues at the two boundary points.

That initial solution is an approximation to the reduced solution and is expected to be smooth near the boundaries. Note that the knowledge and use of the reduced solution as an initial guess for Newton's method does not guarantee its convergence. However, the constructed mesh is right and so Newton's method usually converges in practice.

The input tolerance δ is used to control both the layer meshes construction and the convergence of the nonlinear iteration. Optionally, the condition numbers of the matrices $\mathcal{L}_\Delta$ of (3.13) are calculated. The emphasis in the implementation was on flexibility, rather than efficiency; the efficient implementation of these schemes will be discussed elsewhere.

For the calculations reported below, a floating point system with 14-hexadecimal-digit mantissa was used.

Example 1 (*Carrier* [6], *Chin and Krasny* [7]). Consider the problem

$$\varepsilon^2 u'' = 1 - 2b(1-t^2)u - u^2, \qquad -1 \leqq t \leqq 1, \tag{5.1}$$

$$u(-1) = u(1) = 0, \tag{5.2}$$

where $b \geqq 0$ is a parameter. The reduced solution about which the representation (2.4), (2.5) makes sense is

$$\bar{u}(t) = -b(1-t^2) - \sqrt{b^2(1-t^2)^2 + 1}. \tag{5.3}$$

To convert to a first-order system, set

$$y_1 = u, \qquad y_2 = \varepsilon u'. \tag{5.4}$$

Using symmetry we then obtain the problem

$$\varepsilon y_1' = y_2, \qquad 0 \leqq t \leqq 1, \tag{5.5}$$

$$\varepsilon y_2' = 1 - 2b(1-t^2)y_1 - y_1^2, \tag{5.6}$$

$$y_2(0) = 0, \qquad y_1(1) = 0. \tag{5.7}$$

Thus we have only fast components and the eigenvalues of f_y are

$$\lambda_2(t) = \sqrt{-2(y_1(t) + b(1-t^2))}, \qquad \lambda_1(t) = -\lambda_2(t). \tag{5.8}$$

So, at the reduced solution (5.3), $\lambda_1(t)$ and $\lambda_2(t)$ are real and stay away from 0. Also, $\nu = \sqrt{2}$ in what corresponds to (3.19) for the boundary layer at the right end. (Clearly there is a boundary layer only near $t = 1$.) Note that the boundary conditions (5.2) imply that, evaluated at the exact solution, the eigenvalues of the Jacobian matrix vanish at $t = 1$. This, however, does not cause analytic or computational difficulties for our procedures.

Numerical solutions were calculated for $b = 0, 1$, and $\varepsilon = 10^{-2}, 10^{-3}, 10^{-6}, 10^{-10}$. Some typical values are listed in Table 1. The results in [6], [7] were verified. The number of mesh points, the condition numbers of $\mathcal{L}_\Delta$ and the number of nonlinear iterations needed, all were found to be essentially independent of ε and of b for the

TABLE 1
Selected solution values for Example 1 *with* $b=1$.

ε	$u(0)$	$\varepsilon u'(1)$
10^{-2}	-2.414093	1.174918
10^{-3}	-2.414212	1.156703
10^{-6}	-2.414214	1.154703
10^{-10}	-2.414214	1.154701

above range of parameters. With $\delta=10^{-6}$ and 10 uniform subintervals away from the layers, we tried a number of initial guesses x_Δ^0: With the reduced solution as x_Δ^0, 3 iterations were needed for convergence on a mesh with $N=28$, automatically constructed for a 4-stage Lobatto scheme.

With $x_\Delta^0\equiv\binom{-2}{0}$, which gives an $O(1)$ perturbation to the eigenvalues λ_1, λ_2 used to construct the mesh, a mesh with a similar structure was constructed and nonlinear convergence took 4 iterations. Results in both cases were indeed accurate to within δ. On the other hand, with $x_\Delta^0\equiv 0$ the mesh construction produced an inadequate uniform mesh of 10 subintervals, because $\lambda_2\approx 0$ near $t=1$. Thus, while the process is not very sensitive to inaccuracies in the profile and end values of the reduced solution, of course not every initial guess automatically produces a suitable mesh and some care is needed in the design of the initial solution profile. This observation is even more pronounced in the next example.

Example 2 (*Flaherty and O'Malley* [9]). This example demonstrates that finding the reduced solutions of a problem may help in more ways than one. Here, multiple solutions are detected. Consider the problem

(5.9) $$\varepsilon y_1'=y_2,$$

(5.10) $$\varepsilon y_2'=\alpha^2(z)y_1+\beta(z),\qquad 0\leqq t\leqq 1,$$

(5.11) $$z'=-z+1,$$

(5.12) $$z(0)+y_1(0)=0,\quad -bz(0)+y_2(0)=0,\quad z(1)+y_1(1)=0,$$

where

(5.13) $$\alpha(z)=1+2z,\qquad \beta(z)=8z(1-z)$$

and b is a parameter. Note that the nonlinearities appear as functions of the slow component alone.

Clearly, the eigenvalues at the reduced solution are

(5.14) $$\lambda_1=\alpha(\bar z(t)),\qquad \lambda_2=-\lambda_1.$$

Also, the reduced solution is given by

(5.15) $$\bar y_1(t)=-\frac{8\bar z(t)(1-\bar z(t))}{(1+2\bar z(t))^2},\quad \bar y_2(t)=0,\quad \bar z(t)=1+e^{-t}[\bar z(0)-1].$$

It is much less easy to see what values $\bar z(0)$ may take. One could experimentally use (5.15) with a variety of values for $\bar z(0)$ as initial approximations for (5.9)–(5.12). However, using the technique described in [9], Flaherty and O'Malley obtained that $z(0)$ may have precisely the following three values:

(5.16) $$z(0)=0,\quad \tfrac14[bs-6\pm((bs-4)^2+48)^{1/2}],\quad s=\operatorname{sign}(\alpha(\bar z(0))).$$

The entire construction holds, by (5.14), (5.15), only if

$$(5.17) \qquad \lambda_1(t)=3+2(\bar{z}(0)-1)e^{-t}\neq 0, \qquad 0\leqq t\leqq 1.$$

Following [9] we have calculated 3 solutions for each of the parameter values $b=2, 0, -2$. In [9], the general purpose code COLSYS [2], which implements collocation at Gaussian points, was used with the reduced solution as the initial guess. The authors had some difficulties in carrying out the calculations for small ε, and continuation in ε was needed (see [9], [10]). The reason for these difficulties (as noted by the authors themselves in private communication) is that a uniform mesh was initially used, before allowing COLSYS to adapt it for a given problem. Thus, the approximate solution on the initial mesh had large oscillations throughout the interval [0, 1] and was not close in norm to either the exact differential solution or the initial guess.

Here, using the a priori graded meshes described above, we have encountered no difficulty at all for all cases where (5.17) holds, even with very small ε. No continuation in ε was needed. Using $\delta=10^{-6}$, the mesh construction of § 4 with 10 uniform subintervals away from the boundaries and the reduced solution for the initial guess, solutions were calculated with 3 Gauss, 5 Gauss and 4 Lobatto points per subinterval. The first two choices of collocation points were used by Flaherty and O'Malley [9]. The Lobatto scheme with $k=4$ has, by (4.1), (4.2), the same mesh construction and computational cost as the Gauss scheme with $k=3$, while by the Theorem, its accuracy in h away from the layers is 6, the same as that of the Gauss scheme with $k=5$. It is therefore interesting to compare the actual performance of these 3 schemes.

Computing solutions for $\varepsilon=10^{-3}$, 10^{-6}, 10^{-12}, we have found that, as in the previous example, the number of mesh points ($N=28$ for $k=3$, $N=18$ for $k=5$ Gauss points), the condition numbers of $\mathcal{L}_\Delta$ and the number of nonlinear iterations needed (usually one) were essentially independent of ε and b, as long as (5.17) holds. The Lobatto scheme was particularly accurate for some cases, notably of the negative values of $\bar{z}(0)$ for $b=2$ and $b=0$. To understand why, consider the error term e in (3.23). For the Lobatto scheme $e=Kh^6+\varepsilon h^4$ and the constant K arises from the approximation of the slow components z. Here z is very smooth and is approximated very well. Thus K is very small and, when ε is very small, the error e of (3.23) is very small.

In order to measure computational errors approximately, we have used the reduced solutions for very small ε away from the layers, and additional calculations with denser meshes, to obtain "exact solutions". We have subsequently verified for the above calculations that the layer error tolerance δ has been met to an order of magnitude.

The negative values of $\bar{z}(0)$ provide the more challenging cases. It can be verified that $\lambda_1(t)$ has a zero at

$$(5.18) \qquad \bar{t}=\ln \tfrac{1}{6}(b+10+\sqrt{(b+4)^2+48}),$$

and so we have a turning point at $\bar{t}$ if $0<\bar{t}<1$, and the theory then breaks down. This is the case for $b=-2$ and we note in passing that, while solutions now become unbounded as $\varepsilon\to 0$, for values of ε which are not extremely small solutions can still be calculated using COLSYS, as pointed out by Flaherty and O'Malley [9]. For $b=0$, $\bar{t}=\ln 3>1$, but $\bar{t}$ is close to 1. There results a large boundary layer jump (cf. [9]); however, the condition number of the problem and of $\mathcal{L}_\Delta$ does not blow up as $\varepsilon\to 0$. Accurate solutions were obtained for this case as well, using the Lobatto scheme with $N=28$. Some sample values are given in Table 2.

Example 3 (*Flaherty and O'Malley* [10]). This problem arises when considering a nonlinear elastic beam which rests on a foundation with nonlinear resistance to

TABLE 2
Sample solution values for Example 2 *with* $b=0$, $\bar{z}(0)=-3.5$.

ε	$y_1(1)$	$y_2(1)$
10^{-3}	.6555561	−26.70139
10^{-6}	.6554576	−27.71479
10^{-12}	.6554575	−27.71592

deflection. One is led to the system, for $0 \leqq t \leqq 1$,

(5.19) $$\varepsilon y_1' = -y_2,$$

(5.20) $$\varepsilon y_2' = \phi(z_1)\cos z_2 - y_1(\sec z_2 + \varepsilon y_2 \tan z_2),$$

(5.21) $$z_1' = \sin z_2,$$

(5.22) $$z_2' = y_1$$

where for ϕ we took $\phi(z_1) = z_1 - 1$. See [10] for the development and analysis of this problem.

Now, assuming that y_2 is bounded, we get the eigenvalues

(5.23) $$\lambda_1(t) = \sqrt{\sec \bar{z}_2(t)}, \qquad \lambda_2(t) = -\lambda_1(t)$$

and the reduced solution system

(5.24) $$\bar{z}_1' = \sin \bar{z}_2, \qquad \bar{y}_1 = \phi(\bar{z}_1)\cos^2 \bar{z}_2,$$

(5.25) $$\bar{z}_2' = \bar{y}_1, \qquad \bar{y}_2 = 0,$$

which is referred to as the hanging cable system [10]. The latter system can be integrated if two "reduced" boundary conditions are provided. This can be easily done in the case where the beam is simply supported:

(5.26) $$y_1(0) = y_1(1) = 0,$$

(5.27) $$z_1(0) = z_1(1) = 0.$$

For then, (5.26) is dropped and (5.27) is retained for the reduced solution. Note that in this example, the sharper error bounds (3.22b) or (3.22c) can be used for the slow solution components z.

Applying our numerical schemes to the problem (5.19)–(5.22), (5.26), (5.27), we have once again encountered no difficulty. Rather than integrating the hanging cable system, we simply used the following initial approximation:

(5.28) $$y_1 = t(1-t), \quad y_2 = 0, \quad z_1 = \sin \pi t, \quad z_2 = \tfrac{1}{2}t^2 - \tfrac{1}{3}t^3.$$

With tolerances and mesh sizes as in the previous examples, 3 iterations were needed for convergence (see Table 3).

Other types of boundary conditions are discussed in [10]. One case is of clamped supports at both endpoints, where (5.27) is retained but (5.26) is replaced by

(5.29) $$z_2(0) = z_2(1) = 0.$$

As argued by Flaherty and O'Malley [10], this set of boundary conditions leads to a problem with unbounded inverse as $\varepsilon \to 0$, which is therefore not covered by our theory.

TABLE 3
Selected solution values for Example 3 with simple support boundary conditions.

ε	$y_2(0)$	$z_2(0)$	$y_1(0.5)$	$z_1(0.5)$
10^{-2}	.867460	.426679	−.891701	.108247
10^{-4}	.863935	.434442	−.891686	.108314
10^{-6}	.863899	.434519	−.891686	.108314
10^{-12}	.863899	.434520	−.891686	.108314

The analysis in Schmeiser [19] shows that the reduced problem is still given by (5.24), (5.25) and that there are boundary layers of magnitude $O(1)$ in z_2 and boundary layers of magnitude $O(1/\varepsilon)$ in y_1 and y_2. Hence we expect, with a slight modification of the mesh selection procedure, to be able to solve with the same techniques for the clamped supports as well, with almost the same success as before. (Note, however, that we cannot avoid condition numbers of order $O(1/\varepsilon)$ in $\mathcal{L}_\Delta$; but that alone would only bother us with unrealistically small values of ε.) An extension of our analysis and computations for singular singular-perturbation problems, which covers the above case, will be reported in the near future.

REFERENCES

[1] U. ASCHER, *Solving boundary-value problems with a spline-collocation code*, J. Comp. Phys., 34 (1980), pp. 401–413.

[2] U. ASCHER, J. CHRISTIANSEN AND R. D. RUSSELL, *Collocation software for boundary-value* ODEs, ACM Trans. Math. Software, 7 (1981), pp. 209–222.

[3] U. ASCHER AND R. WEISS, *Collocation for singular perturbation problems* I: *First order systems with constant coefficients*, SIAM J. Numer. Anal., 20 (1983), pp. 537–557.

[4] ———, *Collocation for singular perturbation problems* II: *Linear first order systems without turning points*, Tech. Rep. 82-4, Dept. of Computer Science, Univ. of British Columbia, Vancouver, Canada, 1982.

[5] W.-J. BEYN AND E. DOEDEL, *Stability and multiplicity of solutions to discretizations of nonlinear differential equations*, this Journal, 2 (1981), pp. 404–415.

[6] G. F. CARRIER, *Singular perturbations and geophysics*, SIAM Rev., 12 (1970), pp. 175–193.

[7] R. C. Y. CHIN AND R. KRASNY, *A hybrid asymptotic-finite element method for stiff two-point boundary value problems*, this Journal, 4 (1983), pp. 229–243.

[8] V. A. EPISOVA, *Asymptotic properties of general boundary value problems for singularly perturbed conditionally stable systems of ordinary differential equations*, Differential Equations, 11 (1975), pp. 1457–1465.

[9] J. E. FLAHERTY AND R. E. O'MALLEY, JR., *On the numerical integration of two-point boundary value problems for stiff systems of ordinary differential equations*, in Boundary and Interior Layers—Computational and Asymptotic Methods, Proc. BAIL I Conference, J. J. H. Miller, ed., Boole Press, Dublin, 1980, pp. 93–102.

[10] ———, *Singularly perturbed boundary value problems for nonlinear systems, including a challenging problem for a nonlinear beam*, in Lecture Notes in Mathematics 942, Springer-Verlag, Berlin, 1982, pp. 170–191.

[11] P. M. GRESHO AND R. L. LEE, *Don't suppress the wiggles—they're telling you something*, in AMD 34 (1979)—ASME, T. Hughes, ed.

[12] P. W. HEMKER, H. SCHIPPERS AND P. M. DE ZEEUW, *Comparing some aspects of two codes for two-point boundary-value problems*, NW 98/80, Math. Centrum, Amsterdam, 1980.

[13] H. B. KELLER, *Numerical Solution of Two Point Boundary Value Problems*, CBMS Regional Conference Series in Applied Mathematics, 24, Society for Industrial and Applied Mathematics, Philadelphia, 1976.

[14] R. B. KELLOGG, G. R. SHUBIN AND A. B. STEPHENS, *Uniqueness and the cell Reynolds number*, SIAM J. Numer. Anal., 17 (1980), pp. 733–739.

[15] B. KREISS AND H. O. KREISS, *Numerical methods for singular perturbation problems*, SIAM J. Numer. Anal., 18 (1981), pp. 262–276.

[16] P. A. MARKOWICH AND C. A. RINGHOFER, *Collocation methods for boundary value problems on "long" intervals*, Math. Comp., 40 (1983), pp. 123–150.

[17] R. E. O'MALLEY, JR., *On multiple solutions of singularly perturbed systems in the conditionally stable case*, in Singular Perturbations and Asymptotics, R. E. Meyer and S. V. Parter, eds., Academic Press, New York, 1980, pp. 87–108.

[18] C. RINGHOFER, *On collocation schemes for quasilinear singularly perturbed boundary value problems*, Manuscript, 1982.

[19] C. SCHMEISER, *Behandlung eines nichtlinearen balkenmodelles mit methoden des singulären störungsrechnung*, Ms. thesis, Tech. Univ. Wien, Vienna, Austria, 1981.

[20] P. SPUDICH AND U. ASCHER, *Calculation of complete theoretical seismograms in vertically varying media using collocation methods*, Geoph. J. Roy. Astr. Soc., 75 (1983), pp. 101–124.

[21] F. Y. M. WAN AND U. ASCHER, *Horizontal and flat points in shallow cap dimpling*, Tech. Rep. 80-5, Inst. Appl. Math. and Stat., Univ. of British Columbia, Vancouver, Canada, 1980.

[22] R. WEISS, *An analysis of the box and trapezoidal schemes for linear singularly perturbed boundary value problems*, Math. Comp., to appear.

[23] P. W. HEMKER, *An accurate method without directional bias for the numerical solution of a 2-D elliptic singular perturbation problem*, in Lecture Notes in Mathematics 942, Springer-Verlag, Berlin, 1982, pp. 192–206.

[24] A. ABRAHAMSSON AND S. OSHER, *Monotone difference schemes for singular perturbation problems*, SIAM J. Numer. Anal., 19 (1982), pp. 979–992.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

A SPLINE BASED TECHNIQUE FOR COMPUTING RICCATI OPERATORS AND FEEDBACK CONTROLS IN REGULATOR PROBLEMS FOR DELAY EQUATIONS*

H. T. BANKS†, I. G. ROSEN‡ AND K. ITO§

Abstract. We consider the infinite interval regulator problem for systems with delays. A spline approximation method for computation of the gain operators in feedback controls is proposed and tested numerically. Comparison with a method based on "averaging" approximations is made.

Key words. Riccati operators, regulator problem, delay systems, spline approximations

1. Introduction. The problem of constructing feedback controls for hereditary or delay systems is not new and there is a rather extensive literature pertaining to several aspects of this problem. We refer the reader to the surveys of Ross [26], Alekal et al. [1], and § 5 of Banks and Burns [5] for accounts of some of the previous pertinent results. Among the fundamental earlier contributions are those of Krasovskii [19], [20] (establishment of the functional form of optimal feedback for delay systems and early use of an "averaging" type approximation scheme), Eller et al. [14] and Ross [26], [27] (derivation of Riccati type equations for the feedback gains in the functionals and methods for computing these gains), and Delfour [13] (convergence analysis of an "averaging" scheme for approximate solution of an operator form of the Riccati type equations for the feedback gains). More recently, Gibson [15] and Kunisch [21] have made important contributions which we shall discuss in the context of our presentation below.

Our own renewed interest in feedback controls for delayed systems was motivated by problems arising in the design of controllers for a liquid nitrogen wind tunnel (the National Transonic Facility or NTF) currently under construction by NASA at its Langley Research Center in Hampton, VA. With this wind tunnel it is expected that researchers will be able to achieve an order of magnitude increase in the Reynolds number over that in existing tunnels while maintaining reasonable levels of dynamic pressure. Test chamber temperatures (the Reynolds number is roughly inversely proportional to temperature) will be maintained at cryogenic levels by injection of liquid nitrogen as a coolant into the airstream near the fan section of the tunnel. In addition to a gaseous nitrogen vent to help control pressure, motor driven fans will be used as the primary regulator of Mach number. Fine control of Mach number will be effected through changes in inlet guide vanes in the fan section. Schematically, the tunnel can be depicted as in Fig. 1.1. The basic physical model relating states such as Reynolds number, pressure, and Mach number to controls such as LN_2 input, GN_2

* Received by the editors September 21, 1982, and in revised form April 15, 1983. Parts of this research were conducted while the first two authors were visitors at ICASE which is operated under NASA contracts NAS1-15810 and NAS1-16394.

† Lefschetz Center for Dynamical Systems, Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912, and Department of Mathematics, Southern Methodist University, Dallas, Texas 75275. The research of this author was supported in part by the Air Force Office of Scientific Research under contract 81-0198, in part by the National Science Foundation under grant NSF MCS 7905774-05, and in part by the U.S. Army Research Office under contract ARO-DAAG29-79-C-0161.

‡ Department of Mathematics, Bowdoin College, Brunswick, Maine 04011.

§ Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia 23665.

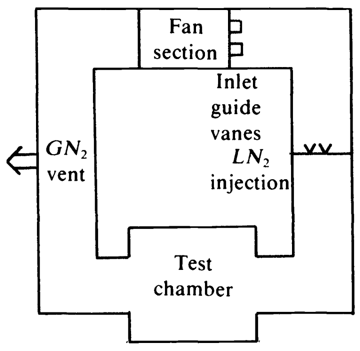


FIG. 1.1

bleed, and fan operation involves a formidable set of partial differential equations (the Navier-Stokes theory) to describe fluid flow in the tunnel and test chamber. This model has, not surprisingly, proved to be very unwieldy from a computational viewpoint and is difficult, if not impossible, to use directly in the design of sophisticated control laws. (Both open loop and feedback controllers are needed for efficient operation of the tunnel—and this is a desirable goal since cost estimates for liquid nitrogen alone are $\$6.5 \times 10^6$ per year of operation.) In addition to the design of both open loop and closed loop controllers, parameter estimation techniques will be useful once data from the completed tunnel is available (current investigations involve use of data from a $\frac{1}{3}$ meter scale model of the tunnel).

In view of the schematic in Fig. 1.1, it is not surprising that engineers (e.g., see [3] and [16]) have proposed design of control laws for subsystems modeled by lumped parameter models (the variables represent values of states and controllers at various discrete locations in the tunnel and test chamber) with transport delays to account for flow times in sections of the tunnel. A specific example is the model [3] for the Mach number control loop in which variations in the Mach no. (in the test chamber) are, to first order, controlled by variations in the inlet guide vane angle setting (in the fan section)—i.e., $\delta M(t) \sim \delta\theta(t-r)$ where r represents a transport time from the fan section to the test chamber. More precisely, the proposed equation relating the variation δM (from steady state operating values) in Mach numbers to the variation $\delta\theta$ in guide vane angle is

$$\tau \delta \dot{M}(t) + \delta M(t) = k\delta\theta(t-r),$$

while the equation relating the guide vane angle variation to that $\delta\theta_A$ of an actuator is

$$\delta\ddot{\theta}(t) + 2\zeta\omega\delta\dot{\theta}(t) + \omega^2\delta\theta(t) = \omega^2\delta\theta_A(t).$$

Rewriting the system in vector notation, one thus finds that the Mach number control loop involves a regulator problem for the equation

$$\dot{x}(t) = A_0x(t) + A_1x(t-r) + B_0u(t) \tag{1.1}$$

where $x = (\delta M, \delta\theta, \delta\dot{\theta})$, $u = \delta\theta_A$. Here the control is the guide vane angle actuator input. A similar 4-vector system problem can be formulated in the case where one treats the actuator rate $\delta\dot{\theta}_A$ as the control (see [3], [12]). We shall return to examples such as (1.1) for the NTF in § 4 below where we present numerical results obtained using the methods we propose.

Several recent contributions to the literature on numerical methods for delay systems prompt the techniques we present in this paper. A rather complete convergence analysis (along with numerical results) of the so-called "averaging" scheme applied to open loop control problems for delay systems was given in Banks and Burns [5]. The analysis was based on approximation results for linear semigroups involving the Trotter–Kato theorem (a functional analytic version of the Lax Equivalence theorem: consistency plus stability implies convergence). Gibson [15] and Kunisch [21] have shown that these same tools can be used to develop a convergence theory for approximations of the feedback gains based on the "averaging" techniques. Subsequent to the development of "averaging" methods for delay systems (which result in essentially first order numerical schemes), Banks and Kappel [9] developed higher order approximation schemes based on spline approximations. In numerous situations ([4], [6], [7], [8], [9]) these methods have proven superior computationally to the popular "averaging" techniques. In this paper we show how one can use spline based computational schemes to obtain the gains in the feedback controllers for delay systems. We present a summary of our numerical findings with these methods which support the efficacy of the proposed schemes.

Our presentation is as follows: In § 2 we summarize those facts from the literature on delay systems needed to discuss and develop our approximation techniques. Section 3 is then devoted to a careful explanation of the proposed schemes, hopefully in sufficient detail to permit readers to develop their own computational packages should they so desire. We report on our numerical experience with the spline based schemes in § 4 where we also compare our findings to those obtained using the "averaging" methods. Finally we discuss briefly in § 5 some of the theoretical aspects of the spline techniques.

The notation we use throughout is rather standard with the following exception. We shall be dealing with vector systems but shall not always make this precise when no loss of understanding results. For example, if x is an n-vector valued function with components in the Sobolev space H^1, we shall simply write $x \in H^1$. We shall only use transpose notation where it is essential; e.g., if Q_0 is an $n \times n$ matrix, we shall write xQ_0x instead of the more conventional x^TQ_0x.

2. Feedback controls for delay system problems.

In light of the motivation above, we consider the control problem of finding an m-vector valued L_2 control $\bar{u}$ which minimizes

$$J(u; \eta, \psi) = \int_0^\infty [x(t)Q_0x(t) + u(t)Ru(t)]\, dt \tag{2.1}$$

subject to the n-vector system

$$\dot{x}(t) = Lx_t + B_0u(t), \qquad t \geqq 0, \tag{2.2}$$

$$x(0) = \eta, \qquad x_0 = \psi, \tag{2.3}$$

where Q_0, R are symmetric $n \times n$ and $m \times m$ matrices, respectively, with $Q_0 \geqq 0$, $R > 0$, B_0 is an $n \times m$ matrix, and ψ is an n-vector function with components in $L_2(-r, 0)$—(we denote this by $\psi \in L_2^n(-r, 0)$). Following standard notation, the symbol x_t denotes the function $\theta \to x(t+\theta)$, $-r \leqq \theta \leqq 0$, and we assume the linear operator L has the form

$$L\phi = \sum_{i=0}^{\nu} A_i\phi(-r_i) + \int_{-r}^{0} D(\theta)\phi(\theta)\, d\theta$$

where $0=r_0<r_1<\cdots<r_\nu=r$, A_i, $i=0,1,\cdots,\nu$, are $n\times n$ matrices, and D is an $n\times n$ matrix function with components in $L_2(-r,0)$. This operator and the system (2.2), (2.3) can be given a proper interpretation for initial data ψ and controls u in L_2 and, indeed, one can establish existence of a unique solution $x\in H^1$ on any finite interval $[0,T]$ where the equation (2.2) is satisfied in the usual Caratheodory sense (i.e., almost all t)—see [9].

Assuming for the moment that a solution to the above control problem exists in closed loop form, it can be shown (see [19], [15]) to have the form

$$\bar{u}(t)=-\left[K_0x(t)+\int_{-r}^{0}K_1(\theta)x(t+\theta)\,d\theta\right]\tag{2.4}$$

where the $m\times n$ gain matrices satisfy certain Riccati-like systems of equations ([14], [26], [1], [15]). Our goal here is to discuss numerical approximations to K_0 and K_1 which, when applied to (2.2), (2.3), (2.4), yield a near optimal performance. It has been understood for some time that we are in this case dealing with feedback controls for an infinite dimensional state system. This system can be succinctly formulated abstractly (e.g., see [5], [9], [13], [15]) in a manner that facilitates convergence analyses for approximation schemes. While we shall not pursue a convergence analysis in this paper, it is convenient in discussing our numerical methods and results to use this formulation and the corresponding notation.

To this end, we let

$$z(t)=(x(t),x_t)\tag{2.5}$$

where x is the solution of (2.2), (2.3). Define Z to be the product space $R^n\times L_2^n(-r,0)$ with the usual product Hilbert space topology (and inner product) and let $D(\mathscr{A})\equiv\{(\xi,\phi)\in Z\colon \xi=\phi(0),\ \phi\in H^1(-r,0)\}$ be the domain for the linear operator $\mathscr{A}$ given by $\mathscr{A}(\phi(0),\phi)=(L\phi,\dot\phi)$. Recalling (2.1) and (2.2), we define the linear operators $Q\colon Z\to Z$ and $\mathscr{B}\colon R^m\to Z$ by $Q(\xi,\phi)=(Q_0\xi,0)$ and $\mathscr{B}v=(B_0v,0)$. Then our original optimization problem for (2.1)–(2.3) can be reformulated as the equivalent problem of minimizing

$$J(u;z_0)=\int_0^\infty\{\langle Qz(t),z(t)\rangle+u(t)Ru(t)\}\,dt\tag{2.6}$$

over $u\in L_2$ subject to the evolution equation constraint

$$\dot z(t)=\mathscr{A}z(t)+\mathscr{B}u(t),\qquad t\geqq 0,\tag{2.7}$$

$$z(0)=z_0=(\eta,\psi).\tag{2.8}$$

It is known [5], [9] that $\mathscr{A}$ generates a C_0-semigroup $\{S(t)\}$ of solution operators and that z defined by (2.5) is the unique mild solution of (2.7), (2.8). That is, z is given by

$$z(t)=S(t)z_0+\int_0^t S(t-\sigma)\mathscr{B}u(\sigma)\,d\sigma.\tag{2.9}$$

If we define an admissible control for our problem corresponding to the initial condition $z_0\in Z$ to be an m-vector function u which is integrable on $(0,\infty)$ and for which $J(u;z_0)$ is finite; and if we make the assumption that the operators Q_0 and L are such that any admissible control corresponding to the initial condition $z_0\in Z$ drives the resulting solution of the state equation (2.7) to zero asymptotically, then we may use results due to Gibson [15] to characterize the solution to the problem in feedback form. More precisely, if there exists an admissible control corresponding to each initial condition $z_0 \in Z$ (or equivalently the system (2.7) is stabilizable, see [15, Def. 2.2 and Cor. 4.1]), then there exists a nonnegative, selfadjoint linear operator Π on Z which satisfies the Riccati algebraic equation

$$(2.10) \qquad \mathcal{A}^*\Pi + \Pi\mathcal{A} - \Pi\mathcal{B}R^{-1}\mathcal{B}^*\Pi + Q = 0.$$

Moreover, under the assumption made above there exists at most one such solution and the unique solution to the problem (2.6)–(2.8) can be given in feedback form by

$$(2.11) \qquad \bar{u}(t) = -R^{-1}\mathcal{B}^*\Pi z(t), \qquad 0 < t < \infty,$$

and

$$\min_{v \text{ admissible}} J(v; z_0) = \langle \Pi z_0, z_0 \rangle.$$

The operator Π can be written as a matrix of linear operators

$$\Pi = \begin{bmatrix} \Pi^{00} & \Pi^{01} \\ \Pi^{10} & \Pi^{11} \end{bmatrix}$$

where $\Pi^{00}: R^n \to R^n$ and $\Pi^{11}: L_2^n(-r, 0) \to L_2^n(-r, 0)$ are nonnegative and selfadjoint, $\Pi^{10} \in L_2^{n\times n}(-r, 0)$ and $\Pi^{01} = \Pi^{10*}$ with

$$(2.12) \qquad \Pi^{01}\phi = \int_{-r}^{0} \Pi^{10}(\theta)^T\phi(\theta)\, d\theta, \qquad \phi \in L_2^n(-r, 0).$$

If we recall the definition of the operator $\mathcal{B}$ and assume that the system (2.7) is stabilizable, then under the assumption made above. (2.5), (2.11) and (2.12) yield that the unique solution to the problem for (2.1)–(2.3) is given in feedback form by

$$(2.13) \qquad \bar{u}(t) = -R^{-1}B_0^T\left[\Pi^{00}x(t) + \int_{-r}^{0} \Pi^{10}(\theta)^T x(t+\theta)\, d\theta\right]$$

with (for $z_0 = (\eta, \psi)$)

$$\min_{v \text{ admissible}} J(v; \eta, \psi) = \langle \Pi z_0, z_0 \rangle.$$

That is, the gains K_0, K_1 of (2.4) are given by $R^{-1}B_0^T\Pi^{00}$ and $R^{-1}B_0^T(\Pi^{10})^T$, respectively, and can be obtained by solving the Riccati equation (2.10).

3. The approximation scheme. In this section we develop and discuss the implementation of a spline based computational scheme which yields a sequence of finite dimensional operators $\{\Pi_N\}$ which approximate Π, the solution to the operator Riccati algebraic equation given by (2.10). The Π_N are found by solving standard matrix Riccati algebraic equations, and are then used to construct feedback controls which approximate (2.11) and which produce near optimal performance by the system (2.7) (2.8) as measured by the functional (2.6).

The approach we take is based largely upon the spline approximation framework developed in [9] for the approximation of solutions of linear functional differential equations. We summarize briefly the essentials of that development. Let Z_N be a sequence of spline based subspaces of Z satisfying $Z_N \subset D(\mathcal{A})$ $N = 1, 2, \cdots$. Let $\mathcal{P}_N: Z \to Z_N$ denote the corresponding sequence of orthogonal projections of Z onto Z_N computed with respect to the weighted inner product $\langle \cdot, \cdot \rangle_g$ on Z given by

$$\langle (\eta, \phi), (\xi, \psi) \rangle_g = \eta^T\xi + \int_{-r}^{0} \phi(\theta)^T\psi(\theta)g(\theta)\, d\theta$$

where

$$g(\theta)=\begin{cases}1, & -r\leqq\theta<-r_{\nu-1},\\ 2, & -r_{\nu-1}\leqq\theta<-r_{\nu-2},\\ \vdots & \\ \nu-1, & -r_2\leqq\theta<-r_1,\\ \nu, & -r_1\leqq\theta\leqq 0.\end{cases}$$

Define the linear operators $\mathcal{A}_N$ and Q_N on Z_N and $\mathcal{B}_N\colon R^m\to Z_N$ by $\mathcal{A}_N=\mathcal{P}_N\mathcal{A}$, $\mathcal{B}_N=\mathcal{P}_N\mathcal{B}$ and $Q_N=\mathcal{P}_NQ$, respectively, and let Π_N be a nonnegative selfadjoint solution to the Riccati algebraic equation in Z_N given by

$$\mathcal{A}_N^*\Pi_N+\Pi_N\mathcal{A}_N-\Pi_N\mathcal{B}_NR^{-1}\mathcal{B}_N^*\Pi_N+Q_N=0. \tag{3.1}$$

The existence and uniqueness of solutions of (3.1), which are related to the existence and uniqueness of solutions of (2.10) and certain properties of the approximation scheme itself, will be discussed in § 5. For the present, however, we assume that for all N sufficiently large, a solution Π_N exists with $\Pi_N\geqq 0$ and $\Pi_N^*=\Pi_N$.

The use of the weighted inner product $\langle\cdot,\cdot\rangle_g$ in place of the standard inner product on Z in computing the projections $\mathcal{P}_N$ (and therefore the operators $\mathcal{A}_N$) insures that the operators $\mathcal{A}_N$ satisfy a uniform dissipative inequality of the form

$$\langle\mathcal{A}_Nz, z\rangle_g\leqq\beta\langle z, z\rangle_g, \qquad z\in Z_N,$$

and hence that the solutions of the finite dimensional ordinary differential equation initial value problem in Z_N

$$\begin{aligned}\dot z_N(t)&=\mathcal{A}_Nz_N(t)+\mathcal{B}_Nu(t), \qquad t\geqq 0,\\ z_N(0)&=\mathcal{P}_Nz_0\end{aligned} \tag{3.2}$$

approximate the solution of (2.7), (2.8) (see [9]). It is this fundamental convergence result which forms the theoretical foundation for the schemes being developed here.

Since the domain of the operators $\mathcal{P}_N$ is $Z=R^n\times L_2^n(-r,0)$, the operators $\Pi_N\mathcal{P}_N$ can be written as the matrix of linear operators given by

$$\Pi_N\mathcal{P}_N=\begin{bmatrix}\Pi_N^{00} & \Pi_N^{01}\\ \Pi_N^{10} & \Pi_N^{11}\end{bmatrix}$$

where the $n\times n$ matrix Π_N^{00} and $\Pi_N^{11}\colon L_2^n(-r,0)\to L_2^n(-r,0)$ are nonnegative and selfadjoint, Π_N^{10} is an $n\times n$ matrix valued function with components in L_2, and $\Pi_N^{01}=\Pi_N^{10*}$ with

$$\Pi_N^{01}\phi=\int_{-r}^0\Pi_N^{10}(\theta)^T\phi(\theta)\,d\theta, \qquad \phi\in L_2^n(-r,0).$$

If the approximating optimal controls in feedback form for the problem involving (2.6)–(2.8) are defined by

$$\bar u_N(t)=-R^{-1}\mathcal{B}_N^*\Pi_N\mathcal{P}_Nz(t), \tag{3.3}$$

then the approximate solutions to our problem take the form

$$\bar u_N(t)=-\left[R^{-1}B_0^T\Pi_N^{00}x(t)+\int_{-r}^0R^{-1}B_0^T\Pi_N^{10}(\theta)^Tx(t+\theta)\,d\theta\right] \tag{3.4}$$

with the corresponding approximating optimal trajectories being given by the solutions to

$$\dot{x}(t)=(A_0-B_0R^{-1}B_0^T\Pi_N^{00})x(t)+\sum_{j=1}^{\nu}A_jx(t-r_j)$$
(3.5)
$$+\int_{-r}^{0}(D(\theta)-B_0R^{-1}B_0^T\Pi_N^{10}(\theta)^T)x(t+\theta)\,d\theta$$

for any initial conditions $x(0)=\eta\in R^n$ and $x_0=\psi\in L_2^n(-r,0)$. In addition, the optimal cost can be approximated by

(3.6) $$\langle \Pi_N\mathcal{P}_N(\eta,\psi),\mathcal{P}_N(\eta,\psi)\rangle.$$

Equation (3.1) is an operator equation, and thus is not suitable for computational purposes in its present form. In order to find the matrix form of (3.1) a basis for Z_N must be chosen and matrix representations for the operators $\mathcal{A}_N$, $\mathcal{A}_N^*$, $\mathcal{B}_N$, $\mathcal{B}_N^*$ and Q_N with respect to this basis must be computed. The adjoint operators $\mathcal{A}^*$ and $\mathcal{B}^*$ (and therefore their approximations $\mathcal{A}_N^*$ and $\mathcal{B}_N^*$) may be computed with respect to either the standard inner product on Z, $\langle\cdot,\cdot\rangle$, or the weighted inner product $\langle\cdot,\cdot\rangle_g$. Indeed, the fact that

$$\langle Qz,z\rangle_g=\langle Qz,z\rangle$$

for all $z\in Z$ implies that the abstract regulator problem given by (2.6), (2.7) and (2.8) can be formulated in the space Z using either inner product and still be equivalent to our original control problem. However, the expressions for the matrix representations for the operators $\mathcal{A}_N^*$ and $\mathcal{B}_N^*$ are simplified if the $\langle\cdot,\cdot\rangle_g$ inner product is employed (in this case of course it must also be used in (3.6)). When the discrete delay part of L consists of only a single delay term (i.e., $\nu=1$), then $g(\theta)\equiv 1$ and the two inner products are the same.

We briefly outline the necessary procedure for finding matrix representations in the case of "linear" or first order spline functions. A more detailed description can be found in [11]. The ideas presented here and in [11] for linear splines are easily extended to the case of cubic or higher order spline functions.

For each $N=1,2,\cdots$, and $j=0,1,\cdots,N$ denote by ϕ_j^N the standard piecewise linear spline basis element corresponding to the partition $\{t_j^N\}$ defined so that ϕ_j^N has support in $[t_{j+1}^N,t_{j-1}^N]$ and value 1 at t_j^N where $t_j^N=-j(r/N)$. Define Z_N to be

$$Z_N=\left\{(\phi(0),\phi)\in Z\colon \phi=\sum_{j=0}^{N}v_j\phi_j^N,\ v_j\in R^N\right\}.$$

Note that dim $Z_N=n(N+1)$ and $Z_N\subset D(\mathcal{A})$ as is required by the theory outlined above. If we define the $n\times n(N+1)$ matrix function $\Phi^N(\cdot)$ by

$$\Phi^N(\theta)=(\phi_0^N(\theta),\phi_1^N(\theta),\cdots,\phi_N^N(\theta))\otimes I_n$$

for $\theta\in[-r,0]$, where I_n denotes the $n\times n$ identity matrix and $\otimes$ is the Kronecker product, then it is easily shown (see [9]) that the matrix representation A^N for the operator $\mathcal{A}_N$ is given by

(3.7) $$A^N=(K^N)^{-1}H^N$$

where the $n(N+1)\times n(N+1)$ matrices K^N and H^N are given by

$$K^N=\Phi^N(0)^T\Phi^N(0)+\int_{-r}^{0}\Phi^N(\theta)^T\Phi^N(\theta)g(\theta)\,d\theta \tag{3.8}$$

$$H^N=\Phi^N(0)^T(L\Phi^N)+\int_{-r}^{0}\Phi^N(\theta)^T\dot{\Phi}^N(\theta)g(\theta)\,d\theta. \tag{3.9}$$

Furthermore it is straightforward to see that the matrix representation A^{*N} for the adjoint operator $\mathscr{A}_N^*$ is given by

$$A^{*N}=(K^N)^{-1}(A^N)^TK^N. \tag{3.10}$$

For the linear spline case it is not difficult to compute the inner products appearing in the definitions of the matrices K^N and H^N analytically, at least for relatively simple forms of the operator L. The forms for these matrices are given explicitly (in terms of the matrices A_j, $j=0, 1, 2,\cdots, \nu$, and the matrix function D appearing in the definition of the operator L) in [9] and [10].

Since the operators $\mathscr{B}_N:R^m\to Z_N$ and $Q_N:Z_N\to Z_N$ are defined by $\mathscr{B}_Nv=\mathscr{P}_N(B_0v,0)$ and $Q_N(\eta,\psi)=\mathscr{P}_N(Q_0\eta,0)$ respectively, it is easily seen that their matrix representations, B^N and Q^N are given by

$$B^N=(K^N)^{-1}\Phi^N(0)^TB_0 \tag{3.11}$$

and

$$Q^N=(K^N)^{-1}\Phi^N(0)^TQ_0\Phi^N(0), \tag{3.12}$$

whereas, B^{*N}, the matrix representation for the operator $\mathscr{B}_N^*$ is given by

$$B^{*N}=(B^N)^TK^N. \tag{3.13}$$

If we let $\tilde{P}^N$ denote the matrix representation for Π_N, the solution to the operator equation (3.1), the matrix form of (3.1) is given by

$$A^{*N}\tilde{P}^N+\tilde{P}^NA^N-\tilde{P}^NB^NR^{-1}B^{*N}\tilde{P}^N+Q^N=0. \tag{3.14}$$

Premultiplying by K^N and using (3.10) and (3.13), (3.14) becomes

$$(A^N)^TK^N\tilde{P}^N+K^N\tilde{P}^NA^N-K^N\tilde{P}^NB^NR^{-1}(B^N)^TK^N\tilde{P}^N+K^NQ^N=0.$$

If the substitutions $P^N=K^N\tilde{P}^N$ and $\tilde{Q}^N=K^NQ^N$ are made in the last equation above, a standard matrix Riccati algebraic equation in $R^{n(N+1)}$ for P^N results and is given by

$$(A^N)^TP^N+P^NA^N-P^NB^NR^{-1}(B^N)^TP^N+\tilde{Q}^N=0. \tag{3.15}$$

Equation (3.15) can be solved for the matrix P^N using standard computational techniques and readily available software packages (see [2] and [22]).

Once P^N has been determined, a simple calculation reveals that the Nth approximating optimal control for our problem given by (3.4) takes the form

$$\begin{aligned}\bar{u}_N(t)=-\Big[&R^{-1}B_0^T\Phi^N(0)(K^N)^{-1}P^N(K^N)^{-1}\Phi^N(0)^Tx(t)\\&+\int_{-r}^{0}R^{-1}B_0^T\Phi^N(0)(K^N)^{-1}P^N(K^N)^{-1}\Phi^N(\theta)^Tx(t+\theta)g(\theta)\,d\theta\Big].\end{aligned} \tag{3.16}$$

Comparing (3.16) to (3.4), it is immediately clear that the approximating feedback gains are given by

$$R^{-1}B_0^T\Pi_N^{00}=R^{-1}B_0^T\Phi^N(0)(K^N)^{-1}P^N(K^N)^{-1}\Phi^N(0)^T \tag{3.17}$$

and

$$(3.18)\qquad R^{-1}B_0^T\Pi_N^{10}(\cdot)^T = R^{-1}B_0^T\Phi^N(0)(K^N)^{-1}P^N(K^N)^{-1}\Phi^N(\cdot)^T g(\cdot).$$

For a given set of initial conditions $x(0)=\eta$, $x_0=\psi$ we also have an approximation to the optimal value of the cost functional

$$(3.19)\qquad J(\bar{u};\eta,\psi)\sim[(K^N)^{-1}h^N(\eta,\psi)]^T P^N(K^N)^{-1}h^N(\eta,\psi)$$

where h^N is given by

$$(3.20)\qquad h^N(\eta,\psi)=\Phi^N(0)^T\eta+\int_{-r}^{0}\Phi^N(\theta)^T\psi(\theta)g(\theta)\,d\theta.$$

The approximation scheme which was developed above is semi-discrete in nature in that the approach taken is based primarily upon the approximation of the infinite dimensional state equation (2.7) in the space Z by a sequence of finite dimensional ordinary differential equations in Z_N of the form (3.2). However it is also possible to develop a parallel theory which is based upon a full discretization of the optimization problem in the spirit of the results presented in [25]. The cost functional (2.6) is discretized and the state equation (in its integrated form (2.9)) is approximated by a finite dimensional difference equation in Z_N resulting in a finite dimensional discrete steady state linear regulator problem which can be solved in feedback form using conventional methods. We sketch briefly the particulars of such an approach.

Let the Nth approximating optimization problem be given by:

Find a sequence $\{\bar{u}_t^N\}_{t=0}^{\infty}$ of m-vectors in l_2 such that $\bar{u}_t^N$ minimizes

$$J^N(\{u_t^N\};z_0)=\sum_{t=0}^{\infty}\langle\hat{Q}_N z_t^N, z_t^N\rangle+\langle R_N u_t^N, u_t^N\rangle$$

subject to

$$(3.21)\qquad z_{t+1}^N=P_{ij}\left(\frac{r}{N}\mathcal{A}_N\right)z_t^N+\frac{r}{N}P_{kl}\left(\frac{r}{2N}\mathcal{A}_N\right)\mathcal{B}_N u_t^N\qquad t=1,2,\cdots,$$

$$(3.22)\qquad z_0^N=\mathcal{P}_N z_0$$

where $\hat{Q}_N=(r/N)Q_N$, $R_N=(r/N)R$ and $\mathcal{A}_N$, $\mathcal{P}_N$, $\mathcal{B}_N$, Q_N, R and z_0 are as they have been defined above. The rational functions $P_{ij}(z)$ and $P_{kl}(z)$ are selected from among the entries in the diagonal or first two subdiagonals of the Padé table of rational function approximations to the exponential.

The basis for the construction of the approximation problems is the fact that the variation of parameters form of the solution to (3.21), (3.22) given by

$$z_t^N=P_{ij}\left(\frac{r}{N}\mathcal{A}_N\right)^t\mathcal{P}_N z_0+\frac{r}{N}\sum_{j=1}^{t}P_{ij}\left(\frac{r}{N}\mathcal{A}_N\right)^{t-s}P_{kl}\left(\frac{r}{2N}\mathcal{A}_N\right)\mathcal{B}_N u_s^N$$

is an approximation to (2.9) in the sense that

$$\left|z_t^N-z\left(\frac{tr}{N}\right)\right|\to 0$$

an $N\to\infty$ uniformly in t for $t\in\{0,1,2,\cdots[t_fN/r]\}$ for any $t_f<\infty$, where the symbol $[\alpha]$ denotes the greatest integer less than or equal to α (see [25]).

The feedback form of the solution (if it exists) to the corresponding approximating problem and the optimal value of the cost functional are given by

$$\bar{u}_t^N = -F_N z_t^N, \qquad t = 0, 1, 2, \cdots,$$

and

$$J^N(\{\bar{u}_t^N\}; z_0) = \langle \Pi_N \mathcal{P}_N z_0, \mathcal{P}_N z_0 \rangle, \tag{3.23}$$

respectively, where the linear operators $F_N : Z_N \to R^m$ and $\Pi_N : Z_N \to Z_N$ are determined by solving the system of operator equations (see [22])

$$\Pi_N = X_N^* \Pi_N X_N + F_N^* R_N F_N + \hat{Q}_N, \tag{3.24}$$

$$X_N = \tilde{\mathcal{A}}_N - \tilde{B}_N F_N, \tag{3.25}$$

$$F_N = (R_N + \tilde{B}_N^* \Pi_N \tilde{B}_N)^{-1} \tilde{B}_N^* \Pi_N \tilde{\mathcal{A}}_N \tag{3.26}$$

in the unknowns Π_N, X_N and F_N where $\tilde{\mathcal{A}}_N = P_{ij}((r/N)\mathcal{A}_N)$ and $\tilde{B}_N = (r/N)P_{kl}((r/N)\mathcal{A}_N)\mathcal{B}_N$.

To actually compute the optimal control law, the system (3.24), (3.25), (3.26) must first be transformed into an equivalent matrix formulation. Adopting the convention that the symbol $\mathcal{T}^N$ will denote the matrix representation for the operator $\mathcal{T}_N$ with respect to the linear spline basis defined above, it is not difficult to show that the system (3.24), (3.25), (3.26) is equivalent to the system of matrix equations given by

$$P^N = (X^N)^T P^N X^N + (F^N)^T R^N F^N + \frac{r}{N} K^N Q^N, \tag{3.27}$$

$$X^N = \tilde{\mathcal{A}}^N - \tilde{B}^N F^N, \tag{3.28}$$

$$F^N = (R^N + (\tilde{B}^N)^T P^N \tilde{B}^N)^{-1} (\tilde{B}^N)^T P^N \tilde{\mathcal{A}}^N \tag{3.29}$$

where $P^N = K^N \Pi^N$, $\tilde{\mathcal{A}}^N = P_{ij}((r/N)A^N)$, $\tilde{B}^N = (r/N)P_{kl}((r/N)A^N)B^N$, $R^N = (r/N)R$ and K^N, A^N, B^N and Q^N are given by (3.8), (3.7), (3.11) and (3.12) respectively. Standard software packages can be used to solve the system (3.27), (3.28), (3.29), see for instance [2]. Once the matrices F^N and P^N have been determined, the Nth approximating solution to our problem is given by

$$\begin{aligned} \bar{u}_N(t) &= -F_N \mathcal{P}_N z(t) \\ &= -\left[F^N (K^N)^{-1} \Phi^N(0)^T x(t) + \int_{-r}^0 F^N (K^N)^{-1} \Phi^N(\theta)^T x(t+\theta) g(\theta)\, d\theta \right] \end{aligned} \tag{3.30}$$

or, using the fact that $\Pi_N \mathcal{P}_N$ approximates Π, by

$$\begin{aligned} \bar{u}_N(t) &= -R^{-1} \mathcal{B}^* \Pi_N \mathcal{P}_N z(t) \\ &= -\Big[R^{-1} B_0^T (K^N)^{-1} P^N (K^N)^{-1} \Phi^N(0)^T x(t) \\ &\quad + \int_{-r}^0 R^{-1} B_0^T (K^N)^{-1} P^N (K^N)^{-1} \Phi^N(\theta)^T x(t+\theta) g(\theta)\, d\theta \Big]. \end{aligned} \tag{3.31}$$

The feedback gains K_0 and K_1 in (2.4) are approximated by

$$F^N (K^N)^{-1} \Phi^N(0)^T \tag{3.32}$$

and

$$F^N (K^N)^{-1} \Phi^N(\cdot)^T g(\cdot), \tag{3.33}$$

respectively, if (3.30) is used and by

$$R^{-1}B_0^T(K^N)^{-1}P^N(K^N)^{-1}\Phi^N(0)^T \tag{3.34}$$

and

$$R^{-1}B_0^T(K^N)^{-1}P^N(K^N)^{-1}\Phi^N(\cdot)^T g(\cdot) \tag{3.35}$$

if (3.31) is used.

Finally for a given set of initial conditions $x(0)=\eta$, $x_0=\psi$, the optimal value of the cost functional can be approximated using (3.23). We have

$$J(\bar{u};\eta,\psi)\sim[(K^N)^{-1}h^N(\eta,\psi)]^T P^N(K^N)^{-1}h^N(\eta,\psi) \tag{3.36}$$

where h^N is given by (3.20).

4. Numerical results. In this section we summarize some of the numerical results obtained by using the linear spline based approximation schemes described above to compute feedback controls for several hereditary regulator problems of the form given in § 2. For the purpose of comparison we have also computed approximate solutions using the finite difference based AVE scheme discussed in [15]. We recall that for the semi-discrete spline scheme, the approximating feedback gains $R^{-1}B_0^T\Pi_N^{00}$ and $R^{-1}B_0^T\Pi_N^{10}(\cdot)^T$ are given by (3.17) and (3.18) respectively while for the AVE scheme they may be computed using the time invariant forms of [15, (7.27) and (7.28)]. For the fully discrete spline scheme, the approximating gains are given either by (3.32) and (3.33) or by (3.34) and (3.35). Analogous formulae can be derived for a fully discrete scheme based upon the AVE approximation.

All computations were performed on a Control Data Corporation Cyber 170 model 730 at the NASA Langley Research Center (LaRC) using software written in Fortran. For the semi-discrete schemes, the approximating matrix Riccati algebraic equations (3.15) were solved using both an iterative Newton technique as it is described in [16] and the Potter method (see [24], [22]) which involves the eigenvalue-eigenvector decomposition of the $2n(N+1)\times 2n(N+1)$ matrix

$$\hat{A}^N=\begin{bmatrix} A^N & -B^N R^{-1}B^{N^T} \\ -\tilde{Q}^N & -A^{N^T}\end{bmatrix}$$

where the matrices A^N, B^N and $\tilde{Q}^N$ are as they were defined in § 3. The implementations of the two methods we used are contained in ORACLS [2], a software package developed at LaRC for the design of multivariable control systems.

Although the Newton algorithm performed well on the equations arising from both the spline and AVE schemes, the Potter method was the more efficient of the two, particularly for large values of N. The ORACLS implementation of the Potter method, however, requires that the matrix $\hat{A}^N$ be diagonalizable. This additional requirement posed no difficulties for the spline schemes in any of the examples we considered. On the other hand, for the AVE scheme, certain classes of problems (including those involving state equations of dimension greater than one of the form considered in Examples 4.2, and 4.3 below) lead to $\hat{A}^N$ which are nondiagonalizable (see [15] Theorems 7.11 and 7.12). In this instance, if one wishes to use the Potter method to solve (3.15), the generalized eigenvectors of $\hat{A}^N$ must be computed.

For the fully-discrete schemes, the system (3.27), (3.38), (3.29) was solved using an iterative Newton algorithm from the ORACLS package. The finite dimensional algebraic Riccati equation arising from either the semi-discrete or the fully discrete approximation schemes could also have been solved using a Schur decomposition technique (see [23]). This method is similar in spirit to the Potter method with the exception of the fact that a Schur decomposition of the Hamiltonian matrix is employed in the place of an eigenvector decomposition. There is evidence that this method for solving algebraic Riccati equations with dense matrices of moderate size (order up to 100) is numerically stable and computationally more efficient than either the eigenvector or Newton–Kleinman approach. In addition the difficulty in applying the Potter method to the algebraic Riccati equation resulting from the AVE approximation scheme described above could be avoided. Since our primary objective was not to study the solution of the finite dimensional Riccati equations but rather the investigation of a spline based finite dimensional approximation to the original infinite dimensional delay system and since the Potter and Newton–Kleinman methods yielded satisfactory results in all of the equations we considered, we were content to make use of these methods which were included in the ORACLS pckage. The Schur method, on the other hand, was not readily available to us. However, we do recommend that in the application of our approximation schemes the use of the Schur technique for solving the resulting approximating finite dimensional algebraic Riccati equation be considered.

We have included results for three examples. Others can be found in [11]. Example 4.1 involves a 1-dimensional state equation while Example 4.2 involves a system of dimension 2. In Example 4.3 we consider the wind tunnel system described in § 1. These examples were all solved using semi-discrete approximations. Example 4.1 was also solved using the fully discrete method and we shall comment further on this later.

Example 4.1. We consider the minimization of

$$J(u; x(0), x_0) = \int_0^\infty [x^2(t) + u^2(t)]\, dt \tag{4.1}$$

subject to the scalar differential equation given by

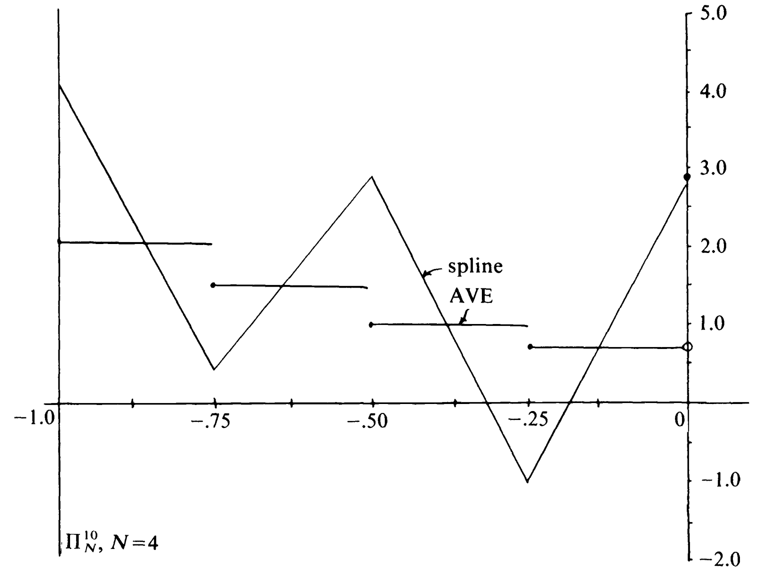
$$\dot{x}(t) = x(t) + x(t-1) + u(t). \tag{4.2}$$

In this example, the Π_N^{00} are scalars and have been tabulated for $N = 4$, 8, 16, and 32 in Table 4.1. The $\Pi_N^{10}(\cdot)$ are scalar valued functions and have been plotted for the same values of N in Figs. 4.1, 4.2, 4.3 and 4.4.

TABLE 4.1

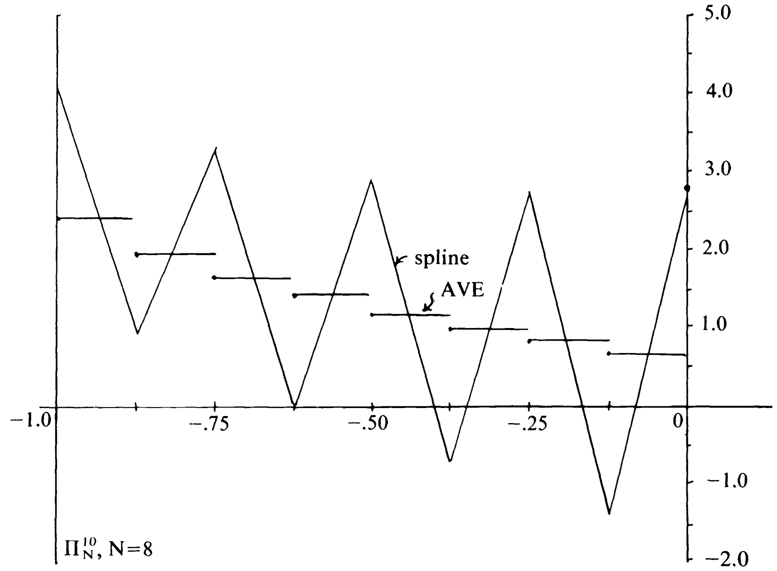
N	AVE	Spline
4	2.8866	2.7940
8	2.8476	2.8054
16	2.8278	2.8084
32	2.8182	2.8091

Although we do not have a true value for Π^{00}, it is immediately clear from Table 4.1 that the values computed using the spline based scheme appear to have converged, while those computed by the AVE scheme are converging much more slowly. The oscillatory behavior exhibited by the spline approximations to Π^{10} is a consequence of the fact that while in general it is not the case that $\Pi^{00} = \Pi^{10}(0)$, the requirement $R(\Pi_N) \subset Z_N \subset D(\mathscr{A})$ imposes the conditions $\Pi_N^{00} = \Pi_N^{10}(0)$ for each N. However, because in the closed loop form of the state equation (see (3.5)) $\Pi_N^{10}(\cdot)$ appears in the form of the kernel of an integral operator, the effect of the oscillations is minimized.



5.0
4.0
3.0
2.0
1.0
−1.0
−2.0
−1.0
−.75
−.50
−.25
0
spline
AVE
Π_N^{10}, $N=4$

FIG. 4.1



5.0
4.0
3.0
2.0
1.0
−1.0
−2.0
−1.0
−.75
−.50
−.25
0
spline
AVE
Π_N^{10}, N=8

FIG. 4.2

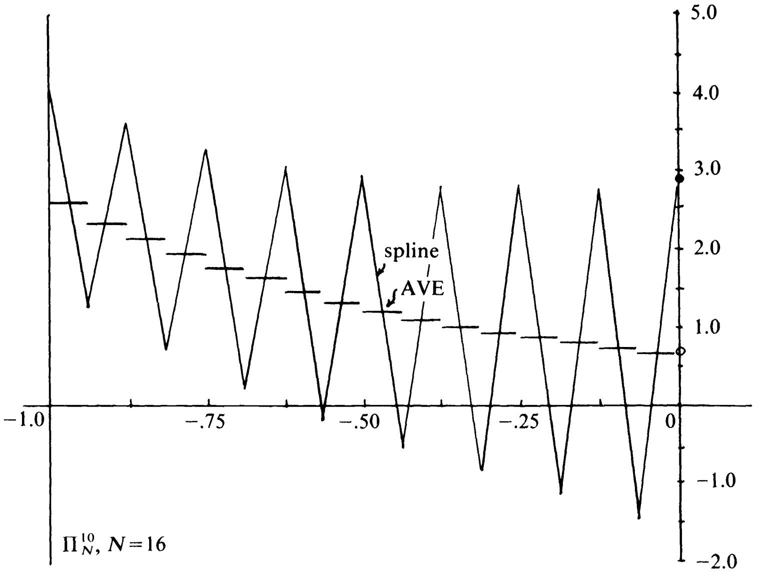

5.0
4.0
3.0
2.0
1.0
0
-1.0
-2.0
-1.0
-.75
-.50
-.25
spline
AVE
Π_N^{10}, $N=16$

FIG. 4.3

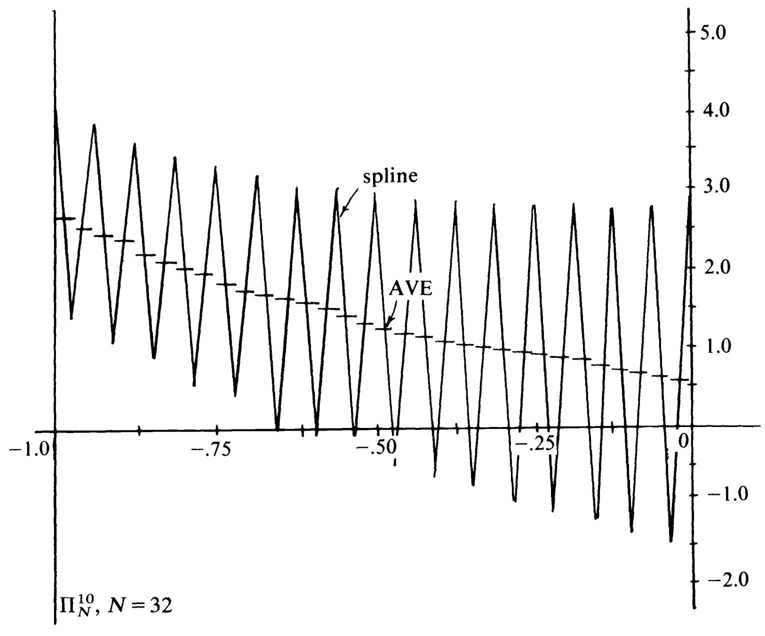

5.0
4.0
3.0
2.0
1.0
0
-1.0
-2.0
-1.0
-.75
-.50
-.25
spline
AVE
Π_N^{10}, $N=32$

FIG. 4.4

We selected the initial data (we also satisfactorily considered other initial data, see [11])

$$x(0)=0 \quad x_0(\theta)=\sin \pi\theta, \quad -1 \leqq \theta \leqq 0, \tag{4.3}$$

and computed the trajectories which result when the approximating optimal feedback controls (computed using either the AVE or the spline approximation schemes) are applied to the system (4.2), (4.3). Approximate values for the cost functional (4.1) were computed two ways: directly using the approximating optimal controls (3.4) with the corresponding approximating optimal trajectories (3.5) in (4.1) and also via (3.19). The integration of the closed loop state equation (3.5) was carried out by first discretizing the integral term and then by applying a modified version of a Runge-Kutta method for the numerical solution of ordinary differential equation initial value problems. We note that the numerical integration method employed to compute these trajectories was completely independent of either of the approximation schemes used to compute the approximating feedback operators, and thus should not have biased our results. For $N=4$, 8, 16, and 32 the approximating optimal trajectories are plotted in Figs. 4.5, 4.7, 4.9 and 4.11 while the open loop form of the approximating optimal controls are plotted in Figs. 4.6, 4.8, 4.10 and 4.12. The approximating values for the cost functional are tabulated in Table 4.2 where columns 1 and 3 contain the values computed directly and columns 2 and 4 the values computed using (3.24).

TABLE 4.2

N	AVE		Spline	
	$J(x(\bar{u}_N))$	$\langle \Pi_N \mathcal{P}_N z_0, \mathcal{P}_N z_0 \rangle$	$J(x(\bar{u}_N))$	$\langle \Pi_N \mathcal{P}_N z_0, \mathcal{P}_N z_0 \rangle$
4	.3309	.2809	.3272	.2484
8	.3281	.3000	.3271	.3027
16	.3275	.3121	.3272	.3163
32	.3274	.3191	.3273	.3196

Based upon the numerical results for the example presented above and several others which we have considered, the following observations concerning the relative performance of the AVE and spline based schemes can be made.

(I) The spline scheme converges faster and is more accurate at low orders. The AVE approximations generate a scheme that appears to converge like $1/N$ while that for the splines is like $1/N^2$. This is not unexpected, given our experience with the AVE and spline approximations in other contexts (e.g., see [5], [9], [7]). Furthermore, the trajectories, controls, and cost functional values obtained using the spline approximations with $N=4$ are competitive with the results produced by the AVE scheme with $N=16$. The computational effort and expense involved in solving a high order Riccati equation makes this an important consideration. In the scalar examples we tested, the amount of CPU time required to solve (using the Newton algorithm) the 5-dimensional Riccati equations (corresponding to $n=1$ and $N=4$) was on the order of 10 seconds while the 17-dimensional equations ($n=1$, $N=16$) required approximately 70 seconds.

(II) If the value of the cost functional J computed using the approximating optimal control and the resulting trajectory is used as a measure of the relative performance of the two approximation methods, then we found that the spline based technique is

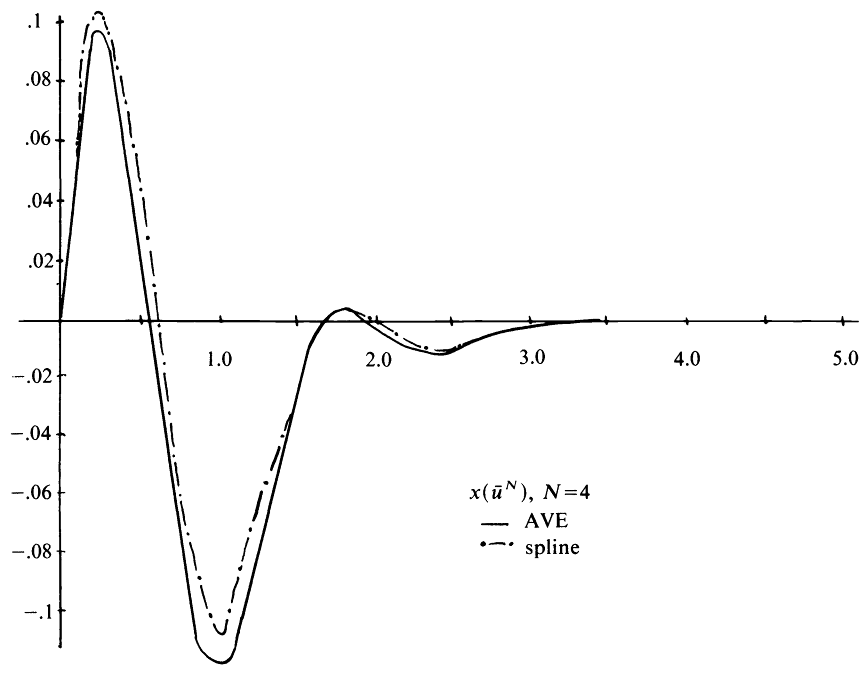

.1
.08
.06
.04
.02
−.02
−.04
−.06
−.08
−.1
1.0
2.0
3.0
4.0
5.0
$x(\bar{u}^N)$, $N=4$
— AVE
•—• spline

FIG. 4.5

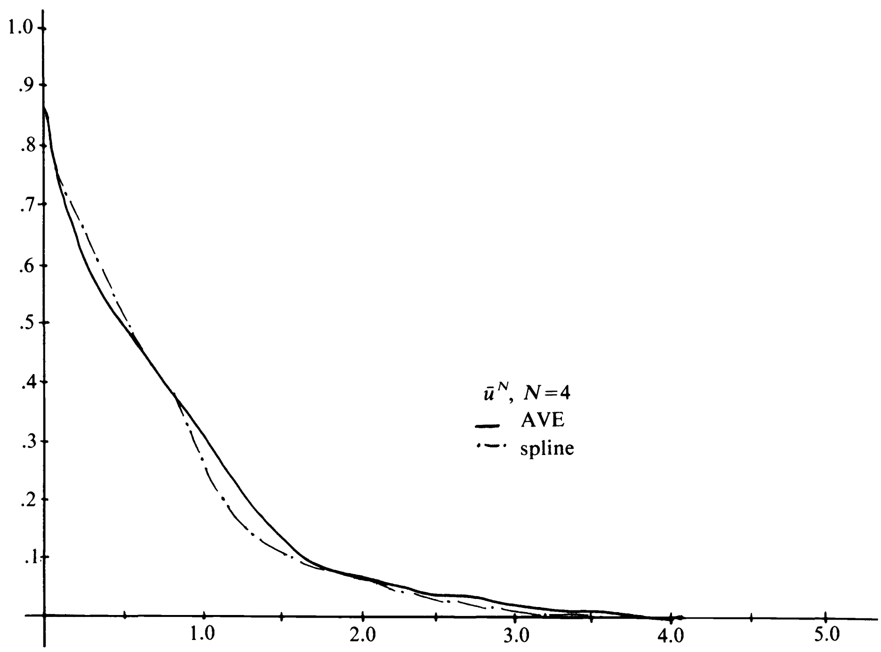

1.0
.9
.8
.7
.6
.5
.4
.3
.2
.1
1.0
2.0
3.0
4.0
5.0
$\bar{u}^N$, $N=4$
— AVE
•—• spline

FIG. 4.6

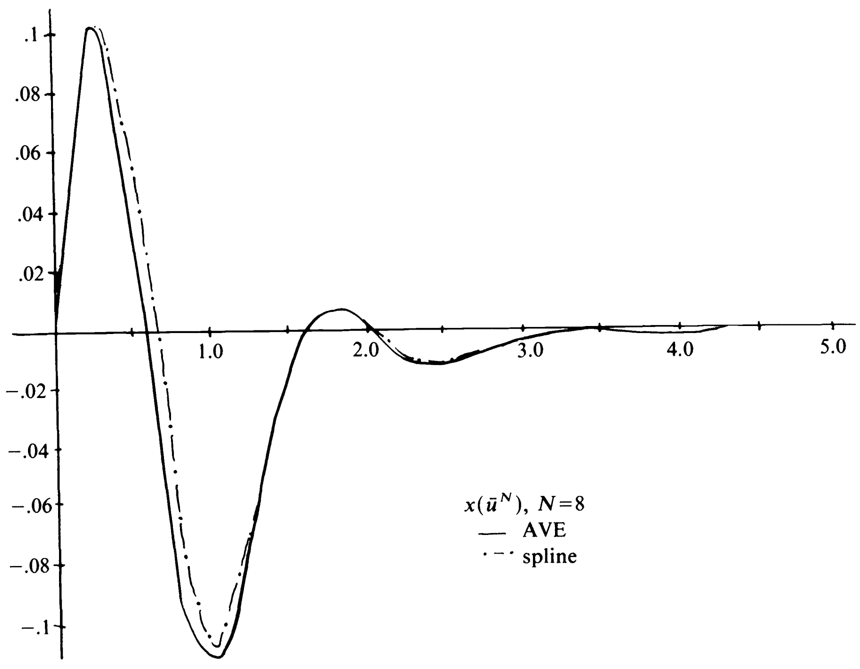

.1
.08
.06
.04
.02
−.02
−.04
−.06
−.08
−.1
1.0
2.0
3.0
4.0
5.0
$x(\bar{u}^N)$, $N=8$
— AVE
·—· spline

FIG. 4.7

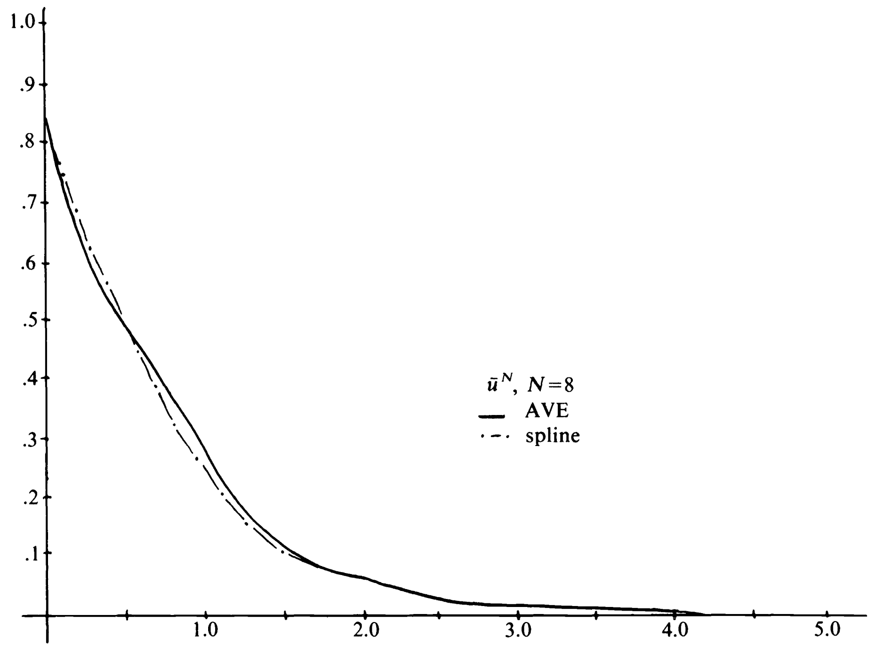

1.0
.9
.8
.7
.6
.5
.4
.3
.2
.1
1.0
2.0
3.0
4.0
5.0
$\bar{u}^N$, $N=8$
— AVE
·—· spline

FIG. 4.8

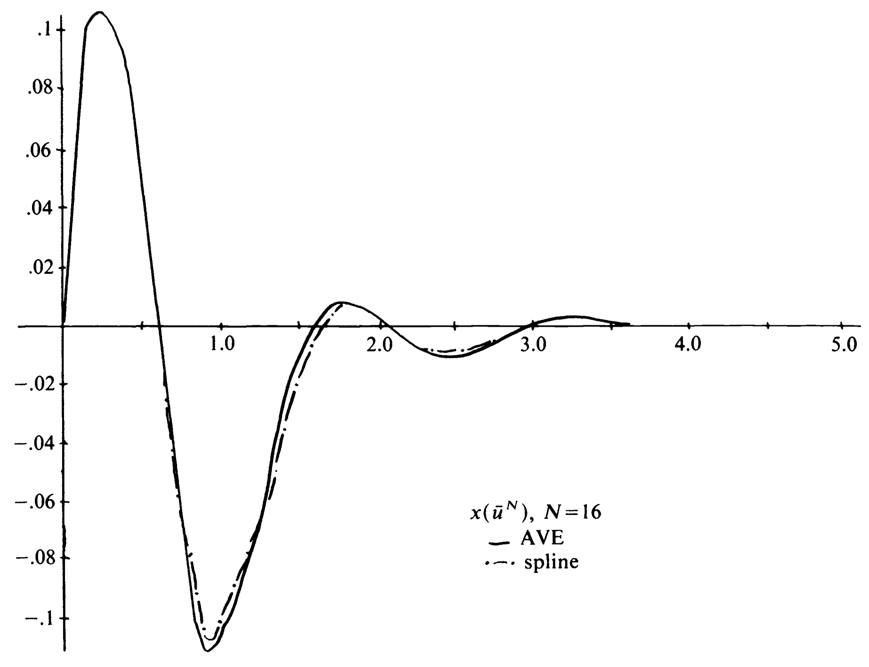


FIG. 4.9

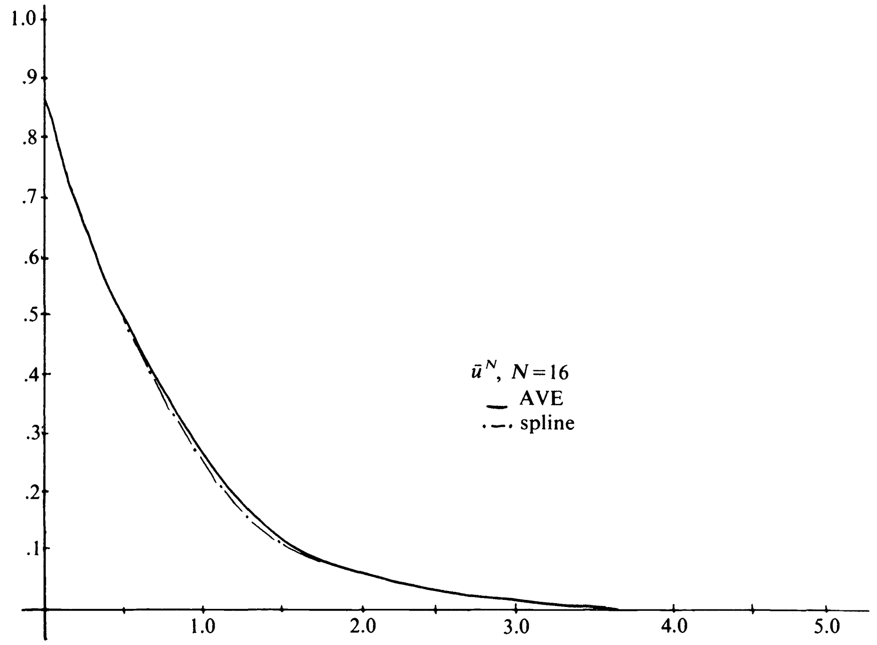


FIG. 4.10

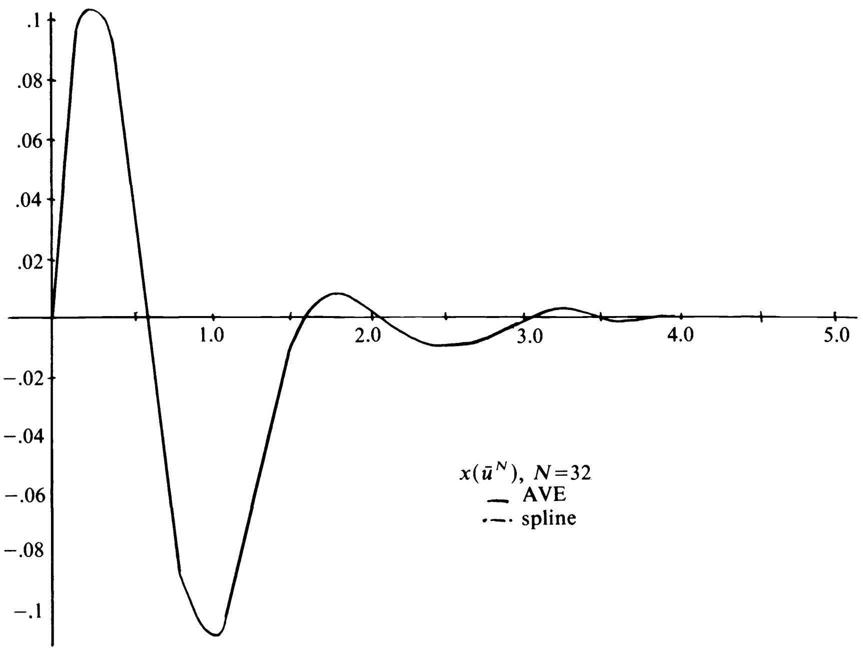


FIG. 4.11

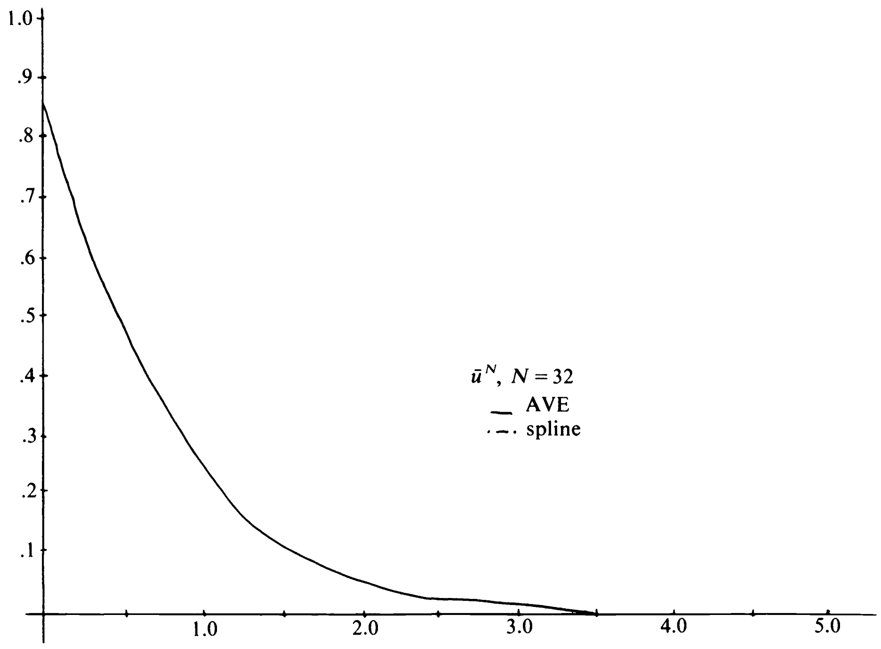


FIG. 4.12

preferable. The spline approximations consistently produced a smaller value for $J(\bar{u}_N; x(0), x_0)$ than did the AVE scheme. Similar conclusions can be drawn in the case of higher dimensional equations. Numerical results for a second order equation is presented in Example 4.2 below. Since the qualitative behavior of each component of the matrix valued functions $\Pi_N^{10}(\cdot)$ was the same as already depicted here in the 1-dimensional case, for the 2-dimensional example we present only the computed values for the 2×2 matrices Π_N^{00}.

Example 4.2. We consider the problem of minimizing

$$J(u; y(0), y_0, \dot{y}(0), \dot{y}_0) = \int_0^\infty [y(t)^2 + \dot{y}(t)^2 + u(t)^2]\,dt \tag{4.4}$$

subject to the harmonic oscillator with delayed restoring force and delayed damping given by

$$\ddot{y}(t) + \dot{y}(t-1) + y(t-1) = u(t), \tag{4.5}$$

or, equivalently

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 \\ -1 & -1 \end{bmatrix} x(t-1) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

where

$$x(t) = \begin{bmatrix} y(t) \\ \dot{y}(t) \end{bmatrix}.$$

In this form J can be written

$$J(u; x(0), x_0) = \int_0^\infty [x(t)^T x(t) + u(t)^2]\,dt.$$

The values of Π_N^{00} for this example are given in Table 4.3 below.

TABLE 4.3

N	AVE	Spline
4	$\begin{bmatrix} 1.9849 & 1.1248 \\ 1.1248 & 1.6538 \end{bmatrix}$	$\begin{bmatrix} 2.1419 & 1.2952 \\ 1.2952 & 1.853528 \end{bmatrix}$
8	$\begin{bmatrix} 2.0511 & 1.1991 \\ 1.1991 & 1.7432 \end{bmatrix}$	$\begin{bmatrix} 2.1394 & 1.296 \\ 1.296 & 1.8568 \end{bmatrix}$
16	$\begin{bmatrix} 2.0914 & 1.2440 \\ 1.2440 & 1.7965 \end{bmatrix}$	$\begin{bmatrix} 2.1389 & 1.2963 \\ 1.2963 & 1.8576 \end{bmatrix}$

We note that using the AVE scheme, Gibson [15] computed Π_{22}^{00} to be

$$\begin{bmatrix} 2.1034 & 1.2574 \\ 1.2574 & 1.8123 \end{bmatrix}$$

with the convergence being monotonic from below in each component of the matrix.

The values of the cost functional (4.4) evaluated using the trajectories obtained by integrating (4.5) with u given by (3.4) and initial conditions

$$y(0)=0, \qquad y(\theta)=\begin{cases} 2(\theta+1), & -1 \leqq \theta \leqq -\frac{1}{2}, \\ -2\theta, & -\frac{1}{2} \leqq \theta \leqq 0, \end{cases}$$

$$\dot{y}(0)=-2, \qquad \dot{y}(\theta)=\begin{cases} 2, & -1 \leqq \theta \leqq -\frac{1}{2}, \\ -2, & -\frac{1}{2} < \theta \leqq 0 \end{cases}$$

are given in Table 4.4.

TABLE 4.4

N	AVE		Spline	
	$J(x(\bar{u}_N))$	$\langle \Pi_N \mathcal{P}_N z_0, \mathcal{P}_N z_0 \rangle$	$J(x(\bar{u}_N))$	$\langle \Pi_N \mathcal{P}_N z_0, \mathcal{P}_N z_0 \rangle$
4	14.4103	11.2650	14.2383	13.8926
8	14.2493	12.3886	14.2201	13.8045
16	14.2165	13.0784	14.2160	13.8308

Example 4.3. In this example we investigate the Mach no. control loop problem described in the introduction. Recall that when the guide vane angle actuator is the control, the state of the system is governed by an equation in R^3 of the form

$$\dot{x}(t)=\begin{bmatrix} -1/\tau & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -\omega^2 & -2\zeta\omega \end{bmatrix} x(t)+\begin{bmatrix} 0 & k/\tau & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} x(t-.33)+\begin{bmatrix} 0 \\ 0 \\ \omega^2 \end{bmatrix} u(t).$$

Here $x=(\delta M, \delta\theta, \delta\dot{\theta})^T$ is made up of the variation in Mach no. δM, the variation in guide vane angle $\delta\theta$, and the variation in guide vane angle velocity $\delta\dot{\theta}$, $u=\delta\theta_A$ is the guide vane angle actuator input and the parameters τ, ω, ζ, and k take on the values 1.964 sec, 6.0 rad/sec, .8, and $-.0117$ deg^{-1} respectively [3].

Parametric studies [3] on the elements of the state weighting matrix in the cost functional revealed that if J was chosen as

$$J(u; x(0), x_0)=\int_0^\infty [x(t)Q_0x(t)+u^2(t)]\,dt$$

where $Q_0=\operatorname{diag}(10^4, 0, 0)$, then the resulting control gains, upon simulation, produced responses which typically did not exceed the physical limitations of the system.

The values for the matrices Π_N^{00} computed using the AVE and spline schemes for $N=2$, 4, and 8 are given in Table 4.5.

TABLE 4.5

N	AVE	Spline
2	$\begin{bmatrix} 8655.2438 & -9.8912 & -.9567 \\ -9.8912 & .0170 & .0017 \\ -.9567 & .0017 & .0002 \end{bmatrix}$	$\begin{bmatrix} 8677.0892 & -9.8201 & -.9477 \\ -9.8201 & .0183 & .0018 \\ -9.477 & .0018 & .0002 \end{bmatrix}$
4	$\begin{bmatrix} 8665.7889 & -9.8528 & -.9523 \\ -9.8528 & .0175 & .0018 \\ -.9523 & .0018 & .0002 \end{bmatrix}$	$\begin{bmatrix} 8676.9237 & -9.8164 & -.9477 \\ -9.8164 & .0185 & .0019 \\ -.9477 & .0019 & .0002 \end{bmatrix}$
8	$\begin{bmatrix} 8671.3161 & -9.8336 & -.9500 \\ -9.8336 & .0179 & .0018 \\ -.9500 & .0018 & .0002 \end{bmatrix}$	$\begin{bmatrix} 8676.9829 & -9.8154 & -.9477 \\ -9.8154 & .0185 & .0019 \\ -.9477 & .0019 & .0002 \end{bmatrix}$

Using the approximating feedback control laws, we computed trajectories for the problem of driving δM from $-.1$ to 0.0 (corresponding to M varying from .8 to .9) and $\delta\theta$ from 8.55 to 0 (corresponding to the guide vane angle varying from 10.48^0 to a steady state of 1.93^0). The initial variation in the guide vane angle velocity, $\delta\dot{\theta}(0)$ was set to 0. The resulting values for the cost functional are given in Table 4.6. The Mach no. and guide vane angle trajectories produced by the approximating control gains computed using the spline scheme with $N = 8$ are plotted in Figs. 4.13 and 4.14, respectively. Our results compare favorably with those obtained by Armstrong and Tripp [3] using an approximating feedback control law produced by a finite difference technique and with those obtained by Daniel [12] using a spline based approximation scheme to solve a similar problem in open loop form.

TABLE 4.6

N	AVE		Spline	
	$J(x(\bar{u}_N))$	$\langle \Pi_N \mathcal{P}_N z_0, \mathcal{P}_N z_0\rangle$	$J(x(\bar{u}_N))$	$\langle \Pi_N \mathcal{P}_N z_0, \mathcal{P}_N z_0\rangle$
2	136.7393	123.9360	136.7355	138.6967
4	136.7369	124.5019	136.7354	138.7345
8	136.7361	124.8019	136.7353	138.7624

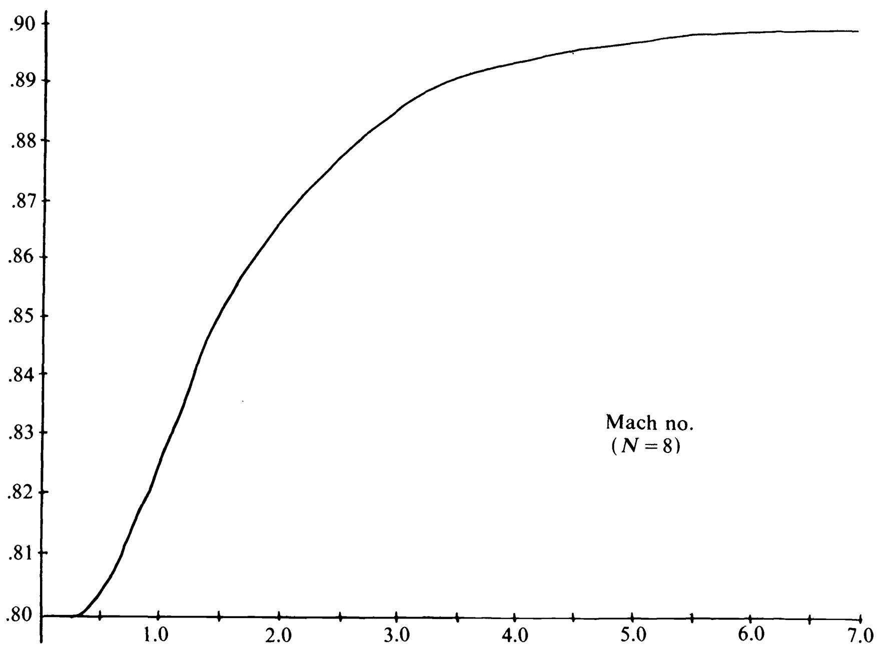


FIG. 4.13

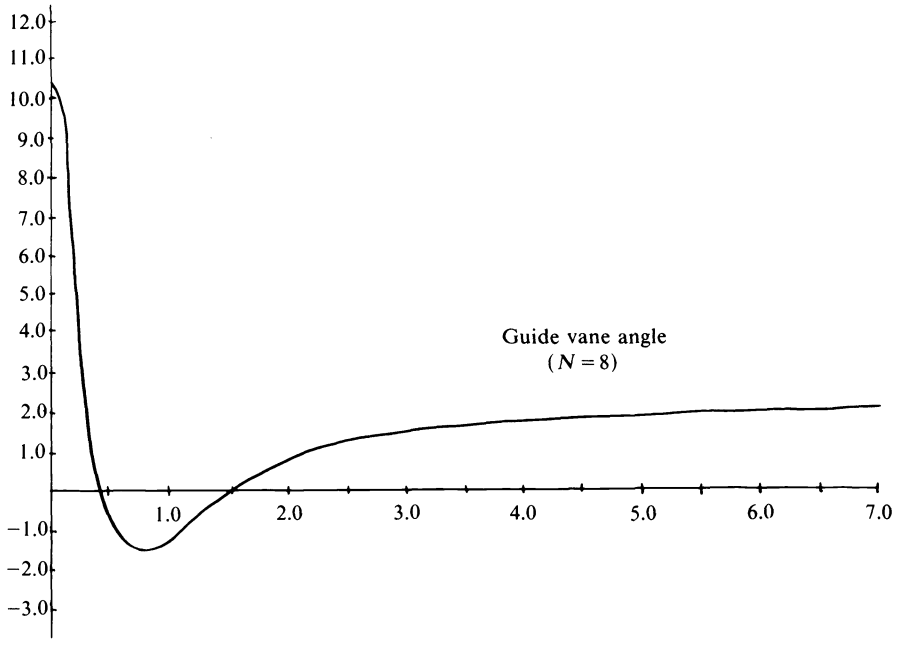


FIG. 4.14

We have also tested the fully discrete schemes on Example 4.1 and compared the results with those obtained with the semi-discrete techniques. The results obtained were not surprising (see [11]). Based on these results and other comparisons of fully discrete vs. semi-discrete versions of these methods we conclude:

(I) For low order approximation (i.e., N small) the results produced by the semi-discrete schemes are better than those produced by the fully discrete methods. As N increases, however, the two techniques yield comparable results.

(II) For N large, using an iterative Newton algorithm, the ORACLS package was able to solve the system (3.27), (3.28), (3.29) arising in the fully discrete methods in roughly half the time it required to solve the matrix Riccati algebraic equation (3.15) resulting from the semi-discrete approximation schemes.

(III) As measured by the magnitude of the cost functional corresponding to a given set of initial conditions and the rate of convergence of the approximating feedback gains, control law (3.31) is preferable to control law (3.30).

(IV) Using the same criteria as in (III), for the fully discrete schemes, as was the case with the semi-discrete schemes, the spline approximations out-perform the AVE approximations.

5. Theoretical considerations and further remarks. We turn to a brief discussion of the convergence exhibited by the "gain" operators in $\Pi_N\mathcal{P}_N$. Careful study of the numerical examples in § 4 reveals that numerically one has weak convergence of $\Pi_N\mathcal{P}_N$, strong (L_2) convergence of the approximating feedback controls $\{\bar{u}_N\}$ of (3.4) and convergence of the performance measures (3.6) and $J(\bar{u}_N; x(0), x_0)$. We have actually proved the weak convergence $\Pi_N\mathcal{P}_N \rightharpoonup \Pi$ in certain special cases that include scalar and second order examples such as those considered in § 4. More precisely, if we consider a hereditary system of the form (1.1) and assume (i) (A_0, B_0) is a controllable system and (ii) Range $(B_0) \supset$ Range (A_1), then $\{\Pi_N \mathcal{P}_N\}$ is uniformly bounded in $L(Z)$. One can then, under assumptions similar to those invoked by Gibson [15, § 6], establish weak convergence of the sequence. For example, if we further assume (iii) the hereditary system is stabilizable, (iv) Q_0, A_0, A_1, B_0 are such that any admissible control drives the state to zero asymptotically, and (v) for N sufficiently large, there exist self-adjoint nonnegative solutions Π_N of the approximate Riccati algebraic equations (3.1), then arguments similar to those in [15] (see Theorem 6.7) can be made to obtain $\Pi_N \mathcal{P}_N \rightharpoonup \Pi$.

If we are willing to assume that Gibson's conjecture 7.1 (essentially, that the approximation scheme when applied preserves uniform asymptotic stability possessed by any original hereditary system) holds for the spline schemes (an assumption that Gibson makes for the "averaging" scheme and one which, based on spectral considerations, we feel confident is valid for both schemes), then we can guarantee existence of solutions Π_N of the approximate Riccati equations (3.1) and furthermore, boundedness of $\{\Pi_N \mathcal{P}_N\}$ follows immediately (e.g., see Theorem 7.5 of [15]).

With respect to the question of strong or trace norm convergence of $\{\Pi_N \mathcal{P}_N\}$, we note that Gibson [15] obtains such convergence for the averaging scheme. This in turn (see Theorem 6.8 and § 7 of [15]) yields convergence of the payoffs. Fundamental to Gibson's arguments (see Theorems 6.1 and 6.9) is the result $S_N^*(t) \to S^*(t)$ where $\{S(t)\}$ is the solution semigroup for the original hereditary system (2.2) with $u=0$, and $\{S_N(t)\}$ is the solution semigroup for the approximating system (e.g., (3.2)) with $u=0$. For our spline schemes we do not believe arguments similar to those of Gibson will suffice to obtain this strong convergence of adjoints; specifically, we do not believe that $\mathcal{A}_N^* \to \mathcal{A}^*$ in a mode sufficient to yield the required convergence of S_N^*. At this time we honestly do not know whether we have strong convergence of $\{\Pi_N \mathcal{P}_N\}$ for the spline schemes. From our numerical results we tend to doubt strong convergence although we *do* observe the desired convergence of the payoffs and can actually establish this theoretically for our spline schemes. If strong convergence of $\{\Pi_N \mathcal{P}_N\}$ is true, we believe a theory somewhat different from that of Gibson's will be required to establish this. In this regard we further note that Kunisch, in his investigation [21] of both the spline and averaging schemes for the finite interval integral quadratic cost control problem for systems with delays, obtains convergence of the payoffs and controls and *weak* convergence of the associated time dependent Riccati operators in a theoretical treatment that is independent of adjoint convergence considerations. However, even if this theory could be extended to treat the infinite interval regulator problems, it would not appear to yield the stronger convergence results for $\{\Pi_N \mathcal{P}_N\}$.

In addition to our numerical findings, there is other evidence that appears to cast doubts on the possibility of strong convergence of $\{\Pi_N \mathcal{P}_N\}$. Recall that $Z_N \subset D(\mathcal{A})$ for each N and since in the representation

$$\Pi_N \mathcal{P}_N = \begin{bmatrix} \Pi_N^{00} & \Pi_N^{01} \\ \Pi_N^{10} & \Pi_N^{11} \end{bmatrix},$$

the first column is "in" $D(\mathcal{A})$, we must have $\Pi_N^{00} = \Pi_N^{10}(0)$. The components of Π do not satisfy such a boundary condition. Indeed (see [15, Theor. 4.4]) col (Π^{00}, Π^{10}) is "in" $D(\mathcal{A}^*)$. This suggest that the convergence of $\Pi_N \mathcal{P}_N$ to Π cannot occur in a very strong mode.

The theoretical considerations above aside, we believe the evidence is quite substantial in support of our contention that the spline methods offer an attractive means for computing feedback gains in delay system regulator problems. We close with a summary of remarks on the merits of our spline schemes over the averaging scheme (we do not mean to discredit the averaging technique—for many problems it should perform quite admirably—rather we wish to argue that in some examples, the spline schemes can offer significant improvements). We recall from the numerical results of § 4 that (on these examples) the linear spline scheme always is as good as the averaging scheme, and in some cases it is better (faster convergence, better approximation at low orders). In some situations the averaging scheme yields a Riccati system that is difficult to solve numerically, while this never (in our experience) occurs with the spline schemes.

As further evidence of the usefulness of the spline schemes, we offer the recent experiences of Gibson (private communication) and Ito [17] in using the averaging and spline approximation schemes as a basis for computation of closed loop eigenvalues for delay systems (e.g., using the system (3.2), (3.3) to compute eigenvalues that approximate those of the feedback system (2.2), (2.13)). In these efforts, the spline based schemes performed in a far superior manner. We interpret this as another argument in favor of construction of feedback gains via our spline schemes.

Acknowledgments. The authors would like to express their appreciation to E. S. Armstrong, J. S. Gibson, and K. Kunisch for fruitful conversations during the course of the research reported above. In particular, Ernest Armstrong stimulated our interest in the NTF control problem and was most helpful with his advice on the software packages (especially some of those in ORACLS) at NASA Langley Research Center. Steve Gibson generously discussed his own computational efforts with the averaging schemes, both for Riccati operator calculations and eigenvalue methods.

REFERENCES

[1] Y. Alekal, P. Brunovsky, D. H. Chyung and E. B. Lee, *The quadratic problem for systems with time delays*, IEEE Trans. Automat. Control, AC-16 (1971), pp. 673–687.

[2] E. S. Armstrong, ORACLS—*A Design System for Linear Multivariable Control*, Marcel Dekker, New York, 1980.

[3] E. S. Armstrong and J. S. Tripp, *An application of multivariable design techniques to the control of the National Transonic Facility*, NASA Tech. Paper 1887, NASA Langley Research Center, Hampton, VA, August, 1981.

[4] H. T. Banks, *Parameter identification techniques for physiological control systems*, Lectures in Applied Mathematics, 19, American Mathematical Society, Providence, RI, 1981.

[5] H. T. Banks and J. A. Burns, *Hereditary control problems: numerical methods based on averaging approximations*, SIAM J. Control Optim., 16 (1978), pp. 169–208.

[6] H. T. Banks, J. A. Burns and E. M. Cliff, *A comparison of numerical methods for identification and optimization problems involving control systems with delays*, LCDS Tech. Rep. 79-7, Brown University, Providence, RI, 1979.

[7] ———, *Parameter estimation and identification for systems with delays*, SIAM J. Control Optim., 19 (1981), pp. 791–828.

[8] H. T. Banks and P. L. Daniel, *Estimation of delays and other parameters in nonlinear functional differential equations*, SIAM J. Control Optim., 21 (1983), pp. 895–915.

[9] H. T. Banks and F. Kappel, *Spline approximations for functional differential equations*, J. Differential Equations, 34 (1979), pp. 496–522.

[10] H. T. Banks and I. G. Rosen, *Spline approximations for linear nonautonomous delay systems*, ICASE Rep. 81-33, NASA Langley Research Center, Hampton, VA, October, 1981; J. Math. Anal. Appl., to appear.

[11] H. T. Banks, I. G. Rosen and K. Ito, *A spline based technique for computing Riccati operators and feedback controls in regulator problems for delay equations*, ICASE Rep. 82-31, NASA Langley Research Center, Hampton, VA, September, 1982.

[12] P. L. DANIEL, *Spline approximations for nonlinear hereditary control systems*, ICASE Rep. 82-10, NASA Langley Research Center, Hampton, VA, April, 1982; J. Optim. Theory Appl., to appear.

[13] M. C. DELFOUR, *The linear quadratic optimal control problem for hereditary differential systems: theory and numerical solution*, Appl. Math. Optim., 3 (1977), pp. 101–162.

[14] D. H. ELLER, J. K. AGGARWAL AND H. T. BANKS, *Optimal control of linear time-delay systems*, IEEE Trans. Automat. Control, AC-14 (1969), pp. 678–687.

[15] J. S. GIBSON, *Linear-quadratic optimal control of hereditary differential systems: infinite dimensional Riccati equations and numerical approximations*, SIAM J. Control Optim., 21 (1983), pp. 95–139.

[16] G. GUMAS, *The dynamic modeling of a slotted test section*, NASA Cr-159069, NASA Langley Research Center, Hampton, VA, 1979.

[17] K. ITO, *On the approximation of eigenvalues associated with functional differential equations*, ICASE Rep. 82-29, NASA Langley Research Center, Hampton, VA, September, 1982.

[18] D. L. KLEINMAN, *On an iterative technique for Riccati equations computation*, IEEE Trans. Automat. Control, AC-13 (1968), pp. 114–115.

[19] N. N. KRASOVSKII, *On the analytic construction of an optimal control in a system with time lags*, J. Appl. Math. Mech., 26 (1962), pp. 50–67.

[20] ———, *The approximation of a problem of analytic design of controls in a system with time-lag*, J. Appl. Math. Mech., 28 (1964), pp. 876–885.

[21] K. KUNISCH, *Approximation schemes for the linear quadratic optimal control problem associated with delay equations*, SIAM J. Control Optim., 20 (1982), pp. 506–540.

[22] H. KWAKERNAAK AND R. SIVAN, *Linear Optimal Control Systems*, John Wiley, New York, 1972.

[23] A. J. LAUB, *A Schur method for solving algebraic Riccati equations*, IEEE Trans. Automat. Control, AC-24 (1979), pp. 913–921.

[24] J. E. POTTER, *Matrix quadratic solutions*, SIAM J. Appl. Math., 14 (1966), pp. 496–501.

[25] I. G. ROSEN, *A discrete approximation framework for hereditary systems*, J. Differential Equations, 40 (1981), pp. 377–449.

[26] D. W. ROSS, *Controller design for time lag systems via a quadratic criterion*, IEEE Trans. Automat. Control, AC-16 (1971), pp. 664–672.

[27] D. W. ROSS AND I. FLUGGE-LOTZ, *An optimal control problem for systems with differential-difference equation dynamics*, SIAM J. Control Optim., 7 (1969), pp. 609–623.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

NOTE ON FINITE DIFFERENCE APPROXIMATIONS TO BURGERS' EQUATION*

H. AREF† AND P. K. DARIPA†

Abstract. Standard finite difference approximations to Burgers' equation are considered from the point of view of dynamical systems theory. Phase plane analyses for discretizations with a few grid points are presented. These show the existence of initial conditions leading to spurious solutions with unlimited amplitude growth due to nonconservation of kinetic energy by the nondissipative terms in the discretizations. It is shown that such solutions may be found even for arbitrarily fine pointwise resolution, i.e., for arbitrarily many grid points. On the other hand, an energy conserving discretization of the nondissipative terms removes all spurious solutions of this kind. The results obtained seem to complement recent investigations of the steady state problem.

Key words. finite difference scheme, Burgers' equation, phase plane analysis

1. Introduction. We have been studying standard finite difference approximations to Burgers' equation [1] as part of an attempt to compare various numerical methods for solving partial differential equations. Burgers' equation is a valuable test case for such studies since it is directly solvable by the Cole–Hopf [2], [3] transformation, and numerically it is accessible by standard finite difference, finite element and spectral methods. One may also formulate a "particle method" for Burgers' equation by appealing to the possibility of a pole decomposition [4] for this equation.

Our primary interest is in the dynamics of two-dimensional flow, particularly the two-dimensional Euler equation and the representation of its solutions by an assembly of point vortices [5]–[7]. The pole decomposition of Burgers' equation can be seen as an analogue of the vortex decomposition of two-dimensional incompressible hydrodynamics formulated as a field theory for the stream function [7]. Hence, a comparison of "standard" numerical techniques, such as finite differences, for Burgers' equation with the pole decomposition solutions suggests itself. As a preliminary to this a study of the finite difference equations themselves was performed, and, since the ideas of pole and vortex decomposition quickly lead to notions from dynamical systems theory, the finite difference equations were considered from this point of view. There has recently been much interest in using results from the theory of dynamical systems to study in greater detail the nature of the instabilities to which numerical schemes are susceptible [8], [9].

A standard finite difference approximation to Burgers' equation consists of a set of ordinary differential equations, one for the field value at each grid point, coupled through quadratic interactions. As is well known, problems of precisely this format may display chaotic solutions [10]. The Lorenz equations [11] are a case in point. If such behavior occurs for a finite difference approximation to Burgers' equation, it must clearly come from the numerical scheme, since the continuum Burgers' equation is in some sense "integrable." (Burgers' equation is dissipative, and so integrability is not immediately defined. However, its pole decomposition equations can be imbedded in an integrable Hamiltonian system, the Calogero–Moser system [4]. Taflin [12] discusses the concept of integrability and Burgers' equation.)

* Received by the editors August 5, 1982, and in revised form April 29, 1983. This research was supported by the National Science Foundation under grant MEA 81-16910 to Brown University.
† Division of Engineering, Brown University, Providence, Rhode Island 02912.

We must state right away that we did not find chaotic behavior for our standard finite difference approximations to Burgers' equation. However, the dynamical systems point of view suggested that we look in detail at the "phase plane" for discretizations with a small number of grid points. This was done for $N = 3, 4, 5$ and 6, and we found for $N = 4$ that there exist initial states which grow in time beyond all bounds. Such solutions are physically unacceptable. Moreover, we are able to show, by a rather obvious argument, that for any N which is a multiple of 4 such initial states with spurious long time behavior will exist. This result seems to complement recent work on the *steady* Burgers' equation [13], [14]. The result is unsettling because it shows that even for arbitrarily fine pointwise spatial resolution the standard finite difference approximation to Burgers' equation can give physically unacceptable results for certain initial conditions. A complete resolution of this "paradox" is not given. However, we do show that if a slightly different finite difference approximation is employed, which conserves the discretized kinetic energy, then no initial conditions can lead to the above pathology of infinite amplitude growth.

We must emphasize that although our entire discussion centers on Burgers' equation our objective is not to solve that equation numerically. Burgers' equation is trivial. However, as just mentioned, *any* finite difference approximation to the material derivative of a field results in a system of ODEs with quadratic couplings. Thus, we submit that the method of analysis exemplified here is of general applicability and usefulness. (For a related discussion, involving the Fourier amplitude equations for two-dimensional flow, see [15]).

2. Preliminaries.

We are concerned with Burgers' equation,

$$u_t + uu_x = \nu u_{xx},$$

for a real field, $u = u(x, t)$, and specifically consider the initial value problem:

$$u(x, 0) = u_0(x)$$

for periodic boundary conditions on an interval of length L:

$$u(x+L, t) = u(x, t).$$

To solve this problem numerically, we introduce the grid values $u_k(t) = u(kL/N, t)$ for $k = 0, \cdots, N-1$, and discretize u_{xx} and uu_x according to

$$u_{xx} = \left(\frac{N}{L}\right)^2 (u_{k+1} + u_{k-1} - 2u_k),$$

$$uu_x = \left(\frac{N}{2L}\right) u_k(u_{k+1} - u_{k-1}).$$

These expressions are accurate to $O((\Delta x)^2)$, where $\Delta x = L/N$. Substituting into Burgers' equation we obtain a system of coupled ODEs of first order for the amplitudes u_k. We may nondimensionalize these equations by setting

$$v_k(s) = \left(\frac{L}{N\nu}\right) u_k\left(\frac{L^2 s}{N^2 \nu}\right), \qquad k = 0, \cdots, N-1.$$

The proposed discretization of Burgers' equation then reads

$$\frac{dv_k}{ds} = -\left(\frac{v_k}{2}\right)(v_{k+1} - v_{k-1}) + v_{k+1} + v_{k-1} - 2v_k. \tag{1}$$

There is one ODE for each grid point amplitude v_k, $k=0,\cdots,N-1$, making a system of N coupled equations in all. As noted previously, the couplings are quadratic. The periodic boundary condition now means that $v_{k+N}=v_k$ for all nondimensional times s.

The discretization (1) conserves momentum in the sense that

$$\sum_{k=0}^{N-1}\frac{dv_k}{ds}=0.$$

Since the value of the momentum of the field may be altered at will by subjecting it to a Galilean transformation, viz.

$$U(x,t)=u(x-ct,t)+c,$$

we shall consistently assume that the total momentum vanishes. For the discrete system we thus assume:

$$\sum_{k=0}^{N-1} v_k=0.$$

This restriction on the sum of the v_k will prove very convenient later.

Kinetic energy, on the other hand, is not conserved by the scheme (1), i.e.

$$\sum_{k=0}^{N-1} v_k\frac{dv_k}{ds}\neq 0,$$

even if the dissipative (linear) terms are omitted. It may be shown that the kinetic energy of the field u,

$$E_{\text{kin}}\equiv\frac{1}{2}\int_0^L u^2\,dx,$$

satisfies the equation of motion

$$\frac{dE_{\text{kin}}}{dt}=-\nu\int_0^L\left(\frac{\partial u}{\partial x}\right)^2 dx$$

and thus decreases unless u is constant in x or $\nu=0$. Hence solutions to the discretized equations that make the energy increase indefinitely or solutions that are steady in time but vary in x are not physically acceptable and must be classified as artifacts of the numerical scheme. We shall meet with such solutions in the next section. In § 4 we shall then trace the origin of these spurious solutions to the fact that scheme (1) does not conserve kinetic energy in the nondissipative limit.

3. Case studies for small N. For $N=2$ we are led to consider the system

$$\dot v_0=-\left(\frac{v_0}{2}\right)(v_1-v_1)+v_1+v_1-2v_0=2(v_1-v_0),$$

$$\dot v_1=2(v_0-v_1)$$

(where the dot signifies a derivative with respect to s) so that

$$\dot v_0-\dot v_1=-4(v_0-v_1).$$

Thus v_0-v_1 decays exponentially and since v_0 and v_1 sum to zero (total momentum is assumed to vanish; see § 2) we see that both v_0 and v_1 must decay to zero. This is completely in accord with our expectations for the continuum equation.

For $N=3$ we get the more interesting system

$$\dot{v}_0 = -\left(\frac{v_0}{2}\right)(v_1 - v_2) + v_1 + v_2 - 2v_0,$$

$$\dot{v}_1 = -\left(\frac{v_1}{2}\right)(v_2 - v_0) + v_2 + v_0 - 2v_1,$$

$$\dot{v}_2 = -\left(\frac{v_2}{2}\right)(v_0 - v_1) + v_0 + v_1 - 2v_2.$$

It is not difficult to see that this system has the integral

$$I \equiv v_0 v_1 v_2 \exp(9s)$$

(when $v_0 + v_1 + v_2 = 0$). This integral immediately shows us that if the equations for $N=3$ have a steady state solution, one of the amplitudes, say v_0, must vanish. But if $v_0 = 0$, we have $v_1 = -v_2 \equiv v$ and the system reduces to

$$\dot{v} = \left(\frac{v}{2}\right)(v-6)$$

with steady states corresponding to $v=0$ and $v=6$ and in general the solution

$$v(s) = \frac{6v(0)\exp(-3s)}{6 - v(0)(1 - \exp(-3s))}.$$

Figure 1 shows projections of several phase space trajectories, which reside in the plane $v_0 + v_1 + v_2 = 0$, onto the (v_1, v_2)-plane. We see that for certain initial conditions

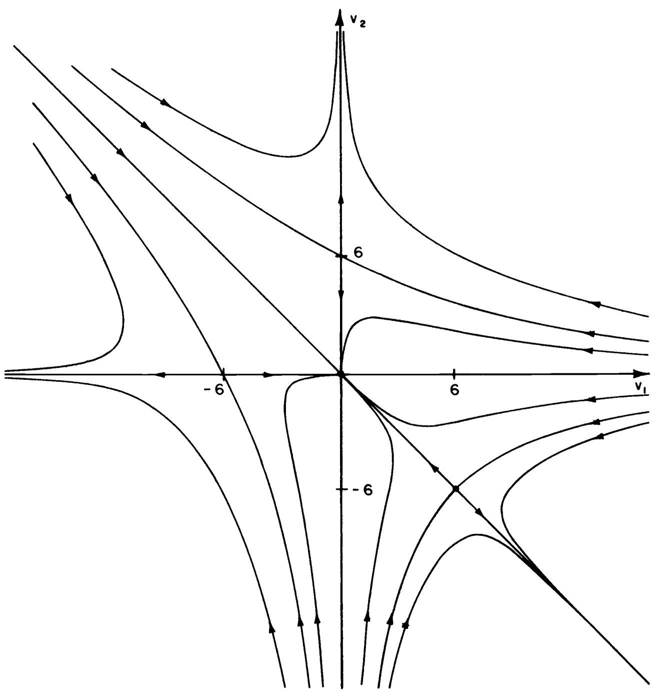


FIG. 1. *Phase trajectories, projected onto the* (v_1, v_2)*-plane for scheme* (1) *with* $N=3$.

the trajectories depart to infinity due to the existence of saddle points at $(0, 6, -6)$, $(-6, 0, 6)$ and $(6, -6, 0)$.

For $N = 4$ we must consider the system

$$\dot{v}_0 = -\left(\frac{v_0}{2}\right)(v_1 - v_3) + v_1 + v_3 - 2v_0,$$

$$\dot{v}_1 = -\left(\frac{v_1}{2}\right)(v_2 - v_0) + v_2 + v_0 - 2v_1,$$

$$\dot{v}_2 = -\left(\frac{v_2}{2}\right)(v_3 - v_1) + v_3 + v_1 - 2v_2,$$

$$\dot{v}_3 = -\left(\frac{v_3}{2}\right)(v_0 - v_2) + v_0 + v_2 - 2v_3.$$

We notice that

$$v_3\dot{v}_1 + \dot{v}_3 v_1 = (v_1 + v_3)(v_2 + v_0) - 4v_1v_3,$$

$$v_0\dot{v}_2 + \dot{v}_0 v_2 = (v_0 + v_2)(v_1 + v_3) - 4v_0v_2.$$

Hence

$$\frac{d}{ds}(v_0v_2 - v_1v_3) = -4(v_0v_2 - v_1v_3)$$

and the system has the integral

$$J \equiv (v_0v_2 - v_1v_3)\exp(4s).$$

We now observe that the full four-dimensional system has a discrete symmetry: The constraint $v_0 = -v_3 \equiv U$, $v_1 = -v_2 \equiv V$ will be preserved by the equations of motion. The evolution of U, V is governed by

$$\dot{U} = -\left(\frac{U}{2}\right)(U + V) + V - 3U,$$

$$\dot{V} = \left(\frac{V}{2}\right)(U + V) + U - 3V.$$

Let

$$X = (U + V)\sqrt{2}, \qquad Y = U - V.$$

Then the equations for U and V may be written as

$$\dot{X} = \frac{\partial G}{\partial X}, \qquad \dot{Y} = \frac{\partial G}{\partial Y}$$

with

$$G(X, Y) = -X^2(1 + \tfrac{1}{4}Y) - 2Y^2.$$

Several level curves of the potential G are shown in Fig. 2a and the (X, Y)-flow, which arises as the family of trajectories orthogonal to the level curves, is shown in Fig. 2b. Again we see spurious steady states and again they apparently give rise to modes of evolution in which some discrete amplitudes grow indefinitely.

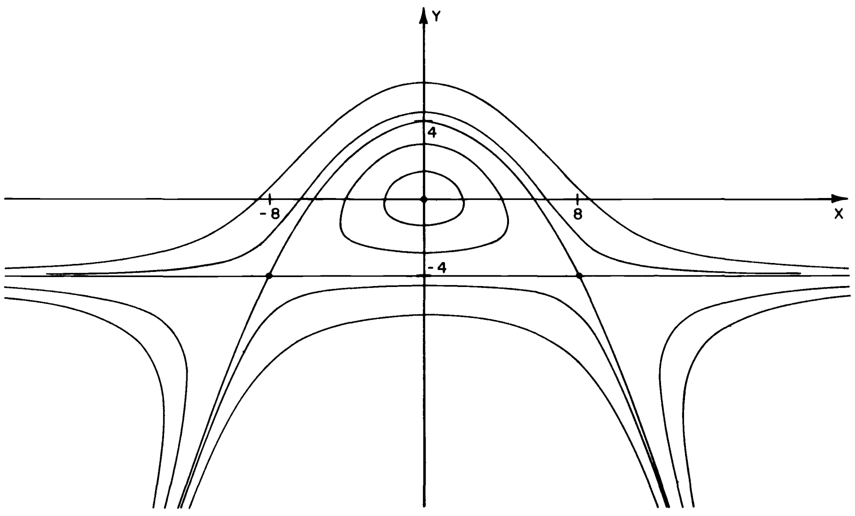


FIG. 2(a). *The potential $G(X, Y)$ pertaining to the analysis of scheme* (1) *with $N = 4$. The separatrix consists of the line $Y = -4$ and the parabola $Y = -\frac{1}{8}X^2 + 4$.*

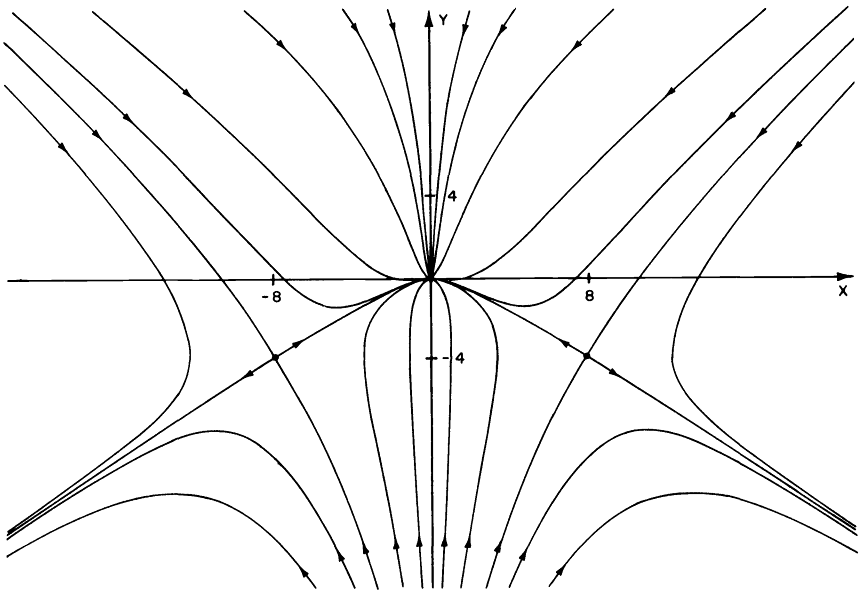


FIG. 2(b). *The (X, Y)-flow pertaining to scheme* (1) *with $N = 4$.*

We conclude this section by observing that increasing N will not rule out the existence of initial conditions leading to physically spurious solutions. This follows, for example, if $N = 4n$. The finite difference amplitude equations will then have a discrete symmetry which consists in every fourth amplitude being the same, i.e., $v_{k+4p} = v_k$ where $k = 0, 1, 2, 3$ and $p = 0, 1, \cdots, n-1$. Within the subspace singled out by this symmetry, the system of ODEs reduces to n replicas of the $N = 4$ system discussed above. Now consider an initial condition with the repeated period-four symmetry in the region that leads to indefinite growth. Such an initial condition can be found in the form

$$(v_0, \cdots, v_{N-1}) = (a, b, -a, -b, a, b, -a, -b, \cdots, a, b, -a, -b)$$

according to our analysis of the $N=4$ system above. Since the region in the $(v_0, v_1, -v_0, -v_1)$ subspace that leads to indefinite growth is obviously an open set (cf. Fig. 2b), it must be possible to find initial conditions for the full $N=4n$ system of the form

$$(v_0, \cdots, v_{N-1}) = (a+e_1, b+e_2, -a+e_3, -b+e_4, \cdots)$$

where e_1, e_2, e_3, e_4, etc. are different, i.e. initial conditions *without* the discrete, period four symmetry, that still lead to indefinitely large amplitudes. We have tried several such initial conditions and checked numerically that they indeed lead to unlimited amplitude growth.

To see what this property of scheme (1) means in terms of the original variables u_k we must refer to the definition of v_k in § 2. Then we see that if $u_k(0) = F_k$, $k = 0, 1, 2, 3$, is a set of initial amplitudes that leads to indefinite growth for a discretization with 4 grid points, $u_{k+4p}(0) = nF_k$, $k = 0, 1, 2, 3$, $p = 0, 1, \cdots, n-1$, will lead to indefinite growth for a discretization with $N = 4n$ grid points. We shall restate this result in terms of the so-called *cell Reynolds number* in § 5.

4. Remedy. Having described the pathologies of the scheme (1), we must now prescribe a cure. We note that all the spurious solutions in § 3 violate the requirement that kinetic energy be dissipated. Indeed with diverging amplitudes the discretized kinetic energy clearly tends to infinity. Hence, if a scheme that conserves energy (when the dissipative terms are neglected) can be found, unbounded spurious solutions, such as those found for scheme (1), should disappear. It is not difficult to find such a scheme: We first recall that Burgers' equation may be written in "conservation form"

$$u_t + (\tfrac{1}{2}u^2)_x = \nu u_{xx},$$

and this form can then be discretized. This leads to

$$\frac{dv_k}{ds} = -\frac{1}{4}(v_{k+1}^2 - v_{k-1}^2) + v_{k+1} + v_{k-1} - 2v_k \tag{2}$$

with the same rescalings as before. This scheme again conserves momentum but not energy. Few-amplitude truncations of the system (2) can be shown to display spurious solutions as in § 3.

However, we now have two schemes and new schemes can be constructed by forming convex combinations of them. In particular, we can ask whether some such combination will conserve energy. Thus we add (1) and (2) with "weights" w and $1-w$ respectively and impose the condition that the combination conserve energy (when the linear, dissipative terms are neglected). This turns out to determine w uniquely ($w = \frac{1}{3}$) and the resulting scheme is [16]

$$\frac{dv_k}{ds} = -\frac{1}{6}(v_{k+1} - v_{k-1})(v_{k+1} + v_{k-1} + v_k) + v_{k+1} + v_{k-1} - 2v_k. \tag{3}$$

Since it arose by linear (convex) combination of (1) and (2), scheme (3) clearly still conserves momentum. Using the Cauchy–Schwarz inequality it is also easy to show from (3) that

$$\sum_{k=0}^{N-1} v_k \frac{dv_k}{ds} \leqq 0$$

where the equal sign only holds if all v_k are identical (and hence zero).

We remark that it may sometimes be undesirable to have an energy-conserving discretization (in the limit of zero viscosity) for physical reasons. For example, if the objective is to track shock formation in an initial value calculation, energy conservation may actually be an unwanted constraint [17, p. 252].

5. Concluding remarks. It is useful to state precisely what the remedy of § 4 was. Essentially it consisted in discretizing the factor u of uu_x by $(v_{k+1}+v_{k-1}+v_k)/3$ instead of just by v_k (compare (3) to (1)). This kind of differencing to produce the value of the field itself at a point, as opposed to the values of derivatives, arises frequently and naturally in the finite element method. In fact, the discretization of the convective derivative in (3) can be obtained using the finite element method with a basis of piecewise linear functions.

It is worth reiterating that we have not found evidence of chaotic behavior. This seems to be due to the absence of any free parameters in our discretized equations. With the relative magnitude of linear and nonlinear terms that is forced upon us here, the phase space flows seem to be dominated by sinks and saddles. We have not attempted to insert a variable parameter into these equations in order to seek out regimes of chaotic solutions since the physical significance of such an exercise seemed unclear.

We may restate our results by saying that close to the origin even the "naive" schemes (1) and (2) give qualitatively acceptable results. In fact, we can identify a certain region, $|v_k| \leqq v_{\max}(N)$ for $k=0, 1, \cdots, N-1$, within which the discretization behaves in a qualitatively correct way compared to the continuum equation. In terms of the field amplitudes u_k and the spatial resolution Δx (§ 2) this criterion takes the form $|u_k|\Delta x/\nu \leqq v_{\max}(N)$, i.e. the cell Reynolds number, $\mathrm{Re} \equiv (\max_k |u_k|)\Delta x/\nu$, must be chosen less than some N-dependent upper bound. Clearly if $v_{\max}(N)$ tends to infinity with increasing N, spurious solutions become less troublesome with increasing resolution. However, we have just seen that for scheme (1), $v_{\max}(4n) \leqq v_{\max}(4)$ (see the argument given at the end of § 3) and thus that $v_{\max}(N)$ does not increase systematically in this case. In terms of the original variables u_k we must go to ever larger amplitudes as N increases to encounter the spurious behavior. But for scheme (1) spurious solutions can be found; for scheme (3), they cannot. We may mention in conclusion that for cell Reynolds number less than 2 scheme (1) is usually found to be adequate [13], [14].

Acknowledgments. We are indebted to Professor T. F. Morse for comments and discussion. One of us (P.K.D.) would like to thank the Graduate School at Brown University for fellowship support.

REFERENCES

[1] J. M. BURGERS, *A mathematical model illustrating the theory of turbulence*, Adv. Appl. Mech., 1 (1948), pp. 171–199.

[2] J. D. COLE, *On a quasilinear parabolic equation occurring in aerodynamics*, Quart. Appl. Math., 9 (1951), pp. 225–236.

[3] E. HOPF, *The partial differential equation $u_t + uu_x = \mu u_{xx}$*, Comm. Pure Appl. Math., 3 (1950), pp. 201–230.

[4] D. V. CHOODNOVSKY AND G. V. CHOODNOVSKY, *Pole expansions of nonlinear partial differential equations*, Il Nuovo Cimento B, 40 (1977), pp. 339–53.

[5] A. J. CHORIN, *Numerical study of slightly viscous flow*, J. Fluid Mech., 57 (1973), pp. 785–96.

[6] A. LEONARD, *Vortex methods for flow simulation*, J. Comp. Phys., 37 (1980), pp. 289–335.

[7] H. AREF, *Integrable, chaotic and turbulent vortex motion in two-dimensional flows*, Ann. Rev. Fluid Mech., 15 (1983), pp. 345–89.

[8] M. YAMAGUTI AND S. USHIKI, *Chaos in numerical analysis of ordinary differential equations*, Physica D, 3 (1981), pp. 618–26.

[9] S. USHIKI, *Central difference scheme and chaos*, Physica D, 4 (1982), pp. 407–24.

[10] S. JORNA, ed., *Topics in Nonlinear Dynamics. A Tribute to Sir Edward Bullard*, Conference Proc. 46, American Institute of Physics, New York, 1978.

[11] E. N. LORENZ, *Deterministic nonperiodic flow*, J. Atmos. Sci., 20 (1963), pp. 130–41.

[12] E. TAFLIN, *Analytic linearization, Hamiltonian formalism, and infinite sequences of constants of the motion for Burgers' equation*, Phys. Rev. Lett., 47 (1981), pp. 1425–8.

[13] A. B. STEPHENS AND G. R. SHUBIN, *Multiple solutions and bifurcations of finite difference approximations to some steady problems of fluid dynamics*, this Journal, 2 (1981), pp. 404–415.

[14] R. B. KELLOGG, G. R. SHUBIN AND A. B. STEPHENS, *Uniqueness and the cell Reynolds number*, SIAM J. Numer. Anal., 17 (1980), pp. 733–739.

[15] O. H. HALD, *Constants of motion in models of two-dimensional turbulence*, Phys. Fluids, 19 (1976), pp. 914–15.

[16] R. D. RICHTMYER AND K. W. MORTON, *Difference Methods for Initial-Value Problems*, Interscience, New York, 1967.

[17] 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. 5, No. 4, December 1984

NUMERICAL METHODS FOR STIFF SYSTEMS OF TWO-POINT BOUNDARY VALUE PROBLEMS*

JOSEPH E. FLAHERTY† AND ROBERT E. O'MALLEY, JR.†

Abstract. We develop numerical procedures for constructing asymptotic solutions of certain nonlinear singularly perturbed vector two-point boundary value problems having boundary layers at one or both endpoints. The asymptotic approximations are generated numerically and can either be used as is or to furnish a general purpose two-point boundary value code with an initial approximation and the nonuniform computational mesh needed for such problems. The procedures are applied to a model problem that has multiple solutions and to problems describing the deformation of a thin nonlinear elastic beam that is resting on an elastic foundation.

1. Introduction.

Initial value problems for stiff systems of ordinary differential equations are now considered to be relatively tractable numerically (cf. Enright et al. [7]). However, codes for stiff (or singularly perturbed) boundary value problems are not readily available, even though these problems arise in a great many applications. (The term stiff is more commonly restricted to singularly perturbed initial value problems, when stability forces stepsizes for explicit integration schemes to be severely restricted in regions where the solution is relatively smooth. We use the term in a broader context where solutions are not smooth everywhere, but are instead characterized by both intervals of smoothness and by rapid changes over narrow intervals.)

In this paper we consider asymptotic and numerical methods for singularly perturbed two-point boundary value problems of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{y}, t, \varepsilon), \qquad \varepsilon\dot{\mathbf{y}} = \mathbf{g}(\mathbf{x}, \mathbf{y}, t, \varepsilon), \tag{1.1}$$

$$\mathbf{a}(\mathbf{x}(0), \mathbf{y}(0), \varepsilon) = \mathbf{0}, \qquad \mathbf{b}(\mathbf{x}(1), \mathbf{y}(1), \varepsilon) = \mathbf{0}, \tag{1.2a,b}$$

where $\mathbf{x}$, $\mathbf{y}$, $\mathbf{a}$, and $\mathbf{b}$ are vectors of dimension m, n, q, and $r = m+n-q$, respectively, and ε is a small positive parameter.

Although many special problems of this form can be solved by known asymptotic or numerical techniques, the general problem is very difficult and beyond our current understanding. The form of equations (1.1), (1.2) imply that, whenever $\mathbf{g}$ is not small, $\mathbf{y}$ varies rapidly relative to $\mathbf{x}$. The behavior of the solution in these zones of rapid transition can be very complicated. For example, $\mathbf{y}$ can "jump" abruptly in a narrow boundary layer near $t=0$ and/or 1. These jumps can also occur at interior locations where solutions or their derivatives will become unbounded as $\varepsilon \to 0$. The locations of the interior layers are generally unknown and must be determined as part of the solution process. Examples of these and other phenomena are discussed in, e.g., O'Malley [21], Kevorkian and Cole [18], Pearson [25], [26], Hemker [15], and Flaherty and O'Malley [11].

The traditional numerical techniques for two-point boundary value problems all have difficulties with singularly perturbed problems unless the grid that is used for the

* Received by the editors March 8, 1983, and in final form August 10, 1983. This research was partially supported by the U.S. Air Force Office of Scientific Research, Air Force Systems Command, USAF, under grant AFOSR 80-0192, the Office of Naval Research under contract N00014-81K-056, and by the U.S. Army Research Office under grant DAAG29-82-K-0197. Partial support was also provided under NASA contracts NAS1-15810 and NAS1-17070 while the authors were in residence at the Institute for Computer Applications in Science and Engineering, Hampton, Virginia.

† Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, New York 12181.

discretization is appropriately fine, at least within boundary or interior layers. If the grid is not fine enough to resolve the layers, the computed solution typically exhibits spurious mesh oscillations. There are, however, special purpose schemes that can solve some singularly perturbed boundary value problems without using a fine discretization in transition regions. Most notable among these are the "upwind" or one-sided finite difference schemes (cf., e.g., Kreiss and Kreiss [19] or Osher [24]) and the exponentially weighted finite difference and finite element schemes (cf., e.g., Flaherty and Mathon [9] or Hemker [15]). These schemes must usually be either restricted to relatively simple problems or employ complicated algebraic transformations.

In view of these theoretical and computational difficulties, we simplify problem (1.1), (1.2) considerably by assuming, in addition to natural smoothness hypotheses, that (i) $\mathbf{g}$, $\mathbf{a}$, and $\mathbf{b}$ are linear functions of the "fast" variable $\mathbf{y}$, (ii) the $n \times n$ Jacobian

$$(1.3) \qquad \mathbf{G}(\mathbf{x}, t) := \mathbf{g}_{\mathbf{y}}(\mathbf{x}, \mathbf{y}, t, 0)$$

has a strict hyperbolic splitting with $k \geqq 0$ stable and $n-k \geqq 0$ unstable eigenvalues for all $\mathbf{x}$ and $0 \leqq t \leqq 1$, and (iii) $q \geqq k$ and $r \geqq n-k$. (By a stable (unstable) eigenvalue, we mean one with strictly negative (positive) real part.) A corresponding theory for problems with quadratic dependence on $\mathbf{y}$ is very limited (cf., e.g., Howes [17] which discusses second-order scalar equations). This, of course, limits extension of a numerical theory, but encourages further numerical experimentation.

The assumed hyperbolic splitting restricts any rapid variations in $\mathbf{y}$ to occur in boundary layer regions near $t = 0$ and/or 1. Thus, we unfortunately have eliminated many important and challenging problems having interior or "shock" layers. Some numerical work on these problems was done by Kreiss and Kreiss [19], Osher [24], and O'Malley [23].

In a series of three papers, Ascher and Weiss [2], [3], [4] (this issue, pp. 811–829) show that symmetric, or centered, collocation schemes could be used on problems that satisfied assumptions similar to ours provided that appropriately fine meshes were used in the endpoint boundary layers. Our approach is somewhat different in that we use the assumed hyperbolic splitting to find an asymptotic solution of problem (1.1), (1.2) which is composed of a limiting outer solution $(\mathbf{X}_0(t), \mathbf{Y}_0(t))$ and boundary layer corrections neat $t = 0$ and 1. The limiting solution satisfies a reduced system, which is obtained from (1.1) by formally setting ε to zero, i.e.,

$$(1.4\text{a,b}) \qquad \dot{\mathbf{X}}_0 = \mathbf{f}(\mathbf{X}_0, \mathbf{Y}_0, t, 0), \qquad \mathbf{0} = \mathbf{g}(\mathbf{X}_0, \mathbf{Y}_0, t, 0).$$

Because $\mathbf{G}$ is everywhere nonsingular, we can solve (1.4b) for $\mathbf{Y}_0 = \mathbf{Y}_0(\mathbf{X}_0, t)$ in a locally unique way, and there remains the mth order differential system (1.4a) for determining $\mathbf{X}_0(t)$.

In order to completely specify the limiting solution, we must prescribe m boundary conditions for (1.4a). We do this in § 2 by providing a "cancellation law" that selects a combination of $q-k$ initial conditions (1.2a) and of $r-n+k$ terminal conditions (1.2b) to be satisfied by $\mathbf{X}_0(t)$. If $k > q$ or $n-k > r$, the linear examples of Wasow [30] show that limiting solutions may not exist. For more nonlinear problems, we note that such a cancellation law is much more difficult to specify (cf. O'Malley [22]). Boundary layer corrections are generally needed to compensate for the cancelled initial and terminal conditions, and these are easily determined once $\mathbf{X}_0(t)$ and $\mathbf{Y}_0(t)$ have been found (cf. § 2).

In § 3 we discuss a numerical procedure for calculating the asymptotic solution of § 2. We implement the cancellation law by using orthogonal transformations to reduce $\mathbf{G}(\mathbf{x}(t), t)$ to a block triangular form with its stable and unstable eigenspaces separated. We also use the general purpose two-point boundary value code COLSYS to solve the reduced problem and then add numerical approximations of the boundary layer corrections. This approximation is considerably less expensive to obtain than solving the full stiff problem numerically and it has the advantage of improving in accuracy, without any additional computational cost, as the small parameter ε tends to zero. However, when ε is only moderately small, our asymptotic approximation may not be sufficiently accurate for some applications, so we have developed a procedure for generating an improved solution by using COLSYS to solve the complete problem (1.1), (1.2) with our asymptotic approximation as an initial guess. In order for this approach to succeed, we must also provide COLSYS with an initial nonuniform mesh that is appropriately graded in the boundary layers. We give an algorithm for constructing such a mesh in § 3.

In § 4 we apply our procedures to a third order model problem that has multiple solutions and to problems involving the deformation of a thin nonlinear elastic beam. These examples show that our methods can calculate accurate solutions of stiff problems for a very modest computational effort. While our algorithm for furnishing COLSYS with an initial guess and a nonuniform mesh does not seem to be optimal, it does offer some advantages over the more standard approach of continuation in ε, where one starts with a large value of ε (e.g., $\varepsilon = 1$) and a crude initial guess of the solution and reduces ε in steps so that the mesh is gradually concentrated into the boundary layer regions.

We also present two examples in § 4 that are beyond the capabilities of our current methods because their solutions become unbounded as $\varepsilon \to 0$. We include numerical results for these problems in this paper in order to show some of the several challenging effects that can occur with singularly perturbed problems. Finally, in § 5, we discuss our results and present some suggestions for future investigations.

2. Asymptotic approximation. With the assumed hyperbolic splitting, we expect solutions of (1.1, 2) to feature boundary layers in the fast $\mathbf{y}$ variable near both endpoints as $\varepsilon \to 0$. Thus, it is natural (cf. O'Malley [21]) to seek bounded uniform asymptotic expansions of the form

$$\begin{aligned} \mathbf{x}(t, \varepsilon) &= \mathbf{X}(t, \varepsilon) + \varepsilon \boldsymbol{\xi}(\tau, \varepsilon) + \varepsilon \boldsymbol{\eta}(\sigma, \varepsilon), \\ \mathbf{y}(t, \varepsilon) &= \mathbf{Y}(t, \varepsilon) + \boldsymbol{\mu}(\tau, \varepsilon) + \boldsymbol{\nu}(\sigma, \varepsilon), \end{aligned} \qquad 0 < t < 1, \tag{2.1}$$

where the outer solution $(\mathbf{X}(t, \varepsilon), \mathbf{Y}(t, \varepsilon))$ represents the solution asymptotically within $(0, 1)$, the initial layer correction $(\varepsilon \boldsymbol{\xi}(\tau, \varepsilon), \boldsymbol{\mu}(\tau, \varepsilon))$ decays to zero as the stretched variable

$$\tau = t/\varepsilon \tag{2.2a}$$

tends to infinity, and the terminal layer correction $(\varepsilon \boldsymbol{\eta}(\sigma, \varepsilon), \boldsymbol{\nu}(\sigma, \varepsilon))$ goes to zero as the stretched variable

$$\sigma = (1 - t)/\varepsilon \tag{2.2b}$$

approaches infinity. The outer solution and the boundary layer corrections are represented by expansions of the form

$$(2.3\text{a–f})\qquad \begin{bmatrix} \mathbf{X}(t,\varepsilon) \\ \mathbf{Y}(t,\varepsilon) \\ \boldsymbol{\xi}(\tau,\varepsilon) \\ \boldsymbol{\mu}(\tau,\varepsilon) \\ \boldsymbol{\eta}(\sigma,\varepsilon) \\ \boldsymbol{\nu}(\sigma,\varepsilon) \end{bmatrix} \sim \sum_{j=0}^{\infty} \begin{bmatrix} \mathbf{X}_j(t) \\ \mathbf{Y}_j(t) \\ \boldsymbol{\xi}_j(\tau) \\ \boldsymbol{\mu}_j(\tau) \\ \boldsymbol{\eta}_j(\sigma) \\ \boldsymbol{\nu}_j(\sigma) \end{bmatrix} \varepsilon^j.$$

The limiting uniform approximation is obtained from (2.1) by letting ε tend to zero, i.e.,

$$(2.4\text{a})\qquad \mathbf{x}(t,\varepsilon)=\mathbf{X}_0(t)+O(\varepsilon),\qquad \mathbf{y}(t,\varepsilon)=\mathbf{Y}_0(t)+\boldsymbol{\mu}_0(\tau)+\boldsymbol{\nu}_0(\sigma)+O(\varepsilon).$$

At $t=0$ the fast vector $\mathbf{y}$ usually has a discontinuous limit, jumping from $\mathbf{y}(0,0)=\mathbf{Y}_0(0)+\boldsymbol{\mu}_0(0)$ to $\mathbf{Y}_0(0)$ at $t=0^+$. An analogous Heaviside discontinuity generally occurs near $t=1$.

The outer expansion (2.3a, b) must satisfy the full problem (1.1) within $(0,1)$ as a power series in ε; thus, the limiting solution $(\mathbf{X}_0,\mathbf{Y}_0)$ will satisfy the nonlinear and nonstiff reduced system (1.4). As previously noted, since $\mathbf{G}(\mathbf{X}_0,t)$ (cf. (1.3)) is nonsingular, we can solve (1.4b) for $\mathbf{Y}_0=\mathbf{Y}_0(\mathbf{X}_0,t)$ in a locally unique way, so there remains the mth order nonlinear system (1.4a) for $\mathbf{X}_0(t)$. Later terms of the expansion (2.3a, b) satisfy linearized versions of the reduced system. For example, the coefficients of order ε give

$$(2.4\text{b})\qquad \begin{aligned} \dot{\mathbf{X}}_1&=\mathbf{f}_{\mathbf{x}}(\mathbf{X}_0,\mathbf{Y}_0,t,0)\boldsymbol{X}_1+\boldsymbol{f}_{\mathbf{y}}(\boldsymbol{X}_0,\boldsymbol{Y}_0,t,0)\mathbf{Y}_1+\mathbf{f}_\varepsilon(\mathbf{X}_0,\mathbf{Y}_0,t,0),\\ \dot{\mathbf{Y}}_0&=\mathbf{g}_{\mathbf{x}}(\mathbf{X}_0,\mathbf{Y}_0,t,0)\mathbf{X}_1+\mathbf{G}(\mathbf{X}_0,t)\mathbf{Y}_1+\mathbf{g}_\varepsilon,(\mathbf{X}_0,\mathbf{Y}_0,t,0).\end{aligned}$$

We can determine $\mathbf{Y}_1(t)$ in terms of $\mathbf{X}_0$, $\mathbf{Y}_0$, and $\mathbf{X}_1$ from (2.4b) and, once again, there remains the mth order linear system (2.4a) for $\mathbf{X}_1$. Similarly, for each $j>1$, we obtain a system of the form

$$(2.5)\qquad \begin{aligned} \dot{\mathbf{X}}_j&=\mathbf{f}_{\mathbf{x}}(\mathbf{X}_0,\mathbf{Y}_0,t,0)\mathbf{X}_j+\mathbf{f}_{\mathbf{y}}(\mathbf{X}_0,\mathbf{Y}_0,t,0)\mathbf{Y}_j+\boldsymbol{\alpha}_{j-1}(\mathbf{X}_0,\cdots,\mathbf{X}_{j-1},t),\\ \dot{\mathbf{Y}}_{j-1}&=\mathbf{g}_{\mathbf{x}}(\mathbf{X}_0,\mathbf{Y}_0,t,0)\mathbf{X}_j+\mathbf{G}(\mathbf{X}_0,t)\mathbf{Y}_j+\boldsymbol{\beta}_{j-1}(\mathbf{X}_0,\cdots,\mathbf{X}_{j-1},t)\end{aligned}$$

with successively determined inhomogeneous terms.

In order to completely specify the outer expansion (2.3a, b), we must prescribe boundary conditions for the m-vectors $\mathbf{X}_j(t)$. Most critically, we need to specify m boundary conditions for the limiting slow vector $\mathbf{X}_0(t)$. It is natural to attempt to determine them by somehow selecting a subset of m combinations of the $m+n$ boundary conditions (1.2) evaluated at $\varepsilon=0$. For scalar higher order linear differential equations, the first such "cancellation law" was obtained by Wasow [30]. Harris [14] obtained a more complicated cancellation law for linear systems with coupled boundary conditions and Ferguson [8] developed a numerical procedure for corresponding linear problems. These early works suggest that we should seek a cancellation law that ignores an appropriate combination of k initial conditions and of $n-k$ terminal conditions. To this end, we must examine the boundary layer corrections and we begin by considering the initial layer correction $(\varepsilon\boldsymbol{\xi},\boldsymbol{\mu})$. Near $t=0$, the terminal layer correction $(\varepsilon\boldsymbol{\eta},\boldsymbol{\nu})$ may be neglected, so the representation of our asymptotic solution (2.1) requires the initial layer correction $(\varepsilon\boldsymbol{\xi}, \boldsymbol{\mu})$ to satisfy the nonlinear system

$$\frac{d\boldsymbol{\xi}}{d\tau}=\frac{dx}{dt}-\frac{d\mathbf{X}}{dt}=\mathbf{f}(\mathbf{X}+\varepsilon\boldsymbol{\xi}, \mathbf{Y}+\boldsymbol{\mu}, \varepsilon\tau, \varepsilon)-\mathbf{f}(\mathbf{X}, \mathbf{Y}, \varepsilon\tau, \varepsilon),$$
(2.6)
$$\frac{d\boldsymbol{\mu}}{d\tau}=\varepsilon\left(\frac{d\mathbf{y}}{dt}-\frac{d\mathbf{Y}}{dt}\right)=\mathbf{g}(\mathbf{X}+\varepsilon\boldsymbol{\xi}, \mathbf{Y}+\boldsymbol{\mu}, \varepsilon\tau, \varepsilon)-\mathbf{g}(\mathbf{X}, \mathbf{Y}, \varepsilon\tau, \varepsilon),$$

on $\tau \geqq 0$ and to decay to zero as $\tau\to\infty$. Substitution of the asymptotic expansion (2.3c, d) into (2.6) provides successive differential equations for the coefficients $(\boldsymbol{\xi}_j, \boldsymbol{\mu}_j)$. In particular, when $\varepsilon=0$, we have the limiting initial layer system

$$\frac{d\boldsymbol{\xi}_0}{d\tau}=\mathbf{f}(\mathbf{X}_0(0), \mathbf{Y}_0(0)+\boldsymbol{\mu}_0, 0, 0)-\mathbf{f}(\mathbf{X}_0(0), \mathbf{Y}_0(0), 0, 0),$$
(2.7)
$$\frac{d\boldsymbol{\mu}_0}{d\tau}=\mathbf{g}(\mathbf{X}_0(0), \mathbf{Y}_0(0)+\boldsymbol{\mu}_0, 0, 0)-\mathbf{g}(\mathbf{X}_0(0), \mathbf{Y}_0(0), 0, 0).$$

The decay requirement determines

(2.8a)
$$\boldsymbol{\xi}_0(\tau)=-\int_\tau^\infty\left(\frac{d\boldsymbol{\xi}_0(s)}{d\tau}\right)ds$$

as a functional of $\boldsymbol{\mu}_0$, while $\boldsymbol{\mu}_0$ satisfies the conditionally stable system

(2.8b)
$$\frac{d\boldsymbol{\mu}_0}{d\tau}=\mathbf{G}(\mathbf{X}_0(0), 0)\boldsymbol{\mu}_0.$$

We used (1.3) and the assumed linearity of $\mathbf{g}$ in $\mathbf{y}$ when obtaining (2.8b). If $\mathbf{g}(\mathbf{x}, \mathbf{y}, t, \varepsilon)$ were not linear in $\mathbf{y}$, the initial layer correction would satisfy a nonlinear differential equation which would generally be difficult to solve (cf. O'Malley [22]). Indeed, it would then be extremely difficult to specify what set of initial vectors $\boldsymbol{\mu}_0(0)$ would lead to decaying solutions of the boundary layer system (2.7b). Here (2.8b) is readily integrated to give

(2.9)
$$\boldsymbol{\mu}_0(\tau)=e^{\mathbf{G}(\mathbf{X}_0(0),0)\tau}\boldsymbol{\mu}_0(0).$$

Thus, $\boldsymbol{\mu}_0$ will decay to zero as $\tau\to\infty$ provided that

(2.10)
$$\boldsymbol{\mu}_0(0)=\mathbf{P}(\mathbf{X}_0(0), 0)\boldsymbol{\mu}_0(0),$$

where $\mathbf{P}$ is a projection onto the k dimensional stable eigenspace of $\mathbf{G}(\mathbf{X}_0(0), 0)$. The representation (2.9) is used here and below for notational convenience, but its direct numerical implementation is not recommended due to the conditional stability of the matrix G.

Substituting (2.10) into (1.2a) and letting $\varepsilon\to 0$, we see that the q limiting initial conditions take the form

(2.11)
$$\mathbf{a}(\mathbf{X}_0(0), \mathbf{Y}_0(0)+\mathbf{P}(\mathbf{X}_0(0), 0)\boldsymbol{\mu}_0(0), 0)=\mathbf{0}.$$

Now, using the linearity of $\mathbf{a}$ in $\mathbf{y}$, we let

(2.12)
$$\mathbf{A}(\mathbf{x}, t)=\mathbf{a}_\mathbf{y}(\mathbf{x}, \mathbf{y}, t, 0)$$

and further assume that $\mathbf{A}(\mathbf{X}_0(0), 0)\mathbf{P}(\mathbf{X}_0(0), 0)$ has its full and maximal rank k. Then we can uniquely determine $\boldsymbol{\mu}_0(0)$ as a function of $\mathbf{X}_0(0)$ from k of the equations (2.11).

Having done this, initial conditions for the reduced problem can be determined from the remaining $q-k$ conditions in (2.11). For the moment, we write these in the form

$$\mathbf{\Phi}(\mathbf{X}_0(0))=\mathbf{0}. \tag{2.13}$$

In § 3, we discuss a numerical procedure for determining $\mathbf{P}$, $\boldsymbol{\mu}_0(0)$ and $\mathbf{\Phi}(\mathbf{X}_0(0))$.

The terminal layer correction can be analyzed in an analogous manner. In particular, the leading term $\boldsymbol{\nu}_0(\sigma)$ satisfies

$$\boldsymbol{\nu}_0(\sigma)=e^{\mathbf{G}(\mathbf{X}_0(1),1)\sigma}\boldsymbol{\nu}_0(0). \tag{2.14}$$

Now, $\boldsymbol{\nu}_0$ will decay to zero as $\sigma\to\infty$ provided that

$$\boldsymbol{\nu}_0(0)=\mathbf{Q}(\mathbf{X}_0(1),1)\boldsymbol{\nu}_0(0), \tag{2.15}$$

where $\mathbf{Q}$ is a projection onto the $n-k$ dimensional unstable eigenspace of $\mathbf{G}(\mathbf{X}_0(1),1)$. Substituting (2.15) into (1.2b) and letting $\varepsilon\to0$ gives the r limiting terminal conditions as

$$\mathbf{b}(\mathbf{X}_0(1),\mathbf{Y}_0(1)+\mathbf{Q}(\mathbf{X}_0(1),1)\boldsymbol{\nu}_0(0),0)=\mathbf{0}. \tag{2.16}$$

We let

$$\mathbf{B}(\mathbf{x},t)=\mathbf{b}_{\mathbf{y}}(\mathbf{x},\mathbf{y},t,0) \tag{2.17}$$

and assume that $\mathbf{B}(\mathbf{X}_0(1),1)\mathbf{Q}(\mathbf{X}_0(1),1)$ has full rank $n-k$. Then we can solve (2.16) for $\boldsymbol{\nu}_0(0)$ and the remaining $r-n+k$ conditions specify terminal conditions for the limiting problem, which we denote by

$$\mathbf{\Psi}(\mathbf{X}_0(1))=\mathbf{0}. \tag{2.18}$$

The reduced problem consists of the nonlinear reduced differential equation and the m separated nonlinear boundary conditions (2.13), (2.18). If it is solvable, it may have many solutions; however, corresponding to any of its isolated solutions $(\mathbf{X}_0(t), \mathbf{Y}_0(t))$, one can expect to find a solution of the original problem (1.1), (1.2) that converges to $(\mathbf{X}_0(t), \mathbf{Y}_0(t))$ on $0<t<1$ as $\varepsilon\to0$. Sufficient hypotheses to obtain an asymptotic solution having the form of (2.1) are provided by Hoppensteadt [16] and others. For this reason, we shall merely indicate the considerations that are involved in obtaining further terms in the initial and terminal layer expansions and boundary conditions for the outer expansion. In the linear case, our hypotheses guarantee well-conditioning of the boundary value problem (cf. Mattheij [20]).

Additional terms of the initial layer expansion (2.3c, d) are determined by equating the coefficients of ε^j in the nonlinear system (2.7), i.e.

$$\begin{aligned}\frac{d\boldsymbol{\xi}_j}{d\tau}&=\mathbf{f}_{\mathbf{y}}(\mathbf{X}_0(0),\mathbf{Y}_0(0)+\boldsymbol{\mu}_0(\tau),0,0)\boldsymbol{\mu}_j+\boldsymbol{\gamma}_{j-1}(\tau),\\ \frac{d\boldsymbol{\mu}_j}{d\tau}&=\mathbf{G}(\mathbf{X}_0(0),0)\boldsymbol{\mu}_j+\boldsymbol{\delta}_{j-1}(\tau),\end{aligned} \tag{2.19}$$

for $j>1$, where the inhomogeneous terms are exponentially decaying as $\tau\to\infty$ because $\boldsymbol{\xi}_{i-1}$ and $\boldsymbol{\mu}_{i-1}$, $i=1,\cdots,j$, and their derivatives behave in this manner. The linear system (2.19) may be integrated to yield

$$\begin{aligned}\boldsymbol{\xi}_j(\tau) &= -\int_\tau^\infty \left(\frac{d\boldsymbol{\xi}_j(s)}{d\tau}\right) ds,\\ \boldsymbol{\mu}_j(\tau) &= e^{\mathbf{G}(\mathbf{X}_0(0),0)\tau}\boldsymbol{\mu}_j(0) + \int_0^\tau e^{\mathbf{G}(\mathbf{X}_0(0),0)(\tau-s)}\boldsymbol{\delta}_{j-1}(s)\, ds.\end{aligned} \tag{2.20}$$

We see that $\boldsymbol{\xi}_j(\tau)$ decays as τ increases and $\boldsymbol{\mu}_j(\tau)$ will decay provided that $\boldsymbol{\mu}_j(0)$ lies in the stable eigenspace of $G(\mathbf{X}_0(0), 0)$, i.e.,

$$\boldsymbol{\mu}_j(0) = \mathbf{P}(\mathbf{X}_0(0), 0)\boldsymbol{\mu}_j(0). \tag{2.21}$$

Using (2.1) and (2.3a, b), we find that the coefficient of ε^j in the initial condition (1.2a) has the form

$$\mathbf{a}_\mathbf{x}(\mathbf{X}_0(0), \mathbf{Y}_0(0)+\boldsymbol{\mu}_0(0), 0)\mathbf{X}_j(0) + \mathbf{A}(\mathbf{X}_0(0), 0)[\mathbf{Y}_j(0) + \mathbf{P}(\mathbf{X}_0(0), 0)\boldsymbol{\mu}_j(0)] = \boldsymbol{\zeta}_{j-1}. \tag{2.22}$$

Since $\mathbf{A}(\mathbf{X}_0(0), 0)\mathbf{P}(\mathbf{X}_0(0), 0)$ has its maximal rank k, we can determine $\boldsymbol{\mu}_j(0)$ from k of these equations, and the remaining $q-k$ equations determine linear equations for $\mathbf{X}_j(0)$. The situation for the terminal layer correction is completely analogous; thus, $\boldsymbol{\nu}_j(0)$ and the terminal conditions for $\mathbf{X}_j(1)$ are determined from linear equations of the form

$$\mathbf{b}_\mathbf{x}(\mathbf{X}_0(1), \mathbf{Y}_0(1)+\boldsymbol{\nu}_0(0), 0)\mathbf{X}_j(1) + \mathbf{B}(\mathbf{X}_0(1), 1)[\mathbf{Y}_j(1) + \mathbf{Q}(\mathbf{X}_0(1), 1)\boldsymbol{\nu}_j(0)] = \boldsymbol{\theta}_{j-1}. \tag{2.23}$$

To summarize, we have shown that the jth($j>1$) term in the outer expansion satisfies an mth order linear boundary value problem consisting of (2.5) and a set of m linear boundary conditions determined from (2.22) and (2.23). It is a linearization of the problem for $\mathbf{X}_0(t)$.

3. Numerical procedure. In this section we discuss a numerical procedure for finding the limiting uniform asymptotic solution (2.4). It consists of solving the limiting outer problem (1.4), (2.13), (2.18) and determining boundary layer corrections from (2.9) and (2.14).

Our first task is to find the projections $\mathbf{P}$ and $\mathbf{Q}$ and we do this by finding the Schur decomposition of the matrix $\mathbf{G}$ at $t=0$ and $t=1$. In particular, at $t=0$ we find an orthogonal matrix $\mathbf{E}(\mathbf{x}(0), 0)$ such that

$$\mathbf{G}(\mathbf{x}(0), 0)\mathbf{E}(\mathbf{x}(0), 0) = \mathbf{E}(\mathbf{x}(0), 0)\begin{bmatrix}\mathbf{T}_-(\mathbf{x}(0),0) & \mathbf{U}(\mathbf{x}(0), 0)\\ \mathbf{0} & \mathbf{T}_+(\mathbf{x}(0), 0)\end{bmatrix} \tag{3.1}$$

where $\mathbf{T}_-$ is $k\times k$ and upper triangular with the stable eigenvalues of $\mathbf{G}(\mathbf{x}(0), 0)$, and $\mathbf{T}_+$ is upper triangular with the remaining $n-k$ unstable eigenvalues. (In (3.1), $\mathbf{x}(0)$ is an unspecified variable.) The decomposition (3.1) can often be obtained analytically; however, when this is not possible or practical, it can be determined numerically by using the QR algorithm (cf. Golub and Wilkinson [13], Ruhe [27], and Bjorck [5] for specific procedures).

We partition $\mathbf{E}$ after its kth column as

$$\mathbf{E} = [\mathbf{E}_- \quad \bar{\mathbf{E}}_-] \tag{3.2}$$

and note that $\mathbf{E}_-$ spans the stable eigenspace of $\mathbf{G}$ at $t=0$ and

$$\mathbf{P} = \mathbf{E}_-\mathbf{E}_-^{\mathrm{T}} \tag{3.3}$$

is the desired projection onto this eigenspace.

Substituting (3.3) into (2.11) gives

$$\mathbf{a}(\mathbf{X}_0(0), \mathbf{Y}_0(0)+\mathbf{E}_-(\mathbf{X}_0(0), 0)\mathbf{E}_-^T(\mathbf{X}_0(0), 0)\boldsymbol{\mu}_0(0), 0)=\mathbf{0}, \tag{3.4}$$

as the equation for determining $\boldsymbol{\mu}_0(0)$ and $\boldsymbol{\Phi}(\mathbf{X}_0(0))$. Since $\mathbf{A}(\mathbf{X}_0(0), 0)\mathbf{E}_-(\mathbf{X}_0(0), 0)$ is of rank k, we construct a $q\times q$ matrix

$$\mathbf{L}^T=[\mathbf{L}_-^T \quad \bar{\mathbf{L}}_-^T] \tag{3.5a}$$

that reduces it to echelon form, i.e.,

$$\begin{bmatrix}\mathbf{L}_-\\ \bar{\mathbf{L}}_-\end{bmatrix}\mathbf{A}(\mathbf{X}_0(0), 0)\mathbf{E}_-(\mathbf{X}_0(0), 0)=\begin{bmatrix}\mathbf{V}_-\\ \mathbf{0}\end{bmatrix}, \tag{3.5b}$$

where $\mathbf{V}_-$ is $k\times k$ and nonsingular. Multiplying (3.4) by $\mathbf{L}$, using the linearity of $\mathbf{a}$ in $\mathbf{y}$, (3.5) implies

$$\mathbf{L}_-\mathbf{a}+\mathbf{V}_-\mathbf{E}_-^T\mu_0(0)=0, \qquad \bar{\mathbf{L}}_-\boldsymbol{\alpha}=0.$$

Thus, we obtain the initial layer jump $\boldsymbol{\mu}_0(0)$ and the $q-k$ initial conditions (2.13) for the reduced problem, respectively, as

$$\begin{aligned}\boldsymbol{\mu}_0(0)&=-\mathbf{E}_-(\mathbf{X}_0(0), 0)\mathbf{V}_-^{-1}\mathbf{L}_-\mathbf{a}(\mathbf{X}_0(0), \mathbf{Y}_0(0), 0),\\ \boldsymbol{\Phi}(\mathbf{X}_0(0))&:=\bar{\mathbf{L}}_-\mathbf{a}(\mathbf{X}_0(0), \mathbf{Y}_0(0), 0)=\mathbf{0}.\end{aligned} \tag{3.6}$$

We find the terminal layer jump and the $r-(n-k)$ terminal conditions for the reduced problem in an analogous fashion with the exception that we define $\mathbf{E}(\mathbf{x}(1), 1)$ such that

$$\mathbf{G}(\mathbf{x}(1), 1)\mathbf{E}(\mathbf{x}(1), 1)=\mathbf{E}(\mathbf{x}(1), 1)\begin{bmatrix}\hat{\mathbf{T}}_+(\mathbf{x}(1), 1) & \hat{\mathbf{U}}(\mathbf{x}(1), 1)\\ \mathbf{0} & \hat{\mathbf{T}}_-(\mathbf{x}(1), 1)\end{bmatrix}, \tag{3.7}$$

which we partition after its $(n-k)$th column as

$$\mathbf{E}=[\mathbf{E}_+ \quad \bar{\mathbf{E}}_+]. \tag{3.8}$$

In parallel with (3.1) and (3.2), the matrices $\hat{\mathbf{T}}_-$, $\hat{\mathbf{T}}_+$, and $\mathbf{E}_+$ contain the k stable eigenvalues, the $n-k$ unstable eigenvalues, and span the unstable eigenspace, respectively, of $\mathbf{G}$ at $t=1$. Our reasons for switching the positions of the matrices containing the stable and unstable eigenvalues of $\mathbf{G}$ is that we are unaware of a simple and stable computational procedure for finding a set of vectors that span a given subspace and are not in the leading columns of an orthogonal matrix like $\mathbf{E}$ (cf. Golub and Wilkinson [13]).

Now, following the procedure that we used for the initial layer, we take

$$\mathbf{Q}(\mathbf{X}_0(1), 1)=\mathbf{E}_+(\mathbf{X}_0(1), 1)\mathbf{E}_+^T(\mathbf{X}_0(1), 1) \tag{3.9}$$

as our projection onto the $(n-k)$ dimensional unstable eigenspace of $\mathbf{G}(\mathbf{X}_0(1), 1)$ and construct an $r\times r$ matrix

$$\mathbf{R}^T=[\mathbf{R}_+^T \quad \bar{\mathbf{R}}_+^T] \tag{3.10a}$$

that reduces the rank $n-k$ matrix $\mathbf{B}(\mathbf{X}_0(1), 1)\mathbf{E}_+(\mathbf{X}_0(1), 1)$ to echelon form, i.e.,

$$\begin{bmatrix}\mathbf{R}_+\\ \bar{\mathbf{R}}_+\end{bmatrix}\mathbf{B}(\mathbf{X}_0(1), 1)\mathbf{E}_+(\mathbf{X}_0(1), 1)=\begin{bmatrix}\mathbf{V}_+\\ \mathbf{0}\end{bmatrix}, \tag{3.10b}$$

where $\mathbf{V}_+$ is $(n-k)\times(n-k)$ and nonsingular. Multiplying (2.16) by $\mathbf{R}$ and using (3.9) and (3.10), we find the terminal layer jump and terminal conditions for the reduced problem, respectively, as

$$(3.11)\qquad \begin{aligned} &\boldsymbol{\nu}_0(0) = -\mathbf{E}_+(\mathbf{X}_0(1), 1)\mathbf{V}_+^{-1}\mathbf{R}_+\mathbf{b}(\mathbf{X}_0(1), \mathbf{Y}_0(1), 0),\\ &\boldsymbol{\Psi}(\mathbf{X}_0(1)) := \bar{\mathbf{R}}_+\mathbf{b}(\mathbf{X}_0(1), \mathbf{Y}_0(1), 0) = \mathbf{0}. \end{aligned}$$

Since the reduced problem (1.4), (3.6b), and (3.11b) is not stiff, we can use any good code for two-point boundary value problems (cf. Childs et al. [6]) to solve it, and we have chosen to use the collocation code COLSYS of Ascher, Christiansen and Russell [1]. The reduced problem is generally nonlinear and since COLSYS solves nonlinear problems using a damped Newton method, we have to supply formulas for evaluating the Jacobians of $\mathbf{f}$, $\mathbf{Y}$, $\boldsymbol{\Phi}$, and $\boldsymbol{\Psi}$ with respect to $\mathbf{X}$. We do this, but introduce an error, by providing analytical formulas for these Jacobians that neglect the influence of the derivatives of $\mathbf{E}$, $\mathbf{L}$, $\mathbf{R}$, and $\mathbf{G}$. (These derivatives will be small when the related subspaces are nearly constant). This procedure failed to converge once on Example 2 of § 4 and a minor modification to the Jacobian of $\boldsymbol{\Phi}$ restored convergence; however, an alternative possibility would be to approximate the Jacobians by finite differences.

We start the Newton iteration with a uniform mesh and an initial guess $\mathbf{X}_0^{(0)}(t)$ for $\mathbf{X}_0(t)$. In § 4, we used the default initial guess that is provided by COLSYS for Example 1 and a constant initial guess for Example 2. This latter choice was necessary as Example 2 has three solutions. At each iteration step, we calculate an approximation $\mathbf{E}(\mathbf{X}_0^{(p)}(t), t)$ to $\mathbf{E}(\mathbf{X}(t), t)$ for $t = 0$ and 1 as the Schur decomposition of $\mathbf{G}(\mathbf{X}^{(p)}(t), t)$. The examples of § 4 were calculated using analytical formulas for $\mathbf{E}$ rather than the numerical procedures of Golub and Wilkinson [13], Ruhe [27], or Bjorck [5]. Finally, $\mathbf{L}^{(p)}$ and $\mathbf{R}^{(p)}$ are obtained by using Gaussian elimination to row reduce $\mathbf{A}(\mathbf{X}_0^{(p)}(0), 0)\mathbf{E}_-(\mathbf{X}_0^{(p)}(0), 0)$ and $\mathbf{B}(\mathbf{X}_0^{(p)}(1), 1)\mathbf{E}_+(\mathbf{X}_0^{(p)}(1), 1)$, respectively.

When this procedure converges to $(\mathbf{X}_0(t), \mathbf{Y}_0(t))$, we calculate boundary layer corrections $\boldsymbol{\mu}_0(\tau)$ and $\boldsymbol{\nu}_0(\sigma)$, for a given value of ε, using (2.9), (3.6a), (2.14), and (3.11a), and add these to the reduced solution in order to get the $O(\varepsilon)$ asymptotic approximation (1.4). For moderately small values of ε, this approximation may not provide a sufficiently accurate representation of the solution and, in this case, we use it as an initial guess to COLSYS and solve the complete problem (1.1), (1.2). However, this procedure may fail unless we also provide COLSYS with an initial nonuniform partition

$$(3.12)\qquad \pi := \{0 = t_0 < t_1 < \cdots < t_N = 1\}$$

that is appropriately graded within the boundary layers. Following Ascher, Christiansen, and Russell [1], we seek to find π such that the error on each subinterval satisfies

$$(3.13)\qquad \|\mathbf{e}\|_i \leqq \delta(1 + \|\mathbf{u}\|_i), \qquad i = 1, 2, \cdots, N,$$

where δ is a prescribed tolerance,

$$(3.14)\qquad \mathbf{u}^T = [\mathbf{x}^T \quad \mathbf{y}^T],$$

$\mathbf{e}(t)$ is the difference between $\mathbf{u}(t)$ and its collocation approximation,

$$(3.15)\qquad \|\mathbf{u}\|_i = \max_{t_{i-1} \leqq t \leqq t_i} |\mathbf{u}(t)| \quad \text{and} \quad |\mathbf{u}(t)| = \max_{i \leqq j \leqq m+n} |u_j(t)|.$$

We assume that the final partition selected by COLSYS to solve the reduced problem satisfies (3.13) outside of boundary layer regions and we seek to refine it within the boundary layers. We further assume that derivatives of $\mathbf{u}$ can adequately be approximated by either $\boldsymbol{\mu}_0(\tau)$ or $\boldsymbol{\nu}_0(\sigma)$ in the left or right boundary layer, respectively.

It is known (cf. Russell and Christiansen [28]) that if the solution of (1.1), (1.2) is smooth,

$$\|\mathbf{e}\|_i = c_j \|\mathbf{u}^{(j+1)}\|_i h_i^{j+1} + O(h^{j+2}) \tag{3.16}$$

for collocation at the image of j Gauss–Legendre points per subinterval. Here c_j is a known constant,

$$h_i = t_i - t_{i-1} \quad \text{and} \quad h = \max_{1 \leq i \leq N} h_i. \tag{3.17}$$

In the left boundary layer we approximate $\mathbf{u}$ in (3.16) by $\boldsymbol{\mu}_0$ using (2.9) and attempt to find a partition that satisfies

$$c_j h_i^{j+1} \|\boldsymbol{\mu}_0^{(j+1)}(t/\varepsilon)\|_i \approx \delta(1 + \|u\|_i). \tag{3.18}$$

Finally, we use (2.9) and (3.1) to approximate $\boldsymbol{\mu}_0$ and the subinterval lengths as

$$t_i - t_{i-1} \approx \left(\frac{\varepsilon}{\alpha}\right) \left[\frac{\delta(1 + \|\mathbf{u}\|_i)}{c_j |\boldsymbol{\mu}_0(t_{i-1}/\varepsilon)|}\right]^{1/(j+1)}, \tag{3.19}$$

where α_- is the magnitude of the largest diagonal element of $\mathbf{T}_-(\mathbf{X}_0(0), 0)$. A similar formula can be obtained for selecting subintervals in the right boundary layer.

Starting with $i = 1$, we use (3.19) to generate a partition until we either reach $t = 1/2$ or a point where a subinterval length selected by (3.19) is larger than that used locally by COLSYS to solve the reduced problem. We then repeat the procedure in the right boundary layer.

We have written a computer code called SPCOL that implements the algorithms that are described in this section; thus, it (i) uses COLSYS to solve the reduced problem, (ii) calculates and adds appropriate boundary layer corrections to the reduced problem, and (iii) (optionally) suggests a mesh that can be used by COLSYS to solve the complete problem.

4. Examples. In order to appraise the performance of SPCOL, we have applied it to a problem involving the deformation of a thin nonlinear elastic beam (Example 1) and a third order model problem that has multiple solutions (Example 2).

Example 1. We consider problems involving the deformation of a nonlinear elastic beam that is resting on an elastic foundation with unit spring constant and is subjected to the combined action of a horizontal end thrust P and a unit uniform lateral load. This problem is discussed in detail in Flaherty and O'Malley [11] and herein we only present the governing differential equations, which in dimensionless form are

$$\dot{x}_1 = \cos x_3, \quad \dot{x}_2 = \sin x_3, \quad \dot{x}_3 = y_1, \tag{4.1a,b,c}$$

$$\varepsilon \dot{y}_1 = -y_2, \quad \varepsilon \dot{y}_2 = (x_2 - 1)\cos x_3 - T y_1, \tag{4.1d,e}$$

where

$$T = \sec x_3 + \varepsilon y_2 \tan x_3. \tag{4.1f}$$

The slow variables (x_1, x_2) and x_3 represent the Cartesian coordinates and the tangent angle of a material particle on the centerline of the beam that was at the Cartesian location $(t, 0)$ in the undeformed state. The fast variables y_1 and y_2 are the internal bending moment and transverse shear force, respectively. Finally, the small parameter is

$$\varepsilon^2 = EI/PL^2, \tag{4.2}$$

where EI is the flexural rigidity and L is the length of the beam; thus, our beam is much stronger in extension than it is in bending.

This example does not precisely fit our hypotheses since the axial force T is a function of the fast variable y_2 and, thus, $\mathbf{g}_\mathbf{y}$ also depends on $\mathbf{y}$. However, our theory and methods will still apply as long as $\mathbf{y}$ remains bounded and $|x_3| < \pi/2$ as $\varepsilon \to 0$. Flaherty and O'Malley [11] show that unbounded solutions can occur when certain types of boundary conditions are prescribed for (4.1). In this paper we present results for the following three sets of boundary conditions:

$$\text{(4.2a)} \quad \text{(i)} \quad x_1(0,\varepsilon) = x_2(0,\varepsilon) = y_1(0,\varepsilon) = x_2(1,\varepsilon) = y_1(1,\varepsilon) = 0,$$

$$\text{(4.2b)} \quad \text{(ii)} \quad x_1(0,\varepsilon) = 0, \quad -10x_2(0,\varepsilon) + y_2(0,\varepsilon) = 0, \quad -x_3(0,\varepsilon) + 10y_1(0,\varepsilon) = 0,$$
$$10x_2(1,\varepsilon) + y_2(1,\varepsilon) = 0, \quad 10x_3(1,\varepsilon) + y_1(1,\varepsilon) = 0,$$

$$\text{(4.2c)} \quad \text{(iii)} \quad x_1(0,\varepsilon) = x_2(0,\varepsilon) = x_3(0,\varepsilon) = x_2(1,\varepsilon) = x_3(1,\varepsilon) = 0.$$

Equations (4.2a) correspond to "simple supports," (4.2c) correspond to "clamped supports," and (4.2b) correspond to elastic supports that are almost simply supported at $t = 0$ and almost clamped at $t = 1$. Conditions (4.2b) could arise because, say, friction introduces some coupling between lateral and rotational effects at the supports. As we shall see, $\mathbf{y}$ remains bounded for conditions (4.2a,b), but becomes unbounded as $\varepsilon \to 0$ when conditions (4.2c) are applied. The problem is that the boundary conditions for the clamped beam only involve the slow variables and the slow vector $\mathbf{x}$ cannot generally satisfy all five of them without having boundary layers. This in turn forces the fast vector $\mathbf{y}$ to become unbounded like $O(1/\varepsilon)$ at the endpoints. Thus, the solution cannot have an asymptotic expansion of the form of (2.1); however, an appropriate asymptotic expansion was obtained by Flaherty and O'Malley [11]. We do not repeat those results here, but in order to emphasize the diverse behavior that can occur with nonlinear singularly perturbed problems, we present solutions for x_2, x_3, y_1, and y_2 corresponding to each of the boundary conditions (4.2a), (4.2b), and (4.2c) in Figs. 1, 2, and 3, respectively.

Our methods apply to problems having boundary conditions (4.2a) and (4.2b) and, in these cases, the orthogonal matrix

$$\text{(4.3a)} \qquad \mathbf{E}(\mathbf{x}(0), 0) = (1+\alpha^2)^{-1/2}\begin{bmatrix} 1 & -|\alpha| \\ |\alpha| & 1 \end{bmatrix}$$

where

$$\text{(4.3b)} \qquad \alpha^2 = \sec x_3(0)$$

reduces

$$\text{(4.4)} \qquad \mathbf{G}(\mathbf{x}(0), 0) = \begin{bmatrix} 0 & -1 \\ -\alpha^2 & 0 \end{bmatrix}$$

to the Schur form given by (3.1) at $t = 0$ while $\mathbf{E}^T(\mathbf{x}(1), 1)$ will reduce $\mathbf{G}(\mathbf{x}(1), 1)$ to the form given by (3.7) at $t = 1$.

We solved (4.1) with conditions (4.2a) and (4.2b) in two ways: (i) using COLSYS to solve the complete problem with continuation from a large to a small value of ε and (ii) using our code SPCOL to compute an initial asymptotic approximation and to recommend a nonuniform mesh and using this with COLSYS to calculate an improved solution. All calculations were performed using double precision arithmetic

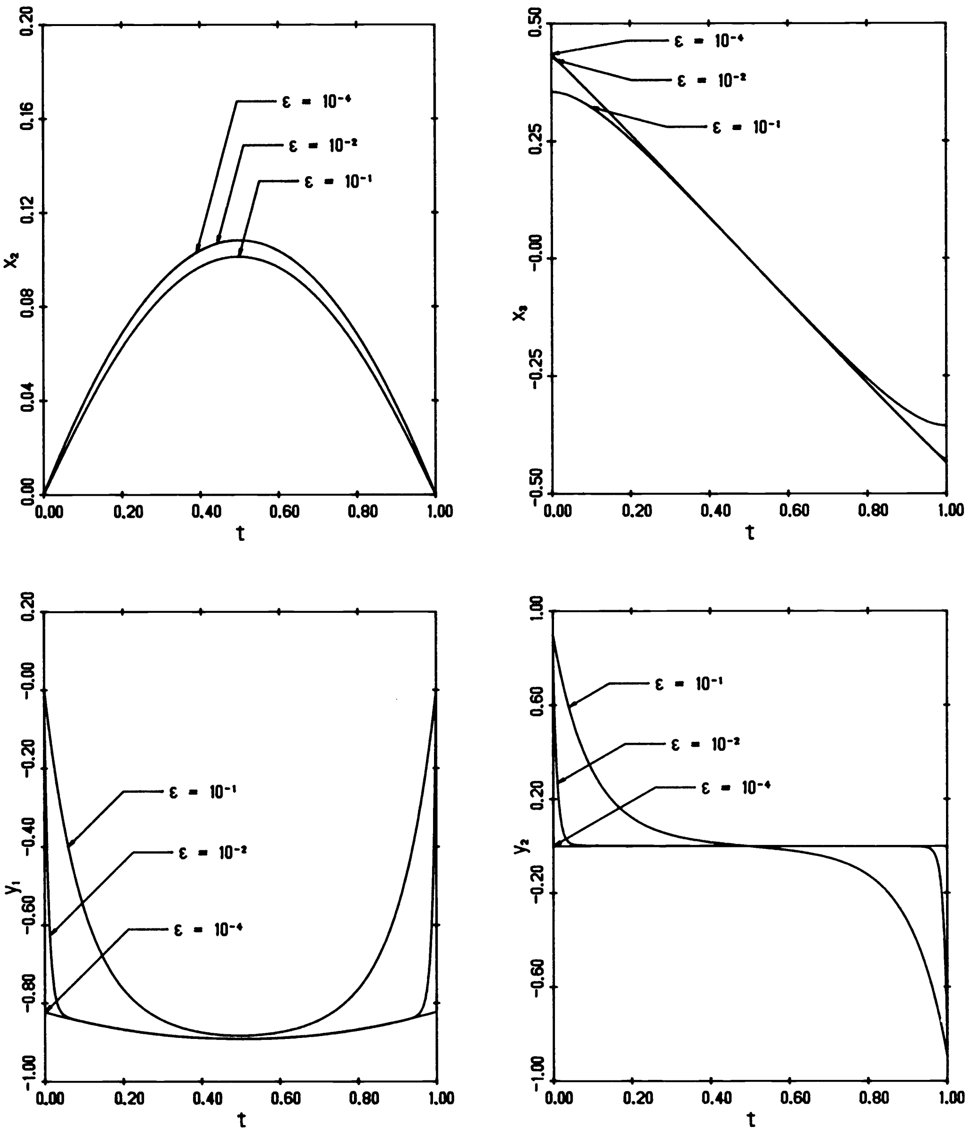


FIG 1. *Numerical solution of simply supported beam, Example* 1 *with boundary conditions given by* (4.2a).

on an IBM 3033 computer, two collocation points per subinterval, and an error tolerance δ (cf. (3.19)) of 10^{-6} for slow variables and 10^{-3} for fast variables.

Our results for the normalized CP times and the number of subintervals (NSUB) that are either used by COLSYS or recommended by SPCOL are shown in Tables 1 and 2 for the simply supported beam and in Tables 3 and 4 for the elastically supported beam. Tables 1 and 3 contain the continuation results and Tables 2 and 4 contain the SPCOL results with COLSYS improvement. The CP times (for all examples) were normalized with respect to the ε sequence in Table 1. Differences between our initial asymptotic approximation and the final solution obtained by COLSYS are shown for $x_2(\frac{1}{2}, \varepsilon)$ and $y_2(0, \varepsilon)$ for the simply supported beam in Table 5 and for $x_3(0, \varepsilon)$ and $y_2(0, \varepsilon)$ for the elastically supported beam in Table 6. All of the differences are

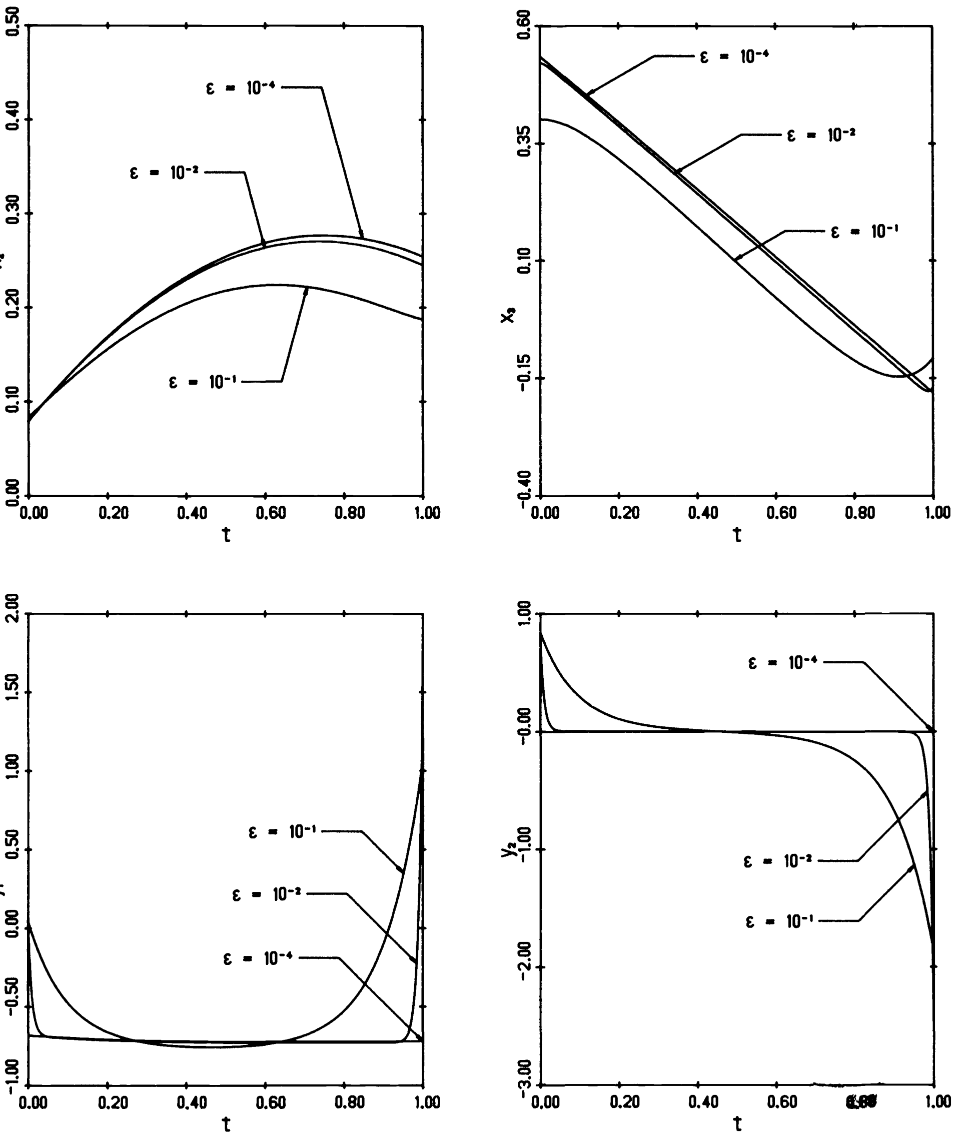


FIG. 2. *Numerical solution of elastically supported beam, Example* 1 *with boundary conditions given by* (4.2b).

decreasing like $O(\varepsilon)$ as expected. Differences that are recorded as zero are less than 10^{-8}.

The results reported in these tables need some additional explanation. The number of subintervals and CP times used with continuation depended heavily on the ε sequence that was used. The results in Tables 1 and 3 are about the best insofar as they gave the smallest total CP time for the sequence. We see in almost every instance that the COLSYS correction is using about twice the number of subintervals that were suggested by SPCOL. This mesh doubling strategy is often used in COLSYS to estimate errors or when the Newton iteration has convergence difficulties. Thus, in some sense our mesh strategy is doing about as well as can be expected; however, it seems that fewer

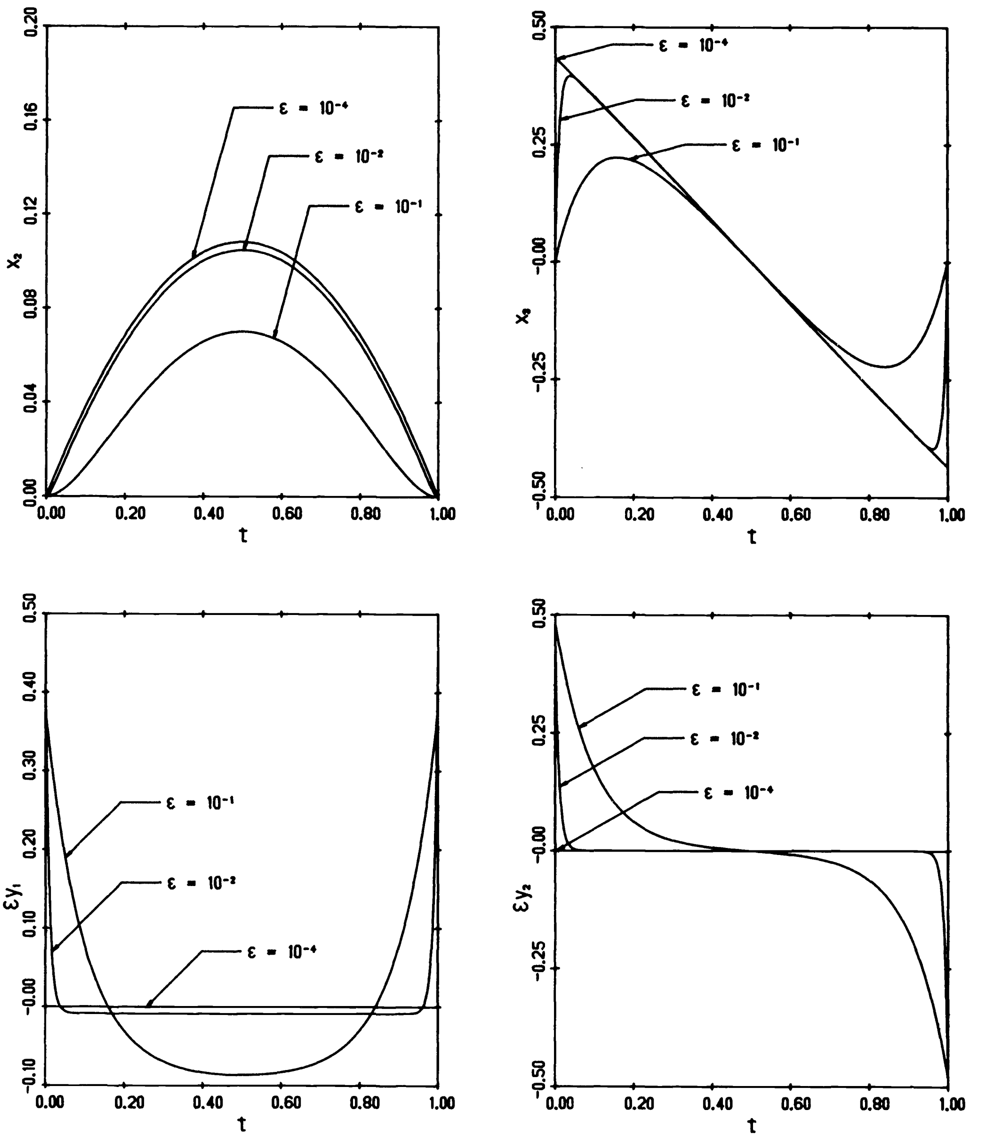


FIG. 3. *Numerical solution of clamped beam, Example* 1 *with boundary conditions given by* (4.2c). *Note that* y_1 *and* y_2 *are multiplied by* ε.

points should be necessary. We tried placing the subintervals according to a pointwise error criteria, as suggested by Ascher and Weiss [2], [3], [4], rather than the global criteria used in (3.19), but this gave very similar results (cf. Flaherty and O'Malley [12]). We also tried suggesting an initial mesh to COLSYS that consisted of every other point in the mesh suggested by SPCOL. This is clearly a risky strategy, since collocation at the Gauss-Legendre points is known to produce oscillations unless the mesh is appropriately fine in the boundary layers (cf. Ascher and Weiss [2]). Nevertheless, this did give some improvement for values of $\varepsilon > 10^{-8}$ (cf. Flaherty and O'Malley [12]). Perhaps the results could be improved further by using higher order collocation

TABLE 1
Example 1 *with simple supports. Number of sub-intervals* (NSUB) *and* CP *times to solve the problem by* COLSYS *with continuation in* ε. *Total* CP *is the accumulated time for the* ε *sequence.*

ε	NSUB	CP	Total CP
10^{-1}	80	6.1	6.1
10^{-2}	72	6.3	12.5
10^{-4}	112	18.4	30.9
10^{-6}	158	27.2	58.1
10^{-8}	254	41.9	100.0

TABLE 2
Example 1 *with simple supports. Number of sub-intervals* (NSUB) *and* CP *times to solve the problem by* SPCOL *and to improve it by* COLSYS. *The* CP *times include the time to calculate the reduced solution, which was* 1.3 *time units. Total* CP *is the sum of the* SPCOL CP *and the* COLSYS CP.

	SPCOL		COLSYS		
ε	NSUB	CP	NSUB	CP	Total CP
10^{-1}	20	1.3	80	5.7	7.0
10^{-2}	28	1.3	112	8.7	10.0
10^{-4}	34	1.3	136	9.0	10.3
10^{-8}	35	1.3	92	9.3	10.6

TABLE 3
Example 1 *with elastic supports. Number of sub-intervals* (NSUB) *and* CP *times to solve the problem by* COLSYS *with continuation in* ε. *Total* CP *is the accumulated time for the* ε *sequence.*

ε	NSUB	CP	Total CP
10^{-1}	80	6.9	6.9
10^{-2}	78	6.3	14.6
10^{-4}	78	16.8	31.4
10^{-6}	156	38.3	69.7
10^{-8}	100	16.4	86.1

and/or collocation at the Gauss–Lobatto points as suggested by Ascher and Weiss [2], [3], [4].

We see from Tables 1 to 4 that for $\varepsilon = 10^{-8}$ the SPCOL solution can be computed in less than 5% of the time of the continuation solution and the COLSYS improvement with the SPCOL solution as an initial guess can be computed in less than 24% of the time of the continuation solution for both simple and elastic supports.

TABLE 4
Example 1 *with elastic supports. Number of sub-intervals* (NSUB) *and* CP *times to solve the problem by* SPCOL *and to improve it by* COLSYS. *The* CP *times include the time to calculate the reduced solution, which was* 3.8 *time units. Total* CP *is the sum of the* SPCOL CP *and the* COLSYS CP.

ε	SPCOL NSUB	SPCOL CP	COLSYS NSUB	COLSYS CP	Total CP
10^{-1}	40	3.9	100	10.2	14.1
10^{-2}	47	3.9	94	10.5	14.4
10^{-4}	56	3.9	112	12.8	16.7
10^{-8}	57	3.9	134	16.8	20.7

TABLE 5
Example 1 *with simple supports. Differences between* SPCOL *and* COLSYS *solutions, i.e.,* $\Delta(\ ) := |(\ )_{\text{SPCOL}} - (\ )_{\text{COLSYS}}|$.

ε	$\Delta x_2(\frac{1}{2}, \varepsilon)$	$\Delta y_2(0, \varepsilon)$
10^{-1}	7.1×10^{-3}	3.2×10^{-2}
10^{-2}	6.7×10^{-5}	3.6×10^{-3}
10^{-4}	0	3.6×10^{-5}
10^{-8}	0	0

TABLE 6
Example 1 *with elastic supports. Differences between* SPCOL *and* COLSYS *solutions, i.e.,* $\Delta(\ ) := |(\ )_{\text{SPCOL}} - (\ )_{\text{COLSYS}}|$.

ε	$\Delta x_3(0, \varepsilon)$	$\Delta y_2(0, \varepsilon)$
10^{-1}	1.3×10^{-1}	4.2×10^{-2}
10^{-2}	1.4×10^{-2}	5.2×10^{-3}
10^{-4}	1.5×10^{-4}	5.4×10^{-5}
10^{-8}	0	0

Example 2. We consider the third order model problem

$$\dot{x} = 1 - x, \quad \varepsilon\dot{y}_1 = y_2, \quad \varepsilon\dot{y}_2 = \alpha^2(x)y_1 + 8x(1-x) \tag{4.5a,b,c}$$

with

$$\alpha(x) = 1 + 2x \tag{4.5d}$$

and the linear boundary conditions

$$x(0,\varepsilon) + y_1(0,\varepsilon) = 0, \quad -\gamma x(0,\varepsilon) + y_2(0,\varepsilon) = 0, \quad x(1,\varepsilon) + y_1(1,\varepsilon) = 0. \tag{4.6}$$

The matrix $\mathbf{G}(\mathbf{x}, t)$ for this example is the negative of that given by (4.4) for Example 1 with α now being given by (4.5d). Thus, $\mathbf{G}$ has one negative and one positive eigenvalue provided that $\alpha(x)$ is nonzero and $\mathbf{G}$ may be reduced to Schur form at $t = 0$ using the orthogonal matrix $\mathbf{E}^T(\mathbf{x}(0), 0)$ and at $t = 1$ using $\mathbf{E}(\mathbf{x}(1), 1)$ (with $\mathbf{E}(\mathbf{x}, t)$ given by (4.3a)).

Flaherty and O'Malley [10] studied this problem and showed that the reduced system is

(4.7) $$\dot{X}_0 = 1 - X_0, \quad Y_{20} = 0, \quad \alpha^2(X_0)Y_{10} + 8X_0(1 - X_0) = 0$$

with the initial condition

(4.8) $$|\alpha(X_0(0))|[X_0(0) + Y_{10}(0)] - \gamma X_0(0) = 0.$$

They show that there are three solutions of (4.7), (4.8) for each value of the constant γ provided that there are no "turning points," i.e., provided that there are no values of $x(t)$ for which $\alpha(x) = 0$ on $0 \leqq t \leqq 1$. The three solutions can be characterized by their value of $X_0(0)$ which is determined as

(4.9) $$X_0(0) = 0, \quad \tfrac{1}{4}[\gamma s - 6 \pm \sqrt{(\gamma s - 4)^2 + 48}], \qquad s = \operatorname{sgn}(\alpha(X_0(0))).$$

For $\gamma = 2$ the three values of $X_0(0)$ are 0, 0.803, and -4.29 and the three corresponding solutions for $y_1(t, \varepsilon)$ are shown in Fig. 4. For $X_0(0) = 0$, the initial layer correction $\mu_0(\tau)$ is trivial; however, the other two solutions have initial layer jumps.

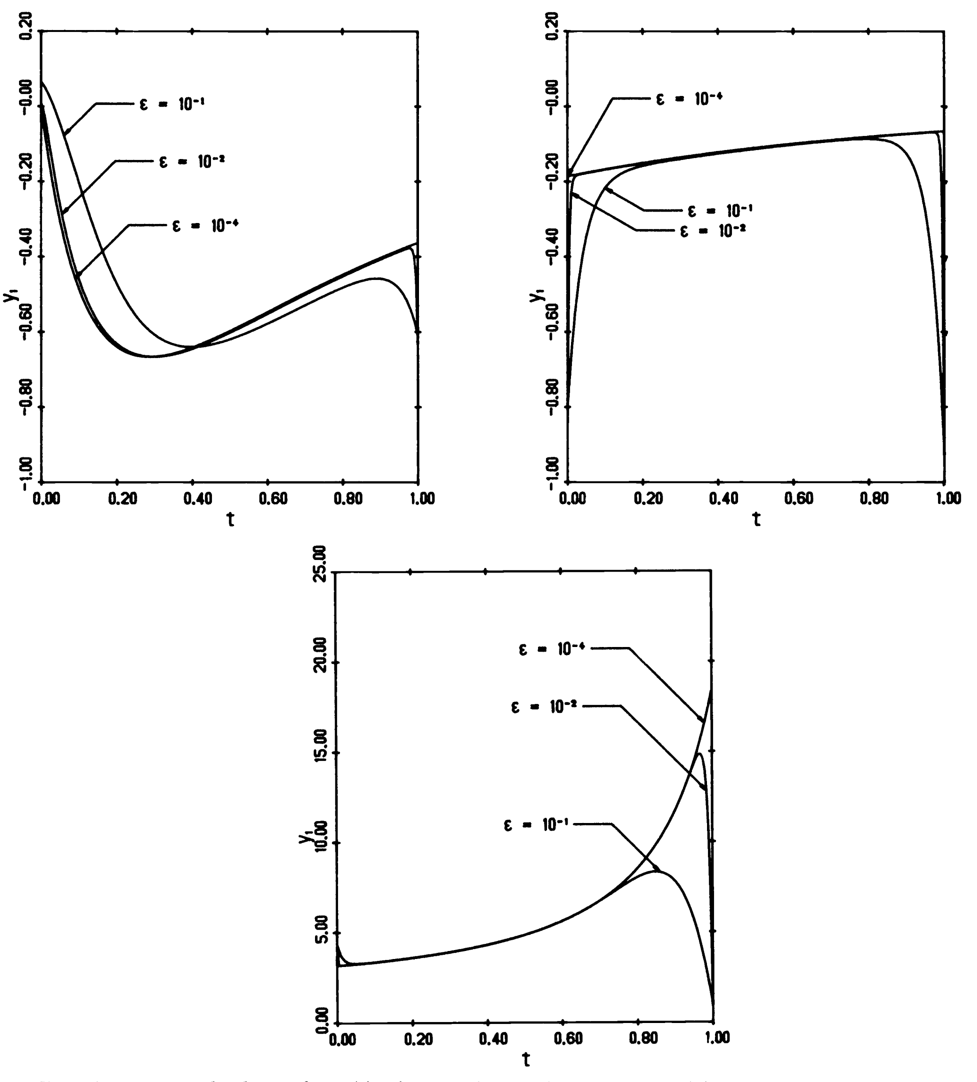


FIG. 4. *Numerical solution for* $y_1(t)$ *of Example* 2 *with* $\gamma = 2$ *and* $X_0(0) = 0$, 0.803, *and* -4.29.

It can be easily verified that $\alpha(X_0(t))$ has a zero on $0 \leqq t \leqq 1$ when $(-3.08\approx)-3e/2+1 \leqq X_0(0) \leqq -\frac{1}{2}$. In this case (4.5) has a turning point and Y_{10} becomes unbounded. Our theory and methods do not apply in this case; however, if ε is not too small, the solution of (4.5) can be calculated using COLSYS. In order to contrast solutions with and without turning points, we illustrate $y_1(t, \varepsilon)$ for $\gamma=-2$ and $X_0(0)=-2.80$ in Fig. 5.

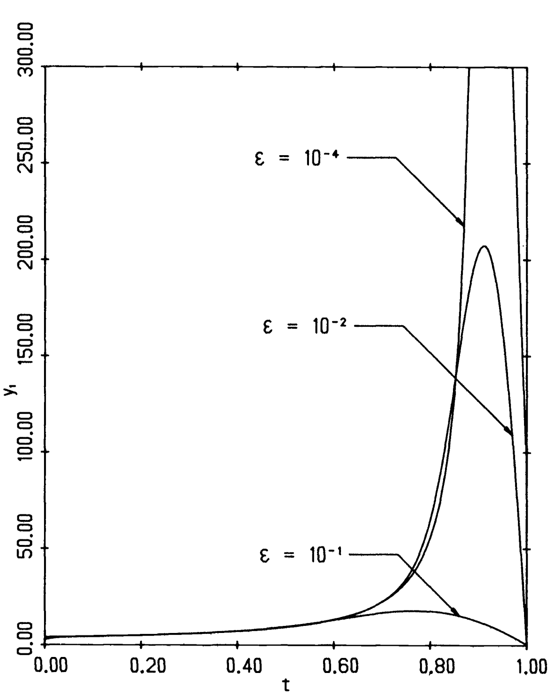


FIG 5. *Numerical solution for $y_1(t)$ of Example 2 with $\gamma=-2$ and $X_0(0)=-2.80$.*

Solutions obtained using SPCOL and the corresponding COLSYS corrections are shown for $\gamma=2$ and $X_0(0)=0$, 0.803, and -4.29 in Tables 7, 9, and 11, respectively. The COLSYS correction failed to converge for $\varepsilon \leqq 10^{-6}$ when $X_0(0)=0$ and -4.29. We have no explanation as to why the solution with $X_0(0)=0.803$ was so much easier to calculate. The relative difference between the SPCOL and COLSYS solutions for $x(1, \varepsilon)$ and $y_2(1, \varepsilon)$ are shown in Table 13 for $\gamma=2$ and $X_0(0)=-4.29$. These results are typical of those that we obtained for all three solutions.

Using COLSYS with continuation in ε and the default initial guess can find at most one solution, and, for this example, it found the $X_0(0)=0$ solution. The results of this calculation are shown in Table 8 for $\gamma=2$. Although several ε sequences were tried, we were unable to obtain convergence for $\varepsilon \leqq 10^{-6}$. Again, this situation could possibly be remedied by using collocation at the Gauss–Lobatto points as in Ascher and Weiss [2], [3], [4]. The other two solutions when $\gamma=2$ can also be calculated using continuation in ε provided that we use a suitable initial guess. Results for the solutions corresponding to $X_0(0)=0.803$ and -4.29 are presented in Tables 10 and 12, respectively, using continuation with SPCOL furnishing an initial guess. These results seem to point to the possibility of using a combination of asymptotics and continuation to solve singular perturbation problems.

TABLE 7

Example 2 *with* $\gamma = 2$ *and* $X_0(0) = 0$. *Number of subintervals* (NSUB) *and* CP *times to solve the problem by* SPCOL *and to improve it by* COLSYS. *The* CP *times include the time to calculate the reduced solution, which was* 0.5 *time units. Total* CP *is the sum of the* SPCOL CP *and the* COLSYS CP.

	SPCOL		COLSYS		
ε	NSUB	CP	NSUB	CP	Total CP
10^{-1}	40	0.6	88	6.3	6.9
10^{-2}	44	0.6	88	6.2	6.8
10^{-4}	47	0.6	192	18.2	18.8
10^{-6}	47	0.6		failed	

TABLE 8

Example 2 *with* $\gamma = 2$ *and* $X_0(0) = 0$. *Number of subintervals* (NSUB) *and* CP *times to solve the problem by* COLSYS *with continuation in* ε *from* $\varepsilon = 10^{-1}$. *The default initial guess that is provided in* COLSYS *was used to start the continuation sequence. Total* CP *is the accumulated time for the sequence.*

ε	NSUB	CP	Total CP
10^{-1}	40	1.8	1.8
10^{-2}	44	3.3	5.2
10^{-4}	264	13.4	18.6
10^{-5}	372	20.2	38.7
10^{-6}		failed	

TABLE 9

Example 2 *with* $\gamma = 2$ *and* $X_0 = 0.803$. *Number of subintervals* (NSUB) *and* CP *times to solve the problem by* SPCOL *and to improve it by* COLSYS. *The* CP *times include the time to calculate the reduced solution, which was* 1.5 *time units. Total* CP *is the sum of the* SPCOL CP *and the* COLSYS CP.

	SPCOL		COLSYS		
ε	NSUB	CP	NSUB	CP	Total CP
10^{-1}	42	1.5	42	3.0	4.5
10^{-2}	52	1.6	52	3.0	4.6
10^{-4}	57	1.6	58	2.6	4.2
10^{-6}	57	1.6	114	10.9	12.5

TABLE 10

Example 2 *with* $\gamma = 2$ *and* $X_0(0) = 0.803$. *Number of subintervals* (NSUB) *and* CP *times to solve the problem by* COLSYS *with continuation in* ε *from* $\varepsilon = 10^{-4}$. *Total* CP *is the accumulated time for the sequence.*

ε	NSUB	CP	Total CP
10^{-4}	58	2.6	2.6
10^{-5}	58	2.4	5.0
10^{-6}	70	4.3	9.3

TABLE 11
Example 2 *with* $\gamma = 2$ *and* $X_0(0) = -4.29$. *Number of subintervals* (NSUB) *and* CP *times to solve the problem by* SPCOL *and to improve it by* COLSYS. *The* CP *times include the time to calculate the reduced solution, which was* 0.9 *time units. Total* CP *is the sum of the* SPCOL CP *and the* COLSYS CP.

	SPCOL		COLSYS		
ε	NSUB	CP	NSUB	CP	Total CP
10^{-1}	44	0.9	62	3.6	4.5
10^{-2}	52	0.9	84	3.8	4.7
10^{-4}	59	0.9	232	15.3	16.2
10^{-6}	59	0.9		failed	

TABLE 12
Example 2 *with* $\gamma = 2$ *and* $X_0(0) = -4.29$. *Number of subintervals* (NSUB) *and* CP *times to solve the problem by* COLSYS *with continuation in* ε *from* $\varepsilon = 10^{-2}$. *Total* CP *is the accumulated time for the sequence.*

ε	NSUB	CP	Total CP
10^{-2}	84	3.8	3.8
10^{-4}	168	21.3	25.1
10^{-6}	322	40.8	65.9

TABLE 13
Example 2 *with* $\gamma = 2$ *and* $X_0(0) = -4.29$. *Relative difference between* SPCOL *and* COLSYS *solutions with* $\Delta(\ ) := |(\ )_{\text{SPCOL}} - (\ )_{\text{COLSYS}}|$.

ε	$\dfrac{\Delta x(1,\varepsilon)}{\lvert x(1,\varepsilon)_{\text{COLSYS}}\rvert}$	$\dfrac{\Delta y_2(1,\varepsilon)}{\lvert y_2(1,\varepsilon)_{\text{COLSYS}}\rvert}$
10^{-1}	9.7×10^{-3}	2.4×10^{-1}
10^{-2}	9.6×10^{-4}	3.9×10^{-2}
10^{-4}	9.6×10^{-6}	4.3×10^{-4}
10^{-6}	1.0×10^{-7}	4.5×10^{-6}

5. Discussion. We have obtained asymptotic approximations for a restricted class of nonlinear singularly perturbed two-point boundary value problems and have shown how to construct approximate solutions numerically and use them to suggest a nonuniform mesh that may be used as input to a two-point boundary value code in order to calculate improved solutions. Clearly this approach offers some advantages over the more standard technique of continuation in ε steps; however, the picture is far from clear and several questions still remain as to how best to use asymptotic analysis in conjunction with numerical analysis.

In Example 2 of § 4 we have shown that asymptotic methods may be used to distinguish different solutions in problems having multiple solutions. These asymptotic approximations may be used to provide initial guesses to a two-point boundary value code.

In Example 1 of § 4 we have shown that unbounded solutions can result from seemingly minor changes in the boundary conditions of singularly perturbed boundary value problems. Other very diverse behaviors can occur when turning point problems are considered (cf., e.g., Kevorkian and Cole [18] or O'Malley [21]). Since phenomena cannot easily be predicted, a sensible course to follow is perhaps to use asymptotic and numerical methods in tandem. For example, a rough numerical solution could be obtained for several values of ε which could then be used to suggest the form of an asymptotic solution. The asymptotic approximation could then be used to refine the numerical solution, and so on. It is also possible that singular perturbation theory could be used to construct special methods that are appropriate for specific problems as e.g., in Flaherty and Mathon [9] and Ascher and Weiss [2], [3], [4].

REFERENCES

[1] U. Ascher, J. Christiansen and R. D. Russell, *Collocation software for boundary value* ODE's, ACM Trans. Math. Software, 7 (1981), pp. 209–222.

[2] U. Ascher and R. Weiss, *Collocation for singular perturbation problems* I: *First order systems with constant coefficients*, SIAM J. Numer. Anal., 20 (1983), pp. 537–557.

[3] ———, *Collocation for singular perturbation problems* II: *Linear first order systems without turning points*, Tech. Rep. 82-4, Dept. Computer Science Univ. British Columbia, Vancouver, 1982.

[4] ———, *Collocation for singular perturbation problems* III: *Nonlinear problems without turning points*, Tech. Rep. 82–9, Dept. Computer Science, Univ. British Columbia, Vancouver, 1982; this Journal, this issue, pp. 811–829.

[5] A. Bjorck, *A block* QR *algorithm for partitioning stiff differential systems*, Rep. LITH-MAT-82-44 Linkoping University, 1982.

[6] B. Childs, M. Scott, J. W. Daniel, E. Denman and P. Nelson, eds., *Codes for boundary-value problems in ordinary differential equations*, Proceedings of a Working Conference, May 14–17, 1978, Lecture Notes in Computer Science, 76, Springer-Verlag, Berlin, 1979.

[7] W. Enright, T. E. Hull and B. Lindberg, *Comparing methods for stiff systems of* ODEs, BIT, 54 (1976), pp. 66–93.

[8] W. E. Ferguson, Jr., *A singularly perturbed linear two-point boundary value problem*, Ph.D. dissertation, California Institute of Technology, Pasadena, 1975.

[9] J. E. Flaherty and W. Mathon, *Collocation with polynomial and tension splines for singularly-perturbed boundary value problems*, this Journal, 1 (1980), pp. 260–289.

[10] J. E. Flaherty and R. E. O'Malley, Jr., *On the numerical integration of two-point boundary value problems for stiff systems of ordinary differential equations*, in Boundary and Interior Layers—Computational and Asymptotic Methods, J. J. H. Miller, ed., Boole Press, Dublin, 1980, pp. 93–102.

[11] ———, *Singularly perturbed boundary value problems for nonlinear systems, including a challenging problem for a nonlinear beam*, in Theory and Applications of Singular Perturbations, W. Eckhaus and E. H. deJager, eds., Lecture Notes in Mathematics, 942, Springer-Verlag, Berlin, 1982, pp. 170–191.

[12] ———, *Asymptotic and numerical methods for vector systems of boundary value problems*, Proc. 1982 Army Numerical Analysis and Computers Conference, ARO Rep. 82-3, 1982, pp. 381–395.

[13] G. H. Golub and J. H. Wilkinson, *Ill-conditioned eigensystems and the computation of the Jordan canonical form*, SIAM Rev., 18 (1976), pp. 578–619.

[14] W. A. Harris, Jr., *Singular perturbations of two-point boundary value problems for systems of ordinary differential equations*, Arch. Rational Mech. Anal., 5 (1960), pp. 212–225.

[15] P. W. Hemker, *A Numerical Study of Stiff Two-Point Boundary Value Problems*, Mathematical Centre Tracts 80, Amsterdam, 1977.

[16] F. C. Hoppensteadt, *Properties of solutions of ordinary differential equations with a small parameter*, Comm. Pure Appl. Math., 24 (1971), pp. 807–840.

[17] F. A. Howes, *Some old and new results on singularly perturbed boundary value problems*, in Singular Perturbations and Asymptotics, R. E. Meyer and S. V. Parter, eds., Academic Press, New York, 1980.

[18] J. Kevorkian and J. D. Cole, *Perturbation Methods in Applied Mathematics*, Applied Mathematical Sciences 34, Springer-Verlag, New York, 1981.

[19] B. KREISS AND H. O. KREISS, *Numerical methods for singular perturbation problems*, SIAM J. Numer. Anal., 18 (1981), pp. 262–276.

[20] R. M. M. MATTHEIJ, *The conditioning of linear boundary value problems*, SIAM J. Numer. Anal., 19 (1982), pp. 963–978.

[21] R. E. O'MALLEY, JR., *Introduction to Singular Perturbations*, Academic Press, New York, 1974.

[22] ———, *On multiple solutions of singularly perturbed systems in the conditionally stable case*, in Singular Perturbations and Asymptotics, R. E. Meyer and S. V. Parter, eds., Academic Press, New York, 1980, pp. 87–108.

[23] ———, *Shock and transition layers for singularly perturbed second order vector systems*, SIAM J. Appl. Math., 43 (1983), pp. 935–943.

[24] S. OSHER, *Numerical singular perturbation problems and one-sided difference schemes*, SIAM J. Numer. Anal., 18 (1981), pp. 129–144.

[25] C. E. PEARSON, *On a differential equation of the boundary layer type*, J. Math. and Phys., 47 (1968), pp. 134–154.

[26] ———, *On non-linear ordinary differential equations of boundary layer type*, J. Math. and Phys., 47 (1968), pp. 351–358.

[27] A. RUHE, *An algorithm for numerical determination of the structure of a general matrix*, BIT, 10 (1970), pp. 196–216.

[28] R. D. RUSSELL AND J. CHRISTIANSEN, *Adaptive mesh strategies for solving boundary value problems*, SIAM J. Numer. Anal., 15 (1978), pp. 59–80.

[29] G. SÖDERLIND, *Theoretical and computational aspects on partitioning in the numerical integration of stiff differential systems*, Report 8115, Department of Numerical Analysis and Computing Science, The Royal Institute of Technology, Stockholm, 1981.

[30] W. WASOW, *On the asymptotic solution of boundary value problems for ordinary differential equations containing a parameter*, J. Math. and Phys., 23 (1944), pp. 173–183.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

UNDERFLOW AND THE RELIABILITY OF NUMERICAL SOFTWARE*

JAMES DEMMEL†

Abstract. We examine the effects of different underflow mechanisms on the reliability of numerical software. Software is considered reliable in the face of underflow if the effects of underflow are no worse than the uncertainty due to roundoff alone. The two primary underflow mechanisms discussed are store zero and gradual underflow, although we consider other mechanisms as well. By examining a variety of codes, including Gaussian elimination, polynomial evaluation, and eigenvalue calculation, we conclude that gradual underflow makes it significantly easier to write good numerical codes than store zero, and that this remains true even if extra range and precision are available for intermediate calculations.

Key words. software reliability, numerical software, roundoff, underflow, error analysis

1. Introduction and summary.

In this paper we examine the effects of underflow on the reliability of codes for solving a wide variety of numerical problems. In particular we demonstrate the utility of gradual underflow for writing more robust codes than are usually written when the conventional "store zero" approach to underflow is used. This paper summarizes the work of several people over a period of several years during which they participated in the IEEE Floating Point Standard subcommittee's deliberations about the proper way to handle underflow. In addition to the author, these people are J. Coonen, D. Hough, W. Kahan and S. Linnainmaa. Some of the results presented here have been published (separately) before; others have not.

When we speak of reliable software, we mean software that ideally produces accurate results whenever they can be represented, and otherwise gives a warning. Needless to say, such software must cope with roundoff, and that may be difficult for many problems even in the absence of underflow. These unavoidable roundoff errors have led to diminished expectations and less stringent definitions of reliability for different kinds of codes. For example, a Gaussian elimination code to solve a system of linear equations is commonly called reliable if it delivers the exact solution of a problem close to the one it received as input (we will discuss this example in more detail below). Users have come to expect no more than these weaker forms of reliability from many of their codes because both experience and sometimes proofs have demonstrated that roundoff errors prevent better performance.

How much further must the notion of reliability be weakened in the face of underflow? For example, does Gaussian elimination still deliver the exact solution of a problem close to the input if underflows occur during the computation? If so, and in general, if we can show that the effects of underflow on a code are no worse than the uncertainty due to roundoff alone, then we consider that code no less reliable in the face of underflow than in the face of roundoff. Thus, our approach during our investigations has been to decide if underflow contributes nothing worse to a code than the uncertainty from the expected effects of roundoff errors which must be tolerated anyway.

To explain our approach and conclusions, we need some notation. A more complete discussion of the following terminology may be found in § 2 of this paper. We describe

* Received by the editors December 14, 1982, and in revised form May 5, 1983. This work was supported by the U.S. Department of Energy, contract DE-AM03-76SF00034, Project Agreement DE-AS03-79ER10358, the Office of Naval Research, contract N00014-76-C-0013, and IBM, contract 82007PLP0446. The author holds an IBM Fellowship. This paper contains an expanded version of "Effects of Underflow on Solving Linear Systems", by Dr. James Demmel, appearing in the Proceedings of the 5th Symposium on Computer Arithmetic, Ann Arbor, MI, May 18–19, 1981, pp. 113–120. Copyright © 1981 IEEE.

† Computer Science Division, University of California, Berkeley, California 94720.

floating point arithmetic with two parameters: ε and λ. ε denotes the difference between 1 and the next larger floating point number; thus ε bounds the rounding error in the operations +, −, * and /. λ denotes the underflow threshold, i.e. the smallest positive normalized floating point number. The two basic underflow mechanisms we have compared are store zero and gradual underflow. Store zero, the standard response to underflow, simply replaces any result that would be smaller than λ in magnitude by 0. Gradual underflow, on the other hand, returns an unnormalized floating point number less than λ in magnitude which approximates the tiny result. These unnormalized numbers form an arithmetic progression between 0 and λ with common separation $\lambda\varepsilon$, and are called *denormalized* to emphasize that they occur only at the bottom of the exponent range. Gradual underflow will henceforth be abbreviated by G.U. and store zero by S.Z.

There are actually many more mechanisms available to the system architect; all underflow mechanisms will be discussed further in § 2 below. For reasons also explained there we selected the following variations on G.U. and S.Z. for analysis in this paper:

We compared using the *same* precision and range for intermediate calculations as are used to represent the inputs and outputs with using *extra* precision and range for intermediate calculations.

We compared using gradual underflow with the underflow flag being set by a *threshold test* (which signals underflow whenever the result is denormalized) with using gradual underflow with the flag being set by an *accuracy test* (which signals underflow only if the denormalized result has a numerical value different from that of the correctly rounded result).

We compared using underflow flags which are *sticky* (which, once set, remain set until explicitly reset by the user) with underflow flags which are *nonsticky* (which are reset prior to each floating point operation).

We have compared the effects of these mechanisms on the robustness of codes written without attention to over/underflow problems, but we occasionally consider highly robust, expert codes as well.

Our main conclusions are given below:

(1) For many algorithms written without attention to over/underflow, only if G.U. is used instead of S.Z. is the algorithm as robust in the face of roundoff and underflow as it is with roundoff alone. More specifically, as long as the data is normalized ($>\lambda$ in magnitude) the results are as good as can be expected just with roundoff when using G.U., but when using S.Z. the data must be at least λ/ε to expect the same performance.

(2) For some computations, one can claim more than in (1). Suppose we measure backwards error in the following combined relative/absolute way:

$$\text{the change in } x \text{ is comparable to} \begin{cases} \varepsilon|x| & \text{if } |x| \geqq \lambda, \\ \varepsilon\lambda & \text{if } |x| < \lambda, \end{cases}$$

for G.U., and

$$\text{the change in } x \text{ is comparable to} \begin{cases} \varepsilon|x| & \text{if } |x| \geqq \lambda/\varepsilon, \\ \lambda & \text{if } |x| < \lambda/\varepsilon, \end{cases}$$

for S.Z. For G.U. this means the change in x is comparable to a few units in the last place stored of x, no matter if x is normalized or not. For S.Z., on the other hand, numbers near λ contain almost no significant digits. Then with respect to this new distance function, many algorithms always deliver the exact solution of a problem close to the input problem, no matter if underflow occurs or not. This statement is true of Gaussian elimination as long as the results themselves do not underflow and lose accuracy, of polynomial evaluation, and of computing the eigenvalues of a symmetric tridiagonal matrix, for example. In other words, these algorithms *always* have a small backwards error with respect to this new definition (and subject to easily testable constraints), no matter what the inputs are. For G.U., this means nearly *every* bit stored in a number is significant, whereas in S.Z. almost no bits in any number of the problem may be significant, if all the numbers are too close to λ.

(3) In addition to extending the effective exponent range of the system by $-\log_2 \varepsilon$ as described in (1), G.U. preserves certain mathematical relationships (such as $x=y$ if and only if $\mathrm{fl}(x-y)=0$) over the *entire* range of floating point numbers. These relationships may occasionally fail with S.Z. Their failure can lead to strange and elusive bugs in codes (see § 4 below), whereas it is easier to write reliable code if these relationships can be depended on.

(4) Availability of extended precision *and* range does not always obviate the advantage of G.U. over S.Z. For some computations, such as polynomial evaluation, an extended format does eliminate almost all worry about intermediate over/underflow, but for others, such as Gaussian elimination and Cholesky decomposition, as long as the solution itself and the triangular factors of the matrix are stored in the basic format, the conclusions in (1) above remain valid even if all intermediate results are computed *exactly*. Thus, G.U. is of advantage to a system with an extended format as well as to a system with just one format.

(5) There are computations for which the accuracy test for G.U. is preferable to the threshold test and computations for which the threshold test is preferable, but the relative advantage is not very great for either type of test. The only advantage of the accuracy test over the threshold test we discovered was in the underflow flag being a false alarm less frequently. These potential false alarms arise from the assignment statement $a := b$ when b is denormalized, negation ($a := -b$ when b is denormalized), addition, subtraction, multiplication when one factor is an integer, and remainder ($a \bmod b$). The only potential advantage of the threshold test over the accuracy test was in helping to automatically verify the constraint that inputs be normalized ($>\lambda$ in magnitude) mentioned in (1) above. It was not clear that this could be used easily in practice (see the discussion of Gaussian elimination in § 8 below).

(6) The sticky underflow flag is much more useful than the nonsticky kind, although there are several applications of nonsticky flags in expert codes (see the discussion of Gaussian elimination below). The sticky flag can be used to simulate a nonsticky one at the cost of resetting it before each relevant operation, a cost which may be severe if resetting requires an expensive system call in a tight loop.

(7) Highly robust, expert codes for problems like polynomial root finding are easier to write using G.U. than S.Z. However, as soon as any scaling is done it is usually as easy to scale to avoid S.Z. underflows as G.U. (see the discussions of Gaussian elimination, Cholesky decomposition, and eigenvalue computations in [2]).

We believe that the evidence weighs clearly in favor of G.U. over S.Z. Presumably that is why gradual underflow is required by the proposed floating point standard.

The evidence shows neither the accuracy test for G.U. nor the threshold test to be uniformly superior to the other, but the choice depends on whether the floating point designer also has control over how the compilers implement assignment and negation statements (see § 5). If he does have control, he should insist on simple bit copying (nonfloating point) operations; if not, choosing the accuracy test over the threshold test eliminates the possibility of spurious underflow messages during assignment and negation. The proposed standard incorporates the accuracy test for lack of control over compiler design.

The sticky underflow flag is preferable to the nonsticky kind if there can be only one; a friendly system would make both available. The proposed standard requires sticky flags for all exception conditions, including underflow.

The rest of this paper is organized as follows. Section 2 presents underflow from a system architect's point of view. We discuss number formats and the options available for handling underflows, both when the underflow occurs and when the result is used later. Section 3 discusses underflow from a numerical analyst's point of view and shows how to extend conventional error analyses to include underflow. Sections 4 through 14 elaborate on the above results (without proofs) for the eleven computations listed below. Sections 4 through 14 may be read independently of one another:

tests and comparisons
the accuracy test versus the threshold test for G.U.
complex arithmetic
inner product calculations
Gaussian elimination
Cholesky decomposition
iterative refinement of linear systems
polynomial evaluation and root finding
eigenvalue computations for symmetric tridiagonal matrices
numerical quadrature
accelerating the convergence of sequences

2. A system architect's view of underflow. In this section we have two goals, first to describe the mechanisms available to the system architect for handling underflow, and second to describe the mechanisms we compare in this paper and why we have chosen them. We will introduce much notation in this section; when a new term is defined it will appear in *italics*.

The design questions facing the system architect are of two kinds: what value should be returned in the destination word when underflow occurs, and what side effects (if any) should underflow have? Options for the destination value are G.U., S.Z., and several other conventions such as exponent wraparound [10] and nonnumeric symbols like UN [4] and NAN [8]. Possible side effects are raising an underflow flag and continuing execution, invoking a trap handler that may execute any code of the system's or user's choice, waiting until an underflowed quantity is to be used to decide what to do, or most simply doing nothing. In case the architect decides to have flags or traps, the efficiency of his implementation will affect how the programmer writes codes to use the flags or traps (see the discussion of Gaussian elimination in § 5, for example). Other side effects arise from design decisions made in the compiler; these are discussed below and in §§ 4 and 5. We will first discuss the different values that can be returned from an underflowed operation, and then possible side effects.

To describe the values that can be returned we need to refer to a specific floating point format which we now describe (the conclusions of this paper apply to similar formats as well). It contains three fields: a *sign bit* σ, a *significand* f, and an *exponent* e, and represents the value $x=(-1)^{\sigma}\cdot f\cdot 2^{e}$. The exponent e satisfies $e_{\min}\leqq e\leqq e_{\max}$. The binary point follows the leading bit of f.

We call f (and the entire floating point number) *normalized* if its leading bit is 1 (or if $e=0$ and $f=0$, which represents 0). This means $1\leqq f<2$. Otherwise $0\leqq f<1$ and is called *unnormalized.*

The *rounding error* of the arithmetic is the largest possible value of

$$\frac{|\mathrm{fl}\,(a\blacksquare b)-a\blacksquare b|}{|a\blacksquare b|},$$

where ■ denotes one of the operations $\{+, -, *, /\}$, and a and b are such that $a \blacksquare b \neq 0$ and fl $(a \blacksquare b)$, which denotes the floating point result of the operation $a \blacksquare b$, is normalized and nonzero. As long as fl $(a \blacksquare b)$ is the first floating point number greater than or equal to $a \blacksquare b$ or the first number less than or equal to $a \blacksquare b$ (e.g. if the arithmetic truncates or rounds), then

$$\varepsilon \equiv 2^{1-n},$$

where n is the number of bits used to represent f, is a bound on the rounding error. In other words, ε is the difference between 1 and the next larger floating point number. Note that ε is twice as big as the rounding error if fl $(a \blacksquare b)$ is the nearest floating point number to the true result $a \blacksquare b$.

The largest normalized number has $e = e_{\max}$ and $f = 1.1 \cdots 1$ (n bits long); it is called the *overflow threshold* and denoted by

$$\Lambda \equiv 2^{e_{\max}}(2-\varepsilon) \approx 2^{e_{\max}+1}.$$

The smallest normalized number, which has $e = e_{\min}$ and $f = 1$, is called the *underflow threshold*, and is denoted by

$$\lambda \equiv 2^{e_{\min}}.$$

Even though λ is called the underflow threshold, we will see that underflow might not always be signalled whenever a result is less than λ in magnitude.

When $e = e_{\min}$ and $f < 1$ we call the number *denormalized*. Denormalized numbers are also called *subnormal* [6], a name which is perhaps more descriptive than denormalized. The denormalized numbers, which are a subset of the unnormalized numbers, form an arithmetic progression between 0 and λ with common separation $\lambda\varepsilon$. Not all floating point systems allow denormalized numbers, or any unnormalized numbers at all. If denormalized numbers are not allowed, we typically handle underflow using *store zero* (S.Z.). This means that if the rounded value of a computation x would lie strictly between $\pm\lambda$ so that we could not represent it as a normalized nonzero number, we return zero. If denormalized numbers are allowed then we can use *gradual underflow* (G.U.), which means rounding such an x to the nearest denormalized number and returning that instead of zero. Gradual underflow is also called *graceful underflow* [6].

Exponent wraparound [10] is another possibility which only makes sense on a system which does not trap on over/underflow but which increments/decrements a counter designated in advance by the user (cf. Kahan's Counting Mode [10]). When a result would underflow, the value returned has the normalized significand of the result stored in f and the result's exponent biased upward by a constant (such as $-3 \cdot e_{\min}/2$) stored in e. (The analogous technique applies to overflow). By examining the counter the user can keep track of the powers of two contributed by wraparound.

Finally, the system may return a nonnumeric symbol such as UN [4] or NAN (*Not A Number*) [8]. A NAN is encoded in the IEEE proposal by an exponent $e = e_{\max} + 1$ and a nonzero significand f that may contain or point to diagnostic information about where and when the underflow occurred. This technique allows the user to defer deciding what to do about an underflow until later when he has more information (this is discussed further below). For more detail on floating point formats and representing underflowed quantities see [1].

The architect also has many options for side effects. Side effects of underflow may be generated on two occasions: when an underflowed quantity is created, and later when it is used. First we describe creation time side effects and then use time side effects.

The creation time options are raising a flag/not raising one, trapping/not trapping, and doing nothing. Doing nothing is the most common response of systems today because underflow is generally presumed to be harmless (were that true, this paper would not have been written).

One attribute a flag can have is "stickiness". An underflow flag is *sticky* if, once set, it remains set until explicitly reset by the user (as in the proposed standard); otherwise it is *nonsticky*, that is reset prior to each operation. A sticky flag is generally much more useful than a nonsticky one because it allows the user to ask if any underflows have occurred anywhere in a section of code (since the last time the flag was reset). This is the proper type of flag for debugging or when underflows are not anticipated. A nonsticky flag, which can always be simulated by a sticky one, is useful only when analysis has shown that underflow in only a certain few operations can matter. This is the case in certain expert codes (see § 8 below) but is rare.

Another attribute a flag can possess is available only with G.U.: it can be set either by a threshold test or an accuracy test:

Threshold test. Signal underflow if the exact result would have been less than λ in magnitude and not zero, and

Accuracy test. Signal underflow if, in addition to the computed result being no more than λ in magnitude, it is different from what would have been the result had exponent range been unbounded.

The reason for the option is as follows. Just because a result of an operation must be represented as a denormalized number does not mean accuracy has been lost. It may be that the error incurred by denormalization is no worse than what roundoff would have caused had exponent range been unlimited. For example, $\lambda/2$ is representable exactly as a denormalized number. In such cases, the architect may decide not to signal underflow, since the error is no worse than what roundoff alone would have caused. This more restrictive definition of underflow has the advantage of signalling underflow less frequently than the threshold test and therefore generates fewer false alarms. For example, the accuracy test will never signal underflow on copy (assigning $a := b$), negation ($a := -b$), addition, subtraction, multiplication where one factor is an integer, or remainder ($a \bmod b$) [16]. On the other hand, a threshold test may be better for an application where any nonzero result less than λ in magnitude causes problems later in the code. In the friendliest system, the user would be able to choose the definition depending on his application. For example, when debugging a new code in which underflow is not expected to occur, a threshold test with a smart trap handler/debugger would be useful, whereas a clever, robust code might exploit the more restrictive definition. We give examples of codes which use both types of flags below.

There are at least as many options available to the designer of a trap handler, because in principle a trap handler can contain any code of the system's or user's choice. For example, one may want a smart trap handler/debugger which lets the user examine his operands and code when underflow occurs, or one which keeps a record of where and when all underflows occur and lets the user examine them at the end of the program, or even one which attempts to perform the computation in a totally different way to avoid underflow. Actually, any given underflow mechanism can be implemented using a trap handler if a trap occurs on every underflow, although this may be slow. Obviously these possibilities involve compiler and operating system questions which would be difficult and interesting even without raising any numerical issues; we will not consider traps further in this paper.

Finally, the system (or user) can decide at the time of use what to do about underflow. This option is not available in an S.Z. system because there is nothing unusual about an underflowed S.Z. value (it is zero) that lets us detect when it is used;

with G.U., however, denormalized numbers mark themselves as underflowed quantities. By delaying a reaction until time of use, the user can defer judgement about the harmfulness or harmlessness of an underflow until he has more information available to help him decide. If a denormalized number is to be added to a much larger number, for example, little or nothing is lost. If it is to be multiplied by a large number, accuracy lost in denormalization might become significant later, especially if cancellation occurs. In general these decisions can better be made when the denormalized number is to be used rather than when it is created. Again, it is advantageous to give the user a choice in response. One approach considered by the IEEE committee was to have two modes: warning and normalizing. *Warning mode* caused a trap whenever

the uncertainty in a denormalized operand ($\pm\lambda\varepsilon/2$) would be magnified relative to the result by multiplication or division by a normalized operand, or
dividing a finite nonzero dividend by a denormalized divisor, or
taking the square root of a denormalized number.

Normalizing mode does not trap in these cases. As with the different definitions of underflow, warning mode may be useful for debugging new codes, and normalizing mode for writing clever, robust ones. We again give examples of such clever codes below. The committee chose not to include warning mode in the standard.

Given this bewildering array of options, how do we intend to compare G.U. and S.Z. systems? It is obviously possible to compute anything using S.Z. that can be computed with G.U. (and vice versa) by testing and scaling each pair of operands before use, but this is hardly a fair comparison since one code may be much harder to write or take much longer to run than the other. One fair comparison is to ask if for a given level of system support and given level of effort the code using G.U. has substantially different reliability than one using S.Z. For the comparisons in this paper, we chose the least effort possible, meaning that we want to compare codes written without regard to underflow at all if possible, or sight modifications of such codes. Furthermore, we chose the least possible system support short of doing nothing: providing a user testable underflow flag (and, of course, not trapping on underflow). We also consider the two ways to raise the G.U. flag described above: the threshold test and the accuracy test (in what follows we will often use the phrase "inaccurate underflows" to refer to both S.Z. underflows and G.U. underflows according to the accuracy test). Applications of nonsticky flags will be noted when they exist; unless a flag is explicitly called nonsticky it should be assumed sticky. In addition to these underflow options, we examine the utility of performing intermediate calculations with extra precision and range to avoid as many underflows as possible.

Finally, a writer of clever library routines may well be interested in how much reliability he can get for a fixed execution time, code size, etc., independent of development cost. We believe several of the codes discussed in this paper (and in more detail in [2]) will provide a basis for such a comparison.

3. A numerical analyst's view of underflow. In this section we show how to extend traditional floating point error analyses to take underflow into account. Let $\blacksquare$ be one of the operations $\{+, -, *, /\}$ and let $\mathrm{fl}\,(a \blacksquare b)$ denote the floating point result of the indicated operation. Traditional error analyses use the formula [23]

$$\mathrm{fl}\,(a \blacksquare b) = (a \blacksquare b)(1+e) \quad \text{unless } a \blacksquare b \text{ underflows or overflows,} \tag{1}$$

where $|e| \leqq \varepsilon$. To take underflow into ccount, we write [13]

$$\mathrm{fl}\,(a \blacksquare b) = (a \blacksquare b)(1+e) + \eta \quad \text{unless } a \blacksquare b \text{ overflows.} \tag{2}$$

In the case of G.U. there are the following constraints on e and η:

(3) $$|e| \leqq \varepsilon \quad \text{and} \quad |\eta| \leqq \lambda\varepsilon,$$

(4) $$\eta \cdot e = 0 \quad \text{(i.e. at most one of } \eta \text{ and } e \text{ is nonzero), and}$$

(5) $$\eta = 0 \quad \text{if } \blacksquare \text{ is either addition or subtraction.}$$

In the case of S.Z. we have the following somewhat different constraints on e and η:

(3′) $$|e| \leqq \varepsilon \quad \text{and} \quad |\eta| \leqq \lambda, \quad \text{and}$$

(4′) $$\eta \cdot e = 0.$$

Let us examine the differences in constraints. The different bounds on $|\eta|$ in (3) and (3′) mean that the error contributed by underflow for S.Z. can be $1/\varepsilon$ times as large as for G.U. (5) means that there can be no underflow error in addition or subtraction for G.U., whereas underflow can cause complete loss of relative accuracy for S.Z.

Formula (2) gives a combined relative/absolute error bound on the error in floating point. For G.U. we have a bound ε on the relative error as long as the true result is bigger than a threshold λ, and an absolute error bound $\lambda\varepsilon$ for smaller results. The bounds match, in that for results at the underflow threshold λ, the absolute magnitude of the largest relative error ($\varepsilon \cdot$ result) is equal to the largest absolute error ($\varepsilon \cdot \lambda$) (see Fig. 1). This property of (2) means that when doing a G.U. error analysis, we are really doing both a floating point and fixed point analysis simultaneously, because

$$\text{fl}\,(a \blacksquare b) = (a \blacksquare b) + \eta$$

is the error formula used in fixed point analyses.

For S.Z. on the other hand, the error jumps at λ. For results just bigger than λ, the largest possible error is $\lambda\varepsilon$ as with G.U., but for smaller results the error leaps up to nearly λ (see Fig. 2). In order to analyze errors in S.Z. arithmetic as in G.U. (relative error above a threshold, absolute below, and at the threshold the errors match), we must raise the threshold to λ/ε (see Fig. 3). Said another way, G.U. reduces underflow errors to the size of roundoff for all normalized results, but S.Z. underflow errors are roundoff size only for results greater than λ/ε in magnitude. This explains why so many of the results to be presented later read as follows:

(6) When using G.U., as long as the data is normalized ($>\lambda$), the results are as good as can be expected just with roundoff, but when using S.Z. the data must be at least λ/ε to expect the same performance. Furthermore, as the data decreases below the threshold (λ or λ/ε) G.U.'s results degrade smoothly rather than abruptly, as do S.Z.'s.

λ is a much more natural threshold (and easier to test for, depending on the definition of underflow) than λ/ε for the range of application of a code.

For some codes one can make a backwards error bound independent of input values if one measure backwards error in the way suggested in conclusion (2) of § 1. For G.U., this measure means *every* number is viewed as uncertain in the last few places stored, whether denormalized or not. For S.Z., it means some numbers near λ are viewed as uncertain in nearly all their places. Gaussian elimination without inaccurate underflows in the solution components themselves, polynomial evaluation, and our algorithm for eigenvalues of symmetric tridiagonal matrices have backwards error bounds of this form. For these codes, G.U. more than extends the apparent exponent range by $-\log_2 \varepsilon$ over S.Z.: it asserts the significance of nearly all bits in every number in the machine.

Horizontal axis: True result of operation $a \blacksquare b$. (Tic marks represent floating point numbers.)
Vertical axis: —— Error in computed result $a \blacksquare b - \mathrm{fl}\,(a \blacksquare b)$. (Arithmetic is binary and chopped with $\varepsilon = \frac{1}{8} =$ maximum rounding error, $\lambda =$ underflow threshold)
—— Error bound.

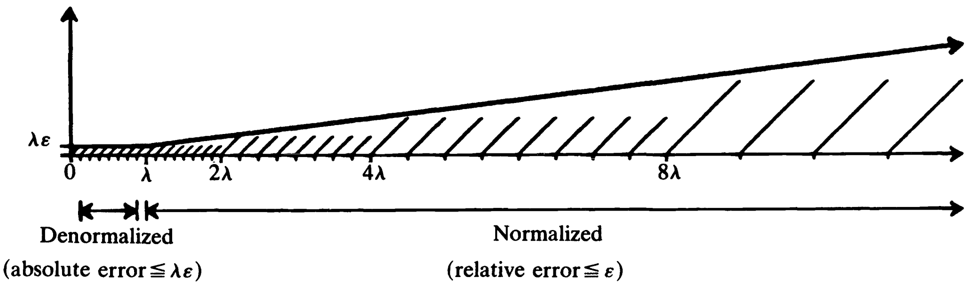


FIG. 1. *Error with gradual underflow* (*see* (2), (3), (4) *for error bound*).

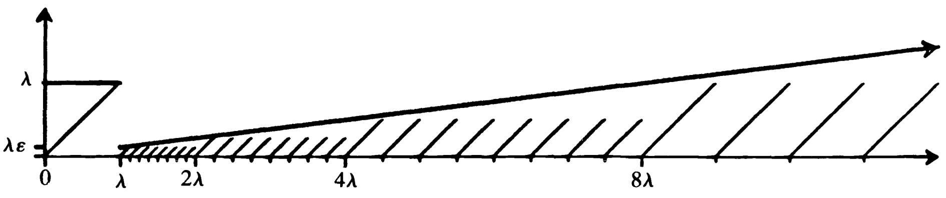


FIG. 2. *Error with store zero* (*see* (2), (3′), (4′) *for error bound*).

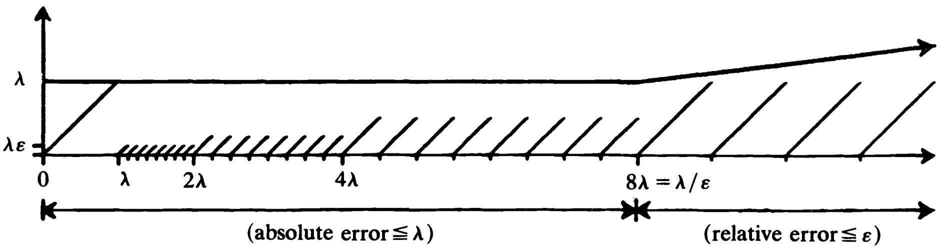

λ
$\lambda\varepsilon$
0
λ
2λ
4λ
$8\lambda = \lambda/\varepsilon$
(absolute error $\leqq \lambda$)
(relative error $\leqq \varepsilon$)

FIG. 3. *Store zero error bounded in same way as gradual underflow error.*

If all G.U. did were to extend the apparent exponent range of the system, then the argument for G.U. over S.Z. would become weaker as the actual exponent range grew larger. As we have just seen, however, there are certain mathematical relationships, preserved by G.U. but not S.Z. over the range of all floating point numbers, which make codes that are to work over the range of all inputs easier to write. Other useful relationships preserved by G.U. but occasionally violated by S.Z. include [1]:

$$(7) \qquad x = y \quad \text{if and only if} \quad \mathrm{fl}\,(x-y) = 0,$$

$$(8) \qquad \mathrm{fl}\,((x-y)+y) \approx x \quad \text{(to within a rounding error in the larger of } x \text{ and } y\text{)},$$

and assuming the exponent range $[e_{\min}, e_{\max}]$ is nearly symmetrical about 0 (as with the proposed IEEE standard), then if no overflow occurs

$$(9) \qquad \mathrm{fl}\,(1/(1/x)) \approx x \quad \text{to within a few rounding errors in } x.$$

Failure to satisfy statements like (7) to (9) can induce strange and elusive bugs in codes (see § 4 and [10]). Their validity makes it much easier to write and maintain codes by eliminating the need for tests for the very rare cirumstances in which they fail.

The combined relative/absolute error measure given in (2) arises naturally in several ways. When solving linear equations with iterative refinement, we stop when the relative error in the solution vector is (hopefully) small. This means large components are known to high relative accuracy, and small components to an absolute accuracy of the same magnitude. In physical problems there is often a noise level which means that only measurements above it can be made relatively accurately, and below it only with absolute accuracy equal to the noise level.

4. Tests and comparisons. To analyze codes containing tests like

$$\begin{array}{ll} \text{if } x \neq y & \text{then } r := \dfrac{f(x)-f(y)}{x-y} \\ & \text{else if } 100 * x \neq 100 * y \text{ then print } \textit{why}? \end{array} \tag{10}$$

or

$$\text{if } x \neq 0 \text{ and } |x-y| \leqq .001|x| \text{ then } z = \text{SQRT}\,(1.5 - y/x) \tag{11}$$

the first of which can produce a divide by zero error and the second of which a square root of negative number error, we must not only know how underflow is handled, but how the compiler implements tests like "$x \neq y$?". There are two possibilities for this: a fixed point (bitwise) comparison of x and y, and a comparison of fl$(x-y)$ with 0.

Let us first analyze (10) and (11) using S.Z. With the first (fixed point) implementation of "$x \neq y$?", any choice of x and y such that $0 < |x-y| < \lambda$ (such as $x = 1.25\lambda$ and $y = 2\lambda$) will pass the test "$x \neq y$?" and cause a divide by zero error in the expression for r in (10). In (11), the same choice of x and y passes both tests but causes $1.5 - y/x$ to equal $-.1$ and gives a square root of negative number error. Using the second, floating point implementation of "$x \neq y$?" the same x and y causes *why*? to be printed by (10). Thus, both implementations and even the more robust looking test in (11) can cause strange results using S.Z.

With G.U., on the other hand, the two implementations of the test "$x \neq y$?" are equivalent (barring overflow of fl$(x-y)$), and neither divide by zero nor *why*? nor square root of negative number messages are possible from (10) or (11). Any underflow flags raised by the threshold test should be ignored in these examples because if an addition or subtraction underflows in G.U. arithmetic, it must give the exact result (thus no underflow flag would be raised with the accuracy test).

The pitfalls of using extended range and precision in comparisons have been well documented in [15].

5. The accuracy test versus the threshold test for G.U. When an operation $a \blacksquare b$ underflows, the denormalized result need not have a different numerical value from the result that would have been returned had the exponent range been unbounded. For example, the results of $\lambda/2, \lambda/4, \cdots, \lambda/(1/\varepsilon)$ are all denormalized yet representable without error. The accuracy test for G.U. will not raise the underflow flag for these operations, or for any others where the denormalized result is identical to the result that would have been returned had the exponent range been unbounded. In contrast, the threshold test raises an underflow flag whenever a nonzero result is less than λ in magnitude (there are slight variations possible on this definition, but they do not effect the results of this analysis).

The accuracy test has the advantage over the threshold test, that if the *only* bad effect of underflow is its abnormally large loss of accuracy, then it avoids raising the underflow flag unnecessarily, whereas the threshold test raises the flag whenever the result is small even if accurate. If, on the other hand, it is the *size* of an underflowed result that can cause difficulty later, the threshold test is more useful. We have found examples where both definitions of underflow are useful.

First we discuss examples where the threshold test appears advantageous. In conclusion (1) of the § 1, we stated that for many algorithms as long as the inputs were normalized ($>\lambda$ in magnitude), they would perform as well as expected with roundoff. This seems like an ideal use for the threshold test, but as described in the section on Gaussian elimination, for example, what we need to test is if *any* entry of the input matrix is normalized, a weaker condition on the matrix, but one requiring testing the underflow flag (and resetting it as well if it is a sticky flag) for each matrix entry. If testing, or more likely resetting, involves an expensive system call, we would not want to include it in such a tight loop. Similar input constraints apply to Cholesky, iterative refinement, inner product calculations and others: we would need to test and possibly reset the underflow flag in a tight loop. If these are expensive operations, the usefulness of the threshold test is undermined. Furthermore, some of these codes satisfy a combined relative/absolute error bound independent of the input values (see conclusion (2) of § 1).

Now we discuss the situations in which the accuracy test appears more useful. In Gaussian elimination, iterative refinement, and complex divide we may use the accuracy test to test intermediate and final results for underflows we know can be harmful only if they are inaccurate. There are also the simple assignment statement $a := b$ and negation $a := -b$. If b is denormalized, *and the compiler implements these statements as floating point operations*, then the accuracy test will raise no flag, but the threshold test will. If they are implemented as fixed point operations, then of course no flags will be raised, but in the unhappily common situation where one designer designs the floating point and another the compiler, the floating point designer may have no control over the compiler design decisions. One may counter that one could just test and reset the underflow flag after assignments and negations, but if this incurs the overhead of a system call, it may not be a good solution. These examples of assignment and negation may well be the major contributor of false alarms on threshold underflow.

6. Complex arithmetic.

In order to make error analysis in complex arithmetic as similar as possible to the analysis in real arithmetic, we would like to have formulas describing the error in complex addition, subtraction, multiplication and division that are nearly identical to (1) to (5) and (3′) and (4′) which describe the error in real arithmetic.

6.1. Complex addition and subtraction.

Here the situation is most satisfying: formulas (1) to (5) and (3′) and (4′) all remain true as long as "$a \blacksquare b$ overflows" is interpreted as "overflows in either component". We repeat these formulas for completeness. In the absence of overflow or underflow we have

(12) $$\mathrm{fl}\,(a \pm b) = (a \pm b)(1+e) \quad \text{unless } a \pm b \text{ underflows or overflows.}$$

To take underflow into account, we write

(13) $$\mathrm{fl}\,(a \pm b) = (a \pm b)(1+e) + \eta \quad \text{unless } a \pm b \text{ overflows.}$$

In the case of G.U. there are the following constraints on e and η:

(14) $$|e| \leqq \varepsilon \quad \text{and} \quad \eta = 0.$$

In the case of S.Z. we have the following somewhat different constraints on e and η:

$$|e| \leqq \varepsilon \quad \text{and} \quad |\eta| \leqq \lambda. \tag{15}$$

It is not true that at most one of e and η can be nonzero, as it was with real addition.

6.2. Complex multiplication. Multiplication is not quite so satisfactory as addition and subtraction because of the possibility of intermediate overflow in the obvious algorithm:

$$(a+i\cdot b)\cdot(c+i\cdot d)=(ac-bd)+i\cdot(ad+bc)\equiv p_r+i\cdot p_i, \tag{16}$$

even though the final product may be a representable number. Since this can only happen if one of p_r or p_i is within a factor of 2 of the overflow threshold Λ anyway, we accept this slight loss of robustness since formula (16) is otherwise so satisfactory, as we now discuss.

In the absence of overflow or underflow (in the intermediate or final results)

$$\mathrm{fl}\,(a*b)=(a*b)(1+e) \tag{17}$$

where a, b, and e are all complex quantities, and $|e|<2\sqrt{2}\varepsilon$. To take underflow into account we again write

$$\mathrm{fl}\,(a*b)=(a*b)(1+e)+\eta \quad \text{in the absence of overflow.} \tag{18}$$

For G.U. we have the following constraints on e and η (to first order in ε):

$$|e|\leqq 2\sqrt{2}\varepsilon \quad \text{and} \quad |\eta|\leqq 2\sqrt{2}\lambda\varepsilon. \tag{19}$$

For S.Z. we have the following slightly different constraints:

$$|e|\leqq 2\sqrt{2}\varepsilon \quad \text{and} \quad |\eta|\leqq 2\sqrt{2}\lambda. \tag{20}$$

Thus, complex multiplication can be analyzed in the identical way as real multiplication but with slightly larger bounds on e and η.

Hence, analyses of algorithms which use only $+$, $-$, and $*$ operations (such as inner product) and the error bounds in (2) extend immediately to the complex case.

It is no longer possible to test for underflow in multiplication with S.Z. by comparing the product to zero as in real multiplication. Indeed, it is possible for a nonzero product computed with S.Z. to be wrong in the second bit in both components due to underflow. For example, consider the product of $2\sqrt{\lambda}+i\cdot 0.5\sqrt{\lambda}$ and $\sqrt{\lambda}+i\cdot\sqrt{\lambda}$. The correct product, produced with G.U., is $1.5\lambda+i\cdot 2.5\lambda$, but S.Z. delivers $2\lambda+i\cdot 2\lambda$. The underflow flag, however, may also be raised spuriously, for S.Z. or G.U., accuracy test or threshold test, even though the product is exemplary.

6.3. Complex division. This case was originally analyzed by Hough [7]. The algorithm is due to Smith and can be found in Knuth [17, p. 195] and avoids almost all unnecessary intermediate overflows in the calculation. We want to compute the quotient $(a+i\cdot b)/(c+i\cdot d)=q_r+i\cdot q_i$:

$$\begin{aligned} &\text{if } |d|<|c| \text{ then compute } q_r+i\cdot q_i=\frac{a+b(d/c)}{c+d(d/c)}+i\cdot\frac{b-a(d/c)}{c+d(d/c)},\\ &\text{else compute } q_r+i\cdot q_i=\frac{b+a(c/d)}{d+c(c/d)}+i\cdot\frac{-a+b(c/d)}{d+c(c/d)}. \end{aligned} \tag{21}$$

As with complex multiplication, it is possible to have intermediate overflows even if q_r and q_i are exactly representable, but this can only happen if either the a and b or c and d are both within a factor of 2 of Λ anyway.

If no overflows or underflows occur, then the relative error in the quotient is bounded by $7\sqrt{2}\varepsilon$, where ε is the error in the underlying arithmetic. In contrast to addition, subtraction and multiplication, however, it is not possible to bound the error in the presence of underflow simply in terms of a few units in the last place of the correct result plus a few underflow errors. If either the dividend $a+i\cdot b$ or divisor $c+i\cdot d$ is entirely denormalized, it is possible to get a normalized quotient that may be wrong in most of its places. If both dividend and divisor are normalized in at least one component, however, then with G.U. the computed quotient does indeed agree with the correct quotient to all but a few units in the last place of $|q_r+i\cdot q_i|$. With S.Z. both divisor and divided have to be at least λ/ε to be assured of the same accuracy. We write these conclusions as follows:

(22) $\quad \mathrm{fl}\,(a/b)=(a/b)^*(1+e)+\eta$ if both $|a|$ and $|b|$ are bigger than τ

where

(23) $$|e|\leqq 7\sqrt{2}\varepsilon \quad \text{for both G.U. and S.Z.}$$

and

(24) $$\tau=\lambda \quad \text{and} \quad \eta=\sqrt{2}\lambda\varepsilon \quad \text{for G.U.}$$

and

(25) $$\tau=\lambda/\varepsilon \quad \text{and} \quad \eta=\sqrt{2}\lambda \quad \text{for S.Z.}$$

Thus, when analyzing algorithms with complex division, more care must be taken than with real division to make sure the constraints given by τ above are satisfied.

Here are some examples to show what happens when the constraints given by τ are violated. We use 6 decimal arithmetic for ease of presentation. First, let $a+i\cdot b=2\lambda+i\cdot 1\lambda$ and $c+i\cdot d=4\lambda+i\cdot 2\lambda$. The correct quotient $(a+i\cdot b)/(c+i\cdot d)=.5$, but in S.Z. the term $\lambda(1/2)$ underflows to 0 and we get the quotient .4 instead of .5. With G.U. we get .5. If we now multiply both dividend and divisor by ε so they are denormalized, G.U. suffers the same fate as S.Z. and delivers .4 instead of .5.

Unfortunately, an underflow flag may be raised even though the product is very accurate. This is true for S.Z. or G.U. with either accuracy test or threshold test.

With extended precision *and* range both the multiplication and division routines can underflow (or overflow) only when storing the final results, thus avoiding all false alarms.

7. Inner product calculations. Consider the two vectors $a=(\Lambda, \lambda, 1/2, \lambda, 0)$ and $b=(0, 1/2, \lambda, 1, \Lambda)$. If we compute their inner product $\sum_{i=1,5} a_ib_i$ in the straightforward way

```
sum := 0
for $i$ := 1 to 5 do sum := sum + $a_i * b_i$
```

we get very different answers if we use G.U. than if we use S.Z. With G.U. we get the exact answers 2λ whereas with S.Z. we get λ because both a_2b_2 and a_3b_3 are less than λ and so flush to zero in a S.Z. system. The difference is large in the forward sense (λ is relatively much different than 2λ) and the backward sense as well, because it cannot be explained by saying that the result obtained from S.Z. is the exact result of a different inner product whose vector components differ from the original ones by a few units in the last place. Note also that there are no scale factors α and β such that the inner product can be calculated as

$$\frac{1}{\alpha\beta}\sum_{1=i}^{5}(\alpha a_i)(\beta b)_i$$

without underflow or overflow.

We can state the following propositions about inner products which generalize the above example [2].

PROPOSITION 1. *Let g' be a bound on the partial sums and individual terms of the inner product $\sum_{i=1}^{n} a_i b_i$:*

$$g' = \max_{1\leqq i\leqq n}\left(\mathrm{fl}\left(\sum_{j=1}^{i} a_j b_j\right), a_i b_i\right). \tag{26}$$

We can bound the error in computing $\sum_{i=1}^{n} a_i b_i$ as follows: In the absence of underflow we have

$$\left|\mathrm{fl}\left(\sum_{i=1}^{n} a_i b_i\right) - \sum_{i=1}^{n} a_i b_i\right| \leqq (2n-1)\varepsilon g \tag{27}$$

where $g = g'/(1-\varepsilon)$.

In the case of G.U. we have

$$\begin{aligned}\left|\mathrm{fl}\left(\sum_{i=1}^{n} a_i b_i\right) - \sum_{i=1}^{n} a_i b_i\right| &\leqq (n-1)\varepsilon g + n\varepsilon \max(\lambda, g)\\ &\leqq (2n-1)\varepsilon g \quad \text{if } g \geqq \lambda\end{aligned} \tag{28}$$

where $g = g'/(1-\varepsilon)$.

In the case of S.Z. we have

$$\begin{aligned}\left|\mathrm{fl}\left(\sum_{i=1}^{n} a_i b_i\right) - \sum_{i=1}^{n} a_i b_i\right| &\leqq (2n-1)\varepsilon \max\left(\frac{\lambda}{\varepsilon}, g\right)\\ &\leqq (2n-1)\varepsilon g \quad \text{if } g > \frac{\lambda}{\varepsilon}\end{aligned} \tag{29}$$

where $g = (g'+\lambda)/(1-\varepsilon)$. Note that the g used in equation (28) may differ from the g used in equation (29) because g depends on the kind of arithmetic used (G.U. or S.Z.). Also, g depends on the order of the terms $a_i b_i$.

The proof is a straightforward extension of the usual error analysis of inner products [23] using formula (2) of § 3.

The significance of this proposition is the following: (27) states the well-known result that the error in an inner product subject only to roundoff errors can be as large as about $2n$ rounding errors in the largest intermediate result g'. The second line of (28) says that the same is true for G.U. as long as the largest intermediate results g' is normalized. In particular, if the final result is normalized, then underflow is no worse than roundoff. (If g' is not normalized, then we have effectively computed the inner product in fixed point and we get only an absolute error bound from the first line of (28) as expected.) If we use the accuracy test with G.U. and the underflow flag is not raised, then (27) holds independent of the size of g. For S.Z. on the other hand (29) says that g' must exceed λ/ε for the same claim to hold. This is an example of statement (9) in § 3.

To analyze the backwards error in an inner product, we need another expression for the error.

PROPOSITION 2. *The floating point result of the inner product $\sum_{i=1}^{n} a_i b_i$ may be written*

$$\mathrm{fl}\left(\sum_{i=1}^{n} a_i b_i\right) = \sum_{i=1}^{n} a_i b_i (1+E_i) + \eta. \tag{30}$$

In the absence of underflow we have

$$\begin{aligned} |E_1| &\leqq n\varepsilon, \\ |E_j| &\leqq (n+2-j)\varepsilon \quad \text{if } j>1. \end{aligned} \tag{31}$$

In the case of G.U. we have the same bounds on the $|E_i|$, and

$$|\eta| \leqq n\lambda\varepsilon. \tag{32}$$

In the case of S.Z. we have the same bounds on the $|E_i|$, and

$$|\eta| \leqq n\lambda. \tag{33}$$

The proof is again a straightforward extension of the usual error analysis [23] using formula (2).

(31) means that in the absence of underflow, an inner product can be computed with small backwards error; in other words the computed result is the exact inner product of two vectors whose components differ by at most n rounding errors from the components of the original vectors. (32) means that with G.U., as long as some intermediate result fl $(a_i b_i)$ is normalized $(>\lambda)$, then the backwards error is also small, because η can be absorbed into the $a_i b_i(1+E_i)$ term, increasing E_i by at most $n\varepsilon$. In particular, if the final result is normalized, underflow is no worse than round off. (33) means that some intermediate term must be as large as λ/ε for a similar claim to hold for S.Z.

Of course, if we are using the accuracy test with G.U. and no flag is raised, then $\eta = 0$ and the roundoff only error bounds in (31) hold.

These two propositions may be used to extend the results of error analyses for many matrix computations to include underflow. The next three sections present the results of such analyses for Gaussian elimination, Cholesky decomposition, and iterative refinement.

8. Gaussian elimination.

8.1. Summary. The algorithm we analyze for solving the system of linear equations $Ax = b$ is a standard form of Gaussian elimination:

(1) Decompose $A = LU =$ (lower triangular) (upper triangular) using pivoting, so that the diagonal of L contains all 1's and no entries of L exceed 1 in absolute value;

(2) Solve $Ly = b$ for y (forward substitution);

(3) Solve $Ux = y$ for x (back substitution).

What kind of reliability do we expect from this algorithm in the absence of underflow? It is well-known that even though we can not expect an accurate solution if the input matrix is ill-conditioned, we can expect to get a residual $A\hat{x} - b$ ($\hat{x}$ is the computed solution) that is small in a sense made precise later. We also expect a small backwards error: $\hat{x}$ will be the exact solution of a problem slightly different from the original, again in a sense to be made precise later.

It turns out that as long as *one* component each of the matrix A and right-hand side b are normalized, then the only gradual underflows that can possibly contribute significantly to the residual or backwards error are inaccurate underflows in the final solution vector $\hat{x}$. Here we are using the accuracy test for underflow (see § 3), but our conclusions are also valid with the threshold test, though we get more false alarms.

This is a situation where the proper choice of underflow test depends on the application: if the output of the Gaussian elimination routine is input for another call to it, the user may choose the threshold test to see if he is passing normalized data to the second call as required for the conclusions above to apply. This may not be easy to do in practice, of course, but it shows that the accuracy test might not be best for all situations.

In contrast, unless one component each of the A and b is greater than λ/ε in magnitude, intermediate underflows with S.Z. during any stage of solution can introduce significant errors, possibly producing reasonable looking results whose error greatly exceeds the uncertainty attributable to roundoff alone (see the examples).

The measure of backwards error on which these conclusions depend is the following: in trying to solve $Ax=b$ we really compute $\hat{x}$ where $(A+\delta A)\hat{x}=b+\delta b$ and

$$\|\delta A\| \text{ is comparable to } \varepsilon\cdot\|A\| \tag{34a}$$

and

$$\|\delta b\| \text{ is comparable to } \varepsilon\cdot\|b\|. \tag{34b}$$

$\|A\|$ is a measure of the size of A (similarly for $\|b\|$), and "comparable to" means not larger by more than a factor $f(n)$ which is a low order polynomial in the dimension n of A (this will be made more explicit later). In other words, under the conditions stated above, Gaussian elimination has an error no larger than $f(n)$ rounding errors in the largest entry of A or b.

If we weaken our measure of backwards error in (34) and ask how much larger $\|\delta A\|$ ($\|\delta b\|$) can be than $\varepsilon\|A\|+\varepsilon\lambda$ ($\varepsilon\|b\|+\varepsilon\lambda$) instead of $\varepsilon\|A\|$ ($\varepsilon\|b\|$), then as long as the solution $\hat{x}$ itself does not underflow inaccurately, we can prove that Gaussian elimination using G.U. always has a small backwards error no matter how big $\|A\|$ or $\|b\|$ is. In other words, δA changes A (δb changes b) in the last few places of the largest entry, no matter if the largest entry is normalized or not. This robustness is not shared by S.Z.: almost all the bits in all the entries of A or b can be insignificant using S.Z. if the entries are too close to λ in size.

Gaussian elimination using G.U. with a warning of (inaccurate) underflows in the solution $\hat{x}$ appears to be a robust enough program to deserve inclusion in a library. If we insist on the traditional measure of backwards error in (34a) and (34b) above, and if we are willing to include an explicit scaling test ("are the largest entries of A and b at least λ in magnitude?") then G.U. offers no great advantage over S.Z. because changing the test threshold from λ to λ/ε makes an equally ironclad program in S.Z. with an only slightly smaller range of application, and eliminates the largest potential advantage of G.U.: making robust code faster to execute or easier to write. Note that we want to test whether *any* input component of A and b exceeds the threshold λ, and whether *any* output component of $\hat{x}$ underflows inaccurately. The most efficient test on the inputs would use a nonsticky flag based on the threshold test, since a sticky flag would have to be reset after each entry was tested. The most efficient test on the outputs would also use a nonsticky flag but be based on the accuracy test instead. Since each component $\hat{x}_i$ of $\hat{x}$ is computed in a loop with other computations, a sticky flag would have to be reset within the loop just before the last operation yielding $\hat{x}_i$.

It is very important to point out that use of extended range and precision for intermediate results does *not* invalidate the results just discussed. As long as the entries of L and $\hat{x}$ must be stored in the basic format the conclusions remain valid, because it is possible underflows in these entries that undermine the code's reliability. Thus, the conclusions of this section are as relevant to a system with just one precision and range available as to one with extended precision and range.

Section 8.2 contains examples and § 8.3 presents the theorems and offers conclusions. This material has been published before [3].

8.2. Examples of Gaussian elimination. $\|A\|_\infty(\|b\|_\infty)$ denotes the infinity norm of the matrix A (vector b):

$$\|A\|_\infty = \max_i \sum_j |A_{ij}| \quad \text{and} \quad \|b\|_\infty = \max_i |b_i|.$$

$|A|$ ($|b|$) denotes the matrix (vector) whose entries are the absolute values of the entries of $A(b)$. Inequalities like $|A| < |B|$ are meant componentwise.

We denote the usual condition number of the matrix A by

$$k(A) = \|A_\infty\| \cdot \|A^{-1}\|_\infty,$$

and a new set of condition numbers by

$$\text{Cond}\,(A, x) = \frac{\| |A^{-1}| |A| |x| \|_\infty}{\|x\|_\infty}$$

$$\text{Cond}\,(A) = \| |A^{-1}| |A| \|_\infty.$$

These new condition numbers, due to Skeel [20], will be discussed more fully below. Note Cond $(A) \geqq$ Cond (A, x) for all x.

In this section we present four examples of the effects of underflow on performing Gaussian elimination. The first example shows how store zero can produce a reasonable looking but completely inaccurate decomposition of a well conditioned matrix, whereas gradual underflow either produces the correct decomposition or correctly decides the matrix is singular. (There are no rounding errors nor pivot growth in this example.) The second example shows that G.U. produces the correct decomposition of a well conditioned matrix which S.Z. incorrectly decides is singular. Third, we present an innocuous looking ordinary differential equation and show that the linear system arising from trying to solve it numerically leads to underflow which is handled correctly by G.U. and not by S.Z. Finally, we present an example which shows that regardless of whether we use G.U. or S.Z., Gaussian elimination can only guarantee small residuals, not an accurate answer, even when the matrix A is well conditioned in the sense that Cond (A) is small.

8.2.1. *Example* 1. Consider the family of matrices $A(x)$ where

$$(35) \qquad A(x) = \lambda \cdot \begin{bmatrix} 2 & & & & 1 \\ & 2 & & & 1 \\ & & 2 & & 1 \\ & & & 2 & 1 \\ 1 & 1 & 1 & 1 & x \end{bmatrix}$$

(blanks denote zero entries). The LU decomposition obtained by G.U. is

$$(36) \quad L^{\text{G.U.}}(x) \cdot U^{\text{G.U.}}(x) = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ .5 & .5 & .5 & .5 & 1 \end{bmatrix} \cdot \lambda \cdot \begin{bmatrix} 2 & & & & 1 \\ & 2 & & & 1 \\ & & 2 & & 1 \\ & & & 2 & 1 \\ & & & & x-2 \end{bmatrix} = A(x)$$

exactly, and by S.Z. is

$$(37)\ L_{\text{S.Z.}}(x)\cdot U^{\text{S.Z.}}(x)=\begin{bmatrix}1&&&&\\&1&&&\\&&1&&\\&&&1&\\.5&.5&.5&.5&1\end{bmatrix}\cdot\lambda\cdot\begin{bmatrix}2&&&&1\\&2&&&1\\&&2&&1\\&&&2&1\\&&&&x\end{bmatrix}=A(x)+E,$$

where the error matrix E equals

$$(38)\qquad E=\lambda\cdot\begin{bmatrix}&&&&\\&&&&\\&&&&\\&&&&\\&&&&-2\end{bmatrix}.$$

We see S.Z. causes a relatively large error in the $U(x)_{55}$ entry, whereas G.U. gives the correct decomposition. When $x=2$, using S.Z. leads us to conclude that the matrix is far from singular, when in fact it is exactly singular. Note that the matrix $A(x)$ is well conditioned when x is far from 2, and if x is a smaller integer no rounding errors occur in either decomposition.

8.2.2. *Example* 2. Let

$$(39)\qquad A=\begin{bmatrix}2\lambda&3\lambda\\\lambda&2\lambda\end{bmatrix},$$

a well conditioned matrix. Using G.U. we obtain

$$(40)\qquad L^{\text{G.U.}}\cdot U^{\text{G.U.}}=\begin{bmatrix}1&\\.5&1\end{bmatrix}\cdot\begin{bmatrix}2\lambda&3\lambda\\&\lambda/2\end{bmatrix}=A,$$

but by using S.Z. we obtain

$$(41)\qquad L^{\text{S.Z.}}\cdot U^{\text{S.Z.}}=\begin{bmatrix}1&\\.5&1\end{bmatrix}\cdot\begin{bmatrix}2\lambda&3\lambda\\&0\end{bmatrix}.$$

Thus, G.U. correctly decomposes the matrix A, whereas S.Z. incorrectly makes the matrix look singular.

8.2.3. *Example* 3. Consider the ordinary differential equation

$$(42)\qquad \dot{x}(t)=\frac{1-(t/T)^M}{T-t}x(t),\qquad x(T_0)=c.$$

We try to solve this equation numerically by replacing $x(t)$ by the truncated power series $\sum_{n=1}^{N}x_nt^n$, the function $(1-(t/T)^N)/(T-t)$ by its (finite) power series, and then equating coefficients of equal powers of t on both sides of equation (42). After we scale the last row (which represents the initial condition) down to have the largest entry equal to 1, we get the linear system $Ax=b$, where

$$(43)\qquad A=\begin{bmatrix}N&-1/T&-1/T^2&\cdots&&-1/T^N\\&N-1&-1/T&\cdots&&-1/T^{N-1}\\&&N-2&\cdots&&-1/T^{N-2}\\&&&\ddots&&\vdots\\&&&&1&-1/T\\1&1/T_0&1/T_0^2&\cdots&1/T_0^{N-1}&1/T_0^N\end{bmatrix},$$

$b^T = (0, \cdots, 0, c/T_0^N)$, and $x^T = (x_N, \cdots, x_0)$.

We chose $M = 15$, $N = 14$, $T = 512.$, $T_0 = 500.$, and $c = 100.$ for this example. We used a single precision implementation of the IEEE Floating Point Standard [8] on a VAX 11/780[1] for which ε was $2^{-23} \approx 1.19_{10}-7$ and λ was $2^{-126} \approx 1.18_{10}-38$. There was a switch on the compiler to enable/disable G.U., so we were able to obtain numerical results using both G.U. and S.Z.

L and U have a simple structure. L will be zero below the diagonal, except for the last row, which is graded from $L_{15,1} \approx 7.14285_{10}-2$ down to $L_{15,14} \approx 5.34726_{10}-35$. U is identical to A in all but its last row.

$$L = \begin{bmatrix} 1 & & & & \\ 0 & 1 & & & \\ 0 & 0 & & & \\ \vdots & \vdots & & & \\ 0 & 0 & \cdots & 1 & \\ L_{15,1} & L_{15,2} & \cdots & L_{15,14} & 1 \end{bmatrix} \tag{44}$$

$$U = \begin{bmatrix} N & -1/T & -1/T^2 & \cdots & \cdot & -1/T^N \\ & N-1 & -1/T & \cdots & \cdot & -1/T^{N-1} \\ & & N-2 & \cdots & \cdot & -1/T^{N-2} \\ & & & \ddots & & \vdots \\ & & & & 1 & -1/T \\ & & & & & U_{15,15} \end{bmatrix} \tag{45}$$

A's columns are badly scaled, although this is not obvious bcause no row nor column is drastically smaller in norm than any other; nonetheless, bad scaling causes A to *appear* very ill conditioned, and this ill conditioning shows up in the last row of U, making $U_{15,15}$ very small, barely above the underflow threshold. S.Z. and G.U. compute all elements of L and U identically except for $U_{15,15}$. In fact, all additions in the computation of L add normalized numbers with like magnitudes and like signs, so no cancellation, loss of significance, nor underflows occur. If the exponent range were unbounded, so underflow never happened, the correct value $U_{15,15} \approx 2.09261_{10}-37$ would be computed. This is the value computed using G.U. But when S.Z. is used instead, the computed value is $U^{\text{S.Z.}}_{15,15} \approx 1.72763_{10}-37$, a relative difference of .174 from the correct value. All additions in the computation of $U_{15,15}$ involve numbers of like magnitude and sign, so cancellation cannot be blamed for the discrepancy. This relative difference in the last entry of U is very important, because one divides by $U_{15,15}$ in the course of solution. Thus, the computed solution $x^{\text{G.U.}}$ is very close to the true x, and the relative difference in solution vectors is

$$\frac{\|x^{\text{G.U.}} - x^{\text{S.Z.}}\|_\infty}{\|x^{\text{G.U.}}\|_\infty} \approx .211.$$

Thus, G.U. obtains markedly better results than S.Z. This example is very interesting because there is nothing obviously wrong with the matrix. All its entries are unexceptional normalized numbers, and every row and every column contains at least one number no tinier than $1/T \approx .00195$ and none larger than $N = 14$, yet 11 out of 14 products $L_{15,j} * U_{j,15}$ in the sum for $U_{15,15}$ underflow just slightly below the underflow

[1] VAX is trademark of the Digital Equipment Corporation.

threshold. Since the true value of $U_{15,15}$ is itself not much larger than the underflow threshold, this makes for a large relative error.

This example was chosen to be simple and realistic; even though it can be solved analytically, it could be changed easily into a two-dimensional problem without an explicit solution, but with the same sensitivity to underflow.

We repeat that even though A appears very ill conditioned, since $k(A) \approx 1/\lambda$ (i.e. near the overflow threshold in most arithmetics), it is also well conditioned in the sense that Cond $(A, x) \approx 5.5$. We will discuss the significance of this example later in § 8.4.

8.2.4. *Example* 4. Let

$$A = \begin{bmatrix} G & G \\ g & 2g \end{bmatrix}, \qquad A^{-1} = \begin{bmatrix} 2/G & -1/g \\ -1/G & 1/g \end{bmatrix},$$

where g/G underflows to 0 using either S.Z. or G.U. The L obtained is thus the identity matrix since $L_{2,1} = \mathrm{fl}\,(g/G) = 0$, and so the L and U obtained are the exact factors of the matrix

$$A + E = \begin{bmatrix} G & G \\ 0 & 2g \end{bmatrix},$$

which is a very different matrix than A. If $b^T = (G, 0)$, then $x = A^{-1}b = (2, -1)^T$, whereas $\hat{x} = (A+E)^{-1}b = (1, 0)^T$, so $\hat{x}$ does not resemble x at all. The residual r is however guaranteed to be small, in the sense that $\|r\|_\infty / \| |A||\hat{x}| + |b| \|_\infty$ is small:

$$\frac{\|r\|_\infty}{\| |A||\hat{x}| + |b| \|_\infty} = \frac{\|E\hat{x}\|_\infty}{\| |A||\hat{x}| + |b| \|_\infty}$$

$$\leq \frac{g|\hat{x}_1|}{G|\hat{x}_1| + G|\hat{x}_2|} \leq \frac{g}{G} \leq \lambda\varepsilon/2.$$

Of course A is an exceedingly ill conditioned matrix in the sense that $k(A) \approx 2G/g$ is beyond the reciprocal of the underflow threshold, so we would be inclined not to trust our results anyway. However, Cond (A) is only 7. This is true because Cond (A) = Cond (DA) for any nonsingular diagonal matrix D, so A has the same condition number as the utterly tame matrix

$$\begin{bmatrix} G^{-1} & \\ & g^{-1} \end{bmatrix} A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}.$$

Needless to say, in the absence of underflow we would compute a very accurate solution. We will return to this example later to explain why we can get inaccurate results from a matrix with a small condition number Cond (A).

8.3. Results of error analysis.

8.3.1. *Approach.* As stated in the introduction, we use backward error analysis. Thus, when Gaussian elimination is used to solve

$$Ax = b \tag{46}$$

for x it generates instead an approximation $\hat{x} = x + \delta x$ which satisfies some perturbed problem

$$(A + \delta A)\hat{x} = b + \delta b. \tag{47}$$

The task of backwards error analysis is to infer bounds on δA and δb from the details of the arithmetic used to implement the elimination process. These bounds can be used in turn to bound the residual

(48) $$r = A\hat{x} - b = -\delta A\hat{x} + \delta b = A\delta x$$

and then the error δx.

Wilkinson's approach [22] is to determine a bound ω_w on the errors

(49) $$\|\delta A\|_\infty \leqq \omega_w \|A\|_\infty \quad \text{and} \quad \|\delta b\|_\infty \leqq \omega_w \|b\|_\infty$$

whence

(50) $$\|r\|_\infty \leqq \omega_w[\|A\|_\infty \|\hat{x}\|_\infty + \|b\|_\infty]$$

and then it will follow that the error δx is bounded:

(51) $$\frac{\|\delta x\|_\infty}{\|x\|_\infty + \|\hat{x}\|_\infty} \leqq \omega_w k(A).$$

The detailed derivation of ω_w from the details of the arithmetic is given elsewhere [2]. Theorem 1 below states simple requirements on A and b that ensure ω_w will be scarcely worse if underflow occurs than if it does not.

Skeel's approach [20], modified slightly here, is to determine a bound ω_s on the relative error in each entry of A and b:

(52) $$|\delta A| \leqq \omega_s |A| \quad \text{and} \quad |\delta b| \leqq \omega_s |b|.$$

From these inequalities follows a bound upon the error δx:

(53) $$\frac{\|\delta x\|_\infty}{\|x\|_\infty} \leqq \omega_s \cdot \frac{\| |A^{-1}||A||x| + |A^{-1}||b| \|_\infty}{(1 - \omega_s \| |A^{-1}||A| \|_\infty) \|x\|_\infty}$$

(provided the denominator is positive). This motivates defining the following condition numbers:

(54a) $$\operatorname{Cond}(A, x) \equiv \frac{\| |A^{-1}||A||x| \|_\infty}{\|x\|_\infty},$$

(54b) $$\operatorname{Cond}(A) \equiv \| |A^{-1}||A| \|_\infty.$$

Cond (A) is an upper bound for Cond (A, x) for all x; the error bounds are useful only if ω_s Cond $(A) < 1$.

Following Oettli and Prager [19] and Skeel [20] we use an expression for ω_s obtainable from (48) in terms of the residual r:

(55) $$\omega_s = \max_i \frac{|r_i|}{(|A||\hat{x}| + |b|)_i}$$

where the max is over those i for which the denominator is nonzero. Following Skeel, we overestimate ω_s by analyzing the elimination process to infer an inequality

(56) $$\|r\|_\infty \leqq \omega_s' \| |A||\hat{x}| + |b| \|_\infty$$

from which we compute the overestimate $\bar{\omega}_s$ as

(57) $$\bar{\omega}_s = \frac{\max_i (|A||\hat{x}| + |b|)_i}{\min_i (|A||\hat{x}| + |b|)_i} \omega_s'$$

(where the min in the denominator is over the nonzero values of $(|A||\hat{x}|)_i$ only). Unfortunately $\bar{\omega}_s$ can be a gross overestimate of ω_s, as we will see when we return to Example 3 later.

The detailed derivation of ω_s' is given in [2]. Theorem 2 below states requirements on A and b that ensure ω_s' will be scarcely worse if underflow occurs than if it does not. These requirements on A and b are nearly identical to the requirements in the Wilkinson style analysis.

8.3.2. *Results.*

THEOREM 1. *Wilkinson style error analysis of solving $Ax = b$ with Gaussian elimination in the presence of underflow: Let $a_{\max} = \max_{ij} |A_{ij}|$, and $g =$ [largest intermediate result appearing in the decomposition]$/a_{\max}$. g is the "pivot growth factor" and is $\leqq 2^{n-1}$.*

Then a bound ω_w for which

$$\|r\|_\infty \leqq \omega_w[\|A\|_\infty \|\hat{x}\|_\infty + \|b\|_\infty] \tag{50}$$

is given as follows. In the absence of underflow, we have

$$\omega_w = n^3 \varepsilon g/2. \tag{58}$$

If underflow occurs then

$$\omega_w = 3n^3 \varepsilon g/2 \tag{59}$$

provided certain conditions are met. For G.U. these conditions are:

(60)

$ga_{\max} \geqq \lambda$	*if there are any underflows during triangular decomposition,*
$\|b\|_\infty \geqq \dfrac{\lambda}{n}$	*if there are any intermediate underflows during forward and back substitutions,*
$\dfrac{\|b\|_\infty}{a_{\max}} \geqq \dfrac{2\lambda}{n^2}$	*if the solution $\hat{x}$ itself underflows in some component.*

For S.Z. the above conditions still apply but λ must be increased to λ/ε.

Proof. See [2].

THEOREM 2. *Skeel style error analysis of solving $Ax = b$ with Gaussian elimination in the presence of underflow: Let $a_j = \max_i |A_{ij}|$, and $g_C = \max_j$ ([largest intermediate result appearing in the decomposition in column j]$/a_j$). g_C is the "columnwise pivot growth factor" and is $\leqq 2^{n-1}$.*

Then a bound ω_s' for which

$$\|r\|_\infty \leqq \omega_s' \| |A| |\hat{x}| + |b| \|_\infty \tag{56}$$

is given as follows. In the absence of underflow we have

$$\omega_s' = n^3 \varepsilon g_C. \tag{61}$$

If underflow occurs, then

$$\omega_s' = 3n^3 \varepsilon g_C/2 \tag{62}$$

provided certain conditions are met. For G.U. *these conditions are:*

(63)

$g_C a_j \geqq \lambda$	*for all j, if there are any underflows during triangular decomposition,*
$\|b\|_\infty \geqq \dfrac{\lambda}{2n}$	*if there are any intermediate underflows during forward and back substitutions,*
$\dfrac{\|b\|_\infty}{a_{\max}} \geqq \dfrac{\lambda}{n^2}$	*if the solution $\hat{x}$ itself underflows in some component.*

For S.Z. *the above conditions apply with except λ must be increased to λ/ε.*

Proof. See [2].

The theorems indicate how to write software that will solve $Ax = b$ reliably despite underflow, and how the requirements for G.U. differ from those for S.Z. To keep the residual small in the sense of a Wilkinson style error analysis, we appeal to Theorem 1. With G.U., as long as one normalized number appears during the decomposition ($ga_{\max} \geqq \lambda$), residual with underflow has a bound not much worse than residual without underflow. If there are intermediate underflows while solving the triangular systems, as long as some component of b is normalized ($\|b\|_\infty \geqq \lambda$), residual with underflow has a bound scarcely worse than without underflow. If the answer $\hat{x}$ itself underflows, we can either issue an error message (which would be very reasonable since the first goal of reliable software is only to compute an answer if it is representable) or test to see if $\|b\|_\infty/a_{\max}$ is not too small.

All these requirements are natural ones to make, since they say that when a problem's inputs and its computed solution are normalized numbers, we should expect the residual to be scarcely worse with underflow than without. Thus, the only gradual underflows which can cause concern in a problem with normalized inputs are underflows in the solution itself. The scaling condition $\|b\|_\infty/a_{\max} \geqq \lambda/n^2$ arises naturally; consider solving the scalar equation $ax = b$ by the division $x = b/a$.

In contrast, the bounds for S.Z. are all higher by a factor of $1/\varepsilon$. Thus, using S.Z. we can neither solve as many problems as the G.U., nor decide so easily which underflows matter. Thus, from the point of view of a Wilkinson style error analysis, G.U. makes writing reliable software easier.

Theorem 2 shows that Skeel style bounds for the residual are scarcely worse with underflow than without provided conditions are satisfied that are almost the same as in Theorem 1. Therefore the previous paragraphs' comments remain valid provided, when underflow is gradual, at least one normalized number appears in each column of A, rather than just somewhere is A, before or during the decomposition process.

8.4. Examples 3 and 4 revisited. We wish to emphasize that we have only derived conditions under which with underflow are about the same as without underflow. There is no way using this analysis to say how closely this bound will be approached with and without underflow, or how accurate the computed solution will be.

In Example 4 above, the matrix A and vector b satisfy all the conditions of Theorems 1 and 2 for G.U. as well as S.Z., so the residual is small, but the answer $\hat{x}$ is totally inaccurate. This inaccuracy can be explained either by the huge condition number $k(A) \approx$ overflow threshold, or the large backwards error in equation (55): $\omega_s = 1$. In this case ω_s's upper bound $\bar{\omega}_s$ in (57) is also 1. Thus, having a small value of Cond (A) is not sufficient to guarantee accuracy given a small residual ω_s' ((56)), although a small value of $k(A)$ combined with a small residual ω_w is enough, as can be seen from (51).

Example 3 is another case where the conditions of Theorems 1 and 2 hold, but now G.U. successfully computes the last pivot $U_{15,15}$ and an accurate solution $\hat{x}$ while S.Z. does not. Again, we have a problem where $k(A)$ is huge and Cond (A, x) is small. Now the ω_s of equation (55) is $\approx 5.23_{10}-8$, verifying the high accuracy of solution. Unfortunately the bad scaling of the matrix causes the upper bound $\bar{\omega}_s$ of equation (57) to be $2.0_{10}20$. This example demonstrates the occasionally intense pessimism of Skeel's approach.

In summary, the significance of Examples 3 and 4 is to show that maintaining a small residual in the face of underflow does not guarantee an accurate solution $\hat{x}$, although we conjecture that for not terribly ill conditioned matrices G.U. will provide answers at least as accurate as provided by S.Z.

We have proven something quite unremarkable: if underflows are gradual, then we continue to get what we have come to expect from Gaussian elimination. That is, we get a small residual as long as the inputs and outputs are all representable (normalized) numbers and there is no indication of singularity or excessive pivot growth. If, however, underflows are handled in the usual way and set to zero, then no such simple guarantee can be made, and some kind of testing on the scaling of the problem is necessary. These results demonstrate that gradual underflow makes it easier to write reliable linear equation solvers than "store zero."

9. Cholesky decomposition.

9.1. Summary. The algorithm we discuss is analogous to Gaussian elimination, but is applicable only to positive definite symmetric matrices A:

(1) Decompose $A = LL^T$ where L is lower triangular;

(2) Solve $Ly = b$ for y (forward substitution);

(3) Solve $L^Tx = y$ for x (backward substitution).

We expect the same kind of reliability from this algorithm in the absence of underflow as we do from Gaussian elimination: a small residual $A\hat{x} - b$ where $\hat{x}$ is the computed solution, and that $\hat{x}$ is the exact solution of a slightly different problem than the original.

With G.U., as long as *one* component each of the matrix A and right-hand side b are normalized the only harmful underflows are underflows in components of x and y (recall that with Gaussian elimination the only harmful underflows were in the solution x). Intermediate gradual underflows contribute an error with a bound scarcely worse than the bound for the error contributed by roundoff alone. As with Gaussian elimination, the accuracy test for underflow (see § 3) leads to fewer false alarms than the threshold test, although the threshold test might make it easier to test the inputs to the Cholesky routine ("are the largest components of A and b at least λ in magnitude?") for the applicability of this analysis.

In contrast, with S.Z. intermediate underflows during any stage of solution can introduce significant errors, possibly producing reasonable looking results whose error greatly exceeds the uncertainty attributable to roundoff alone (see the examples). In fact, one can show that S.Z. can only produce a decomposition of a matrix when G.U. fails if the matrix is so ill conditioned that the computed solution cannot be trusted, or if it is not positive definite at all (see § 9.2.2).

As with Gaussian elimination, the results of this section remain true even if intermediate products are computed to extra range and precision, as long as the entries of L, y and $\hat{x}$ are stored in the range and precision of A and b.

Section 9.2 contains examples and § 9.3 contains theorems and conclusions. Proofs of these results can be found in [2].

9.2. Examples.

9.2.1. *Example* 1. Let m be the smallest floating point number $\geqq \sqrt{\lambda}$, so that m^2 does not underflow. Consider the family of symmetric matrices:

$$A(x) = m^2 \cdot \begin{bmatrix} 4 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & x \end{bmatrix}$$

which has the exact lower triangular factor

$$L(x)=m\cdot\begin{bmatrix}2 & & \\ 1 & 1 & \\ .5 & .5 & \sqrt{x-.5}\end{bmatrix}.$$

$L^{\text{G.U.}}(x)$, the factor provided by Cholesky using G.U., is the same as $L(x)$ except for the rounding error incurred by having to represent $(x-.5)^{1/2}$. $L^{\text{S.Z.}}(x)$, the factor provided by S.Z., is

$$L^{\text{S.Z.}}(x)=m\cdot\begin{bmatrix}2 & & \\ 1 & 1 & \\ .5 & 1 & L_{33}^{\text{S.Z.}}(x)\end{bmatrix}$$

where

$$L_{33}^{\text{S.Z.}}(x)=\begin{cases}\sqrt{x-1} & \text{if } x\geqq 2,\\ 0 & \text{if } 2>x\geqq 1\end{cases}$$

so S.Z. computes a totally wrong value for $L_{33}(x)$, incorrectly labelling the matrix singular for $2>x\geqq 1$ when in fact it is well conditioned.

9.2.2. *Example* 2. Let m be as before. Consider the family of matrices

$$A(x)=m^2\cdot\begin{bmatrix}4 & & & & 1\\ & 4 & & & 1\\ & & 4 & & 1\\ & & & 4 & 1\\ 1 & 1 & 1 & 1 & x\end{bmatrix}.$$

Its correct factor $L(x)$, if it exists, is

$$L(x)=m\cdot\begin{bmatrix}2 & & & & \\ & 2 & & & \\ & & 2 & & \\ & & & 2 & \\ .5 & .5 & .5 & .5 & \sqrt{x-2}\end{bmatrix}.$$

This matrix is positive definite if $x>2$, positive semidefinite if $x=2$, and has both positive and negative eigenvalues if $x<2$. Both G.U. and S.Z. compute all entries of the factor $L(x)$ except the (5,5) entry correctly (using Cholesky decomposition). G.U. obtains the correct value $(x-2)m^2$ for its value of L_{55}^2, whereas S.Z. computes xm^2. Thus, as x decreases from 3 to 2 to 1, G.U. correctly decides the matrix is positive definite when $x=3$, and becomes nonpositive definite when $x\leqq 2$. S.Z., on the other hand, produces an (incorrect) decomposition all the way down to $x=1$. Thus, S.Z. cannot only produce an inaccurate decomposition, but produces it after G.U. has correctly decided no such decomposition exists.

S.Z. can produce a decomposition of a matrix when G.U. fails only if the matrix is either 1) so ill conditioned that the decomposition cannot be trusted, or 2) not positive definite at all. Here is the reason. Assume $a_{\max}\geqq\lambda$, since otherwise the matrix is identically 0 in S.Z. arithmetic. G.U. fails when its computed value of L_{jj}^2 either rounds to 0 or is negative for some j. L_{jj}^2 rounds to 0 when $L_{jj}^2<\lambda\varepsilon$. It is easy to see that $a_{\max} \leqq \lambda_{\max}(A)$ and $L_{jj}^2 \geqq \lambda_{\min}(A)$, because

$$\frac{1}{\min_j L_{jj}^2} = (\lambda_{\max}(L^{-1}))^2 \leqq |L^{-1}|_2^2 = |A^{-1}|_2 = \frac{1}{\lambda_{\min}(A)}.$$

Therefore

$$k_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}} > \frac{a_{\max}}{L_{jj}^2} > \frac{1}{\varepsilon},$$

which means that the matrix is so ill conditioned as to make it difficult to even recognize an accurate inverse, let alone compute one. If L_{jj}^2 is in fact negative, the matrix is not positive definite.

9.3. Results of error analysis.

9.3.1. *Approach.* Our approach is essentially identical to the one we used to analyze Gaussian elimination with the following additions. The Cholesky decomposition uses the square root operation which Gaussian elimination does not. We model the error in square root as follows:

$$\text{SQRT}(x) = \sqrt{x} \cdot (1+e) \text{ for all } x \tag{64}$$

where $|e| < \varepsilon$. (SQRT denotes the floating point square root.) (64) holds because SQRT compresses the exponent range, making overflow and underflow impossible. We make an extra assumption about λ and ε we did not need before; it also arises from the use of square roots in the Cholesky decomposition. This relationship is satisfied by all single precision arithmetics known to the author (but not by a number of double precision arithmetics, such as D format on the VAX, for example) and is only needed to analyze Cholesky decomposition using S.Z.: $\lambda < \varepsilon^3$.

9.4. Results.

THEOREM 3. *Wilkinson style error analysis of solving $Ax = b$ with Cholesky Decomposition in the presence of underflow. Let $a_{\max} = \max_{ij} |A_{ij}|$. Then a bound ω_w for which*

$$\|r\|_\infty \leqq \omega_w[\|A\|_\infty \|\hat{x}\|_\infty + \|b\|_\infty] \tag{50}$$

is given as follows. In the absence of underflow, we have

$$\omega_w = n^3\varepsilon/2. \tag{65}$$

If underflow occurs then

$$\omega_w = 4n^3\varepsilon/2 \tag{66}$$

provided certain conditions are met. For G.U these conditions are:

$$
\begin{array}{lll}
 & a_{\max} \geqq \lambda & \textit{if there are any underflow during Cholesky decomposition,} \\[2ex]
 & \|b\|_\infty \geqq \dfrac{2\lambda}{n^2} & \textit{if there are any intermediate underflow during forward substitution,} \\[2ex]
(67) & \dfrac{\|b\|_\infty}{\sqrt{a_{\max}}} \geqq \dfrac{\lambda}{n} & \textit{if some } y_i \textit{ underflow or there are any intermediate underflows during back substitution,} \\[2ex]
 & \dfrac{\|b\|_\infty}{a_{\max}} \geqq \dfrac{2\lambda}{n^2} & \textit{if the solution } \hat{x} \textit{ itself underflows in some component.}
\end{array}
$$

For S.Z. the above conditions still apply but λ must be increased to λ/ε.

Proof. See [10].

The above theorem shows how to write software that will solve $Ax = b$ with Cholesky reliably despite underflow just as Theorems 1 and 2 in § 8.3.2 did for Gaussian elimination. With G.U., as long as there is one normalized component in A ($a_{\max} > \lambda$) residual with underflow has a bound scarcely worse than without underflow. If there are intermediate underflows during forward substitution, the residual bound is again scarcely worse than without underflow as long as some component of b is normalized ($\|b\|_\infty \geqq \lambda$). Intermediate underflows during back substitution or in y require a scaling condition ($\|b\|_\infty/a_{\max} \geqq \lambda/n$) to be satisfied, as do underflows in the final solution ($\|b\|_\infty/a_{\max} \geqq 2\lambda/n^2$). It is clear that some such scaling condition needs to be satisfied from considering the $n = 1$ case (i.e. solving the scalar equation $ax = b$ by two divisions $x = (b/\sqrt{a})/\sqrt{a}$). If there are underflows in the back substitution, y, or x, then we can either issue an error message or check the scaling.

For S.Z. all the bounds are higher than the ones for G.U. by a factor of $1/\varepsilon$.

The situation with Cholesky is not as satisfactory as for Gaussian elimination, where only underflows in the final solution x could matter for G.U.

10. Iterative refinement. We study the following algorithm for refining the solution of the linear system $Ax = b$. The phrase "in precision (ε, λ)" means that particular computation is to be done in arithmetic with rounding error ε and underflow threshold λ. x_0 is an arbitrary starting vector.

```
$i := 0$
repeat
    $r_i := Ax_i - b$ in precision $(\varepsilon_r, \lambda_r)$
    solve $Ad_i = r_i$ for $d_i$ in precision $(\varepsilon, \lambda)$
    $x_{i+1} := x_i - d_i$ in precision $(\varepsilon, \lambda)$
    $i := i + 1$
until convergence.
```

Double precision computation of the residual (the traditional algorithm) corresponds to $\varepsilon_r = \varepsilon^2$, and single precision to $\varepsilon_r = \varepsilon$. We also assume $\lambda_r \leqq \lambda$.

In order to understand the effects of underflow on this algorithm, we need a theorem due to Skeel [21] which shows, contrary to popular belief, that computing r_i in single precision ($\varepsilon_r = \varepsilon$) does improve the solution in a significant way.

THEOREM 4. *Analysis of iterative refinement in the absence of underflow for both single and double precision computation of the residual: As long as the condition number* Cond $(A) = \| |A^{-1}||A| \|_\infty$ *is sufficiently less then $1/\varepsilon$, then*

1) *If $\varepsilon_r = \varepsilon^2$ (double precision residual computation) then*

$$\limsup_{i\to\infty} \|x - x_i\|_\infty \leqq 2\varepsilon \|x\|_\infty \tag{68}$$

where x denotes the exact solution;

2) *If $\varepsilon_r = \varepsilon$ (single precision residual computation) then*

$$\limsup_{i\to\infty} |Ax_i - b| \leqq 4n\varepsilon |A| |x_i|. \tag{69}$$

Furthermore, this inequality is almost always attained after just one application of iterative refinement.

Proof. See [21].

This last inequality means that for large enough i, x_i is the solution of a slightly perturbed problem

$$(A+\delta A)x_i = b$$

where $|\delta A_{ij}| < 4n\varepsilon|A_{ij}|$. In other words, the perturbed problem agrees with the original problem up to a few rounding errors *in each component* [19]. This is a very strong notion of backwards error, and so Skeel's theorem shows that single precision iterative refinement does lead to a significantly more reliable code than no refinement at all.

How does underflow effect this reliability? For G.U., we can say the following: If the inputs A and b and the output x are normalized and if either double or single precision residuals are computed, then gradual underflows can degrade the algorithm's performance to the level of single precision residual computation but no worse. To guarantee double precision performance, both b and x need to exceed λ/ε. Specifically, it is underflow in $r_i = Ax_i - b$ that contributes to the lower bound on b and underflow in d_i that contributes to the lower bound in x. Using this information, the accuracy test for G.U. could be used to decide when underflow might degrade the performance more precisely than the threshold test. For S.Z., all thresholds are increased by $1/\varepsilon$.

The use of extended range and precision in intermediate computations does not change these conclusions. Assuming r_i and d_i are stored in the same format as A, b and x, underflows in r_i and d_i have the same potential effects on performance as they did when they were not computed in extended range.

We have not yet considered underflow's effect on the *rate* of convergence of the iteration. There are matrices for which the iteration converges only if underflows do not occur, but the matrices are so ill conditioned as to make the computed solution untrustworthy anyway. It follows from the analysis of § 8 that as long as some entry of A is large enough (λ for G.U. and λ/ε for S.Z.) then underflows will have an effect on the convergence rate comparable to round-off.

11. Polynomial evaluation and root finding.

11.1. Horner's rule for polynomial evaluation. We consider Horner's rule for evaluating the polynomial $\sum_{i=0}^n a_i x^i$ for real a_i and x:

(70)
```
sum := $a_n$
for $i := n-1$ to 0 do sum := sum*$x + a_i$.
```

We have the following very satisfying theorem.

THEOREM 5 (Analysis of Horner's rule for polynomial evaluation). *Let P denote the result of applying Horner's rule to the polynomial* $\sum a_i x^i$ *above. Then in the absence of underflow and overflow we have*

$$P = \sum_{i=0}^{n} a_i(1+E_i)x^i \tag{71}$$

where

$$|E_n| \leqq 2n\varepsilon \quad \textit{and} \quad |E_i| \leqq (2i+1)\varepsilon \quad \textit{if } i<n. \tag{72}$$

In the presence of underflow we write

$$P=\sum_{i=0}^{n}(a_i+\eta_i)(1+E_i)x^i \tag{73}$$

where E_i has the same bound as in (72), $\eta_n=0$ *for both* G.U. *and* S.Z., *and*

$$|\eta_i|\leqq\lambda\varepsilon \textit{ for } \text{G.U. } \textit{and } |\eta_i|\leqq 2\lambda \textit{ for } \text{S.Z.} \tag{74}$$

for $i<n$.

The proof is a straightforward extension of the usual error analysis of Horner's rule [23] using formula (2) of § 3.

Thus, in the absence of underflow and overflow, Horner's rule delivers the exact value of a new polynomial each coefficient a_i of which differs by a few rounding errors from the corresponding original a_i. This is a strong backwards error bound.

For G.U., we can make the same kind of statement providing we define backwards error as motivated by the last paragraph of § 3: a relative error no greater than ε for values $>\lambda$ and an absolute error no greater than $\lambda\varepsilon$ for smaller values. Thus, for example, we treat the value 0 as indistinguishable from any value in the interval $[-\lambda\varepsilon/2, \lambda\varepsilon/2]$. By this definition of backwards error, Horner's rule with G.U. delivers the exact value of a new polynomial each of whose coefficients differs by a small relative/absolute error from the corresponding original coefficient. We can further guarantee each new coefficient has a small *relative* error with respect to the original if each a_i is a *nonzero* normalized number.

For S.Z. all thresholds in the last paragraph increase by $1/\varepsilon$ to be able to make corresponding statements.

Here, extended range and precision is extremely beneficial, eliminating most concerns about over/underflow. Indeed, any overflows in extended range would have occurred with the original range, and any underflows in extended range would contribute an uncertainty far less than a unit in the last place of even the smallest denormalized number to any a_i.

11.2. Polynomial root finding. Linnainmaa [18] has analyzed Newton's method for root finding and shown that it is much easier to write an underflow/overflow proof code if G.U. is available than if it is not. An essential feature of his code is evaluating $\sum a_{n-i}z^i$ at $z=1/x$ instead of $\sum a_ix^i$ when $x>1$. This changes almost all potential overflow problems to underflow problems, which are handled by G.U. The advantage of evaluating polynomials at points $x<1$ is that any rounding or underflow errors made early in Horner's recurrence are multiplied down by factors of x. In particular, underflow errors, already at the level of roundoff in the smallest normalized number, only decrease in significance so that if the final value P is normalized we know that any gradual underflows must be completely harmless.

12. Computing eigenvalues of symmetric tridiagonal matrices.

Given the symmetric tridiagonal matrix:

$$T=\begin{bmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & & & & \\ & & & b_n & a_n \end{bmatrix}, \tag{75}$$

how do we compute its eigenvalues? One way is to use the following program which, given a real value z, computes (in exact arithmetic) $v(z)=$ the number of eigenvalues of T that are $<z$:

(76)
```
$u := 1$
$v := 0$
for $j := 1$ to $n$ do
    $u := a_j - z - (b_j/u)b_j$
    if $u < 0$ then $v := v + 1$,
```

where we define $b_1 \equiv 0$. We assume $b_i \neq 0$ for $i > 1$, since otherwise T is block diagonal and its eigenvalues are those of its diagonal blocks. We also use the conventions $\pm 1/0 = \pm\infty$ and $1/\pm\infty = 0$ (which are part of the proposed IEEE floating point standard). A proof that this algorithm computes what we claim is based on Sylvester's inertia theorem and can be found in [5]. It can be used to obtain eigenvalues to any desired accuracy by bisecting an interval in which $v(z)$ increases (which means the interval contains an eigenvalue) until the interval is narrow enough.

What does this algorithm compute when implemented in floating point? There are two interesting questions:

Is $v(z)$ a monotone increasing function of z as it is in exact arithmetic?

Do we compute accurate eigenvalues either of our original matrix or a matrix very close to our original matrix?

In the absence of overflow and underflow, the answer to both questions is yes [11]:

The function $v(z)$ computed by algorithm (76) in the absence of overflow and underflow is an increasing function of z. Furthermore, the value of $v(z)$ computed is the exact value of $v(z)$ for a matrix T' whose diagonal entries a_i' are identical to the diagonal entries a_i of T, and whose off diagonal entries b_i' satisfy $b_i' = b_i(1+e_i)$ where $|e_i| \leqq 2\varepsilon$. T' will in general depend on z.

This is a very strong backwards error bound. It says we can compute the exact number of eigenvalues less than z of a matrix differing from the original by a small relative error in the off diagonal entries, and with no difference on the diagonal.

What can be said in the presence of underflow? Barring overflow, $v(z)$ remains monotonic using either S.Z. or G.U. The only property of the arithmetic needed to prove $v(z)$ monotonic is monotonicity of the arithmetic: if $a \geqq b$ are the exact results of two different arithmetic operations, then $\mathrm{fl}(a)$ must be $\geqq \mathrm{fl}(b)$ as well.

The monotonicity of $v(z)$ is an appealing property but not necessary for the correct functioning of a bisection algorithm for determining one eigenvalue [22]. Lack of monotonicity could lead to lower bounds exceeding upper bounds in codes for determining such bounds for all eigenvalues at once, but since $v(z)$ is monotonic, we will not discuss this possibility further.

Kahan [11] discusses an ironclad version of (76) which scales the problem and inserts tests against carefully chosen thresholds into the inner loop to guarantee that overflow and underflow (G.U. or S.Z.) cannot degrade the results appreciable more than roundoff. Here, we discuss the robustness of the unadorned code in (76) which differs from the most obvious algorithm only in using $(b_i/u)b_i$ in the inner loop instead of b_i^2/u. At the end we will say why this change is important. We assume we have a balanced exponent range, i.e. $\lambda\Lambda$ cannot be larger than a small integer m ($m=4$ in the proposed IEEE standard). The backwards error in (76) is given as follows:

The function $v(z)$ computed by algorithm (76) is the exact value of $v(z)$ for a matrix T' whose entries a_i' and b_i' satisfy:

$$
(77)\qquad
\begin{aligned}
a_i' &= a_i + \eta_i \quad \text{where } |\eta_i| \leqq (1+m)\lambda\varepsilon \\
b_i' &= b_i(1+e_i) \quad \text{where } |e_i| \leqq 2\varepsilon
\end{aligned}
$$

when using G.U., and

$$(78)\quad \begin{aligned} a_i' &= a_i + \eta_i \quad \text{where } |\eta_i| \leqq (3+m)\lambda \\ b_i' &= b_i(1+e_i) \quad \text{where } |e_i| \leqq 2\varepsilon \end{aligned}$$

when using S.Z.

Thus, in order to claim that we are computing the exact $v(z)$ for a matrix T' which differs from T by at most a few rounding errors in each component, which is the case in the absence of underflow, we need to make the following constraints on a_i:

$$|a_i| \geqq \begin{cases} \lambda & \text{for G.U.,} \\ \lambda/\varepsilon & \text{for S.Z.} \end{cases}$$

If we adopt the relative/absolute error measure suggested in the last paragraph of § 3 and discussed further in § 11.1 in connection with polynomial evaluation, then there is no constraint at all on the a_i if we ue G.U. in order to claim that a_i' differs from a_i by a small error.

These backwards error bounds are so strong that it does not seem the accuracy test for G.U. could be of much more use than the threshold test, if indeed it is of any use at all.

A weaker form of backwards error often used in analyses of matrix computations [22] is

$$(79)\quad \frac{\max_{i,j} |T_{ij}' - T_{ij}|}{\max_{i,j} |T_{ij}|}.$$

With respect to this definition, underflow is insignificant if

$$\max_{i,j} |T_{ij}| \geqq \begin{cases} \lambda & \text{for G.U.,} \\ \lambda/\varepsilon & \text{for S.Z.} \end{cases}$$

What would happen if we used b_i^2/u instead of $(b_i/u)b_i$ in the inner loop? In that case, any $|b_i|$ smaller than $\sqrt{\varepsilon\lambda} \gg \lambda$ would underflow to zero when squared whether we used G.U. or S.Z., and the resulting perturbation could not always be explained as a small change in either b_i or a_i. Thus, a seemingly small change in the code effects the robustness a great deal.

If extended range and precision are available, then almost all concerns with over/underflow vanish, as with Horner's rule for polynomial evaluation.

13. Numerical quadrature. Quadrature, along with the matrix algorithms discussed earlier, benefits from the ability to compute inner products more robustly with G.U. than S.Z. This is because most quadrature codes, when asked to compute

$$(80)\quad \int_a^{a+h} w(x)f(x)\,dx$$

evaluate an inner product

$$(81)\quad h\cdot \sum_{i=1}^{n} w_n f(x_n).$$

From the analysis of inner products in § 6, we see that as long as the inner product in (81) is a normalized number, the effects of gradual underflows are no worse than roundoff, but that some intermediate result in the inner product must exceed λ/ε to make the same claim about S.Z. All the benefits of extended range and precision to inner products also accrue to numerical quadrature. A more detailed analysis can be found in [14].

14. Accelerating the convergence of sequences. Methods to accelerate convergence of sequences often do so by extrapolating an estimated error to zero. This requires taking the ratio of differences of successive elements in the sequence. If the sequence is converging to a value near the underflow threshold, these differences can underflow to zero using S.Z. but not G.U. We illustrate with Aitken's δ^2 method.

Given a sequence $\{x_n\}$ which converges to a finite nonzero x, Aitken's δ^2 method produces a new sequence $\{x'_n\}$

$$x'_n = x_n - \left(\frac{x_{n+1}-x_n}{(x_{n+2}-x_{n+1})-(x_{n+1}-x_n)}\right)(x_{n+1}-x_n) \tag{82}$$

which will converge to x faster than $\{x_n\}$ under certain conditions [9]. We have written the term following x in (82) (the correction term) as it appears instead of as in

$$x'_n = x_n - \frac{(x_{n+1}-x_n)^2}{x_{n+2}-2x_{n+1}+x_n} \tag{83}$$

because of the latter's much greater susceptibility to over/underflow. (83) is likely to cause over/underflow if $|x|$ is much outside the range $[\sqrt{\lambda}, \sqrt{\Lambda}]$. (82) is much more robust. In fact, if N is large enough so that

$$\frac{1}{\sqrt{2}} < \frac{|x_n|}{|x|} < \sqrt{2}$$

for $n > N$ and we use G.U., then the correction term in (82) will be computed to within 2 rounding errors in x if $\lambda \leqq |x| \leqq \Lambda$ and to within $\pm\lambda\varepsilon$ if $|x| < \lambda$. In contrast, $|x|$ must exceed λ/ε to make the same claim for S.Z. The use of extended range and precision would not make S.Z.'s disadvantages disappear, since if $|x|$ is very close to λ, the correction term, even if calculated to extra precision, may make x'_n underflow.
A more detailed analysis can be found in [14].

15. Acknowledgments. The results in this paper are the culmination of several years of discussions among all the members of the P754 Floating Point Subcommittee, not just the author and the four others mentioned in the introduction, so thanks are due them and their various institutions which supported their participation in the committee. The author wishes to acknowledge Prof. W. Kahan in particular for his suggestions and comments.

REFERENCES

[1] J. T. Coonen, *Underflow and the denormalized numbers*, Computer, 14 (1981), pp. 75–87.

[2] J. Demmel, *Effects of underflow on solving linear systems*, Computer Science Division, Univ. California, Berkeley, 1980.

[3] ———, *Effects of underflow on solving linear systems*, Fifth Symposium on Computer Arithmetic, Ann Arbor, MI, May 18–19, 1981.

[4] R. A. Fraley and J. S. Walther, *A proposed standard for binary floating point arithmetic: Alternate 3*, IEEE Floating Point Subcommittee Working Document P754/80-1.24, 1980.

[5] F. R. Gantmacher, *The Theory of Matrices*, trans. K. A. Hirsch, Chelsea, New York, 1959.

[6] J. B. Gosling, J. H. P. Zurawski and D. B. G. Edwards, *A chip-set for a high-speed low-cost floating-point unit*, Fifth Symposium on Computer Arithmetic, Ann Arbor, MI, May 18–19, 1981.

[7] D. Hough, *Errors and error bounds*, IEEE Floating Point Subcommittee Working Document P754/80-3.2, 1980.

[8] *A proposed standard for binary floating point arithmetic*, Draft 10.0 of IEEE Task P754, December 2, 1982.

[9] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, John Wiley, New York, 1966.

[10] W. Kahan, 7094-II *system support for numerical analysis*, SHARE Secretarial Distribution SSD-159, Item C4537, 1966.

[11] ———, *Accurate eigenvalues of a symmetric tridiagonal matrix*, Technical Report no. CS41, Computer Science Dept., Stanford University, Stanford, CA, 1966.

[12] ———, *A Survey of Error Analysis*, in Information Processing 71, North-Holland, Amsterdam, 1972, pp. 1214–1239.

[13] W. Kahan and J. Palmer, *On a proposed floating point standard*, SIGNUM Newsletter, Special Issue, October 1979.

[14] W. Kahan, *Aitken's extrapolation and Gaussian quadrature*, IEEE Floating Point Subcommittee Working Document P754/80-2.23, 1980.

[15] ———, *Why do we need a floating point arithmetic standard?*, IEEE Floating Point Subcommittee Working Document P754/81-2.8, 1981.

[16] ———, *Three questions about underflow*, IEEE Floating Point Subcommitee Working Document P754/82-7.6, 1982.

[17] D. Knuth, *The Art of Computer Programming, Vol.* 2, Addison-Wesley, Reading, MA, 1969.

[18] S. Linnainmaa, *Combatting the effects of underflow and overflow in determining real roots of polynomials*, IEEE Floating Point Subcommittee Working Document P754/80-2.23, 1980.

[19] W. Oettli and W. Prager, *Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides*, Numer. Math., 6 (1964), pp. 405–409.

[20] R. D. Skeel, *Scaling for numerical stability in Gaussian elimination*, J. Assoc. Comput. Mach., 26, (1979), pp. 494–526.

[21] ———, *Iterative refinement implies numerical stability for Gaussian elimination*, Dept. Computer Science Report, Univ. Illinois, Urbana, 1979.

[22] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

[23] ———, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

A COMPARISON OF SEVERAL FORMULAS ON LIGHTLY DAMPED, OSCILLATORY PROBLEMS*

C. A. ADDISON†

Abstract. We examine members of three different formula families in an attempt to find a set of formulas that could be the basis of a code suitable for lightly damped, oscillatory problems. We first obtain estimates for the number of arithmetic operations per integration step required by a member of each family and then use these estimates to help us compare the performance of several trial codes, which are based on certain members from each family.

Key words. ordinary differential equations, lightly damped, oscillatory

1. Introduction.

Consider the first order linear ordinary differential equation (ODE)

$$y' = Ay + R(t), \qquad y(a) = y_0 \quad \text{for } t \in [a, b], \tag{1.1}$$

where A is a constant n by n matrix and $R(t)$ is a forcing term. We assume that (1.1) is a stable system, in the sense that the eigenvalues of A have negative real part, but that some of the eigenvalues could have large imaginary parts. We further assume that such high frequency components are significant for only a small portion of the overall range of integration.

These assumptions present a difficulty similar to stiffness (see Lambert (1980)) when attempting to solve such a problem numerically. The ability of a numeric code to handle problems such as (1.1) depends largely upon certain properties possessed by its underlying formulas. We now define the properties that will be mentioned in our discussion.

DEFINITION 1.1 (Dahlquist (1963)). The region of stability of a given formula is defined to be the set of all $h\lambda$, λ complex, such that the formula, applied to the scalar problem $y' = \lambda y$ with a constant step-size h, produces a sequence $\{y_i\}$ such that $\lim_{i\to\infty} y_i = 0$.

DEFINITION 1.2 (Dahlquist (1963)). A formula is said to be A-stable if its region of absolute stability contains the entire left half of the $h\lambda$-plane (i.e. $\operatorname{Re}(h\lambda) \leqq 0$).

DEFINITION 1.3. The asymptotic damping rate of a formula is defined to be $\lim_{|h\lambda|\to\infty} |y_{i+1}|/|y_i|$. If this limit is zero, then we say that the formula is strongly stable at infinity. We refer to an A-stable formula that is strongly stable at infinity as being strongly A-stable (or equivalently, L-stable (see Lambert (1980))).

The particular solution to (1.1) has the form

$$y(t) = \sum_{j=1}^{n} \alpha_j e^{\lambda_j t} s_j + \psi(t), \tag{1.2}$$

where $\lambda_1, \lambda_2, \cdots, \lambda_n$ are the eigenvalues of $A, s_1, \cdots, s_n$ are its eigenvectors, $\alpha_1, \cdots, \alpha_n$ are scalars determined by the initial conditions and $\psi(t)$ is a particular integral. When the high frequency components, associated say with $\lambda_l, \cdots, \lambda_m$ are significant, then, in order to follow the solution accurately, the step-size, h, used in our numeric code would have to be sufficiently small so that $|h\lambda_l|, \cdots, |h\lambda_m|$ are all smaller than one.

* Received by the editors January 20, 1982, and in revised form June 14, 1983.

† Department of Computer Science, University of Victoria, Victoria, British Columbia, Canada V8W 2Y2. Present address: Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1.

As these components are damped, their contribution to the true solution, $y(t)$, will be sufficiently small that we would like $|h\lambda_l|, \cdots, |h\lambda_m|$ to all be much larger than one. That is, we do not want to follow these components accurately, but we only want such components to be damped. Thus both A-stability and an asymptotic damping rate less than one can be seen as desirable properties.

Unfortunately, many of the widely available ODE codes, such as LSODE (Hindmarsh (1980)) seem to be inappropriate when used on such problems. This is due in part to the fact that LSODE's higher order formulas are not absolutely stable on problems such as (1.1) unless the step-size either is very large, in which case the computed solution may be inaccurate, or is very small, in which case the time necessary to obtain the computed solution may be excessive.

We therefore examine members of three different formula families in an attempt to find formulas that may be the basis of a code suitable for problems such as (1.1). The three formula families which we consider are linear multistep formulas, second derivative formulas and singly implicit Runge–Kutta formulas. From the members of each family, we select a subset for a trial code. For each such code, we estimate the cost involved in performing one step of the integration and then use this estimate to help us assess the performance of the trial codes on a few test problems. In our analysis we do not consider the nontrivial issues involved in the iteration schemes for nonlinear problems. Instead, our trial codes are designed for the linear problem, (1.1) so that no iteration is required. For completeness therefore we also briefly consider the additional complications which would be involved in solving the general nonlinear problem $y' = f(t, y)$ using extensions of the methods programmed in our trial codes.

2. Linear multistep formula. The general form of members of the family of linear multistep formulas is

$$\sum_{j=0}^{m} (a_j y_{i-j} + h b_j y'_{i-j}) = 0.$$

Part of the appeal of formulas of this type is that they have a low amount of computational overhead associated with them and also that many of the discussions on how to implement the backward differentiation formulas, BDF's, (e.g. Byrne and Hindmarsh (1975)) can be adapted to this more general form. They are also easily modified to handle implicit problems such as $My' = f(t, y)$. The major difficulty, as far as our discussion is concerned, is that only the first and second order implicit formulas (that is with $b_0 \neq 0$) can be A-stable (Dahlquist (1963)). As we will illustrate in our numerical testing, a code based on such lower order formulas is often not as cost effective in producing an answer to a given accuracy as a code that includes higher order formulas from the same family. We therefore need to consider the importance of the property of A-stability. We observe that if a formula is not A-stable then, for a given step-size, errors associated with certain solution components may grow. The way in which this instability manifests itself is illustrated by considering the problem $y' = \lambda y$. For a given step-size, h, the stability properties associated with λ are determined by the zeros of the characteristic polynomial

$$p(h\lambda, w) = \sum_{j=0}^{m} (a_j + h\lambda b_j) w^{m-j},$$

(see Hall and Watt (1976)). If the largest zero of $p(h\lambda, w)$ has a magnitude greater than one, then $h\lambda$ will lie in the region of instability and any errors made in previous steps will grow.

Note that as $h\lambda$ approaches infinity, $p(h\lambda, w)/(h\lambda)$ approaches $\sum_{j=0}^{m} b_j w^{m-j}$, so that the asymptotic damping rate of the formula is the magnitude of the largest zero of this polynomial called $b(w)$. The BDF's are obtained by defining $b(w) = b_0 w^m$ for $m = 1, 2, \cdots$ so that all the BDF's have an asymptotic damping rate of zero.

The first and second order BDF's are A-stable but as mentioned earlier, the higher order formulas are not. Figure 2.1 shows the relevant portion of the instability region for the fourth order BDF, which is plotted in the complex $h\lambda$ plane. The instability region is symmetric about the real axis and the stability boundary is formed by finding those values of $h\lambda$ such that

$$p(h\lambda, e^{i\theta t}) = 0 \quad \text{for } 0 \leqq \theta < 2\pi.$$

While the location of the stability boundary is important, so is the rate of error growth within the region of instability. One way to assess the error growth pattern of a formula is to plot boundaries of several growth rates. In order to do this for an error growth rate $x > 0.0$, we plot the values of $h\lambda$ such that

$$p(h\lambda, (1+x)\, e^{i\theta t}) = 0 \quad \text{for } 0 \leqq \theta < 2\pi.$$

The boundaries for the growth rates of 2, 6, 12 and 20% have been computed in this fashion and are shown in Figs. 2.1, 2.2 and 2.3 to illustrate the difference among formulas.

As an illustration, consider the problem $y' = Ay$, where one of the solution components is related to the eigenvalues $-10 \pm 200i$ so that it has the form

$$\exp(-10t) \times (u \times \cos(200t) + v \times \sin(200t)).$$

Suppose we try to solve this problem using the fourth order BDF with a step-size of 2×10^{-2}. The $h\lambda$ of interest is then $-.2 + 4i$, which is marked on Fig. 2.1. At this point there will be about a 4% error growth. After a sufficiently large number of steps, the error growth associated with this component will be detected by the error control and the step will be rejected.

The standard approach in most codes, including LSODE, is to reduce the step-size with the extent of the reduction based only on accuracy considerations. Thus, the

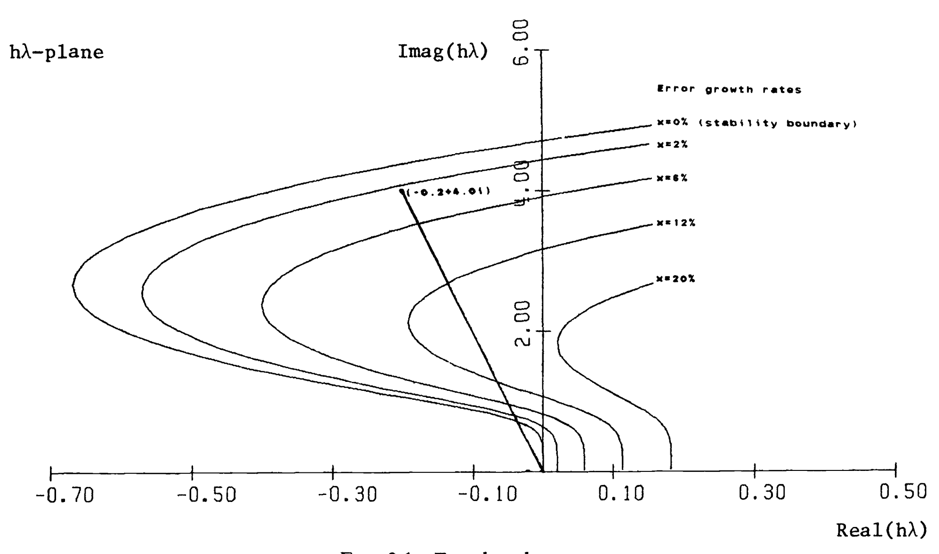


FIG. 2.1. *Fourth order* BDF.

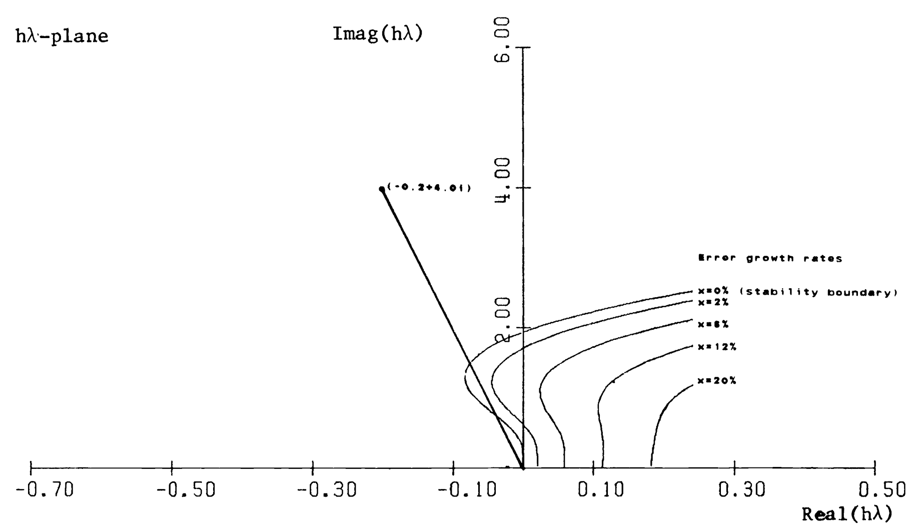


FIG. 2.2. *Third order* BDF.

change in step-size may increase rather than decrease the rate of error growth. In our example, any decrease in h will result in a new $h\lambda$ value that lies along the line between $-.2+4i$ and the origin. Therefore, if the step size is halved, the rate of error growth is increased to about 15%, making the difficulty even worse. Indeed it would be necessary to reduce the step-size to about 7×10^{-4} before there was no error growth associated with this component.

A better strategy in the presence of numerical instability, is to reduce the order of formula used, and perhaps to leave the step-size unchanged. From the stability region of the third order BDF, in Fig. 2.2, we can see why this might be advantageous even though the third order formula is not A-stable. Of course a reduction to second order would eliminate the instability problem altogether but then a much smaller step-size may be necessary to achieve the required accuracy.

As Skelboe (1977) has discussed, there are heuristics available for detecting a point at which instability is becoming a problem so that it is possible to know when decreasing the order is to be preferred to decreasing the step-size. These checks are inexpensive, expecially if the error estimate is based upon difference approximations to higher derivatives but, as we will see in a later section, they are not perfect and the presence of numerical instability may not always be detected.

The likelihood of instability being a difficulty is reduced if we use multistep formulas with regions of instability superior to those of the BDF's. By superior we mean that for a given formula, both the area of the région and the maximum error growth rate in the left half plane are smaller than they are for the corresponding BDF. Addison (1980) describes an attempt to derive a suitable set of formulas. In these new formulas, the zero asymptotic damping rate of the BDF's is traded-off to obtain better stability properties. The idea used there is to incorporate measures of several of the desirable properties in an objective function and then to use a minimization routine to find a suitable set of coefficients. There are two major difficulties with this approach. It is not clear what objective function is "best," in the sense of giving useful formulas, and for any chosen objective function it is difficult to show that the solution found yields a global minimum. Despite these disadvantages, formulas with superior regions of instability were obtained; the stability plot for the fourth order formula derived by Addison is shown in Fig. 2.3.

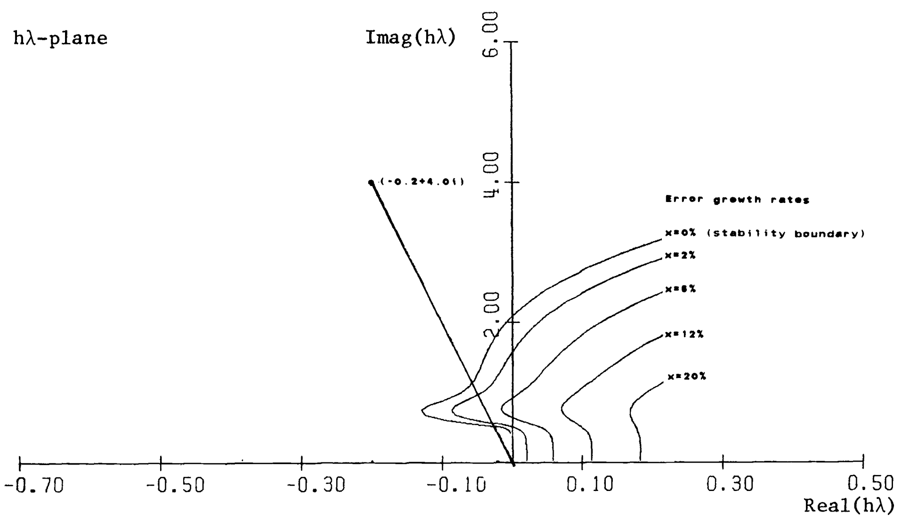


FIG. 2.3. *Addison's fourth order formula.*

Our trial code for linear multistep formulas, MSF14, is based on a set of formulas found by this technique. In selecting this particular set only formulas up to fourth order are used on the assumption that the modest accuracy obtainable will suffice. The leading error terms of the first through third order formulas are smaller than those of the corresponding BDF and that of the fourth order formula is a factor of about a third larger. The asymptotic damping factor for each formula is between .6 and .9. The local error estimates are difference approximations to higher derivatives and the code is organized so that the order selection strategy incorporates stability checking.

We now consider nonlinear problems briefly. Each step of the integration involves predicting the trial value of y, obtaining the derivative associated with this predicted value and then iterating, using a modified Newton scheme. The system of equations that is solved on each iteration is shown below

$$(2.1)\qquad (I-hb_0J)(y_i^{(l)}-y_i^{(l-1)}) = -y_i^{(l-1)} + \sum_{j=1}^{m} (a_jy_{i-j}+hb_jy'_{i-j}) + hb_0y_i'^{(l-1)},$$

where the matrix J is an approximation to the Jacobian, $\partial f/\partial y$ and where we assume that the matrix on the left-hand side of (2.1) is updated and refactored when the step-size, h, or the order, and hence the coefficient b_0, change. If the difference between successive iterates is sufficiently small and if the iteration is converging, then the last iterate may be taken as the trial y-value and its associated derivative is calculated so that the formula is satisfied exactly. If the iteration is not progressing satisfactorily, then the iteration matrix may be updated or the step may be rejected outright (for further details of the strategy that may be used, see Shampine (1980)). The cost per iteration is dominated by a function evaluation and a backsubstitution.

The additional issues involved in handling the general nonlinear problem $y' = f(t, y)$ are well detailed in the literature. In our implementation, we use a predictor based on past y-values (as suggested in Robertson and Williams (1975)) combined with an iteration scheme based on the suggestions of Shampine (1980). While not explicitly designed for solving regular stiff problems, we have found this code to be competitive with LSODE for solving the STIFF DETEST problems (Enright, Hull and Lindberg (1975)) at the less stringent error tolerances.

On problems of the form $y' = Ay + R(t)$, only a single iteration is required and we can use zero for the predicted y-value as convergence is guaranteed for any initial guess. This saves a matrix-vector multiplication but might be more sensitive to rounding errors in the Gaussian elimination process than if we predict normally and carry out some sort of correction as a form of iterative refinement. If we assume that the forcing function, R, is available as a separate function call, then the cost per step is roughly n^2, the cost to solve the system of equations by backsubstitution. Step or order changes involve an additional matrix factorization, which requires about $n^3/3$ operations. It should be noted that the cost per step of a code based on linear multistep formulas is the lowest of all the trial codes.

3. Second derivative formulas. The general form of the members that we consider from this family is

$$y_i = y_{i-1} + h \sum_{j=0}^{m} b_j y'_{i-j} + h^2(c_0 y''_i + c_1 y''_{i-1}). \tag{3.1}$$

The set of order $m+2$ formulas obtained by setting c_1 to zero has been studied in Enright (1972) and implementations are discussed in Addison (1979) and Sacks-Davis (1980). These formulas have many desirable properties. The third and fourth order formulas are A-stable, all of them are strongly stable at infinity and all have good accuracy properties as the step-size, h, approaches zero. What then are the difficulties with them? Consider the problem $y' = Ay$. Equation (3.1) can be written as

$$(i - hb_0A - h^2c_0A^2)y_i = g, \tag{3.2}$$

where g contains the necessary information from past steps. Notice that this equation requires us to square A, but that this is an expensive operation and may destroy any special structure that A possesses. Two ways of avoiding this operation are discussed in Enright (1974). The first approach is to factor the matrix on the left-hand side of (3.2). For Enright's set of formulas, these factors are complex, so that we obtain

$$-c_0(hA - rI)(hA - \bar{r}I)y_i = g, \tag{3.3}$$

where

$$r = -\frac{b_0}{2c_0} + i\text{SQRT}\left(-\left(\frac{b_0}{2c_0}\right)^2 - \frac{1}{c_0}\right)$$

and g is as before. By using complex arithmetic, we can solve the system of equations

$$-c_0(hA - rI)z = g,$$

and then, observing that hA and y are both real, we have

$$y_i = \text{imag}\,(z)/\text{imag}\,(r),$$

and

$$hAy_i = \text{real}\,(z) + \text{real}\,(r)y_i.$$

The additional cost of using complex arithmetic is highly machine dependent. The LINPACK *Users' Guide* (Dongarra et al. (1980)) suggests that complex arithmetic may be as much as eight times slower than real arithmetic on IBM equipment, but only 2.5 times slower on CDC equipment. On machines such as the IBM, it may be preferable to use real arithmetic to carry out the complex arithmetic operations (see Duff (1981)). Four real multiplies are used for each complex multiplication and, in FORTRAN, the differences can be transparent to the user. Therefore the cost of solving a complex system of equations should be no worse than 4 times more than solving a comparable real system.

An alternative approach is to use formulas such that the matrix on the left hand side of (3.2) is a perfect square. This occurs when $c_0 = -(b_0)^2/4$ which allows us to write (3.2) as

$$\left(I - \frac{hb_0}{2}A\right)^2 y = g. \tag{3.4}$$

This can be solved in about $2n^2$ real operations once $(I-(hb_0/2)A)$ has been factored. Unfortunately, by requiring our formulas to have such properties, we lose a great deal. There are one-step, third order formulas that have the above perfect square property and are A-stable but they are less accurate and are not strongly stable at infinity (see Lambert (1973), pp. 245–246). One-step, second order formulas that are both A-stable and strongly stable at infinity do exist, but their higher order analogues are not A-stable (see Enright (1974)). Therefore, if we want to use third and fourth order perfect square formulas, we must be content with formulas processing poorer asymptotic properties both in terms of the leading error coefficient and the asymptotic rate of damping, than Enright's original formulas.

One of our trial codes, SDFR34, is based upon the one-step, third order perfect square formula and a three-step fourth order formula which we have derived. The other code, SDFC34, employs complex arithmetic so that Enright's original formulas can be used. Our order changing strategy is extremely simple—we use the third order formula to start and then switch to the fourth order formula as quickly as possible.

For the problem $y' = Ay + R(t)$ matters become a little more complicated. Notice that $y'' = A^2y + AR(t) + R'(t)$. It turns out that $AR(t)$ need never be formed explicitly. If we let g be as before for equation (3.2), then at each step we need to solve the equation

$$(I - hb_0A - h^2c_0A^2)y_i = g + h^2c_0(AR(t_i) + R'(t_i)) + h^2c_1(AR(t_{i-1}) + R'(t_{i-1})).$$

Using Enright's formulas, where $c_1 \equiv 0$ and where complex arithmetic is used we need to solve

$$-c_0(hA - rI)z = g + h^2c_0(AR(t_i) + R'(t_i)),$$

which can be written

$$-c_0(hA - rI)(z + hR(t_i)) = g + hc_0(rR(t_i) + hR'(t_i)).$$

Since $hR(t_i)$ is real, $\text{imag}\,(z) = \text{imag}\,(z + hR(t_i))$ and hence $y_i = \text{imag}\,(z + hR(t_i))/\text{imag}\,(r)$. A similar rearrangement of terms can be used with the perfect square formulas.

The main issue involves computing $R'(t)$. Asking the user to provide the analytic form of $R'(t)$ may be too demanding. Alternatively, an approximation to it could be obtained, for example

$$R'(t_i) \cong (R(t_i + \sigma) - R(t_i))/\sigma,$$

where σ is the square root of the unit roundoff. This approximation does not give the correct order of accuracy and may not be appropriate for a production code. The most effective way of approximating $R'(t)$ to obtain the correct order of accuracy in a general setting is an open question. Some alternatives are discussed in Addison and

Gladwell (1982). If we again assume that the evaluation of $R(t)$ and $R'(t)$ take roughly n operations, then the cost per step is dominated by the solution of one complex system of equations or two real systems of equations.

For the problem $y' = Ay$ there are several classes of formulas which can be written as second derivative formulas and so our earlier discussion can be applied to them. These formulas begin to differ from true second derivative formulas when considered for the problem $y' = Ay + R(t)$ since $R'(t)$ is only approximated and not used explicitly. On nonlinear problems, $y' = f(t, y)$, these formulas differ considerably from second derivative formulas; nevertheless, the ideas discussed earlier concerning the handling of the system of linear equations that arise on each iteration are still relevant.

As an illustration, consider the blended formulas of Skeel and Kong (1977). These can be given by

$$y_i - y_{i-1} - h \sum_{j=0}^{m} b_j y'_{i-j} + h\gamma_m J\left(\sum_{j=0}^{m} c_j y_{i-j} + h y'_i\right) = 0, \tag{3.5}$$

where the first half of the formula is an Adams–Moulton formula of order $m+1$, the portion in parenthesis is a backward differentiation formula of order m and J is an approximation to the Jacobian evaluated at (t_i, y_i). On the problem $y' = Ay$, with $J = A$ and with an appropriate choice of γ_m, these formulas are the same as an $m-1$ step second derivative formula of Enright (Skeel and Kong (1977)). By varying γ_m, different m step formulas of order $m+1$ with differing stability and accuracy properties can be obtained. If a forcing term, $R(t)$, is included, then the blended formulas are equivalent to a second derivative formula where $R'(t_i)$ is approximated by the m step Adams–Moulton formula.

On nonlinear problems, second derivative formulas can be difficult to implement and expensive to use if terms involving t and y are coupled. Existing second derivative codes (see for example Sacks-Davis (1980)) can really only be used if the problem is written in autonomous form (where an additional equation for t is added to the system of equations) and if the analytic Jacobian is available. Blended formulas offer less restrictive possibilities and can be employed in much the same way as any multistep formula. The difficulty here is that the effects of using an approximation to the Jacobian on the properties of the blended formulas have not been analysed and they could be severe, particularly on problems with strong nonlinearities. This potential difficulty with blended formulas is not present in regular multistep formulas such as the BDF's. In the former case, J is an intrinsic part of the formula itself. In the latter, J is used only to form the iteration matrix and need only be updated if the rate of convergence of the iteration is not satisfactory.

Another alternative is to exploit the fact that on the autonomous problem $y' = Ay$, second derivative formulas are equivalent to certain Runge–Kutta or hybrid formulas (Norsett (1979)). (For example, the third order perfect square formula given in Lambert (1973) is equivalent to a third order semi-explicit formula (see Hall and Watt (1976, pp. 148–151).) These related formulas do no require the analytic Jacobian and could probably form the basis of a code suitable for the problem $y' = f(t, y)$.

4. Singly implicit Runge–Kutta formulas. The standard form of implicit Runge–Kutta formulas is

$$y_i = y_{i-1} + h \sum_{l=1}^{m} b_l f(t_{i-1} + c_l h, Y_l),$$

$$Y_l = y_{i-1} + h \sum_{j=1}^{m} a_{lj} f(t_{i-1} + c_j h, Y_j).$$

For our special first order problem, such a formula would require the solution of a system of equations of order mn per step. The singly-implicit Runge–Kutta formulas (see Burrage (1978)) are designed so that the coefficient matrix (a_{ij}) is similar to a matrix having an m-fold eigenvalue θ. Thus each step involves the solution of m equations of the form

$$(4.1) \qquad (I - h\theta A)d_l = a_l.$$

This requires approximately mn^2 operations and the necessary similarity transformations require about $3m^2n$ operations. Furthermore, m evaluations of R are required. As Burrage has shown, an error estimate can be obtained by adding one additional stage to the step. These formulas all possess good stability properties (there are A-stable formulas for orders one through six and for some higher orders as well) and all are strongly stable at infinity.

Our trial code, SRK45, is a modified version of STRIDE (Butcher, Burrage and Chipman (1979)), which is itself a preliminary implementation of these formulas. The modifications which we carried out were designed to make the code more efficient on linear constant coefficient problems. Rejected steps are expensive in STRIDE and great care was taken by the original authors to avoid wasted effort. Theoretically, at each order m, there are m possible values of θ, each with its associated formula, that can be used. Some of these θ values yield formulas with unacceptable regions of instability but there are at least two useful θ values for any of the second or higher order formulas. Thus, there are at least two step-sizes that can be used at each step for a given matrix $(I - h\theta A)$ in (4.1), and STRIDE checks each useful θ value, from smallest to largest, to find the largest step-size that yields an acceptable error estimate. (We refer to the formula associated with the smallest θ value as the principal formula.) If an acceptable step-size using the current order cannot be found, then step-sizes associated with lower order formulas are checked. A similar procedure is used to predict the step-size and $h\theta$ to be used on the next step.

To obtain SRK45, the step selection part of STRIDE is modified so that only step-sizes associated with the current order are considered. In addition, an order decrease is not allowed on a successful step and the step-size is only allowed to change on a successful step (after the selection process described above) if the increase to be made is greater than 20%. As well, the matrix on the left-hand side of (4.1) is updated whenever $h\theta$ is changed, so that one iteration of Newton's method is sufficient for each stage and the initial guess is chosen so that the cost of the transformations needed is only m^2n operations. Another point is that SRK45 uses only fourth and fifth order formulas. On our test problems, SRK45 was the fastest and most accurate modified version of STRIDE that we tested (we tried various combinations of starting and maximum orders using the first through sixth order formulas). Originally, we had planned to use the fourth order formulas alone, but the principal fourth order formula used in STRIDE is not A-stable and this caused difficulties on some problems. These difficulties were overcome by including the fifth order formulas, as the principal one of these is A-stable. While we have concentrated on this particular type of Runge–Kutta formula, there may be semi-explicit formulas (where a_{ij} is zero if $i < j$ and $a_{ii} = \theta$, for $i, j = 1$ to m) of the same order that are equally useful (see Norsett (1974)).

On nonlinear problems, where usually at least two iterations per step are needed, a production implementation of this formula family may not always be competitive with LSODE because of its high cost per step. As we will see, however, such a code has a great deal of potential on problems where the solution contains lightly damped oscillating components.

5. Testing. In the previous three sections, we have described formula families that could form the basis of a code for lightly damped systems. In Table 5.1 we summarize the cost per step associated with the four trial codes which we compare. We assume that the problem is linear with a constant n by n matrix A and that an mth order formula is used. The miscellaneous column includes estimates for the overhead associated with the error control as well as that associated with the transformations in the singly-implicit formulas. The "c" in the costs column indicates the additional cost of complex arithmetic. Our estimates for the number of operations per step are not precise; they assume that A is a full matrix and that R is inexpensive to compute. We have also assumed that n is large relative to m and have therefore only considered the $O(n^2)$ terms. This assumption must be kept in mind in the pursuant discussion because the $O(n)$ terms in the operation counts can be significant on small problems. Nevertheless, these estimates do provide some guidance. For example, we see that the cost per step of using an mth order singly implicit Runge–Kutta formula is about $(m+1)$ times the cost of a step using an mth order multistep formula. A useful rule of thumb therefore is that a code that uses an mth order Runge–Kutta formula is superior to one that uses an mth order multistep formula only if it requires fewer than $1/(m+1)$ times as many steps as the multistep code to achieve the same accuracy.

We present detailed test results for two problems. The results are representative of those obtained on a larger set of eight test problems, which consists of several variations on the two reported problems and some test problems contrived to test the influence of high frequency, low amplitude components with different forcing terms. All of this testing was carried out on a CDC-7600 computer using single precision. Identical results were obtained using an IBM 4341 and double precision. Both of the reported problems have appeared in the literature. For each problem we present two sets of results. The first set gives the results for a specific local error tolerance. The second set are normalized results which indicate the cost if the maximum global error permitted is a certain value. The normalized results for the first problem were obtained using STIFF DETEST (see Enright (1979)) and those for the second problem were obtained by piecewise linear interpolation (and not from STIFF DETEST, for reasons discussed below). For each set, we include the number of steps attempted, the number of backsubstitutions performed, the number of matrix factorizations performed, and an estimate of the normalized overall cost, based on Table 5.1 and an estimated cost of $n^3/3+n^2$ operations for a matrix factorization. The normalized cost is obtained by dividing the actual cost by n^2. The cost associated with a matrix factorization is consistent with our stated assumptions that A is full.

Our first problem is B5 from the STIFF DETEST set of test problems (Enright et al. (1975)).

Problem 1.

$$y' = Ay, \; y(0) = y_0 \quad \text{on the range } 0, 20],$$

where

$$A = \begin{bmatrix} -10 & 100 & 0 & 0 & 0 & 0 \\ -100 & -10 & 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -.1 \end{bmatrix}$$

and

$$y_0 = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T.$$

This problem has received a great deal of discussion in the literature (see for example Gaffney (1981) or Skelboe (1977)) partly because it is one on which BDF based codes tend to have a great deal of difficulty in providing an accurate approximation to the solution in a reasonable number of steps. An important feature of this problem is that none of the components active outside of the transient region are oscillatory. Once the oscillatory components are sufficiently small, the step-size used by all of our trial codes increases rapidly.

TABLE 5.1
Cost per step of mth order formulas from each family.

Formula	Back-solves	Evals of R	Misc.	Approx. cost
Linear multistep	1	1	$\frac{m^2 n}{2}$	n^2
Second der. (complex)	1	2	$\frac{m^2 n}{2}$	cn^2
Second der. (perfect sq.)	2	2	$\frac{m^2 n}{2}$	$2n^2$
Singly im. Runge–Kutta	$m+1$	$m+1$	$m^2 n$	$(m+1)n^2$

The results for Problem 1 are shown in Tables 5.2 and 5.3. The results in Table 5.3 are the ones from which comparisons between codes will be made. We see that SRK45 and MSF14 have roughly the same cost—$(276+7n)$ units, 43 steps, as opposed to $(293+12n)$ units, 258 steps, while the second derivative codes are somewhat more expensive; SDFC34 requires $(99+6n)c$ units, 80 steps, and SDFR34 $(472+7n)$ units, 225 steps. (When n is small, forming A^2 and using real arithmetic in SDFC34 would make it competitive in cost with the other two codes.) The formulas used in SDFR34 have leading error coefficients that are similar in magnitude to those of the formulas used in MSF14 and consistent with this, the two codes require about the same number of steps. At the larger tolerances, MSF14 uses the second order formula for a large part of the integration. Not surprisingly, the code MSF12, formed by restricting the maximum order in MSF14 to two, requires roughly the same number of operations as MSF14 to obtain a global error of 10^{-2}. When the global error is 10^{-3}, however, MSF12 requires over $(700+13n)$ units, which compares poorly with the $(457+12n)$ units required by MSF14.

As mentioned in § 2, MSF14 uses formulas that are not A-stable. On Problem 1, this shortcoming leads to few difficulties but such is not always the case. Consider a modified version of this problem where the eigenvalues associated with the oscillating components are $-10 \pm 1000i$. When used with a local error tolerance of 10^{-2}, MSF14 has a maximum global error of .66. Even more sgnificant observations are that the code requires over $(2500+2n)$ units to reach $t=1.0$ and at this point it is using a step-size of $3.5 * 10^{-4}$ and has a global error of .24. The oscillatory components are not being damped. By way of a contrast, SRK45 with a local error tolerance of 10^{-2} requires about $(900+6n)$ units, 141 steps, to complete the problem and has a maximum global error of .23. When the local error tolerance is decreased to 10^{-3}, SRK45 requires about $(1920+8n)$ units, 314 steps, with a maximum global error of $4.2 * 10^{-2}$.

TABLE 5.2
Unnormalized results for Problem 1.

Tolerance is 10^{-2}

Code	Steps	Back-solves	Matrix factors	Max. global error	Cost
MSF14	176	176	33	4.2×10^{-2}	$209+11n$
SDFC34	81	81	19	9.3×10^{-3}	$(100+6n)c$
SDFR34	172	344	22	1.9×10^{-2}	$366+7n$
SRK45	34	200	19	2.4×10^{-2}	$219+6n$

Tolerance is 10^{-3}

Code	Steps	Back-solves	Matrix factors	Max. global error	Cost
MSF14	310	310	38	5.2×10^{-3}	$348+13n$
SDFC34	124	124	23	1.4×10^{-3}	$(147+8n)c$
SDFR34	256	512	23	4.7×10^{-3}	$535+8n$
SRK45	54	320	23	4.2×10^{-3}	$343+8n$

Tolerance is 10^{-4}

Code	Steps	Back-solves	Matrix factors	Max. global error	Cost
MSF14	459	459	37	5.7×10^{-4}	$496+12n$
SDFC34	221	221	24	1.4×10^{-4}	$(245+8n)c$
SDFR34	441	828	26	9×10^{-4}	$854+9n$
SRK45	86	511	27	6.2×10^{-4}	$539+9n$

These results call into question the use of formulas that are not A-stable for solving this type of problem, yet the second order linear multistep formulas are not wholly satisfactory either. Again, considering the modified problem, MSF12 requires about $(1215+10n)$ units, 1197 steps, to obtain roughly the same global accuracy as SRK45 when used with a tolerance of 10^{-2}. When MSF12 is modified to use the second order BDF it requires about $(2040+12n)$ units, 2004 steps, at a specified error

TABLE 5.3
Normalized results for Problem 1.

Max. global error is 10^{-2}

Code	Steps	Back-solves	Matrix factor	Cost
MSF14	258	258	35	$293+12n$
SDFC34	80	80	19	$(99+6n)c$
SDFR34	225	450	22	$472+7n$
SRK45	43	256	20	$276+7n$

Max. global error is 10^{-3}

Code	Steps	Back-solves	Matrix factor	Cost
MSF14	420	420	37	$457+12n$
SDFC34	155	155	23	$(178+8n)c$
SDFR34	410	820	26	$846+9n$
SRK45	78	465	25	$490+8n$

tolerance of 10^{-3} to match SRK45's results. LSODE, restricted to second order, performs similarly to MSF12 in this case. Thus, our implementation of these formulas for solving this particular problem is inferior to SRK45.

Our second problem is a model for the second order systems encountered in vibration studies and is taken from Trujillo (1977) (it has also been used by Enright (1980)). It is an inhomogeneous problem and its Jacobian has only pure imaginary eigenvalues. The high frequency components are not appreciably excited by the forcing term so that we would prefer a code that uses step-sizes which are essentially independent of these components.

Problem 2.

$$y' = \begin{bmatrix} 0 & I \\ K & 0 \end{bmatrix} y + R(t), \qquad y(0) = y_0 \quad \text{on the range } [0, 40],$$

where

$$K = \begin{bmatrix} k1+k2+k3 & -k2 & -k5 \\ -k2 & k2+k3+k4 & -k3 \\ -k5 & -k3 & k3+k5 \end{bmatrix}$$

and where

$$k1 = -1, \quad k2 = -100, \quad k3 = -10000, \quad k4 = -2, \quad k5 = -200.$$

The forcing term is

$$R(t) = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad f(t)]^T,$$

$$f(t) = \begin{cases} 2t & \text{if } 0 \leqq t < 5, \\ 20 - 2t & \text{if } 5 \leqq t < 10, \\ 0 & \text{if } 10 \leqq t \leqq 40, \end{cases}$$

and the initial conditions are

$$y(0) = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T.$$

The results for this problem are given in Table 5.4 and 5.5. The unnormalized results show that the cost of solving this problem rises sharply at some value of the local error tolerance for each of the codes. This sharp rise occurs when the error tolerance is sufficiently small that the step-size is appreciably influenced by the high frequency components in y'. The precise tolerance at which these components become significant depends upon several factors. Amongst these factors are the accuracy properties of the underlying formulas, the rate at which high frequency components are numerically damped by the formula, the type of error control used (Enright (1980) illustrates the change in performance when the local error in hy' rather than y' is controlled) and the way in which the error estimate is computed.

Such dramatic changes in the computational effort needed for a code to solve a problem make it difficult to obtain meaningful normalized results from three widely spaced data points (error tolerances) as we used in problem 1. We had planned to use piecewise linear interpolation on the results in Table 5.4 to obtain those in Table 5.5. This was not sufficiently accurate to determine a normalization so we ran each code on a narrower range of tolerances and then used a piecewise linear fit to this more closely spaced data.

At a global accuracy of 10^{-2}, MSF14 performs well, requiring about $(286+5n)$ units, 272 steps, (versus $(386+7n)$ units and 62 steps for SRK45). If the global

TABLE 5.4
Unnormalized results for Problem 2.

Tolerance is 10^{-2}

Code	Steps	Back-solves	Matrix factor	Max. global error	Cost
MSF14	141	141	9	.33	$150+3n$
SDFC34	74	74	15	.10	$(89+5n)c$
SDFR34	143	286	19	.10	$305+6n$
SRK45	30	175	10	.12	$185+3n$

Tolerance is 10^{-3}

Code	Steps	Back-solves	Matrix factor	Max. global error	Cost
MSF14	277	277	14	9×10^{-3}	$291+5n$
SDFC34	133	133	18	3.5×10^{-3}	$(151+6n)c$
SDFR34	444	888	23	8×10^{-3}	$1111+8n$
SRK45	50	293	16	1.6×10^{-2}	$309+5n$

Tolerance is 10^{-4}

Code	Steps	Back-solves	Matrix factor	Max. Global error	Cost
MSF14	1445	1445	30	2.0×10^{-3}	$1475+10n$
SDFC34	535	535	26	1.2×10^{-4}	$(561+9n)c$
SDFR34	1502	3004	22	5.0×10^{-4}	$3026+7n$
SRK45	149	880	35	1.6×10^{-3}	$915+12n$

TABLE 5.5
Normalized results for Problem 2.

Max. global error is 10^{-2}

Code	Steps	Back-solves	Matrix factor	Cost
MSF14	272	272	14	$286+5n$
SDFC34	110	110	17	$(127+6n)c$
SDFR34	383	766	22	$788+7n$
SRK45	62	366	20	$386+7n$

Max global error is 10^{-3}

Code	Steps	Back-solves	Matrix factor	Cost
MSF14	1932	1932	27	$1959+9n$
SDFC34	164	164	21	$(185+7n)c$
SDFR34	1316	2632	16	$2648+5n$
SRK45	179	1067	44	$1111+15n$

accuracy is required to be 10^{-3}, MSF14 requires about $(1959+9n)$ units, 1932 steps, while SRK45 requires about $(1111+15n)$ units, 179 steps, so that MSF14 is the more expensive for moderate sized n. Again, if real arithmetic is used with Enright's strongly A-stable formulas (forming A^2 in the process), the resulting code is competitive as SDFC34 requires $(127+6n)c$ units to obtain a global error of 10^{-2}. Even with the additional expense of complex arithmetic, SDFC34 is very competitive at the more stringent accuracy requirements. For example, it only takes about $(185+7n)c$ units, 164 steps, to obtain a global accuracy of 10^{-3}.

6. Conclusions. Of the four codes we have tested, the singly-implicit Runge–Kutta code, SRK45, has consistently been one of the most accurate and efficient for a variety of test problems and tolerances. Furthermore, we have seen that codes based on Runge–Kutta formulas can be almost as flexible as codes based on linear multistep formulas. A major exception to this flexibility concerns implicit problems of the form $My'=f(t, y)$. Most Runge–Kutta formulas would yield a system of the form

$$My_i = My_{i-1} + h \sum_{l=1}^{m} b_l f(t_{i-1}+c_l h, Y_l),$$

$$MY_l = My_{i-1} + h \sum_{j=1}^{m} a_{lj} f(t_{i-1}+c_j h, Y_j).$$

Thus, the Y_l values could be easily obtained (the associated system of equations would involve matrices of the form $(M-h\theta J)$, where J is the Jacobian of f, rather than $(I-h\theta J)$). Obtaining y_i at each step would be expensive unless one of the Y_l's corresponded to y_i. Our error control would therefore need to be in terms of My but we would still need to obtain an approximation to y at output points. One way to do this would be to factor M and solve the appropriate system of equations, which is expensive in terms of storage.

By contrast, when using linear multistep formulas, the cost per step for an implicit problem is greater than for a regular one, but the modifications needed are not complicated. To see this, consider the problem $y'=M^{-1}f(t, y)$, form the associated system of equations that is needed on each iteration and then multiply each side of the system by M. Second derivative formulas can be similarly modified to solve $My'=Ay+R(t)$, but difficulties arise for nonlinear problems, where y'' is expicitly computed (see Addison and Gladwell (1982)). On nonlinear problems a suitably modified code for the blended formulas should be used.

MSF14 has proved the most unreliable code. It performed well on some problems and at certain tolerances but poorly at other times. Part of the reason for this difficulty can be attributed to the heuristics used to detect the presence of instability as there were problems on which these heuristics failed. If it was possible to derive checks that could be proven to detect the presence of instability for a practical range of problems, then a reliable code based on linear multistep formulas could be written. Also, our testing has led us to believe that our code based on the second order linear multistep formulas is not sufficiently accurate to be competitive with codes such as SRK45.

The inadequacies of the lower order multistep formulas are compounded by the fact that, stability issues aside, we have found it more difficult to write a good variable-step, second order code than to write a variable-step code that also uses higher order formulas. For example, in most of our codes, a reduction in the tolerance by a factor of ten, say, led to a similar reduction in the global error. Such was not the case in MSF12, where reducing some values of the tolerance by a factor of ten only led to a small reduction in the global error. (Part of the reason for this is that the step control is designed to preserve efficiency rather than to maintain a smooth relationship between global and local error.) This is clearly undesirable behavior. It means that given an estimate of global accuracy obtained by runs at large tolerances, the user would have little to guide him on selecting the appropriate tolerance to obtain his desired global error.

The difference in performance between the two second derivative codes is pronounced. Part of the reason for this difference can be attributed to the difference in the formulas themselves. The formulas used in SDFC34 have very small leading error terms whilst those in SDFR34 are relatively large. This implies that SDFC34 will take fewer steps to solve a given problem than SDFR34, at the expense of using complex arithmetic, but it also suggests that SDFR34 will take roughly the same number of steps as a linear multistep code so that there is little compensation for the additional cost per step.

The above conclusions are based on tests involving linear, constant coefficient problems and experimental codes designed to exploit the special properties such a problem possesses. It is therefore prudent to consider some of the implications of solving nonlinear problems with an appropriate code. Rather than one iteration per step of a Newton-like scheme, there would probably be two or more iterations. A multistep method would require a function evaluation and a forward–back substitution per iteration. Methods based upon blended formulas would have a similar increase in cost per step while a code like STRIDE would require the solution of m systems of equations and m function evaluations per iteration for an mth order formula. Additionally, for even moderate sized n, the cost of performing the transformations in STRIDE can be significant.

Our goal throughout has been to obtain a code, which can handle problems with lightly damped oscillatory components, that is suitable for inclusion in a library. An interesting discussion of some of the implications of this can be found in Hull (1980). As outlined there, we would like a code to be consistently accurate, in the sense that a reduction in the local error would produce a similar reduction in the global error. We would also like the code to be robust, so that it can solve problems with awkward forcing terms and extreme user requirements (such as large error tolerances). Because a code cannot possibly solve all of the problems that might be put to it, part of our robustness criterion would be that the code be able to identify situations when it is experiencing difficulty and pass this information onto the user. An implementation of any of the formulas discussed here could be made robust, but not all of the formulas could lead to a code that is simultaneously efficient and robust. On the basis of our testing, a part of which we have presented here, when solving linear constant coefficient problems, we feel that a code based on a small set of A-stable singly-implicit or semi-explicit Runge–Kutta formulas could meet these requirements and might in fact be superior to any other production code on many practical problems. The case is not as clear for nonlinear problems and for these the best choice in general may well be a code based on the blended formulas.

Acknowledgments. The author would like to thank the Leverhulme Trust for the financial support given during the preparation of this report and would like to acknowledge the benefits obtained from Dr. I. Gladwell's careful criticism of the manuscript.

REFERENCES

C. A. ADDISON (1979), *Implementing a stiff method based upon the second derivative formulas*, M.Sc. thesis, Dept. Computer Science Report 130/79, University of Toronto, Toronto.

———, (1980), *Numerical methods for a class of second order* ODE'*s arising in structural dynamics*, Ph.D. thesis, Dept. Computer Science Report 147/80, University of Toronto, Toronto.

C. A. ADDISON AND I. GLADWELL (1981), *Second derivative methods applied to first and second order implicit systems*, Numerical Analysis Report 70, Dept. of Mathematics, Univ. Manchester.

K. BURRAGE (1978), *A special family of Runge–Kutta methods for solving stiff differential equations*, BIT, 18, pp. 22–41.

J. C. Butcher, K. Burrage and F. H. Chipman, STRIDE: *Stable Runge–Kutta integrator for differential equations*, Dept. Mathematics Report Series No. 150, University of Auckland, Auckland.

G. D. Byrne and A. C. Hindmarsh (1975), *A polyalgorithm for the numerical solution of ordinary differential equations*, ACM Trans. Math. Software, 1, pp. 71–96.

G. Dahlquist (1963), *A special stability property for linear multistep methods*, BIT, 3, pp. 27–43.

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

I. S. Duff (1981), ME28. *A Sparse Unsymmetric solver for Complex Equations*, ACM Trans. Math. Software, 7, pp. 505–511.

W. H. Enright (1972), *Studies in the numerical solution of stiff ordinary differential equations*, Ph.D. thesis, Dept. Computer Science Report No. 46, University of Toronto, Toronto.

———, (1974), *Optimal second derivative methods for stiff systems*, in Stiff Differential Systems, R. A. Willoughby, ed., Plenum Press, New York, pp. 95–109.

———, (1979), *Using a testing package for the automatic assessment of numerical methods for* O.D.E.'*s*, in Performance Evaluation of Numerical Software, L. D. Fosdick, ed., North-Holland, Amsterdam, pp. 199–213.

———, (1980), *On the efficient time integration of systems of second order equations arising in structural dynamics*, Internat. J. Numer. Meth. Engng, 16, pp. 13–18.

W. H. Enright, T. E. Hull and B. Lindberg (1975), *Comparing numerical methods for stiff systems of* O.D.E.'*s*, BIT, 15, pp. 10–48.

P. W. Gaffney (1981), *A survey of* FORTRAN *subroutines suitable for solving stiff oscillatory ordinary differential equations*, Oak Ridge National Laboratory/CSD/TM-134, Oak Ridge, TN.

G. Hall and J. W. Watt (1976), *Modern Numerical Methods for Ordinary Differential Equations*, Clarendon Press, Oxford.

A. C. Hindmarsh (1980), LSODE *and* LSODI: *Two new initial value ordinary differential equation solvers*, ACM SIGNUM Newsletter, December issue.

T. E. Hull (1980), *Comparison of algorithms for initial value problems*, in Computational Techniques for Ordinary Differential Equations, I. Gladwell and D. K. Sayers, eds., Academic Press, New York.

J. D. Lambert (1973), *Computational Methods in Ordinary Differential Equations*, John Wiley, New York.

———, (1980), *Stiffness*, in Computational Techniques for Ordinary Differential Equations, I. Gladwell and D. K. Sayers, eds., Academic Press, New York, pp. 19–76.

S. P. Norsett (1974), *Semi-explicit Runge–Kutta methods*, Mathematics and Computation Report No. 6/74, University of Trondheim, Trondheim.

———, (1979), Private communication.

H. H. Robertson and J. Williams (1975), *Some properties of algorithms for stiff differential equations*, J. Inst. Math. Appl., 16, pp. 23–24.

R. Sacks-Davis (1980), *Fixed Leading Coefficient Implementation of* SD-*Formulas for Stiff* ODE'*s*, ACM Trans. Math. Software, 6, pp. 540–562.

R. D. Skeel and A. K. Kong (1977), *Blended linear multistep methods*, ACM Trans. Math. Software, 3, pp. 326–345.

S. Skelboe (1977), *The control of order and step length for backward differentiation methods*, BIT, 17, pp. 91–107.

D. M. Trujillo (1977), *An unconditionally stable explicit algorithm for structured dynamics*, Int. J. Num. Meth. Eng., 11, pp. 1579–1592.

L. F. Shampine (1980), *Implementation of implicit formulas for the solution of* O.D.E.'s, this Journal, 1, pp. 103–118.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

LEAST SQUARES ESTIMATION WITH QUANTIZED INTEGRATED SAMPLES*

ARTHUR DAVID SNIDER†

Abstract. A method for estimating the slope of a linear function from quantized, integrated samples of the function in noise is analyzed, together with error estimates.

Key words. least squares, discretization, approximation, stochastic estimation

1. Introduction.

The customary theory of least squares estimation of linear data buried in random noise is premised on the knowledge of sampled functional values $\{f(t_i); i=1, 2, \cdots, n\}$. However, for a large class of instrumentation devices, particularly systems designed to operate in a "nulling" mode, the random signal is analog-integrated before data processing is initiated. For example, a typical inertial accelerometer has a readout which indicates the time integral of acceleration, rather than acceleration itself [1]. The readout is canceled, or nulled, by feeding back and recording (in effect) known acceleration levels over certain periods; thus the net integral of the external acceleration over these periods is determined. The extension of the customary least squares estimation techniques to the case of integrated sample data is straightforward.

The present paper is concerned with the effects of quantization in the nulling process. Often the feedback mechanism does not possess the flexibility to provide arbitrary levels of nulling control, but instead delivers the nulling signal in quantized units. In other words, the integral of the input signal is not compensated until its intensity reaches a certain level, at which time a "quantum" of feedback signal is delivered (and recorded) to null the readout. The data available for least squares processing, then, constitutes a discrete time series of "quantized integrated samples." In the next section we illustrate this effect with a realistic model and demonstrate how this nonlinear discretization can play havoc with the determination of the slope or "drift rate" when the latter is small. The remainder of the paper is then devoted to the analysis and resolution of this problem in both stochastic and deterministic contexts.

2. The quantized integrated sample model.

For concreteness, consider the simplified model of a pendulous accelerometer depicted in Fig. 1.

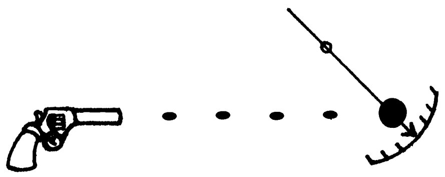


FIG. 1

Gravity or inertial forces tend to rotate the pendulum off of the "null" point, but the pellets fired by the pistol serve to maintain its position. In effect we can say that the external forces work on the pendulum over a period of time until it acquires enough momentum to cross the null point. This crossing is detected by a sensor which commands

* Received by the editors May 21, 1982, and in revised form July 29, 1983.

† Department of Mathematics, University of South Florida, Tampa, Florida 33620.

the pistol to fire, imparting a fixed quantum of momentum to the pendulum and driving it back. Of course the pulse rate must be high enough to overcome the external force, while the individual pulse intensity must be low enough so as not to drive the pendulum too far beyond the null point. Thus the pendulum is essentially maintained fixed, and our assignment is to deduce the external force from the time series record of the pulse firings. (Here we are simplifying the mechanics of the process in an attempt to provide a concrete visualization.)

The mathematical generalization is this. We have an unknown function $f(t)$ and a known function $g(t)$ expressed as a sum of delta functions with constant amplitude

$$g(t)=\sum_i \eta_i\delta(t-t_i), \qquad |\eta_i|=\eta. \tag{1}$$

The supports $\{t_i\}$ of the delta functions are located so that

$$\int_{t_{i-1}}^{t_i} f(t)\,dt=\eta_i,$$

while

$$\left|\int_\lambda^\mu f(t)\,dt\right|<\eta \quad \text{for } t_{i-1}<\lambda<\mu<t_i.$$

See Fig. 2.

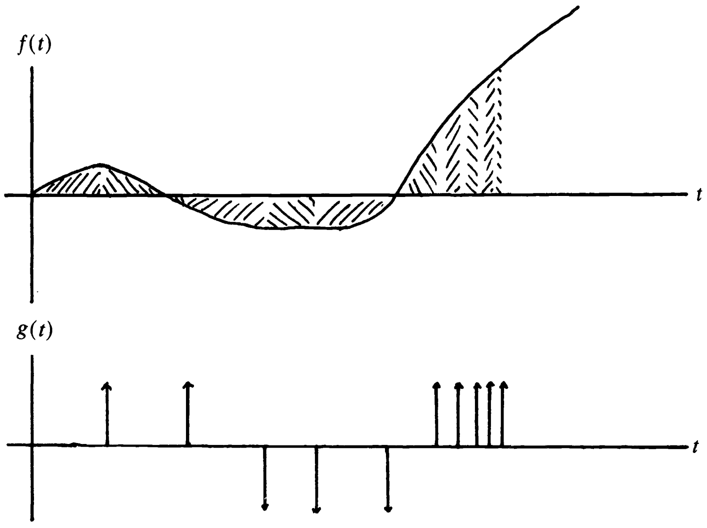


FIG. 2

Thus the pulse function $g(t)$ "cancels" $f(t)$ in a "quantized integral" sense; i.e. for *any* interval $[\lambda, \mu]$,

$$\left|\int_\lambda^\mu f\,dt-\int_\lambda^\mu g\,dt\right|\leqq\eta. \tag{2}$$

We want to estimate f from a knowledge of g. In particular, we seek values for the intercept a and slope b so that the linear function $a+bt$ is the best approximation to f in a least squares sense.

It is instructive to compare this problem with the usual situation where sampled data $\{f(t_j); j=1,2,\cdots,n\}$ are available. Expressions for the values a and b which minimize $\sum |f(t_j)-a-bt_j|^2$ ("l_2-norm") are easy to determine, and estimates of their accuracy have been derived. Naturally the accuracy improves as the number and density of sample points increase, and in the limit we would obtain highest accuracy if we could minimize

$$\int |f(t)-a-bt|^2\,dt \quad (\text{“}L_2\text{-norm”}).$$

In the present context we do not have sampled data of f available, but we do have (approximate) data for integrals of f. Thus we anticipate that the extra accuracy afforded by the L_2 approach will work to our advantage.

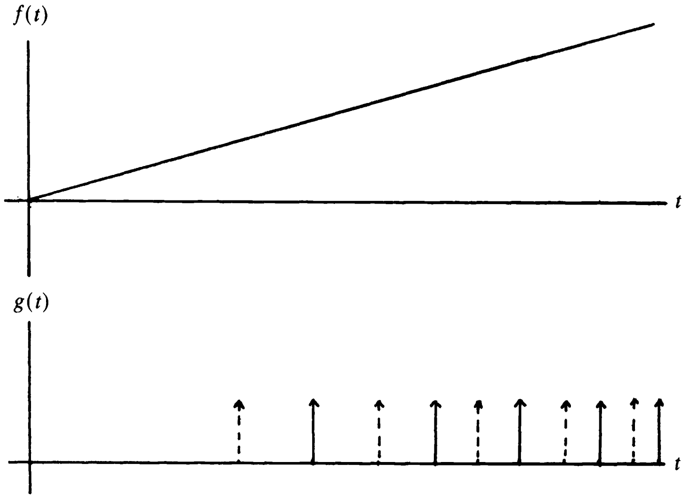


FIG. 3

On the other hand, we can see the difficulties that confront us for the case of functions with low drift rates (b) in Fig. 3. In fact, if $f(t)=bt$ and the pulser $g(t)$ is synchronized with f at $t=0$, then the support points t_i are given by

$$t_i=\left(2\eta\frac{i}{b}\right)^{1/2} \quad \left(\text{because} \int_0^{t_i} bt\,dt=i\eta\right).$$

For low values of b/η these points are sparse and very nonuniform, and one must integrate g over very long intervals to observe the linear trend in f. Indeed, note that the effect of doubling f is to intersperse more support points in g, as indicated by the dashed arrows in Fig. 3; the mapping from f to g is extremely nonlinear, and unless g is integrated over long intervals, one will misinfer the nature of f. Just how long the intervals must be will be determined in this paper.

In fact our main result will demonstrate that one can use $g(t)$ to construct approximations for unbiased estimators of the intercept and slope to within $O(\eta/T)$ and $O(\eta/T^2)$, respectively (for an interval of length $2T$). The variances of these estimators will also be determined.

The fact that we shall be considering the linear least squares fit in the L_2 sense adds an interesting interpretation to our analysis. On the one hand, if we regard $f(t)$

as a sum of a linear function plus a stationary random process [2]

$$f(t)=\alpha+\beta t+e(t), \tag{3}$$

then we are dealing with a statistical estimator of α and of β. On the other hand, if we regard $f(t)$ as deterministic, it has an expansion in terms of Legendre polynomials [3]

$$f=\sum_{j=0}^{\infty} A_j P_j. \tag{4}$$

Since $\{P_j\}$ form an orthogonal basis for the Hilbert space L_2, and since $P_0(x)=1$ and $P_1(x)=x$, the theory implies that $A_0P_0+A_1P_1$ is the best linear fit to f (in L_2). Thus our computations can also be interpreted as projecting f into the subspace spanned by the first two Legendre polynomials. This effect will be included in the analysis.

3. Survey of sampled data theory. Our technique for constructing and analyzing estimators for the "linear part" $\alpha+\beta t$ of f in terms of the quantized-integral approximation g will be based on comparisons with the classical estimators. Thus it is convenient to consider a brief survey of the latter.

To keep the notation simple, we presume $f(t)$ is defined on $[-T, T]$, and we have sampled values at the n equidistant points

$$t_i=-T+(2i-1)\frac{T}{n}, \qquad i=1,2,\cdots,n \tag{5}$$

so that $t_{i+1}-t_i=\Delta t=2T/n$. Then the formulas for a and b minimizing $\sum|f(t_i)-a-bt_i|^2$ are given by [4]

$$a=\left(\frac{1}{n}\right)\sum f(t_i), \qquad b=\frac{3}{((n-1/n)T^2)}\sum t_i f(t_i) \tag{6}$$

(summation limits understood to be $i=1$ to n).

For the stochastic interpretation, if we assume that

$$f(t)=\alpha+\beta t+e(t), \tag{7}$$

where $e(t)$ is a stationary random process with mean zero and autocorrelation ρ

$$\overline{e(t)}=0, \qquad \overline{e(t_1)e(t_2)}=\rho(t_1-t_2)=\rho(|t_1-t_2|),$$

then a and b are unbiased estimators of α and β:

$$\bar{a}=\alpha, \qquad \bar{b}=\beta. \tag{8}$$

(It follows that a and b are *exact* for linear functions.) The variances of a and b are computed to be

$$\begin{aligned}\overline{(a-\alpha)^2}&=\frac{1}{n^2}\sum\sum\overline{e(t_i)e(t_j)}\\&=\frac{1}{n}\rho(0)+\frac{2(n-1)}{n^2}\rho(\Delta t)+O(\rho(p\Delta t)), \qquad p\geqq 2\\\overline{(b-\beta)^2}&=\frac{9}{(n-1/n)^2T^4}\sum\sum t_it_j\overline{e(t_i)e(t_j)}\\&=\frac{3n}{(n^2-1)T^2}\rho(0)+\frac{6(n-3)}{(n^2-1)T^2}\rho(\Delta t)+O(\rho(p\Delta t)).\end{aligned} \tag{9}$$

On the other hand, if $f(t)$ is deterministic and has the Legendre polynomial expansion

$$(10) \qquad f(t)=\sum_{m=0}^{\infty} A_m P_m\left(\frac{t}{T}\right)=A_0+A_1\frac{t}{T}+\sum_{m=2}^{\infty} A_m P_m,$$

then

$$(11) \qquad A_0=\frac{1}{2T}\int_{-T}^{T} f(t)\,dt, \qquad A_1=\frac{3}{2T^2}\int_{-T}^{T} tf(t)\,dt.$$

The estimator a in (6) is then a composite midpoint rule approximation to the integral for A_0 [5], and has accuracy

$$(12) \qquad |A_0-a|\leqq\frac{1}{2T}\frac{\Delta t^3}{24}n\|f''\|_\infty=\frac{T^2}{6n^2}\|f''\|_\infty.$$

The estimator b is not a midpoint rule approximation for A_1/T; in fact it has higher precision, since it integrates linear f exactly (the midpoint rule's degree of precision is 1, and (tf) has degree 2 for linear f). A tedious computation, similar to one we sketch in the Appendix, shows that for large n

$$(13) \qquad \left|\frac{A_1}{T}-b\right|\leqq\frac{4T}{n^2}\|f''\|_\infty.$$

We summarize as follows.

LEMMA 1. *If $f(t)=\alpha+\beta t+e(t)$, where e is a stationary random process, then a and b in* (6) *are unbiased estimators for α and β with variances given roughly by*

$$(14) \qquad \overline{(a-\alpha)^2}\approx\frac{[\rho(0)+2\rho(\Delta t)]}{n}, \qquad \overline{(b-\beta)^2}\approx\frac{3}{T^2}\overline{(a-\alpha)^2}.$$

LEMMA 2. *If $f(t)\in C^2[-T,T]$ and $f=\sum_{m=0}^{\infty}A_mP_m(t/T)$ in L_2, then a and b in* (6) *are estimators of A_0 and A_1/T, with errors bounded roughly by*

$$(15) \qquad |A_0-a|\leqq\|f''\|_\infty\frac{T^2}{6n^2}, \qquad \left|\frac{A_1}{T}-b\right|\leqq\|f''\|_\infty\frac{4T}{n^2}.$$

4. Extension to integrated samples. Again for notational simplicity we take the domain to be $[-T,T]$, and we have N integrated samples F_j,

$$(16) \qquad F_j=\int_{\Delta j} f(t)\,dt, \qquad j=1,2,\cdots,N,$$

with

$$(17) \qquad \Delta_j=\left[\mathcal{T}_j-\frac{\Delta\mathcal{T}}{2},\mathcal{T}_j+\frac{\Delta\mathcal{T}}{2}\right], \qquad \mathcal{T}_j=-T+\frac{(2j-1)T}{N}, \qquad \Delta\mathcal{T}=\frac{2T}{N}.$$

(We presume $N\neq n$ in § 2, since we will be comparing the estimates.) The estimators we propose are constructed by analogy with (6):

$$(18) \qquad A=\left(\frac{1}{N\Delta\mathcal{T}}\right)\sum F_j, \qquad B=\frac{3}{(N-1/N)\Delta\mathcal{T}T^2}\sum \mathcal{T}_jF_j$$

(summation understood for $j=1$ to N).

For a stochastic f as in (7) it is easy to show that these, too, are unbiased estimators for α and β (and hence are exact for linear functions). Their variances are calculated analogously to (9), with $e(t_i)$ replaced by $E_j = \int_{\Delta_j} e(t)\,dt$:

$$\begin{aligned}\overline{(A-\alpha)^2} &= \frac{1}{N^2\Delta\mathcal{T}^2}\sum\sum \overline{E_iE_j}\\ &= \frac{1}{N\Delta\mathcal{T}^2}\overline{E_0^2} + \frac{2(N-1)}{N^2\Delta\mathcal{T}^2}\overline{E_0E_1} + O(\overline{E_0E_p}), \qquad p \geqq 2,\\ \overline{(B-\beta^2)} &= \frac{3N\overline{E_0^2}}{(N^2-1)T^2\Delta\mathcal{T}^2} + \frac{6(N-3)}{(N^2-1)\mathcal{T}^2\Delta\mathcal{T}^2}\overline{E_0E_1} + O(\overline{E_0E_p}).\end{aligned} \tag{19}$$

In terms of the autocorrelation function,

$$\begin{aligned}\overline{E_0^2} &= \int_{\Delta_j}\int_{\Delta_j} \overline{e(t)e(t')}\,dt'\,dt = \int_{\Delta_j}\int_{\Delta_j} \rho(t-t')\,dt'\,dt\\ &= \int_0^{\Delta\mathcal{T}}\int_0^{\Delta\mathcal{T}} \rho(t-t')\,dt'\,dt = 2\int_0^{\Delta\mathcal{T}} (\Delta\mathcal{T}-t)\rho(t)\,dt.\end{aligned} \tag{20}$$

Similarly,

$$\begin{aligned}\overline{E_0E_1} &= \int_{\Delta_j}\int_{\Delta_{j+1}} \overline{e(t)e(t')}\,dt'\,dt = \int_0^{\Delta\mathcal{T}}\int_{\Delta\mathcal{T}}^{2\Delta\mathcal{T}} \rho(t-t')\,dt'\,dt\\ &= \int_0^{2\Delta\mathcal{T}} [\Delta\mathcal{T}-|t-\Delta\mathcal{T}|]\rho(t)\,dt.\end{aligned} \tag{21}$$

In the deterministic case, A is *precisely* the first Legendre coefficient A_0 (11), since

$$\sum F_j = \int_{-T}^{T} f(t)\,dt. \tag{22}$$

In the Appendix we sketch the tedious computation leading to the following accuracy estimate for B:

$$\left|\frac{A_1}{T} - B\right| \leqq \frac{11}{8}T\|f''\|_\infty|(N^2-1). \tag{23}$$

To summarize:

LEMMA 3. *If $f(t) = \alpha + \beta t + e(t)$ as in Lemma 1, then A and B in (18) are unbiased estimators for α and β, with variances given roughly by*

$$\begin{aligned}\overline{(A-\alpha)^2} &\approx \frac{2}{N\Delta\mathcal{T}^2}\left\{\int_0^{\Delta\mathcal{T}} (\Delta\mathcal{T}-t)\rho(t)\,dt + \int_0^{2\Delta\mathcal{T}} [\Delta\mathcal{T}-|t-\Delta\mathcal{T}|]\rho(t)\,dt\right\},\\ \overline{(B-\beta)^2} &\approx \frac{3}{T^2}\overline{(A-\alpha)^2}.\end{aligned} \tag{24}$$

LEMMA 4. *If $f(t) \in C^2 \cap L_2$ as in Lemma 2, then A in (18) equals A_0 precisely and B approximates A_1/T with an error bounded roughly by*

$$\left|\frac{A_1}{T} - B\right| \leqq \|f''\|_\infty \frac{T}{N^2}. \tag{25}$$

5. Comparison of point and integrated sample estimators. The superiority of the integrated sample estimators in the deterministic case is clearly expressed in Lemmas 2 and 4, and we shall not dwell on this.

For the stochastic case the superiority may not be so clear; for example, if the autocorrelation function ρ is constant, Lemmas 1 and 3 reveal no difference between the estimators (for $n=N$). Of course this is physically unreasonable, since we expect $\rho(t)$ to decrease with it. As a result, the integrals in Lemma 3 "soften" the factor $\rho(0)$ in Lemma 1 and demonstrate the superiority of the integrated sample estimators. For example, if $\rho(t)=\sigma^2 e^{-\gamma t}(\gamma>0)$, then the variance of the point sample estimator (14) is

$$\overline{(a-\alpha)^2}\approx\frac{[1+2\,e^{-2\gamma T/n}]\sigma^2}{n}, \tag{26}$$

while the integrated sample estimator (24) has a variance given by

$$\overline{(A-\alpha)^2}\approx\frac{2\sigma^2}{N\Delta\mathcal{T}^2}\left\{\frac{\Delta\mathcal{T}}{\gamma}-\frac{e^{-2\gamma T/N}}{\gamma^2}[1-e^{-2\gamma T/N}]\right\}. \tag{27}$$

For large γ the difference is quite pronounced:

$$\overline{(a-\alpha)^2}\to\frac{\sigma^2}{n},\qquad \overline{(A-\alpha)^2}\approx\frac{\sigma^2}{T\gamma}\to 0. \tag{28}$$

For small γ (and $n=N$) both variance estimates approach $3\sigma^2/n$, but the justification for dropping terms in (9) and (19) is no longer valid.

Equations (28) indicate another advantage of the integrated sample approach. Since the asymptotic forms for the variances of the integrated sample estimators are independent of N, one can possibly achieve a given tolerance level for the variances with a value of N much lower than the corresponding n required by the point sample approach, when the noise autocorrelation is weak. This feature will surface in another context in the following section.

6. Accuracy of the quantized integrated sample estimators. Of course one would expect the integrated samples to yield superior estimates, since they contain so much more information about f. Note that the integrals must be performed *exactly*, e.g. by analog, for the estimates in § 4 to hold; if Riemann sums or interpolatory quadrature are used we are back in the point-sample mode. The analysis of the quantized integrated sample data will be developed by comparison with the integrated sample approach.

Thus in accordance with the model of § 2 we are given intervals Δ_j and approximations ϕ_j to F_j satisfying (recall (2), (16) and (17))

$$\phi_j=\int_{\Delta_j} g(t)\,dt,\qquad |\phi_j-F_j|\leqq\eta. \tag{29}$$

We approximate A and B of § 4 with the formulas

$$A'=\frac{1}{N\Delta\mathcal{T}}\sum\phi_j,\qquad B'=\frac{3}{(N-1/N)\Delta\mathcal{T}T^2}\sum\mathcal{T}_j\phi_j. \tag{30}$$

The accuracy of A' is easy to assess. Although each ϕ_j can differ from F_j by η, the sum of the ϕ_j equals the integral of g, and by (2),

$$|A-A'|=\frac{1}{N\Delta\mathcal{T}}|\sum F_j-\sum\phi_j|=\frac{1}{N\Delta\mathcal{T}}\left|\int_{-T}^{T}f\,dt-\int_{-T}^{T}g\,dt\right|\leqq\frac{\eta}{N\Delta\mathcal{T}}=\frac{\eta}{2T}. \tag{31}$$

For B' we have

$$(32)\qquad |B-B'|=\frac{3}{(N-1/N)\Delta\mathcal{T}T^2}|\Sigma\,\mathcal{T}_j(F_j-\phi_j)|.$$

Acknowledging the constraints (derived from (2))

$$(33)\qquad \left|\sum_{j_1}^{j_2}(F_j-\phi_j)\right|\leqq\eta,\qquad 1\leqq j_1\leqq j_2\leqq N,$$

one can derive with a little combinatorics that the sum in (32) is maximal when $F_1-\phi_1=\pm\eta$, $F_n-\phi_n=\mp\eta$, $F_j-\phi_j=0$ otherwise. Thus

$$(34)\qquad |B-B'|\leqq\frac{3N\eta}{(N^2-1)\Delta\mathcal{T}T^2}(|\mathcal{T}_1|+|\mathcal{T}_n|)=\frac{3n}{(1+1/N)T^2}<\frac{3\eta}{T^2},$$

and we summarize the observations of this paper as follows.

Summary. If $f(t)=\alpha+\beta t+e(t)$, where $e(t)$ is a stationary random process with autocorrelation $\rho(t)$, then A' and B', as formed from quantized integrated samples via (30), are within $\eta/2T$ and $3\eta/T^2$, respectively, of unbiased estimators for α and β possessing variances governed by (24). If $f(t)\in C^2\cap L_2$ on $[-T, T]$, then A' approximates the zeroth Legendre coefficient of f to within $\eta/2T$, and $(B'T)$ approximates the first coefficient to within $3\eta/T+T^2\|f''\|_\infty/N^2$.

7. Conclusion. The derivation of (34) vindicates an empirical observation made by the author in 1981 while analyzing accelerometer data for the Honeywell Corporation. The fact that the accuracy estimates (31) and (34) depend only on T (and η) is fortunate. It shows that the quantizing effect depicted in Fig. 2 can be compensated by the proper choice of T alone. N can be chosen subsequently for noise control in accordance with (24) (keep in mind $\Delta\mathcal{T}=2T/N$). Thus, for example, if one wishes to detect a very low drift rate β to within 25% in low-noise data, one would choose

$$T=\left(\frac{12\eta}{\beta}\right)^{1/2}.$$

Similarly, if the data accumulation intervals $\Delta\mathcal{T}$ are fixed, the error equations (31) and (34) specify a lower bound on T, and hence on $N=2T/\Delta\mathcal{T}$. N (and T) can then be increased if necessary to reduce noise in (24).

The analysis herein can be modified to cover the situation where the Δ_j are uneven, and the $\|f''\|_\infty$ estimates can be replaced by $\|f''\|_2$ estimates, but the results are not particularly illuminating.

Appendix. Sketch of derivation of (23). From (11) and (18) we have

$$(35)\qquad \left|\frac{A_1}{T}-B\right|=\frac{3}{2T^3}\Sigma\int_{\Delta_j}tf(t)\,dt-\frac{3N^2}{(N^2-1)2T^3}\Sigma\,\mathcal{T}_j\int_{\Delta_j}f(t)\,dt.$$

Writing $f(t)=f(0)+tf'(0)+\int_0^t f''(\xi)(t-\xi)\,d\xi=f(0)+tf'(0)+G(t)$, we find that the constant and linear terms cancel in (35) (as they must for unbiased estimators!). Then by adding and subtracting the same term we find

$$(36)\qquad \left|\frac{A_1}{T}-B\right|=\frac{3}{2T^3}\left[1-\frac{N^2}{N^2-1}\right]\Sigma\int_{\Delta_j}G(t)t\,dt$$
$$+\frac{3}{2T^3}\frac{N^2}{N^2-1}\Sigma\int_{\Delta_j}G(t)(t-\mathcal{T}_j)\,dt.$$

In the first term we estimate $|G(t)| \leqq \|f''\|_\infty t^2/2$. In the second, using the Taylor form

$$G(t) = G(\mathcal{T}_j) + G'(\zeta)(t - \mathcal{T}_j),$$

we estimate

$$\left| \int_{\Delta_j} G(t)(t - \mathcal{T}_j)\, dt \right| \leqq \max_{\Delta_j} |G'(t)| \frac{\Delta \mathcal{T}^3}{12} \leqq \|f''\|_\infty T \frac{\Delta \mathcal{T}^3}{12}.$$

Assembling these in (36) we derive (23). (Q.E.D.)

REFERENCES

[1] C. Broxmeyer, *Inertial Guidance Systems*, McGraw-Hill, New York, 1965.
[2] Y. W. Lee, *Statistical Theory of Communication*, Wiley, New York, 1960, Chapter 7.
[3] F. B. Hildebrand, *Advanced Calculus for Applications*, 2nd, ed., Prentice-Hall, Englewood Cliffs, NJ, 1976, Chapters 4, 5.
[4] J. E. Freund, *Mathematical Statistics*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1971, Chapter 13.
[5] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*, John Wiley, New York, 1966, Chapter 7.

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

NUMERICAL SOLUTIONS OF THE GOOD BOUSSINESQ EQUATION*

V. S. MANORANJAN†, A. R. MITCHELL† AND J. Ll. MORRIS‡

Abstract. The "good" Boussinesq equation is studied numerically using Galerkin methods and soliton solutions are shown to exist. An analytic formula for the two-soliton interaction is presented and verified by numerical experiment.

Key words. soliton, Boussinesq equation, Galerkin methods, blow up

1. Introduction.

In recent years a remarkable development has taken place in the study of nonlinear evolutionary partial differential equations. It is the realization that many such equations possess special solutions in the form of pulses which retain their shapes and velocities after interaction amongst themselves. Such solutions are called solitons. The reader interested in the theory behind soliton solutions should consult texts such as Whitham [16], Lamb [9], Ablowitz and Segur [1], etc.

An example of a soliton producing equation is the Boussinesq equation

$$u_{tt} = u_{xx} + u_{xxxx} + (u^2)_{xx} \tag{1.1}$$

which describes shallow water waves propagating in both directions. Unfortunately, due to the presence of exponentially growing Fourier components, (1.1) is linearly unstable and so numerical computations based on (1.1) are at best unreliable. If, however, we change the sign of the fourth order space derivative in (1.1), we obtain

$$u_{tt} = u_{xx} - (u)_{xxxx} + (u^2)_{xx} \tag{1.2}$$

an equation which is linearly stable and computations are now possible. An alternative method of obtaining (1.2) is to transform (1.1) according to the formulae $x = iX$, $t = iT$ ($i = \sqrt{-1}$), thus obtaining (1.2) with x and t replaced by X and T respectively.

It is the aim of this paper to show that numerical studies of the Cauchy problem based on (1.2) with certain initial conditions produce soliton solutions. In so doing the existence of "blow up" inherent in the problem is also demonstrated. So far theory involving (1.2) has been restricted to periodic boundary conditions on a circular region in space (McKean [10]), the Cauchy problem with specific initial data (Tomei [15]), the Cauchy problem with general initial data and a specific initial boundary value problem (Kalantarov and Ladyzhenskaya [8]). None of these authors offer soliton solutions for (1.2), with Tomei deriving nonreal solutions, and Kalantarov and Ladyzhenskaya predicting blow up in most cases unless possibly for short time solutions.

Equation (1.2) is the nonlinear string equation (Zakharov [17]) or the "good" Boussinesq (G.B.) equation (McKean [10]). In contrast, (1.1) is referred to as the "bad" Boussinesq (B.B.) equation. Closely related to (1.1) is the Korteweg–de Vries (K.d.V.) equation

$$u_t + (u^2)_x + u_{xxx} = 0. \tag{1.3}$$

This equation models shallow water waves propagating in a single direction and has the same order of dispersion relation as (1.1) if the wave number is small. It has soliton

* Received by the editors October 1, 1982, and in revised form June 2, 1983.

† Department of Mathematical Sciences, University of Dundee, Scotland.
‡ Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

solutions but unlike (1.1) is linearly stable and so numerical calculations can be carried out (Mitchell and Schoombie [12] and references therein).

Before proceeding to numerical studies of (1.2), we show that the linear term u_{xx} can be removed from (1.2) by putting

$$u = v - \tfrac{1}{2} \tag{1.4}$$

resulting in the equation

$$v_{tt} = -v_{xxxx} + (v^2)_{xx}. \tag{1.2a}$$

It is immaterial with numerical methods whether we use (1.2) or (1.2a), although in theoretical considerations, it may be simpler to use (1.2a).

2. Petrov–Galerkin method with linear "hat" trial functions. We now consider numerical schemes for the Cauchy problem consisting of (1.2) together with the initial conditions

$$\begin{aligned} u(x,0) &= f(x), \\ u_t(x,0) &= g(x), \end{aligned} \qquad -\infty < x < +\infty. \tag{2.1}$$

We approximate $u(x, t)$ by

$$U(x,t) = \sum_i U_i(t)\phi_i(x) \tag{2.2}$$

where the trial functions $\{\phi_i(x)\}$ are the usual piecewise linear "hat" functions, and determine the unknown functions $U_i(t)$ from the system of ordinary differential equations

$$(U_{tt} - U_{xx} + U_{xxxx} - (U^2)_{xx}, \psi_j(x)) = 0 \quad \forall j \tag{2.3}$$

where $\{\psi_j(x)\}$ are suitably chosen test functions. As far as the continuity of the test functions is concerned, the important term in (2.3) is $(U_{xxxx}, \psi_j(x))$ which after two integrations by parts and neglecting boundary terms gives $(U_{xx}, (\psi_j)_{xx})$. This suggests either C^2 cubic B-splines or shifted C' quadratic splines (Mitchell and Schoombie [12]), and for ease of application, we choose the former. With standard time discretization, we arrive at the following predictor-corrector (P.C.) numerical scheme based on (2.3).

$$\begin{aligned} P&: \left(\frac{1}{k^2}(U^{n+1^*} - 2U^n + U^{n-1}), \psi_j\right) + \left(\frac{\partial}{\partial x}\left(\frac{U^{n+1^*} + U^{n-1}}{2}\right), \psi_j'\right) \\ &\qquad + \left(\frac{\partial}{\partial x}((U^n)^2), \psi_j'\right) - \left(\frac{\partial}{\partial x}\left(\frac{U^{n+1^*} + U^{n-1}}{2}\right), \psi_j'''\right) = 0, \\ C&: \left(\frac{1}{k^2}(U^{n+1} - 2U^n + U^{n-1}), \psi_j\right) + \left(\frac{\partial}{\partial x}\left(\frac{U^{n+1} + U^{n-1}}{2}\right), \psi_j'\right) \\ &\qquad + \left(\frac{\partial}{\partial x}\left(\left(\frac{U^{n+1^*} + U^n}{2}\right)^2\right), \psi_j'\right) - \left(\frac{\partial}{\partial x}\left(\frac{U^{n+1} + U^{n-1}}{2}\right), \psi_j'''\right) = 0, \end{aligned} \tag{2.4}$$

where $t = nk$, $n = 0, 1, 2, \cdots$. In arriving at (2.4), we have used product approximation on the nonlinear term i.e.

$$\{U(x,t)\}^2 = \sum_i \{U_i(t)\}^2 \phi_i(x). \tag{2.5}$$

Product approximation was first suggested by Swartz and Wendroff [14] and further details of it can be obtained in Chin et al. [2] and Christie et al. [3]. Once again we remind the reader that the trial functions $\{\phi_i(x)\}$ are linear "hat" functions and the test functions $\{\psi_j(x)\}$ are cubic B-splines.

Numerical calculations are carried out using (2.4) together with a selection of initial conditions $f(x)$ and $g(x)$. The aim is to obtain soliton solutions which so far have not been shown to exist from theoretical analysis of the "good" Boussinesq equation.

3. The single soliton solution. Direct verification shows that

$$u = A\,\mathrm{sech}^2\{\sqrt{A/6}\,(x-ct+x_0)\}+(b-\tfrac{1}{2}), \qquad c=\pm\sqrt{2(b+A/3)} \tag{3.1}$$

(x_0 gives the initial position of the pulse)

exactly satisfies the B.B. equation (1.1), where b is an arbitrary parameter and A is the amplitude of the pulse and that if $b > -\frac{1}{3}A$, (3.1) is a real soliton solution (cf. the case $b=\frac{1}{2}$ given by Scott, Chu and McLaughlin [13]). For the G.B. (1.2) an exact solution is

$$u = -A\,\mathrm{sech}^2\{\sqrt{A/6}\,(x-ct+x_0)\}-(b+\tfrac{1}{2}), \qquad c=\pm\sqrt{-2(b+A/3)} \tag{3.2}$$

which is real if $b < -\frac{1}{3}A$, where again b is an arbitrary parameter and A is the pulse amplitude. Tomei [15], whose analysis was based on (1.2a), chose (3.2) with $b=0$ leading to $c=\pm(-\frac{2}{3}A)^{1/2}$ and so showed the existence only of imaginary solutions. We choose for our numerical experiments $b=-\frac{1}{2}$ and so $c=\pm(1-\frac{2}{3}A)^{1/2}$, giving in theory real solutions for $A\leqq\frac{3}{2}$. The relevant theoretical solution becomes

$$u = -A\,\mathrm{sech}^2\{\sqrt{A/6}\,(x\pm\sqrt{1-\tfrac{2}{3}A}\,t+x_0)\}. \tag{3.3}$$

We now solve the G.B. equation (1.2) by using the numerical scheme (2.4) together with the initial data (see (2.1)) $f(x)$ and $g(x)$ taken from (3.3) for the case $c=0$, viz.

$$f(x) = -A\,\mathrm{sech}^2\{\sqrt{A/6}\,(x+x_0)\}, \qquad g(x)=0.$$

This is carried out for a range of values of A mainly in the vicinity of $A=1.5$, the theoretical value of the amplitude for $c=0$ (see (3.3)), and a variety of mesh lengths h and k in the x- and t-directions respectively. As time evolves the initial pulse either splits into two pulses moving in opposite directions or "blows up" according to $A \lessgtr A_s$. A typical set of values for A_s is given in Table 3.1 where the extrapolated value of A_s as $k\to 0$ is 1.5017, which is approximately 1.5, the value corresponding to the theoretical steady state solution. In Fig. 3.1, we show an initial stationary pulse of amplitude 1.5165 ($h=0.5$, $k=0.05$) breaking up into two pulses moving in opposite directions and each having eventually a steady speed of 0.8639 and an amplitude of 0.3816 (note that $c=(1-\frac{2}{3}A)^{1/2}=0.8635\sim 0.8639$). The shape and behaviour of each pulse agrees with the theoretical solution (3.3) of the single soliton.

TABLE 3.1

$-100\leqq x\leqq 100,\quad h=0.5,\quad x_0=0$

k	0.100	0.050	0.025
A_s	1.5169	1.5165	1.5109

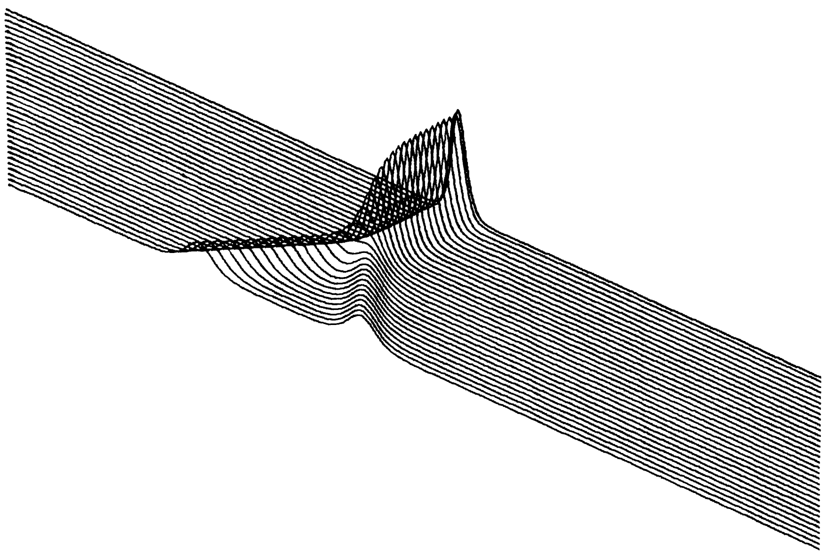


FIG. 3.1. *Break up of the pulse.* $-100 \leqq x \leqq 100$, $0 \leqq t \leqq 40$, $h = 0.5$, $k = 0.05$, *amplitude* $= 1.5165$. *Initial position of the pulse is* $x = 0$. *No radiation loss is observed.*

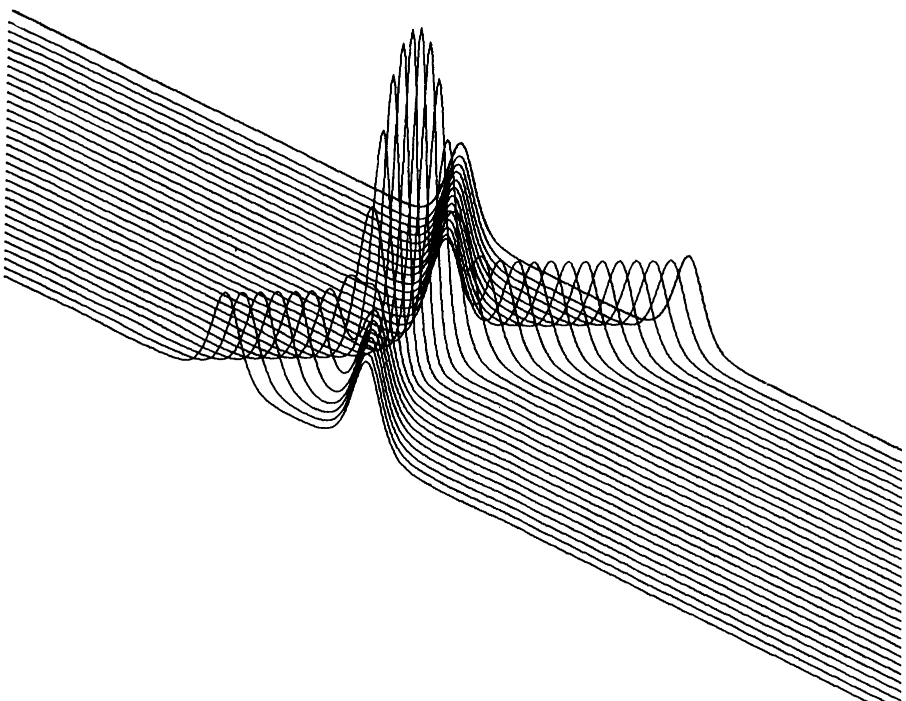


FIG. 3.2. *Interaction of two pulses.* $-100 \leqq x \leqq 100$, $0 \leqq t \leqq 60$, $h = 0.5$, $k = 0.05$, $A = 0.369$. *Initial positions of the pulses are* $x = 0$ *and* $x = 50$. "*Blow up*" *is observed when* $A > 0.3691$.

As a second numerical experiment we allow two solitons of the same amplitude (of the form (3.3)) placed reasonably apart to move towards each other. After interaction the pulses emerge with their original shape and speed (see Fig. 3.2).

Finally we choose the initial data for the numerical scheme (2.4) from the single soliton solution of the B.B. equation ((3.1) with $b=\frac{1}{2}$). As time evolves the amplitude of the pulse drops while the breadth increases ($\int_{-\infty}^{+\infty} u\,dx$ is conserved with time). Also small pulses of increasing amplitudes appear in front of the main pulse, and the complete picture (Fig. 3.3) is similar to the *dispersive wave train* that arises from the K.d.V. equation (1.3) subject to the initial condition $u(x, 0) \leqq 0$ for all x (Hammack and Segur [5]). Another interesting aspect of this solution is that when two dispersive wave trains of the above form collide, the waves seem to emerge from the collision and propagate in their respective directions as if the collision had not taken place (Fig. 3.4).

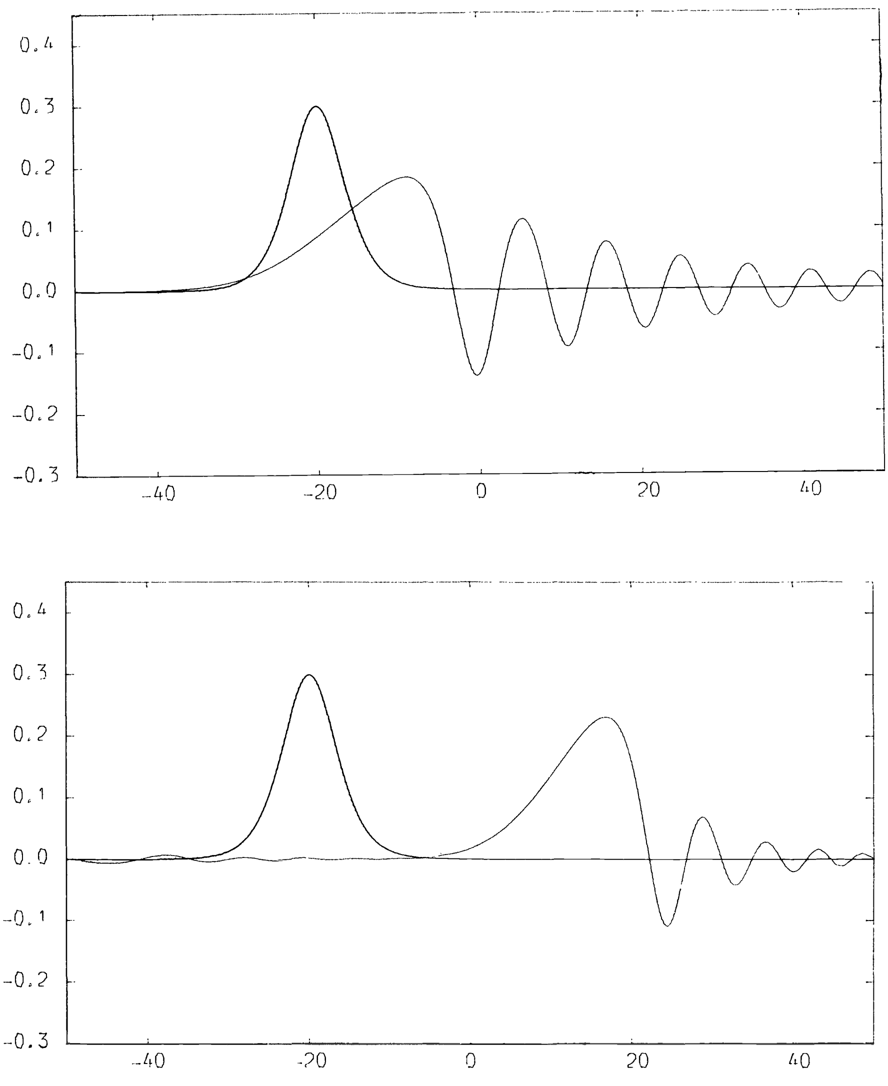


FIG. 3.3. *Dispersive wave trains of* G.B. *and* K.d.V. *equations.* $-50 \leqq x \leqq 50$, $h=0.25$, $k=0.1$, $t=30$.

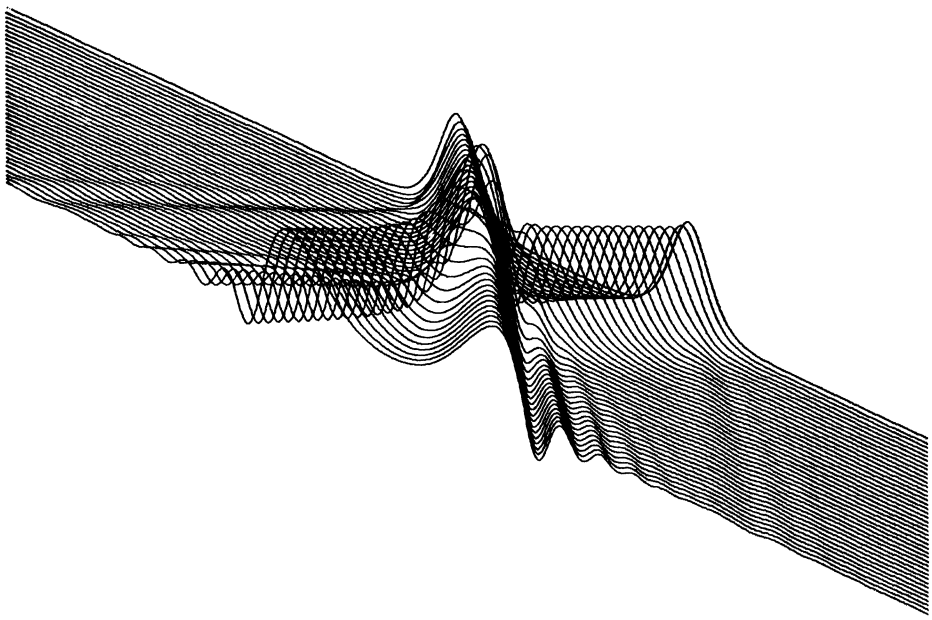


FIG. 3.4. *Collision of two dispersive wave trains.* $-100 \leqq x \leqq 100$, $0 \leqq t \leqq 40$, $h = 0.5$, $k = 0.2$, *initial amplitude* $= 0.2$. *Initial positions of the pulses are* $x = 0$ *and* $x = 50$.

4. The two-soliton solution. Following Hirota [6] who obtained a multi-soliton solution for the B.B. equation, we obtain for the G.B. equation the two-soliton solution

$$u = -6 \frac{\partial^2}{\partial x^2} \{\log_e f(x, t)\}$$

where

$$f(x, t) = 1 + \exp(\eta_1) + \exp(\eta_2) + a \exp(\eta_1 + \eta_2) \tag{4.1}$$

with

$$\begin{gathered} \eta_i = P_i\{x - \varepsilon_i(1 - P_i^2)^{1/2} t + \eta_i^0\} \qquad (\varepsilon_i = \pm 1), \\ a = \{(\varepsilon_1 v_1 - \varepsilon_2 v_2)^2 - 3(P_1 - P_2)^2\}/\{(\varepsilon_1 v_1 - \varepsilon_2 v_2)^2 - 3(P_1 + P_2)^2\} \end{gathered} \tag{4.2}$$

and

$$v_i = (1 - P_i^2)^{1/2}.$$

In the above $i = 1, 2$ and we note that

$$u = -6(ff_{xx} - f_x^2)/f^2 \tag{4.3}$$

and

$$P_i^2 = \tfrac{2}{3} A_i \tag{4.4}$$

where A_i is the amplitude of the ith soliton. (See § 3 for the single soliton.)

From now on we consider the case of two solitons travelling towards each other and so we put $\varepsilon_1 = +1$, $\varepsilon_2 = -1$. From (4.3), the solution u will not "blow up" if f retains the same sign for all x and t. This will certainly be true if $a \geqq 0$ (see (4.1)) which from (4.2) leads to either

(4.5) (i) $(v_1+v_2)^2>3(P_1+P_2)^2$,

or

(4.6) (ii) $(v_1+v_2)^2\leqq 3(P_1-P_2)^2$.

Note that, when $(v_1+v_2)^2=3(P_1+P_2)^2$, the solution u becomes undetermined. The inequality (4.5) gives

$$4(A_1+A_2)+3\sqrt{A_1A_2}-3<\sqrt{(3-2A_1)(3-2A_2)} \tag{4.7}$$

which covers the range $\frac{1}{4}<a<\infty$. For *real* solitons however, the values of A_1 and A_2 must lie in the shaded region in Fig. 4.1 in which $1\leqq a<\infty$. The inequality (4.6) which is valid for $0\leqq a<\frac{1}{4}$ does not lead to real solitons and so can be ignored. Finally, although it may not seem obvious from Fig. 4.1, it is easy to show that "blow-up" always occurs for $a<0$. Numerical experiments confirming the theoretical results for two solitons are now carried out using the numerical method outlined in (2.4) along with appropriate initial data.

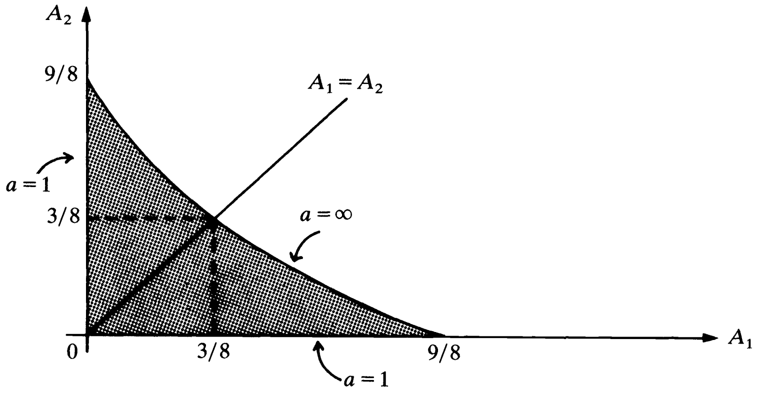


FIG. 4.1

Let us choose the initial data $f(x)$ and $g(x)$ from (4.3) with

$$P_1=P_2=0.4 \quad (\text{i.e. } A_1=A_2=0.24), \quad \varepsilon_1=1, \quad \varepsilon_2=-1.$$

The constants $\eta_i^0(i=1,2)$ are chosen in such a way that the pulses are well separated initially. After the nonlinear interaction the waves emerge without any change in shape and with an eventual speed of 0.9156 ($\sim(1-\frac{2}{3}\times 0.24)^{1/2}=0.91652$). Figure 4.2 illustrates this interaction, whilst Table 4.1 gives the L_2 and L_∞ errors at different times for various mesh sizes h and k. We observe that no significant improvement in the solution is obtained with the reduction of h. However, the reduction in k increases the accuracy of the numerical solution even though the number of time steps increases.

For the rest of the numerical experiments in this section $f(x)$ is chosen to be the linear superposition of two solitary waves of the form (3.3) moving towards each other, but placed well apart initially, and $g(x)$ the corresponding derivative.

When the initial amplitudes of the waves are equal (A, say) "blow up" of the solution is witnessed for A greater than some value A_m. A typical set of values for A_m is tabulated in Table 4.2 and the extrapolated value of A_m as $k\to 0$ is obtained as

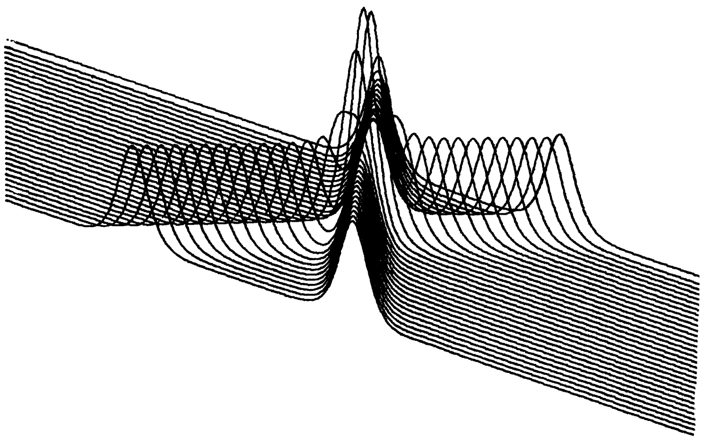


FIG. 4.2. *Interaction of two pulses. Initial conditions chosen from the exact formula* (4.3). $-100 \leqq x \leqq 100$, $0 \leqq t \leqq 60$, $h = 0.5$, $k = 0.1$, $A = 0.24$. *Initial positions of the pulses are* $x = 0$ *and* $x = 50$. *No difference is observed between this interaction and the two-soliton solution from* (4.3) (*cf. Table* 4.1).

TABLE 4.1

$-100 \leqq x \leqq 100$, $0 \leqq t \leqq 60$, $\eta_1^0 = 0, \eta_2^0 = -50$, $A_1 = A_2 = 0.24$

Error $\times 10^2$

	t	L_2	L_∞
	2	0.2586	0.0648
$h = 0.5$	20	3.1664	0.8419
$k = 0.2$	40	6.8686	1.4182
	60	9.0379	1.8051
	2	0.1359	0.0343
$h = 0.5$	20	1.3447	0.3503
$k = 0.1$	40	3.1411	0.6442
	60	4.0386	0.7976
	2	0.0747	0.0192
$h = 0.5$	20	0.5635	0.1417
$k = 0.05$	40	1.5595	0.3172
	60	2.0550	0.3859

TABLE 4.2

$-100 \leqq x \leqq 100$, $h = 0.5$, $x_{01} = 0, x_{02} = -50$

k	0.05	0.10	0.25
A_m	0.3691	0.3630	0.3425

0.37486, which agrees with the "blow up" condition that can be obtained from (4.5), namely $A > \frac{3}{8} = 0.375$. Unlimited growth of the amplitude for an initial sinusoidal disturbance for an equation related to (1.2) has been shown to exist by Kako and Watanabe [7] in the context of a beam-plasma system near the marginally stable state for periodic boundary conditions. This however is unrelated to the "blow up" mentioned above which is due to the singularity in the analytical expression and *not* due to any linear instability of the equation as was the case in Kako and Watanabe [7]. (x_{01} and x_{02} give the initial locations of the solitary waves.) It is also observed during these numerical experiments that the joint amplitude (A_J) of the interacting solitons when they collide and overlap is ***greater*** than twice the amplitude of the individual solitons which of course agrees with the analytical result; this is very different from the case of the B.B. equation where the joint amplitude is *less* than twice the amplitude of the individual solitons. Table 4.3 gives the values of A_J for a set of values of A. Numerical interactions of solitary waves of unequal amplitudes which do not satisfy the "blow up" conditions are performed as well and Figs. 4.3 and 4.4 illustrate these.

TABLE 4.3

$-100 \leqq x \leqq 100,\quad 0 \leqq t \leqq 60,\quad h = 0.5,$
$k = 0.1,\quad x_{01} = 0, x_{02} = -50$

A	$2A$	A_J
0.05	0.100	0.103
0.10	0.200	0.213
0.24	0.480	0.592
0.30	0.600	0.829
0.35	0.700	1.177

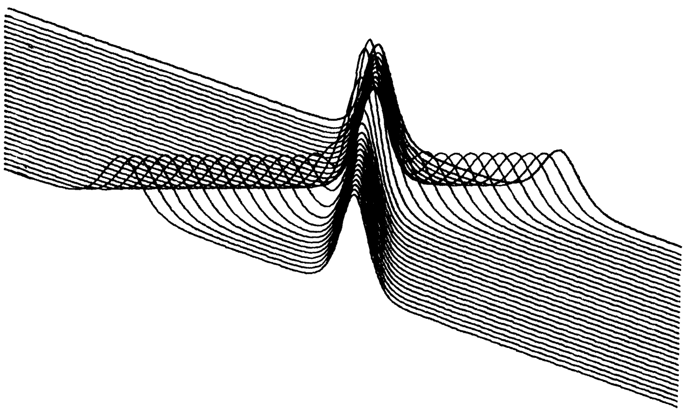


FIG. 4.3. *Interaction of two pulses of unequal amplitudes. Both the amplitudes are less than* 0.375, (*cf.* § 4) $-100 \leqq x \leqq 100$, $0 \leqq t \leqq 60$, $h = 0.5$, $k = 0.1$, *amplitudes* 0.24 *and* 0.135. *Respective initial positions are* $x = 0$ *and* $x = 50$.

5. The "good" Boussinesq equation as a system. As an alternative, the Cauchy problem involving the good Boussinesq equation (1.2) can be written in the form of

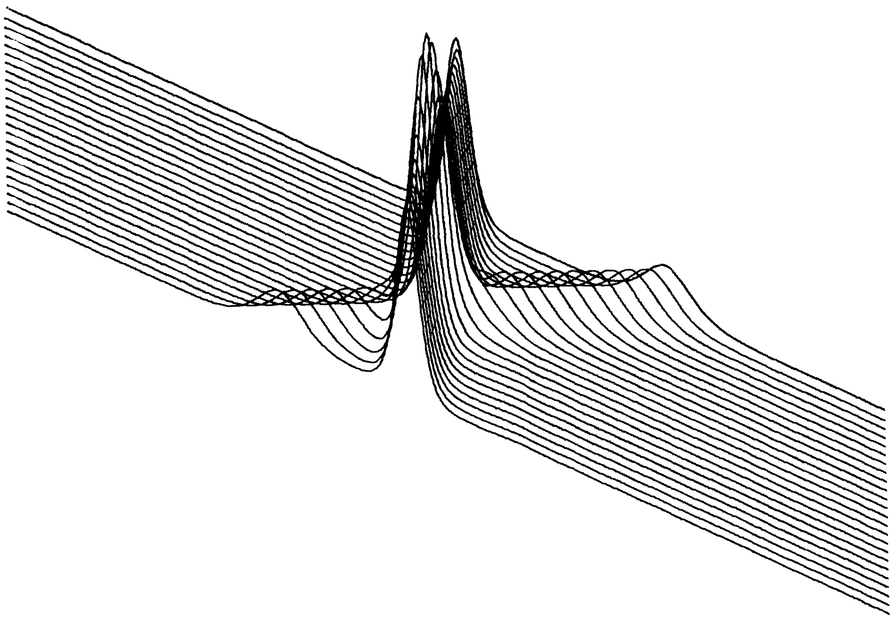


FIG. 4.4. *Interaction of two pulses of unequal amplitudes. One amplitude is greater than* 0.375, *while the other is less than* 0.375. $-100 \leqq x \leqq 100$, $0 \leqq t \leqq 60$, $h = 0.5$, $k = 0.05$, *amplitudes* 0.54 *and* 0.135. *Respective initial positions are* $x = 0$ *and* $x = 50$.

a Schrödinger type system

$$w_t = u_{xx} - u^2, \qquad u_t = -w_{xx}, \tag{5.1}$$

with initial conditions

$$u(x, 0) = f(x), \qquad w(x, 0) = -\iint g(x)\, dx\, dx. \tag{5.2}$$

In vector notation, (5.1) can be rewritten as the Schrödinger type system

$$\mathbf{u}_t + B\mathbf{u}_{xx} + \mathbf{F}(\mathbf{u}) = \mathbf{0} \tag{5.3}$$

where

$$\mathbf{u} = [u, w]^T, \quad B = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad \mathbf{F}(\mathbf{u}) = [0, u^2]^T.$$

We discretize (5.3) in space using a Galerkin finite element procedure based on linear "hat" basis functions and obtain the system of ordinary differential equations

$$M\frac{d\mathbf{U}}{dt} + \frac{1}{h^2} S\mathbf{U} + M\mathbf{F}(\mathbf{U}) = \mathbf{0} \tag{5.4}$$

where

$$M = \frac{1}{6}\begin{bmatrix} 2I & I & & & 0 \\ I & 4I & I & & \\ & \ddots & \ddots & \ddots & \\ & & I & 4I & I \\ 0 & & & I & 2I \end{bmatrix}, \qquad S = \begin{bmatrix} -B & B & & & 0 \\ B & -2B & B & & \\ & \ddots & \ddots & \ddots & \\ 0 & & B & -2B & B \\ & & & B & -B \end{bmatrix}.$$

with I the 2×2 unit matrix and h the grid spacing in x. The lower order of the system (5.3) compared to the scalar equation (1.2) allows us to use piecewise linear trial *and* test functions for the space discretization. In order to solve (5.4) we introduce a uniform time step k and use a two stage algorithm

$$(5.5)\qquad \begin{aligned} M\underset{\sim}{U}^* &= [M-\beta rS]\underset{\sim}{U}^n - \beta kM\underset{\sim}{F}(\underset{\sim}{U}^n),\\ \left[M+\frac{r}{2}S\right]\underset{\sim}{U}^{n+1} &= \left[M-\frac{r}{2}S\right]\underset{\sim}{U}^n - kM\underset{\sim}{F}\left(\frac{\underset{\sim}{U}^*+\underset{\sim}{U}^n}{2}\right)\end{aligned}\qquad n=0,1,2,\cdots,$$

where $\underset{\sim}{U}^n$ is an approximation to $\underset{\sim}{U}(nk)$, β is an arbitrary parameter and $r=k/h^2$ is the grid ratio. Complete details of the above numerical procedure for solving a system of the type (5.3) can be found in Griffiths, Mitchell and Morris [4].

When some of the numerical experiments carried out in § 4 are repeated using the scheme (5.5), the solutions are found to behave similarly except for some minor differences. As in the scalar case reduction of k gives more accurate results and for a single soliton, the value A_s, the bound on the amplitude, below which the pulse splits into two, is found to be 1.5150 for $h=0.5$ and $k=0.0625$ (compare with Table 3.1 where $A_s=1.5165$ for $h=0.5$ and $k=0.05$). The results for the dispersive wave trains are also similar. However, in the case of two solitons of equal amplitudes the "blow up" amplitude 0.375 is approached from above as k is reduced, whereas in the scalar case it is approached from below (see Table 4.2). Figures 5.1, 5.2 and 5.3 illustrate some of these findings. In all these experiments $\beta=1$.

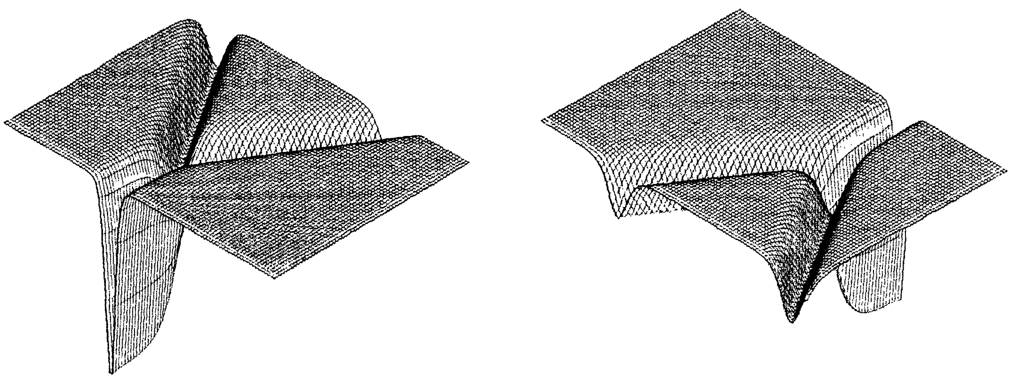


FIG. 5.1. *Break up of the pulse.* $-50\leqq x\leqq 75$, $0\leqq t\leqq 48.75$, $h=0.5$, $k=0.0625$, *amplitude* $=1.5150$, $\beta=1.0$. *Initial position of the pulse is* $x=0$. (*Compare with Fig.* 3.1.)

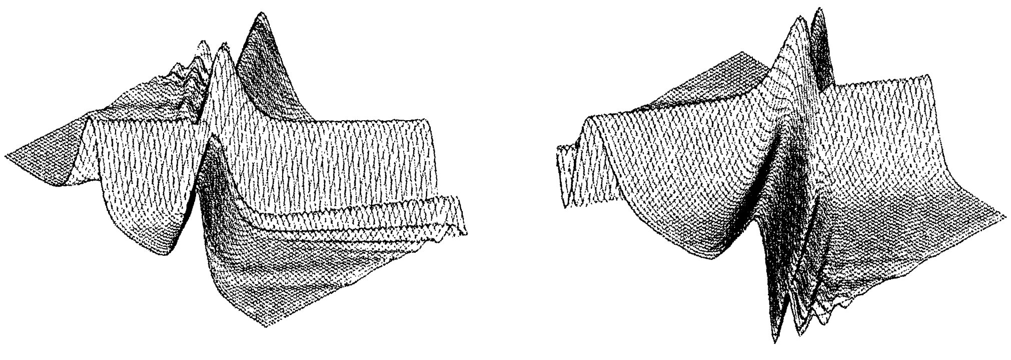


FIG. 5.2. *Collision of two dispersive wave trains.* $-50\leqq x\leqq 75$, $0\leqq t\leqq 48.75$, $\beta=1.0$, $h=0.625$, $k=0.25$, *initial amplitude* $=0.2$. *Initial positions of the pulses are* $x=-15$ *and* $x=30$. (*Compare with Fig.* 3.4.)

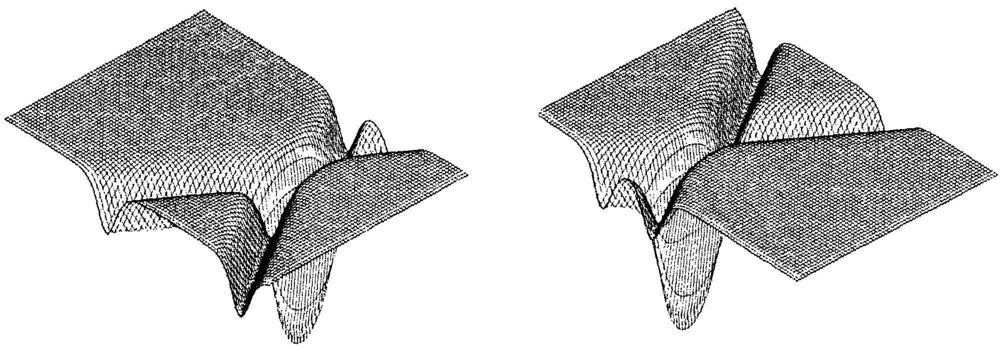


FIG. 5.3. *Interaction of two pulses.* $-50 \leqq x \leqq 75$, $0 \leqq t \leqq 48.75$, $\beta = 1.0$, $h = 0.5$, $k = 0.0625$, $A = 0.376$. *Initial positions of the pulses are* $x = -15$ *and* $x = 30$. "*Blow up*" *is observed when* $A > 0.380$. (*Compare with Fig.* 3.2.)

Acknowledgment. The authors are indebted to Professor B. D. Sleeman for a number of stimulating discussions.

REFERENCES

[1] M. J. ABLOWITZ AND H. SEGUR, *Solitons and the Inverse Scattering Transform*, SIAM Studies in Applied Mathematics 4, Society for Industrial and Applied Mathematics, Philadelphia, 1981.
[2] R. C. Y. CHIN, G. W. HEDSTROM AND K. E. KARLSSON, *A simplified Galerkin method for hyperbolic equations*, Math. Comp., 33 (1979), pp. 647–658.
[3] I. CHRISTIE, D. F. GRIFFITHS, A. R. MITCHELL AND J. M. SANZ-SERNA, *Product approximation for nonlinear problems in the finite element method*, IMA J. Numer. Anal., 1 (1981), pp. 253–266.
[4] D. F. GRIFFITHS, A. R. MITCHELL AND J. Ll. MORRIS, *A numerical study of the nonlinear Schrödinger equation*, Dundee University, Mathematics Department Report NA/52, 1982.
[5] J. L. HAMMACK AND H. SEGUR, *The Korteweg-deVries equation and water waves, Part* 2, *comparison with experiments*, J. Fluid Mech., 65 (1974), pp. 289–314.
[6] R. HIROTA, *Exact N-soliton solutions of the wave equation of long waves in shallow-water and in nonlinear lattices*, J. Math. Phys., 14 (1973), pp. 810–814.
[7] M. KAKO AND T. WATANABE, *Nonlinear behaviors of ion-waves*, J. Phys. Soc. Japan, 32 (1972), pp. 535–542.
[8] V. K. KALANTAROV AND O. A. LADYZHENSKAYA, *Occurrence of blow-up for quasilinear* P.D.E.*s of parabolic and hyperbolic types*, Zap. Nauk Sem. LOMI, 69 (1977), pp. 77–102. (In Russian.)
[9] G. L. LAMB, JR., *Elements of Soliton Theory*, John Wiley, New York 1980.
[10] H. P. MCKEAN, *Boussinesq's equation on the circle*, Comm. Pure Appl. Math., 34 (1981), pp. 599–691.
[11] A. R. MITCHELL AND D. F. GRIFFITHS, *The Finite Difference Method in Partial Differential Equations*, John Wiley, New York, 1980.
[12] A. R. MITCHELL AND S. W. SCHOOMBIE, *Finite element studies of solitons*, Dundee University, Mathematics Department Report NA/47, 1981.
[13] A. C. SCOTT, F. Y. F. CHU AND D. W. MCLAUGHLIN, *The soliton: A new concept in applied science*, Proc. IEEE, 61 (1973), pp. 1443–1483.
[14] B. SWARTZ AND B. WENDROFF, *Generalised finite difference schemes*, Math. Comp., 23 (1969), pp. 37–40.
[15] C. TOMEI, *The Boussinesq equation*, Ph.D. thesis, New York Univ., New York, 1982.
[16] G. B. WHITHAM, *Linear and Nonlinear Waves*, Wiley-Interscience, New York, 1974.
[17] V. E. ZAKHAROV, *Randomization problem for 1-dimensional chains of nonlinear operators*, Zh. E.T.F., 65 (1973), pp. 219–228. (In Russian.)

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

FINITE DIFFERENCE SOLUTIONS FOR INTERNAL WAVES IN ENCLOSURES*

HOWARD R. BAUM† AND RONALD G. REHM‡

Abstract. Finite difference approximations to the set of partial differential equations governing internal waves are investigated. Analytical solutions describing waves in an enclosure in two and three dimensions are obtained. The schemes considered are second order accurate in space and include first order explicit and second order time differencing. The solutions are used to investigate the temporal stability and long term accuracy of all schemes. The mode frequencies and wave shapes obtained from each difference scheme are compared with the solutions both to the corresponding partial differential equations and to equations obtained by discretizing in space only. The solutions have been used by the authors to help develop a finite difference code designed to compute nonlinear buoyancy driven flows of the type that arise in enclosure fires.

Key words. analytical solutions, finite difference equations, internal waves, numerical solution, stability, stratified fluid

1. Introduction. Development of finite difference solutions to nonlinear partial differential equations requires great care. Generally, assessment of the stability and accuracy of numerical solutions represents a major task, because no independent solutions are available against which to make comparisons. Solutions to linearized versions of the differential equations then become a major analytical tool for determining stability and accuracy. When analytical solutions of equations representing finite difference approximations to the linearized equations can be found, these solutions represent a very valuable tool with which to test computational procedures and programming.

In this paper solutions to finite difference approximations to the linear equations describing internal waves in a stratified fluid are given, and the properties of these solutions are discussed. In § 2 the continuous problem, both the linear partial differential equations and their solutions, is presented. This problem and its solution is well known, dating back to the turn of the century; several excellent texts discuss internal waves [4], [6], [7], [11]–[13], and areas of application in which they arise. Discrete approximations to the linear equations are given in § 3, and the corresponding solutions are presented in § 4. These schemes are dispersive but not dissipative; an interesting review of the features of dispersive schemes has recently been published [10]. In § 5 the solutions to the discrete problems are discussed and compared with the solution of the continuous problem. These analytical results have been compared with results obtained by straight numerical solution of one set of discretized equations; in the concluding remarks, § 6, this comparison is discussed.

The comparison represents one of the most valuable reasons for conducting such an analysis: it provides a basis for assessing both the stability and accuracy of several candidate difference schemes for the solution of time dependent nonlinear problems. In particular, it tests all the transient terms, the convection of density, and the buoyancy force. Moreover, it provides a test case for the computation of the pressure perturbation, which required the development of a nonseparable elliptic equation solver [5]. Finally, the interaction of all these terms is tested in the context of a physical phenomenon (internal wave motion) which is widely encountered in the study of buoyancy driven flows.

* Received by the editors April 5, 1983, and in revised form August 2, 1983.

† Center for Fire Research, National Bureau of Standards, Washington, DC 20234.

‡ Center for Applied Mathematics, National Bureau of Standards, Washington, DC 20234.

2. Formulation of continuous equations and continuous solution. The problem may be stated as follows: Consider a room of unit height $0 \leqq y \leqq 1$ and of rectangular planform $0 \leqq x \leqq l$, $0 \leqq z \leqq d$. The gas in the room is stably stratified vertically with a known density $\rho_0(y)$, normalized to unity at the floor (see Fig. 1). If the height is not too large, the hydrostatic pressure variation associated with the density $\rho_0(y)$ is small compared with the ambient pressure. The low frequency fluid motion induced by a small disturbance may then be determined from the solution of the system [4]:

$$\frac{\partial \rho}{\partial t} + v\frac{d\rho_0}{dy} = 0, \quad \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0,$$
(1)
$$\rho_0\frac{\partial u}{\partial t} + \frac{\partial p}{\partial x} = 0, \quad \rho_0\frac{\partial v}{\partial t} + \frac{\partial p}{\partial y} + \rho = 0, \quad \rho_0\frac{\partial w}{\partial t} + \frac{\partial p}{\partial z} = 0.$$

The first equation of (1) is a linearized statement of incompressibility, while the second ensures conservation of mass for an incompressible fluid. The last three equations are linearized momentum equations in the horizontal (x-z) plane and in the vertical (y) direction. Here, u, v, and w are respectively the x, y, and z components of the velocity perturbation, ρ is the density perturbation, and p the nonhydrostatic pressure perturbation. The time scale t is set by the unit height and the requirement that the gravitational acceleration is unity in the scaled variables. The density perturbation is normalized so that it is a small fraction of the ambient floor density while the velocity perturbations are correspondingly scaled with respect to the ratio of length to time scales. The pressure perturbation is small compared with the product of the ambient floor density and the square of the velocity scale.

The boundary conditions appropriate to the system (1) are that the normal component of the perturbation velocity **u** vanishes at the enclosure walls. Two types of initial disturbances are of interest, vertical displacements and velocities. Let $\zeta(x, y, z, t)$ be the vertical displacement of a fluid element. Within the linear approxima-

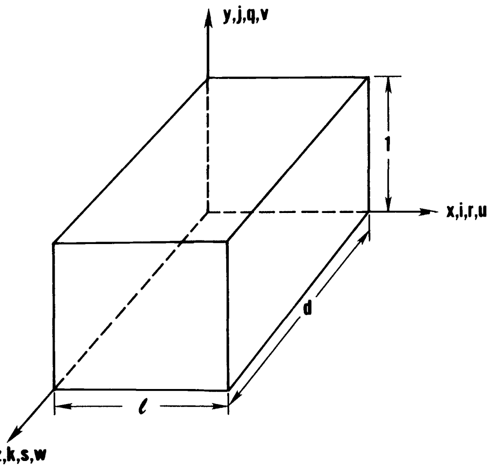


FIG 1. *An enclosure of unit height and rectangular planform $0 \leqq x \leqq l$, $0 \leqq z \leqq d$. The fluid in the room is stably stratified vertically with density $\rho_0(y)$. The velocity components of the internal waves are u, v and w; the modal wavenumbers are r, s and q; and the integer discretization variables are i, j and k where $1 \leqq i \leqq N$, $1 \leqq j \leqq M$ and $1 \leqq k \leqq L$ in the x-, y- and z-directions respectively.*

tion ζ is related to the vertical velocity by

$$\frac{\partial \zeta}{\partial t} = v. \tag{2}$$

From the first equation of (1),

$$\zeta \frac{d\rho_0}{dy} + \rho = 0. \tag{3}$$

Thus, specifying the vertical displacement field initially is equivalent to prescribing the initial density perturbations. Alternatively, the velocity disturbance could be prescribed initially. This choice is not entirely arbitrary, however. The initial velocity field must be divergence free and satisfy the boundary conditions.

The pressure perturbation p cannot be prescribed arbitrarily initially. An equation for p can be obtained by dividing the momentum equations by ρ_0 and taking the divergence of the resulting system to obtain

$$\nabla \cdot \left(\frac{1}{\rho_0} \nabla p \right) = -\frac{\partial}{\partial y} \left(\frac{\rho}{\rho_0} \right). \tag{4}$$

Boundary conditions appropriate to (4) can be obtained from the momentum equations and the velocity boundary conditions. Let $\mathbf{n}$ be the unit outward normal to the enclosure and $\mathbf{j}$ the unit vector in the vertical direction. Then, at any point on the surface:

$$\frac{\partial p}{\partial n} + \mathbf{n} \cdot \mathbf{j} \rho = 0. \tag{5}$$

Equations (4) and (5) show that the pressure at any instant of time is determined by the density distribution at that instant. In particular, if ρ is initially zero, so is the pressure.

The system of homogeneous equations and boundary conditions can be reduced to an eigenvalue problem. Since the solution to any initial value problem of interest can be represented as a sum of eigenmodes, the degree of accuracy to which any numerical method represents the continuous solution is determined by the ability of the method to compute individual eigenmodes. In the following sections of the paper, the eigenmodes of the continuous problem are first displayed and the chief features noted. Next, several candidate difference schemes are proposed and the corresponding solutions to the difference equations are obtained analytically. These solutions are then used to evaluate the stability and accuracy of the difference schemes.

3. The continuous solutions. Each eigenmode is constrained by (1) and the boundary conditions to have the form

$$\begin{aligned}
u &= \{a \cos(\sigma t) - b \sin(\sigma t)\} \sin(r\pi x/l) \cos(s\pi z/d) U(y),\\
v &= \{a \cos(\sigma t) - b \sin(\sigma t)\} \cos(r\pi x/l) \cos(s\pi z/d) V(y),\\
w &= \{a \cos(\sigma t) - b \sin(\sigma t)\} \cos(r\pi x/l) \sin(s\pi z/d) W(y),\\
p &= \{a \sin(\sigma t) + b \cos(\sigma t)\} \cos(r\pi x/l) \cos(s\pi z/d) P(y),\\
\rho &= \{a \sin(\sigma t) + b \cos(\sigma t)\} \cos(r\pi x/l) \cos(s\pi z/d) R(y).
\end{aligned} \tag{6}$$

Substitution of above forms into (1) yields a system of ordinary differential equations which can be manipulated into the following eigenvalue problem for $V(y)$.

$$\begin{aligned}&\frac{d^2V}{dy^2}-N^2(y)\frac{dV}{dy}-\tilde{\lambda}^2[1-N^2/\sigma^2]V=0,\\ &N^2(y)=-(\rho_0)^{-1}\frac{d\rho_0}{dy},\\ &\tilde{\lambda}^2=(r\pi/l)^2+(s\pi/d)^2,\\ &V(0)=V(1)=0.\end{aligned}\tag{7}$$

The eigenvalue to be determined in (7) is the angular frequency σ of the mode under consideration. The coefficients a and b in (6) are the amplitudes of the initial velocity and displacement disturbances respectively. The quantity $N(y)$ is the buoyancy (Brunt–Vaisala) frequency. The y dependence of the mode shape is determined by the ambient stratification $\rho_0(y)$:

$$\rho_0(y)=e^{-\alpha y}.\tag{8}$$

The form yields the simplest solutions describing internal waves in an enclosure:

$$V(y)=\exp(\alpha y/2)\sin(q\pi y),\tag{9a}$$

$$\sigma^2=\frac{\alpha\tilde{\lambda}^2}{(\alpha^2/4)+(q\pi)^2+\tilde{\lambda}^2}.\tag{9b}$$

The flow pattern generated by the solution given in (9) is illustrated in two dimensions by Prandtl [7]. The results are repeated in Turner [11], who gives numerous references to other studies of internal waves. There are two points worth noting for later reference. First, the motion is cellular, each mode containing cells with mode numbers srq. The "resolving power" of any finite difference grid can then be discussed in terms of the number of cells that the grid can approximate to a given accuracy. Second, the eigenvalue σ represents the frequency of oscillation of the mode. Since the frequency is different for each mode, it is necessary to avoid computational procedures which introduce substantial phase shift. Such phase shifts will inevitably throw the modes out of synchronization, leading to meaningless results. *This can occur even if the spatial resolution of each mode is calculated with great accuracy.*

The solution given above represents the standard against which those produced by the finite difference computations must be tested. We now turn to the finite difference formulation of the problem.

4. Discrete equations. In this section several finite difference approximations are presented for linear equations (1) and boundary conditions of no outflow at the boundaries. The difference approximations are either first-order or second-order accurate in time and are second-order accurate in space. Two spatial discretizations are considered for the flux term in the first equation of (1), a central difference and a conservation form; all other spatial derivatives are approximated by their natural second-order differences. The difference equations are written on a "staggered mesh" in the style of [2].

For the finite difference scheme, the enclosure is overlaid with a uniform mesh: in general there are N mesh cells in the x-direction, M mesh cells and the y-direction and L mesh cells in the z-direction. In Fig. 2 a typical mesh cell is shown, illustrating

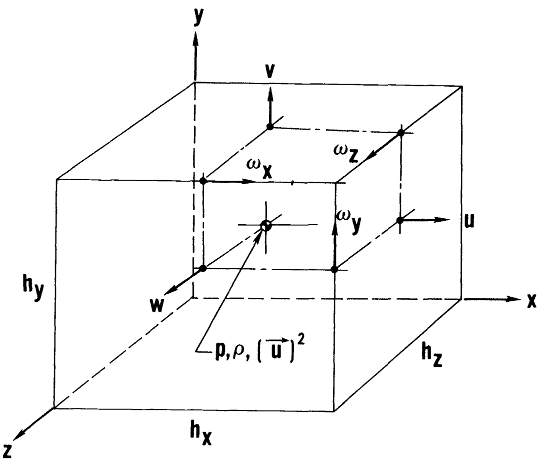


FIG. 2. *A typical mesh cell illustrating where all dependent variables in the finite difference scheme are defined.*

where all of the dependent variables in the finite difference scheme are defined relative to the cell.

The following discretely evaluated functions will denote approximations to the corresponding analytical solutions to (1):

$$\begin{aligned}
u_{ijk}^n &\cong u(ih_x, (j-1/2)h_y, (k-1/2)h_z, n\delta),\\
v_{ijk}^n &\cong v((i-1/2)h_x, jh_y, (k-1/2)h_z, n\delta),\\
w_{ijk}^n &\cong w((i-1/2)h_x, (j-1/2)h_y, kh_z, n\delta),\\
p_{ijk}^n &\cong p((i-1/2)h_x, (j-1/2)h_y, (k-1/2)h_z, n\delta),\\
\rho_{ijk}^n &\cong \rho((i-1/2)h_x, (j-1/2)h_y, (j-1/2)h_z, n\delta),\\
\rho_{0,j} &\cong \rho_0((j-1/2)h_y) \equiv \rho_0(j).
\end{aligned}$$

Furthermore,

$$u_{ijk} \cong u(ih_x, (j-1/2)h_y, (k-1/2)h_z, t), \quad \text{etc.}$$

represents approximating functions which are discretely evaluated in space but continuous in time.

Here $h_x = l/N$, $h_y = 1/M$ and $h_z = d/L$ are the mesh cell sizes in the x-, y- and z-directions respectively and δ is the time-step size so that $n\delta = t$.

With this notation (1) are approximated in the following manner. At first we consider discretization in space, the approximations remaining continuous in time. As noted above, all spatial derivatives, except the flux term in the first of (1), are approximated by a difference that is second-order accurate. Therefore, for example $\partial u/\partial x$, which is to be evaluated at the center of a mesh cell, is approximated naturally as

$$\left(\frac{\partial u}{\partial x}\right)_{ijk} \cong \frac{u_{ijk} - u_{i-1,jk}}{h_x}.$$

Then (1) are approximated by the discrete-continuous equations

$$\frac{\partial \rho_{ijk}}{\partial t}+\left(v\frac{d\rho_0}{dy}\right)_{ijk}=0,$$

$$\frac{u_{ijk}-u_{i-1,jk}}{h_x}+\frac{v_{ijk}-v_{i,j-1,k}}{h_y}+\frac{w_{ijk}-w_{ij,k-1}}{h_z}=0,$$

$$\rho_0(j)\frac{\partial u_{ijk}}{\partial t}+\frac{p_{i+1,jk}-p_{ijk}}{h_x}=0, \tag{10}$$

$$\frac{1}{2}[\rho_0(j)+\rho_0(j+1)]\frac{\partial v_{ijk}}{\partial t}+\frac{p_{i,j+1,k}-p_{ijk}}{h_y}+\frac{1}{2}(\rho_{i,j+1,k}+\rho_{ijk})=0,$$

$$\rho_0(j)\frac{\partial w_{ijk}}{\partial t}+\frac{p_{ij,k+1}-p_{ijk}}{h_z}=0.$$

Boundary conditions are

$$\begin{aligned} u_{0,jk}&=u_{Njk}=0, & 1\leqq j\leqq M, 1\leqq k\leqq L\\ v_{i,0,k}&=v_{i,M,k}=0, & 1\leqq i\leqq N, 1\leqq k\leqq L\\ w_{ij,0}&=w_{ij,L}=0, & 1\leqq i\leqq N, 1\leqq j\leqq M.\end{aligned} \tag{11}$$

The flux term $(v\,d\rho_0/dy)_{ijk}$ is evaluated at the center of a cell, and two difference approximations are used.

(i) Central differences:

$$\left(v\frac{d\rho_0}{dy}\right)_{ijk}\cong\frac{1}{2}(v_{ijk}+v_{i,j-1,k})\frac{\rho_0(j+1)-\rho_0(j-1)}{2h_y}. \tag{12}$$

(ii) Conservative differences:

$$\left(v\frac{d\rho_0}{dy}\right)_{ijk}\cong\frac{1}{2}\left[v_{ijk}\frac{\rho_0(j+1)-\rho_0(j)}{h_y}+v_{i,j-1,k}\frac{\rho_0(j)-\rho_0(j-1)}{h_y}\right]. \tag{13}$$

The last difference approximation is called a conservation form because it can be derived by subtracting the difference form of the zero-divergence relation for the velocity (incompressibility condition) from a difference form of the continuity equation in conservation form. In particular, this scheme conserves the total mass in the enclosure.

Solutions with three forms of behavior with time are found. First, solutions are found which depend on time as a continuous variable, namely solutions of (10) with either (12) or (13). Then solutions are found when the dependent variables are completely discretized using approximations to the time derivatives which are either first-order or second-order accurate. When first-order approximations are used, the time differences in (10) are replaced by differences in the following manner:

$$\frac{d\rho_{ijk}}{dt}\to\frac{\rho_{ijk}^{n+1}-\rho_{ijk}^{n}}{\delta},$$

and all other variables are evaluated at time level n except for the buoyancy term (the last term) in the fourth of (10) which is evaluated at $n+1$ so that the difference approximation is stable (as will be shown in the next section). When second-order approximations, leap-frog, are used, all time derivatives are approximated as follows:

$$\left(\frac{\partial \rho_{ijk}}{\partial t}\right)^n \cong \frac{\rho_{ijk}^{n+1} - \rho_{ijk}^{n-1}}{2\delta}$$

and the buoyancy term is now evaluated at time level n.

For both first-order and second-order time differencing, the flux term is given by (12) or (13) with the vertical velocities evaluated at time level n. The boundary conditions are (11) approximated at time level n.

5. Solutions to the finite difference equations. In this section solutions will be derived to the difference equations presented in the last section. The combinations of spatial and temporal difference schemes considered are listed in Table 1. These equations are linear and separable. They have coefficients which depend only on j so that the dependence of each solution on i, k and t (or n) is simple.

TABLE 1
Table of methods.

Temporal difference \ Spatial difference	Second-order conservative	Second-order central
Continuous	A (38a) and (40)	B (38a) and (39)
First-order semi-implicit	C (38b) and (40)	D (38b) and (39)
Second-order leap-frog	E (38c) and (40)	F (38c) and (39)

For (10) with either (12) or (13), the following form of solution can be assumed

$$\begin{aligned}
u_{ijk} &= \sin\left(\frac{r\pi i}{N}\right)\cos\left[\frac{s\pi}{L}(k-1/2)\right]U_j f(t),\\
v_{ijk} &= \cos\left[\frac{r\pi}{N}(i-1/2)\right]\cos\left[\frac{s\pi}{L}(k-1/2)\right]V_j f(t),\\
w_{ijk} &= \cos\left[\frac{r\pi}{N}(i-1/2)\right]\sin\left(\frac{s\pi k}{L}\right)W_j f(t),\\
p_{ijk} &= \cos\left[\frac{r\pi}{N}(i-1/2)\right]\cos\left[\frac{s\pi}{L}(k-1/2)\right]P_j g_p(t),\\
\rho_{ijk} &= \cos\left[\frac{r\pi}{N}(i-1/2)\right]\cos\left[\frac{s\pi}{L}(k-1/2)\right]R_j g_\rho(t).
\end{aligned} \tag{14}$$

This solution form has been assumed because it satisfies the first and last boundary conditions (11) on u_{ijk} and w_{ijk}. The form of the dependence upon i and k of the remaining variables follows from the equations. As in the case of the continuous problem, the solution represents a (discretized approximation to a) wave mode which is an eigenmode with r half-wavelengths in the i-direction and s half-wavelengths in the k-direction of the problem. The frequency of oscillation σ will appear as the eigenvalue.

Substitution of (14) into (10) yields the following reduced equations:

$$
\begin{aligned}
&R_j\frac{dg_p}{dt}+F_jf(t)=0,\\
&\frac{2}{h_x}\sin\left(\frac{r\pi}{2N}\right)U_j+\frac{1}{h_y}(V_j-V_{j-1})+\frac{2}{h_z}\sin\left(\frac{s\pi}{2L}\right)W_j=0,\\
(15)\qquad &\rho_0(j)U_j\frac{df}{dt}-\frac{2}{h_x}\sin\left(\frac{r\pi}{2N}\right)P_jg_p(t)=0,\\
&\frac{1}{2}[\rho_0(j)+\rho_0(j+1)]V_j\frac{df}{dt}+\frac{1}{h_y}(P_{j+1}-P_j)g_p(t)+(1/2)(R_{j+1}+R_j)g_p(t)=0,\\
&\rho_0(j)W_j\frac{df}{dt}-\frac{2}{h_z}\sin\left(\frac{s\pi}{2L}\right)P_jg_p(t)=0,
\end{aligned}
$$

where the flux term F_j

(i) for the centered difference scheme is

$$
(16a)\qquad F_j=\frac{1}{2}[V_j+V_{j-1}]\left[\frac{\rho_0(j+1)-\rho_0(j-1)}{2h_y}\right];
$$

(ii) for the conservative difference scheme is

$$
(16b)\qquad F_j=\frac{1}{2h_y}\{V_j[\rho_0(j+1)-\rho_0(j)]+V_{j-1}[\rho_0(j)-\rho_0(j-1)]\}.
$$

Now consider the time dependence $f(t)$ and $g(t)$ and note that these two functions will be related exactly as in the continuous solution, (6). Hence, take

$$
(17)\qquad
\begin{aligned}
f(t)&=A_{rqs}\cos(\sigma_\alpha t)-B_{rqs}\sin(\sigma_\alpha t),\\
g_p(t)=g_p(t)&=A_{rqs}\sin(\sigma_\alpha t)+B_{rqs}\cos(\sigma_\alpha t)
\end{aligned}
$$

where A_{rqs} and B_{rqs} are two arbitrary constants representing the internal-wave mode amplitude for the mode (r, q, s). σ_α represents either σ_A or σ_B, the eigenfrequency for the mode (r, q, s) (see Table 1). The frequency σ_A corresponds to the choice of the centered difference scheme used for the flux equation (16a), and σ_B is this frequency when the conservative difference scheme is used, (16b). When equations (17) are substituted into (15), these equations become:

$$
\begin{aligned}
&R_j\sigma_\alpha+F_j=0,\\
&\frac{2}{h_x}\sin\left(\frac{r\pi}{2N}\right)U_j+\frac{2}{h_z}\sin\left(\frac{s\pi}{2L}\right)W_j+\frac{1}{h_y}(V_j-V_{j-1})=0,\\
(18)\qquad &-\rho_0(j)U_j\sigma_\alpha-\frac{2}{h_x}\sin\left(\frac{r\pi}{2N}\right)P_j=0,\\
&-\frac{1}{2}[\rho_0(j)+\rho_0(j+1)]V_j\sigma_\alpha+\frac{1}{h_y}(P_{j+1}-P_j)+\frac{1}{2}(R_{j+1}+R_j)=0,\\
&-\rho_0(j)W_j\sigma_\alpha-\frac{2}{h_z}\sin\left(\frac{s\pi}{2L}\right)P_j=0.
\end{aligned}
$$

At this point, we note that the time dependences for the totally discretized systems can be obtained simply by the following observations. When the system of equations is totally discretized, the solution has the same form as (14) with $f(t)$ replaced by f^n, $g_p(t)$ replaced by g_p^n and $g_\rho(t)$ replaced by g_ρ^n. When this form is substituted into (10) with the time derivatives replaced by the first-order differences, a semi-implicit system of equations arises. The semi-implicit nature of these equations shows up because of the buoyancy term in the vertical momentum equation where g_ρ^{n+1} is found. If this term were taken to be g_ρ^n rather than g_ρ^{n+1}, so that the scheme is completely explicit, then, as will be seen below, the difference scheme is unstable. Similarly, if assumed forms (14) with $f(t)\to f^n$, $g_\rho(t)\to g_\rho^n$ and $g_p(t)\to g_p^n$, are substituted into the leap-frog scheme, the equations which result are the same as (15), but all time derivatives are replaced by leap-frog time differences, namely

$$\frac{dg_\rho}{dt}\to\frac{g_\rho^{n+1}-g_\rho^{n-1}}{2\delta},\quad \text{etc.},$$

and in the last term of the fourth equation in (15), $g_\rho\to g_\rho^n$.

With these identifications, the first-order, semi-implicit method has the following dependences upon discretized time $n\delta$:

$$\begin{aligned} f^n &= A_{rqs}\cos[\sigma_\alpha\delta(n-1/2)]-B_{rqs}\sin(\sigma_\alpha\delta n),\\ g_\rho^n &= A_{rqs}\sin[\sigma_\alpha\delta(n-1)]+B_{rqs}\cos[\sigma_\alpha\delta(n-1/2)],\\ g_p^n &= A_{rqs}\sin(\sigma_\alpha\delta n)+B_{rqs}\cos[\sigma_\alpha\delta(n+1/2)]. \end{aligned} \tag{19}$$

As before, A_{rqs} and B_{rqs} are arbitrary constants and σ_α represents either σ_C, the eigenfrequency for the conservative difference form for the flux term or σ_D, the central difference form for the flux.

Similarly for the leap-frog, the dependences upon discretized time can be shown to be

$$\begin{aligned} f^n &= A_{rqs}\cos(n\sigma_\alpha\delta)-B_{rqs}\sin(n\sigma_\alpha\delta)+(-1)^nC_{rqs}\cos(n\sigma_\alpha\delta)-(-1)^nD_{rqs}\sin(n\sigma_\alpha\delta)\\ g_p^n &= g_\rho^n = A_{rqs}\sin(n\sigma_\alpha\delta)+B_{rqs}\cos(n\sigma_\alpha\delta)+(-1)^{n+1}C_{rqs}\sin(n\sigma_\alpha\delta)\\ &\quad +(-1)^{n+1}D_{rqs}\cos(n\sigma_\alpha\delta). \end{aligned} \tag{20}$$

Here A_{rqs}, B_{rqs}, C_{rqs} and D_{rqs} are all arbitrary constants. Since the leap-frog is a three level scheme, there are four independent solutions to the homogeneous equations and hence four arbitrary constants. The two solutions multiplied by $(-1)^n$ are parasitic solutions to the difference equations which do not approximate the solutions to the differential equations. They are solely due to the fact that the leap-frog difference scheme is of higher order than the original continuous system. In choosing initial conditions, we desire to make C_{rqs} and D_{rqs} small to suppress these parasitic solutions. As before σ_α represents either σ_E (the conservative scheme) or σ_F (the central difference scheme).

Substitution of (19) into the first-order discretized equations in time yields (18) with σ_α replaced by $(2/\delta)\sin(\sigma_\alpha\delta/2)$. Similarly, substitution of the time dependences given in (20) into the version of (15) corresponding to a leap-frog discretization in time also yields (18) with σ_α now replaced by $(1/\delta)\sin(\sigma_\alpha\delta)$. Therefore, the eigenfrequency σ_α in (18) will be determined when $\alpha=A$ or B and all other eigenvalues, for $\alpha=C$, D, E and F, can be determined from these values by simple operations.

Determination of the j dependence of the dependent variables and of the eigenfrequencies σ_α depends upon the form considered for F_j. Therefore, the two solutions will be treated separately; but first, note that for either form, the continuity equation and the two horizontal momentum equations (the second, third and fifth of (18)) can be used to eliminate U_j and V_j, yielding

(21) $$-\frac{\lambda^2}{\rho_0(j)\sigma_\alpha}P_j+\frac{1}{h_y}(V_j-V_{j-1})=0$$

where

(22) $$\lambda^2=4\left[\left(\frac{1}{h_x}\sin\frac{r\pi}{2N}\right)^2+\left(\frac{1}{h_z}\sin\frac{s\pi}{2L}\right)^2\right].$$

First, the central difference approximation for the flux, (16a), is considered. Then, the first and fourth equations of (18) and (21) are three relations for V_j, P_j and R_j:

(23) $$\begin{aligned}&R_j\sigma_\alpha+\frac{1}{2}(V_j+V_{j-1})\frac{1}{2h_y}[\rho_0(j+1)-\rho_0(j-1)]=0,\\&\frac{1}{2}[\rho_0(j)+\rho_0(j+1)]V_j\sigma_\alpha+\frac{1}{h_y}(P_{j+1}-P_j)+\frac{1}{2}(R_{j+1}+R_j)=0,\\&\lambda^2P_j-\frac{\rho_0(j)\sigma_\alpha}{h_y}(V_j-V_{j-1})=0.\end{aligned}$$

Both R_j and P_j can be eliminated using these relations to obtain a single equation for V_j.

For a simple, exponential ambient stratification

(24) $$\rho_0(j)=\exp\{-\alpha h_y(j-\tfrac{1}{2})\}$$

and a change of variables

(25) $$V_j=\exp\left(\frac{\alpha h_y j}{2}\right)\tilde{V}_j$$

this equation becomes

(26) $$\begin{aligned}&\tilde{V}_j\cosh\left(\frac{\alpha h_y}{2}\right)\left[\frac{h_y^2\lambda^2}{2\sigma_\alpha^2}\sinh(\alpha h_y)-(2+h_y^2\lambda^2)\right]\\&+(\tilde{V}_{j+1}+\tilde{V}_{j-1})\times\left[1+\frac{h_y\lambda^2}{4\sigma_\alpha^2}\sinh(\alpha h_y)\right]=0.\end{aligned}$$

The nonvanishing solution to this equation which satisfies the boundary conditions that $V_j=0$ for $j=0$, M is

(27) $$\tilde{V}_j=\sin\left(\frac{jq\pi}{M}\right)$$

where q is an integer, the mode number or number of half wave-lengths that fit between the bottom and top of the enclosure. The relationship that remains when (25) and (27) are substituted into (26) is a quadratic equation for the eigenvalue σ_α.

Hence

(28) $$\sigma_\alpha^2=\frac{\lambda^2}{2}\frac{h_y\sinh(\alpha h_y)[\cos(q\pi/M)+\cosh(\alpha h_y/2)]}{(2+h_y^2\lambda^2)\cosh(\alpha h_y/2)-2\cos(q\pi/M)}.$$

When $h_y \to 0$, σ_α^2 becomes equal to σ^2, the eigenfrequency in the continuous case, which is given in (9).

A similar procedure is used when the conservative difference approximation for the flux, (16b), is considered. The simple exponential stratification, (24), for $\rho_0(j)$ again, is used. The equation for the vertical velocity V_j, is such that we can change variables as follows:

$$V_j = \exp(\beta h_y j)\tilde{V}_j \tag{29}$$

and choose β so that the coefficients of $\tilde{V}_{j+1}$ and $\tilde{V}_{j-1}$ are equal. Then

$$\exp(\beta h_y) = \exp\left(\frac{\alpha h_y}{2}\right)\left\{\frac{1+(h_y^2/\sigma_\alpha)^2[\exp(\alpha h_y)-1]/4h_y}{1+(h_y^2/\sigma_\alpha)^2[1-\exp(-\alpha h_y)]/4h_y}\right\}^{1/2} \tag{30}$$

and the equation for $\tilde{V}_j$ becomes

$$\begin{aligned}(\tilde{V}_{j+1}+\tilde{V}_{j-1})&\left[\left(\frac{\sigma_\alpha}{h_y\lambda}\right)^2+\frac{1}{4h_y}(1-\exp(-\alpha h_y))\right]^{1/2}\left[\left(\frac{\sigma_\alpha}{h_y\lambda}\right)^2+\frac{1}{4h_y}(\exp(\alpha h_y)-1)\right]^{1/2}\\ &-\tilde{V}_j\left\{\sigma_\alpha^2\left[\frac{1}{2}+\frac{1}{(h_y\lambda)^2}\right]2\cosh\left(\frac{\alpha h_y}{2}\right)-\frac{1}{h_y}\sinh\left(\frac{\alpha h_y}{2}\right)\right\}=0.\end{aligned} \tag{31}$$

Again, the solution of (31) satisfying boundary conditions $V_j=0$ at $j=0, M$ is

$$\tilde{V}_j = \sin\left(\frac{jq\pi}{M}\right) \tag{32}$$

and the eigenvalue relationship, (31), is

$$\begin{aligned}2\cos\left(\frac{q\pi}{M}\right)&\left[\left(\frac{\sigma_\alpha}{h_y\lambda}\right)^2+\frac{1}{4h_y}(1-\exp(-\alpha h_y))\right]^{1/2}\left[\left(\frac{\sigma_\alpha}{h_y\lambda}\right)^2+\frac{1}{4h_y}(\exp(\alpha h_y)-1)\right]^{1/2}\\ &=\sigma_\alpha^2\left[1+\frac{2}{(h_y\lambda)^2}\right]\cosh\left(\frac{\alpha h_y}{2}\right)-\frac{1}{h_y}\sinh\left(\frac{\alpha h_y}{2}\right).\end{aligned} \tag{33}$$

Real solutions to this equation require $\sigma_\alpha^2>0$. Squaring and rearranging gives a quadratic equation for σ_α^2, and this equation has the solution

$$\sigma_\alpha^2 = \frac{1}{2a}\{b+\sqrt{b^2-4ac}\} \tag{34}$$

where

$$\begin{aligned}a&=[2+(h_y\lambda)^2]^2\cosh^2\left(\frac{\alpha h_y}{2}\right)-4\cos^2\left(\frac{q\pi}{M}\right),\\ b&=\frac{\sinh(\alpha h_y)}{h_y}(h_y\lambda)^2\left[2+(h_y\lambda)^2+2\cos^2\left(\frac{q\pi}{M}\right)\right],\\ c&=(h_y\lambda)^4\frac{\sinh^2(\alpha h_y/2)}{h_y^2}\sin^2\left(\frac{q\pi}{M}\right),\end{aligned} \tag{35}$$

and where the + sign has been chosen by noting that the continuous limit is recovered when h_y, h_x, $h_z \to 0$.

The results for all six cases, $A-F$, can be summarized as follows. In (14) the solutions for all dependent variables are given with the dependences on i and k given explicitly. The j dependences are

(36)
$$U_j = -\frac{2}{h_x}\sin\left(\frac{r\pi}{2N}\right)\frac{1}{\lambda^2}\frac{V_j - V_{j-1}}{h_y},$$

$$W_j = -\frac{2}{h_z}\sin\left(\frac{s\pi}{2L}\right)\frac{1}{\lambda^2}\frac{V_j - V_{j-1}}{h_y},$$

$$V_j = \exp(j\beta h_y)\sin\left(\frac{q\pi j}{M}\right),$$

$$P_j = \frac{\Gamma}{\lambda^2}\rho_0(j)\frac{V_j - V_{j-1}}{h_y}.$$

For cases A, C and E

(37a)
$$R_j = -\frac{1}{2h_y\Gamma}\{V_j[\rho_0(j+1) - \rho_0(j)] + V_{j-1}[\rho_0(j) - \rho_0(j-1)]\}$$

and for cases B, D and F

(37b)
$$R_j = -\frac{1}{4\Gamma h_y}(V_j + V_{j-1})[\rho_0(j+1) - \rho_0(j-1)]$$

where

(37c)
$$\lambda^2 = 4\left[\left(\frac{1}{h_x}\sin\frac{r\pi}{2N}\right)^2 + \left(\frac{1}{h_z}\sin\frac{s\pi}{2L}\right)^2\right].$$

For cases A and B,

(38a)
$$\Gamma^2 = \sigma^2,$$

for cases C and D,

(38b)
$$\Gamma^2 = \frac{4}{\delta^2}\sin^2\left(\frac{\sigma_\alpha\delta}{2}\right),$$

and for cases E and F

(38c)
$$\Gamma^2 = \frac{\sin^2(\sigma_\alpha\delta)}{\delta^2}.$$

The expressions for Γ and β are for cases B, D and F

(39a)
$$\Gamma^2 = \frac{\lambda^2}{2}\frac{h_y\sinh(\alpha h_y)\{\cos(q\pi/M) + \cosh(\alpha h_y/2)\}}{(2 + h_y^2\lambda^2)\cosh(\alpha h_y/2) - 2\cos(q\pi/M)},$$

(39b)
$$\beta = \alpha/2,$$

and for cases A, C and E

$$\Gamma^2 = \frac{1}{2a}(b + \sqrt{b^2 - 4ac}),$$

$$a = [2 + (h_y\lambda)^2]^2 \cosh^2\left(\frac{\alpha h_y}{2}\right) - 4\cos^2\left(\frac{q\pi}{M}\right),$$

(40a)
$$b = \frac{(h_y\lambda)^2}{h_y} \sinh(\alpha h_y)\left[2 + 2\cos^2\left(\frac{q\pi}{M}\right) + (h_y\lambda)^2\right],$$

$$c = (h_y\lambda)^4 \frac{\sinh^2(\alpha h_y/2)}{h_y^2} \sin^2\left(\frac{q\pi}{M}\right),$$

(40b)
$$\beta = \frac{\alpha}{2} + \frac{1}{2}\log\left\{\frac{1 + (h_y\lambda)^2[\exp(\alpha h_y) - 1]/(4h_y\Gamma^2)}{1 + (h_y\lambda)^2[1 - \exp(-\alpha h_y)]/(4h_y\Gamma^2)}\right\}.$$

6. Discussion. The set of partial differential equations (1) and associated initial and boundary conditions represent a mixed hyperbolic and elliptic system. Such mixed initial, boundary value problems are of general interest, and linear systems such as this are often used as models to analyze the properties of difference approximations for numerical integration of the systems [3], [10]. The exact solutions presented in the previous section provide the opportunity to examine the stability and accuracy of several difference schemes for approximating linearized differential equations representing a physical problem of interest, internal waves in an enclosure. The solutions also provide an excellent comparison for results obtained using direct numerical integration using one of these difference schemes [1], [8], [9].

Difference approximations to (1) introduce spatial and temporal discretization errors into the description of the solution, and these will generally be of two types, dissipative and dispersive. The difference schemes discussed here are not dissipative, but are dispersive [10]. The solutions found in the last section allow us to determine the magnitude of the dispersion introduced by the difference approximation as a function of the parameters of the continuous problem and the number of discretization points.

As noted above, for internal waves in an enclosure, there are four parameters which specify completely these waves: $\sqrt{\alpha}$, the Brunt–Vaisala frequency; $r\pi/l$, the x-direction wavenumber; $s\pi/d$; the z-direction wavenumber; and πq, the wavenumber in the vertical direction. The solutions for the difference equations approximating these internal waves become more complicated because there are additional parameters required to describe the solution in the discretized problem: h_x, h_y, and h_z, the mesh cell dimensions in the x-, y- and z-directions respectively; and, when the equations are also discretized with respect to time, δ, the time step size.

The solutions presented in § 4 show that the parameters are found in certain groupings which can easily be interpreted. For example, the spatial dependences in the horizontal directions are determined by two parameters $N/r\pi$ and $L/s\pi$, which represent respectively the number of mesh points per half-wavelength in the x- and z-directions. Noting the $h_x = l/N$ and $h_z = d/L$, the reciprocals of these parameters can also be written $r\pi h_x/l$ and $s\pi h_z/d$, and can be interpreted, therefore, as the ratios of the mesh size to the half-wavelengths in the x- and z-directions. As is expected, the accuracy with which the discretized solutions represent the solution to the partial differential equation depends upon $r\pi/N = r\pi h_x/l$ and $s\pi/L = s\pi h_z/d$ being small. It should be noted, however, that these two parameters do not appear independently, but only in the combination λ defined in (22), which represents the horizontal wavenumber in the discretized scheme.

In the vertical direction there are two length scales which are relevant: as in the horizontal direction, there is the internal-wave modal half-wavelength $1/q\pi$, but also, because the density decreases exponentially, there is the stratification length scale $1/\alpha$. Therefore, there are two parameters $q\pi h_y = q\pi/M$ and $\alpha h_y = \alpha/M$ which represent the ratio of the mesh size to the vertical half-wavelength and the ratio of the mesh size to the statification length respectively. Again, for accuracy, these ratios must be small.

Finally, there are the ratios of the mesh sizes in the various directions h_x/h_y and h_z/h_y. The solutions presented in the last section to the equations discretized in space depend upon these six parameters, $r\pi/N$, $s\pi/L$, $q\pi/M$, αh_y, h_x/h_y and h_z/h_y. When the equations are discretized also with respect to time, they also depend upon a seventh parameter $\delta\sqrt{\alpha}$, the ratio of the time step size δ to the natural period of oscillation $1/N = 1/\sqrt{\alpha}$. The stability and accuracy of the solutions to the discretized equations are determined by the values of these parameters.

Stability. Once the forms of the solutions are known, it is instructive to derive the differential equations for the time dependences $f(t)$, $g_p(t)$ and $g_\rho(t)$ in (14). This is accomplished by substituting the known dependences on all discrete variables, i, k and j into (10) and determining the relations for these functions of time. The determination of the stability of the temporal discretization schemes becomes trivial then. In this subsection we determine the stability characteristics of the difference schemes by this means.

To determine the time dependences either for the central difference scheme or for the conservative scheme, we substitute assumed forms (14) into (10) with either (12) or (13) just as was done in § 4. Substitution of assumed forms (36) and (37) for the j dependences then results in the following equations for $f(t)$, $g_p(t)$ and $g_\rho(t)$.

From either the third or fifth of equation (15), we obtain

$$\text{(41a)} \qquad \frac{1}{\sigma_\alpha}\frac{df}{dt} + g_p(t) = 0,$$

where σ_α has been used to denote the frequency in the continuous case. Because of the assumed forms, the second of (15) is identically satisfied. From the first of (15) and either the central difference or conservative difference scheme (and the corresponding solution form), we obtain

$$\text{(41b)} \qquad \frac{1}{\sigma_\alpha}\frac{dg_\rho}{dt} - f(t) = 0.$$

Finally, from the fourth of (15) and the assumed j dependences, we obtain, for either central or conservative differences, a linear relationship between df/dt, $g_\rho(t)$ and $g_p(t)$. When this latter relationship is combined with (41a) and the assumed solutions (36) are used, it can be shown that

$$\text{(41c)} \qquad g_p(t) = g_\rho(t)$$

(if σ_α is to be the oscillation frequency of the system).

Equations (41) are three equations for the time dependences $f(t)$, $g_p(t)$ and $g_\rho(t)$. Combining these equations, we find

$$\text{(42)} \qquad \frac{d^2f}{dt^2} + \sigma_\alpha^2 f = 0$$

representing simple harmonic motion, as expected and as assumed in (17) of the previous section.

The interesting feature of the exercise outlined above is that the stability properties of the first-order and second-order time differencing schemes now become simple to understand. Replacing the time derivatives by a first-order explicit Euler method leads to a situation in which the solutions for the amplification factors in a stability analysis have magnitudes greater than one; hence the scheme is unstable:

$$(43)\qquad \begin{aligned} \frac{df}{dt} &\to \frac{f^{n+1}-f^n}{\delta} = -\sigma_\alpha g_p^n,\\ \frac{dg_\rho}{dt} &\to \frac{g_\rho^{n+1}-g_\rho^n}{\delta} = \sigma_\alpha f^n \end{aligned}$$

in (41a) and (41b) and $g_p^n = g_\rho^n$ in (41c); we find solutions

$$f^n = f_0\beta^n, \quad g_\rho^n = g_0\beta^n, \quad g_p^n = \tilde{g}_0\beta^n,$$

where $\beta = -1 \pm i\sigma_\alpha\delta$ so that both values of β have $|\beta| > 1$.

Choosing a first-order Euler method for the time derivatives, but making the scheme semi-implicit, as discussed in the last section, will remove this instability and make the difference scheme nondissipative. In this case, the equations (43) are still used to approximate (41a) and (41b), but now the semi-implicit scheme requires that $g_p^n = g_\rho^{n+1}$, and the corresponding amplification factors β are given by solutions to the equation

$$\beta^2 - \beta(2-\sigma_\alpha^2\delta^2) + 1 = 0.$$

Hence, for both values of β, $|\beta| = 1$, provided only that $\sigma_\alpha\delta < 2$. Therefore, stability is guaranteed for this first-order, semi-implicit method when

$$(44)\qquad \sigma_\alpha\delta < 2$$

where σ_α is given by (28) for the central difference scheme and by (34) for the conservative scheme.

When the leap-frog method is used for the time differencing,

$$(45)\qquad \begin{aligned} \frac{df}{dt} &\to \frac{f^{n+1}-f^{n-1}}{2\delta} = -\sigma_\alpha g_p^n,\\ \frac{dg_\rho}{dt} &\to \frac{g_\rho^{n+1}-g_\rho^{n-1}}{2\delta} = \sigma_\alpha f^n,\\ g_p^n &= g_\rho^n. \end{aligned}$$

The scheme is stable provided only that

$$(46)\qquad \sigma_\alpha\delta < 2.$$

The analyses outlined above have determined the same conditions for stability which are not very restrictive for representing the internal waves; *accuracy of the finite difference solutions is much more restrictive.*

Accuracy. A convenient dependent variable with which to make comparisons between the solutions to the continous problem and the discretized problem is the vertical displacement ζ of a fluid element at any specified point; ζ is determined in terms of the density perturbation and the background density gradient in (3):

$$\zeta = -\rho \Big/ \left(\frac{d\rho_0}{dy}\right)$$

where the density perturbation for the continuous solution or for any of the discretized solutions can be evaluated using expressions given in §3 and 5. As with all dependent variables, the vertical displacement undergoes sinusoidal oscillations with respect to time, and the accuracy with which these oscillations will be reproduced depends upon the parameters discussed earlier, namely the Brunt–Vaisala frequency and the wavenumbers in the horizontal and vertical directions plus the parameters required to define the discretization, the mesh cell dimensions and the time step size.

A sample calculation comparing the time dependences of the vertical displacement in the continuous and two discretized cases has been performed. The sample calculation is for a two-dimensional case: there is no dependence of any variables upon z and the z-component of velocity, w, is zero (formally, d, $L \to \infty$ with h_z held fixed). The two-dimensional rectangular enclosure is one unit high and one unit wide ($l = 1$) and is covered by a mesh composed of 31 increments in the horizontal or x-direction and 30 in the vertical or y-direction ($h_x = \frac{1}{31}$, $h_y = \frac{1}{30}$). The stratification parameter $\alpha = 1$ so that the Brunt–Vaisala frequency is unity and the integers r and q defining the horizontal and vertical wavenumbers are 5 and 2 respectively. Finally, the centered difference, (16a) is used for the flux term and the time step size $\delta = 0.5$.

In Fig. 3 the variation with time in the vertical displacement at a fixed location $x = 0.21$ and $y = 0.15$ is shown. The solid line is the solution to the continuous problem. The circled points represent results obtained by the first-order method, and the points enclosed in triangles results obtained by the second-order method. The discrepancy between the continuous solution and that obtained by the first-order method is a result of a small phase shift introduced at the start of the calculation. The phase shift is proportional to the time step size and can therefore be reduced by taking smaller time steps in the computation. Ignoring this phase shift, the solution follows rather faithfully the exact solution. For the time step selected, the second-order solution reproduced

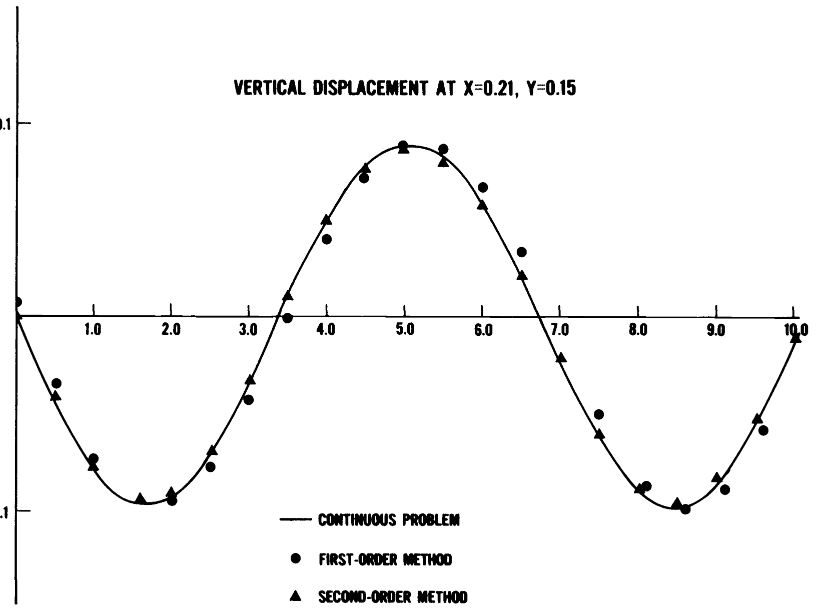


FIG. 3. *Comparison of the time dependences of the vertical displacement in the continuous and two discretized cases. The sample calculation is for a two-dimensional configuration in which the Brunt–Vaisala frequency is unity.* (*See the text for other conditions defining the calculations.*)

the exact one very well. Again, smaller time steps would produce still better agreement. The values of σ listed on this figure are the eigenvalues for the mode, in the continuous and each discrete case, and represent the frequency of each oscillation. Similar results are found when the conservative difference approximation is used for the flux term.

The magnitude of the dispersion introduced by a difference scheme for a particular mode is determined by the variation of the eigenvalue for the scheme from the eigenvalue σ for the continuous problem: the closer the eigenvalues are, the less numerical dispersion for a particular mode and the more faithful the representation of the true solution by the difference approximation.

In Figs. 4 through 6 plots are given of the eigenfrequencies for the continuous problem, as given by (9b), and for two completely discretized problems, one with conservative differences for the flux term and leap-frog in time (σ_E as given by (38c) and (40)), and the other with central differences for the flux term and leap-frog in time (σ_F as given by (38c) and (39)). For all three plots, σ is shown as a solid line while the approximation for σ_E is given by a dotted line and σ_F is given by a dashed line. In each of these plots a parameter increases along the abscissa while all other parameters are fixed. The parameters chosen for the figures are inversely proportional to a length scale in each case and have been selected to demonstrate the magnitude of the dispersion introduced by the difference approximations as the number of mesh points relative to a length scale is decreased.

In any computation the "resolving power" of the numerical scheme will be limited for practical purposes by the maximum number of mesh points which can be utilized:

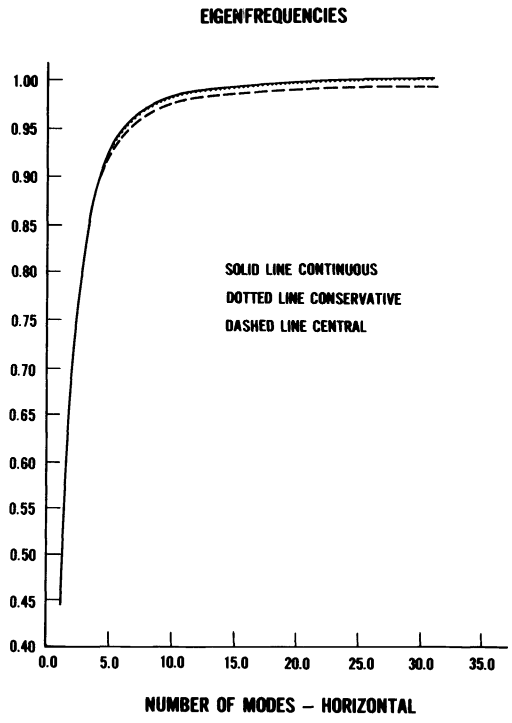


FIG. 4. *Eigenfrequencies for the continuous internal wave solution and for two completely discretized solutions, one with conservative differences for the flux term and leap-frog in time* (σ_E *as given by* (38c) *and* (40)) *and the other with central differences for the flux term and leap-frog* (σ_F *as given by* (38c) *and* (39)). (*See text for other conditions defining the calculations.*) *Eigenfrequencies plotted as functions of horizontal mode number.*

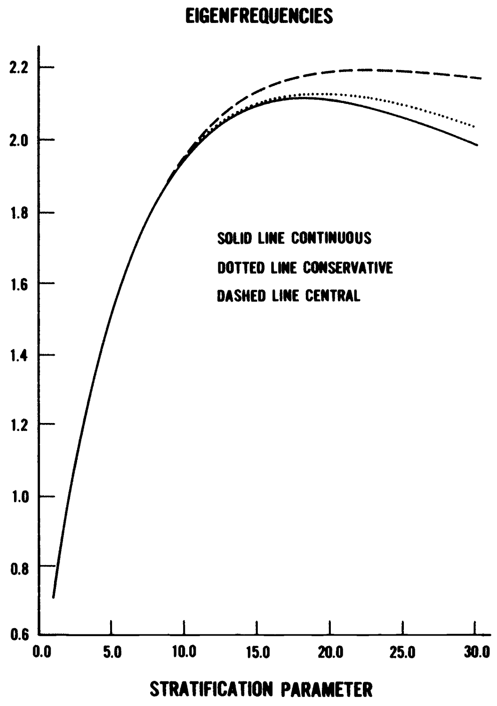


FIG. 5. *Eigenfrequencies for continuous and two discretized internal wave solutions (same as in Fig. 4) as functions of the stratification parameter.*

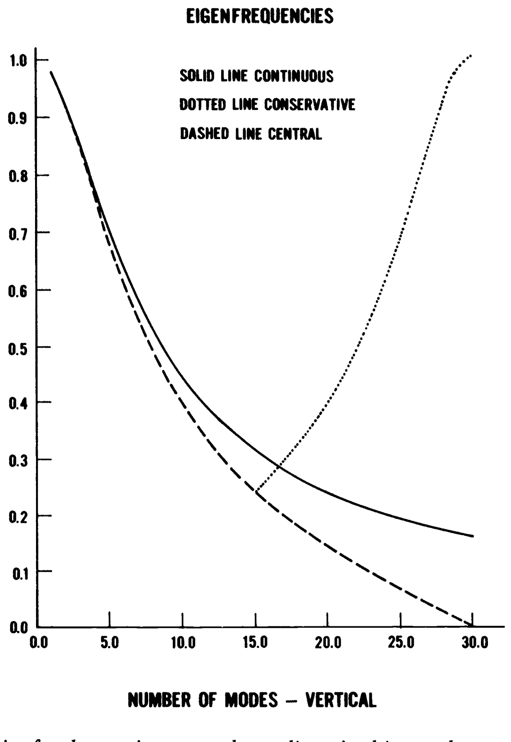


FIG 6. *Eigenfrequencies for the continuous and two discretized internal wave solutions (same as in Fig. 4) as functions of the vertical mode number.*

the length scales which can be resolved by the computation will be those described by at least a few mesh points. The number of mesh points in each direction is 30 for the sample calculations shown. In all three figures, the dispersion is expected to be better for small values of the parameters (large values of the corresponding length scales) and to exhibit increased dispersion due to the difference approximations for larger values of the parameters.

In Fig. 4 the eigenfrequencies are plotted as functions of the mode number, $\tilde{\lambda}$ in (9) or λ in (37c) in the horizontal direction. Clearly, both approximations behave qualitatively the same as the eigenfrequency for the continuous problem, that for the conservative difference scheme being indistinguishable from this frequency while that for the central difference scheme is only a few percent below.

In Fig. 5 the eigenfrequencies are plotted as functions of the stratification parameter α. The greater the stratification, the smaller the length scale over which the density decreases by a factor l^{-1}. The two difference approximations determine eigenfrequencies which behave qualitatively the same as the exact eigenfrequency; however, the conservative scheme is quantitatively better over the range of the stratification parameter shown.

In Fig. 6 the eigenfrequencies are plotted as functions of the vertical mode number q. Both approximated eigenfrequencies are indistinguishable from each other until $q=15$, which is a mode number equal to half the number of mesh points. These approximations diverge from the correct value with increasing vertical mode number until they are about twenty percent below the continuous eigenfrequency at $q=15$. Beyond this value of the vertical mode number the central difference scheme is qualitatively correct but quite inaccurate whereas the conservative scheme is not qualitatively correct.

7. Concluding remarks. The analysis of difference equations describing linear internal waves presented in this paper has been used as a guide in the selection of a finite difference scheme to numerically integrate nonlinear partial differential equations describing buoyant convection [1], [8] and [9]. This analysis has provided a basis for estimating the stability and accuracy for such a scheme. In the two-dimensional case, it has also been used to provide a detailed check on the numerical method of solution of the linearized finite difference equations and the implementation of the solution.

The numerical method of solving the linear difference equations is composed of a time marching portion and a linear algebraic (or elliptic) portion. The time marching difference equations represented by the temporally and spatially differenced versions of (10) (the first, third, fourth and fifth equations) are solved by explicit calculation of the dependent variable (density and the three components of velocity) at a new time level, given the dependent variables at the preceding two time levels. The pressure must be determined from a matrix equation obtained by applying the second of (10) to the remaining time marching equations. This matrix equation arises as a discretization of an elliptic equation for the pressure and must be solved iteratively in the nonlinear case [5]. Details of the algorithm to solve the linear, inhomogeneous problem in two dimensions are presented in [8] and to solve the nonlinear, inhomogeneous problem are presented in [1].

In two dimensions, the homogeneous, initial value problem for an arbitrary eigenmode, the internal waves, was solved for one discretization by direct numerical integration using the algorithm described above. The results of this computation for a few of the lower eigenmodes (Fig. 3 is a plot prepared from these computations) were compared with the analytical results determined in this paper. The agreement between the numerical and analytical results was exceedingly good, this agreement being limited only by the accuracy specified by the iterative solution to the matrix equation for the pressure. We have specified regularly a tolerance of 10^{-6} (limited by the computer word length) for the tolerance of the solution to the matrix equation on a UNIVAC 1108 or 1100/82 and obtained agreement after a few time steps between the analytical and the numerical solution to the difference equations of a few parts in the sixth significant figure. This agreement is to be contrasted with that obtained between the solution to the discretized equations and the solution to the continuous problem which, as shown earlier, is a few percent for a mesh of 30 grid points in each direction.

REFERENCES

[1] H. R. BAUM, R. G. REHM, P. D. BARNETT AND D. M. CORLEY, *Finite difference calculations of buoyant convection in an enclosure, Part* I: *the basic algorithm*, this Journal, 4 (1983), pp. 117–135.

[2] F. H. HARLOW AND A. A. AMSDEN, *Fluid Dynamics*, A LASL Monograph, Los Alamos Scientific Laboratory Report LA 4700, Los Alamos, NM, 1971.

[3] H. KREISS AND J. OLIGER, *Methods for the approximate solution of time dependent problems*, Global Atmospheric Research Programme (GARP) Publication Series No. 10, 1973.

[4] H. LAMB, *Hydrodynamics*, Dover, New York, 1945, Article 378, pp. 378–9.

[5] J. G. LEWIS AND R. G. REHM, *The numerical solution of a nonseparable elliptic partial differential equation by preconditioned conjugate gradients*, J. Res. National Bureau of Standards, 85 (1980), pp. 367–390.

[6] M. J. LIGHTHILL, *Waves in Fluids*, Cambridge Univ. Press, New York, 1978, Chapter 4.

[7] L. PRANDTL, *Essentials of Fluid Dynamics*, Hafner, New York, 1952, pp. 374–5.

[8] R. G. REHM, H. R. BAUM, J. LEWIS AND M. R. CORDES, *A linearized finite-difference computation of fluid heating in an enclosure*, National Bureau of Standards, NBSIR79-1754, Washington, DC, 1979.

[9] R. G. REHM, H. R. BAUM AND P. D. BARNETT, *Buoyant convection in a vorticity, stream-function formulation*, J. Res. National Bureau of Standards, 87, 1982.

[10] L. N. TREFETHEN, *Group velocity in finite difference schemes*, SIAM Rev., 24 (1982), pp. 113–136.

[11] J. S. TURNER, *Buoyancy Effects in Fluids*, Cambridge Univ. Press, New York, 1973, Chapter 2.

[12] G. B. WHITHAM, *Linear and Nonlinear Waves*, John Wiley, New York, 1974.

[13] C. S. YIH, *Dynamics of Nonhomogeneous Fluids*, Macmillan, New York, 1965, Chapter 2.

SIAM J. SCI. STAT. COMPUT.
Vol. 5. No. 4, December 1984

INCOMPLETE METHODS FOR SOLVING $A^TAx=b$*

A. JENNINGS† AND M. A. AJIZ‡

Abstract. Incomplete methods are considered for the solution of $A^TAx = b$. One possibility is to construct the product A^TA and proceed with an ICCG solution. However, it is also possible to utilize incomplete orthogonal decompositions of A obtained by modifying either the Gram–Schmidt or the Givens rotation method. These three methods are applied to four small structural analyses and the numerical efficiencies are compared.

Key words. simultaneous equations, least squares, incomplete factorization, conjugate gradient method, symmetric positive definite matrices

1. Introduction.

Equations of the form

$$A^TAx=b \tag{1.1}$$

where A is of order $m\times n$ $(m\geqq n)$ may be obtained in the analysis of any conservative system although in many formulations the coefficient matrix A^TA is constructed directly. For instance, in the solution of linear elastic structural problems [1], [2], [3] the stiffness method gives equations which can be written in the form

$$\bar{A}^T\bar{K}\bar{A}x=b. \tag{1.2}$$

Here $\bar{K}$ is a block diagonal matrix of the member stiffnesses expressed in terms of member coordinates, $\bar{A}$ is a transformation matrix which defines these member coordinates in terms of the global displacement coordinates x (normally joint displacements). The vector b represents the forces applied at the nodes. If $\bar{K}$ is nonsingular, it may be factorized into the form

$$\bar{K}=\bar{L}\bar{L}^T \tag{1.3}$$

where $\bar{L}$ is a Choleski matrix of diagonal block form. If $\bar{K}$ is singular, rows of $\bar{L}^T$ which would otherwise be null may be omitted yielding a rectangular matrix. In either case equations of the form of (1.1) may be obtained by multiplying

$$A=\bar{L}^T\bar{A}. \tag{1.4}$$

It is also possible to obtain finite element and some finite difference formulations of conservative systems in the form of (1.1). If an overdetermined set of linear equations is written in the form

$$\tilde{A}x=\tilde{b}+e \tag{1.5}$$

where $\tilde{A}$ is of order $m\times n$ $(m\geqq n)$, $\tilde{b}$ is of order m and e is an error vector, the normal equations for a least squares fit is given by the solution of

$$\tilde{A}^TW\tilde{A}x=\tilde{A}^TW\tilde{b} \tag{1.6}$$

where W is a diagonal matrix of weighting factors. This also reduces to the form of (1.1) with

$$A=W^{1/2}\tilde{A} \tag{1.7a}$$

* Received by the editors November 16, 1982 and in revised form February 16, 1984. This work was presented at the Sparse Matrix Symposium, Fairfield Glade, Tennessee, October 25–27, 1982.

† Civil Engineering Department, Queens University, Belfast, Northern Ireland BT7 1NN.

‡ Department of Computer Science, Yarmouk University, Irbid, Jordan.

and

$$b=\tilde{A}^TW\tilde{b}. \tag{1.7b}$$

Hence it is not only possible, but in many cases may be more convenient to express an analysis in the form of (1.1).

In cases where the resulting simultaneous equations have a sparse coefficient matrix, the matrix A will have an even lower density of nonzero elements. For instance a set of five-point Laplacean finite difference equations for a rectangular mesh with rectangular boundaries may be generated using an A matrix containing no more than two nonzero elements/row and the analysis of a plane pin-jointed structural frame may be formulated by using an A matrix containing no more than four nonzero elements/row. The work reported here is part of an investigation directed towards finding efficient methods of solving large order structural analyses where the equations are generally of such large bandwidth that factorization techniques require a prohibitive amount of store and computing time and yet where classical iterative techniques converge too slowly to provide a satisfactory alternative.

2. Preconditioning. Most of the classical iterative methods for the solution of simultaneous equations have convergence rates which are linked in some way or other to the eigenvalue spectrum of the coefficient matrix [4]. In particular the P-condition number is important, being the ratio of the largest to the smallest eigenvalue. In general, the greater the P-condition number, the slower will be the convergence rate. Preconditioning is a process of transforming the equations so that their eigenvalue spectrum is improved. Equations (1.1) can be preconditioned by introducing a transformation matrix L giving

$$(L^{-1}A^TAL^{-T})(L^Tx)=(L^{-1}b). \tag{2.1}$$

Because of the need to carry out forward and back substitutions with L and its transpose, this matrix is usually triangular. An iterative solution of these equations to determine the transformed variables (L^Tx) has $(L^{-1}A^TAL^{-T})$ as coeficient matrix and hence the rate of convergence is related to the eigenvalue spectrum of this matrix. Early methods for which A^TA needs to be constructed were block relaxation, SSOR [5] and Evans' preconditioning method [6], the latter two having been shown to be equivalent to each other.

The solutions of sparse linear least squares equations without forming A^TA have received considerable attention in recent years [7], [8]. Two methods of preconditioning which have been developed are the SSOR method of Björck [9] and the LU factorization method by Saunders [10]. Of these Saunders claims the LU factorization to be more effective for equations which are badly conditioned.

3. ICCG methods. From the standpoint of convergence rate, the ideal choice of transformation matrix L is the Choleski factor of A^TA, for then

$$L^{-1}A^TAL^{-T}=I \tag{3.1}$$

giving a P-condition number of 1 and convergence in one iteration whichever iteration method is used. The objective of incomplete Choleski methods is to obtain a transformed coefficient matrix which is as close to I as possible without invoking the large amounts of fill-in within L that would be obtained from accurate Choleski factorization. The conjugate gradient method is the most appropriate iterative method to use with incomplete Choleski preconditioning because it can be implemented without the transformed coefficient matrix (which is likely to be fully populated) being fully formed.

Instead it can be implemented with A^TA and L (or A and L) stored in sparse form [11]. Furthermore the conjugate gradient method is universally convergent for any symmetric positive definite coefficient matrix and has a rate of convergence which is generally superior to other classical iterative methods. In particular, when significant gaps in the eigenvalue spectrum exist, the conjugate gradient method will accelerate as iteration proceeds [12].

Variations in IGCG methods arise through:

(a) Rejection criteria for omitting off-diagonal elements from the factorization. In some methods [13] all elements which fill-in are automatically rejected, thus the storage pattern of L^T and the upper triangle of A^TA will be the same. In other methods elements are rejected if they are weak according to some prescribed criterion.

(b) Diagonal correction. In some methods no modifications of the diagonal elements are made during incomplete factorization. This is justifiable when the coefficient matrix is an M-matrix [14]. In other methods diagonal modifications are made to ensure that the reduced matrix remains positive definite throughout the incomplete factorization [15].

The method adopted in these tests is that given by Jennings and Malik [16] in which a rejection criterion based on element size is used and the magnitudes of diagonal elements are increased when rejections occur. The modifications to the diagonal elements are made sufficiently large to ensure that the eigenvalues of LL^T are at least as large as the corresponding eigenvalues of A^TA. This has the effect of guaranteeing that the incomplete factorization will not break down due to negative or zero pivots.

Unfortunately there is no known way of predicting the influence of any particular element in the factorization on the eigenvalue spectrum of the resulting coefficient matrix without employing extensive computations. Hence elements are rejected if their magnitude relative to the geometric mean of the relevant current values of the two diagonal elements is lower than a rejection parameter. Furthermore the diagonal modifications can be proved to be unnecessary in certain special cases, such as when the coefficients form an M matrix. Hence it is possible that alternative incomplete methods could yield more efficient convergence rates. However variation in the rejection parameter gives more control over the process than is possible with either SSOR or LU methods in which variations in performance are only possible by conducting row interchanges.

4. Orthogonal decomposition.

Consider the decomposition

$$A = QR \tag{4.1}$$

where Q is an $m \times n$ matrix comprising orthogonal vectors and R is an upper triangular matrix with positive diagonal elements. The orthonormal condition

$$Q^TQ = I \tag{4.2}$$

gives

$$A^TA = R^TR. \tag{4.3}$$

Hence R is the upper triangular Choleski matrix associated with A^TA, and the orthogonal decomposition gives an alternative way of obtaining it. It is well-known that this technique will normally involve more computation than will the standard use of Choleski factorization. However the process is less subject to rounding error effects.

Normal implementation of the above ICCG method generally entails much less time being spent in the incomplete factorization phase of the algorithm than in the iteration phase. It is therefore possible that generation of an incomplete Choleski matrix by modifying methods of orthogonal decomposition could be advantageous if the eigenvalue spectrum of the resulting transformed matrix yields a faster convergence rate for the iteration phase of the solution which more than offsets any possible extra time taken for the decomposition. Therefore incomplete orthogonal decomposition is not adopted in order to reduce the effects of rounding error. Whether there would be any benefits in this respect have not been investigated.

5. Incomplete Gram–Schmidt. In the Gram–Schmidt method matrix A is converted to Q by a series of vector orthonormalizations. The matrix R represents the coefficients of the orthonormalization process. A possible algorithm for implementation is

```
for $i=1$ to $n$ do $a_i^{(1)}=a_i$;
for $i=1$ to $n$ do
begin $r_{ii}=\|a_i^{(i)}\|_E$;
    $q_i=a_i^{(i)}/r_{ii}$;
    for $j=i+1$ to $n$ do
    begin $r_{ij}=q_i^Ta_j^{(i)}$;
        $a_j^{(i+1)}=a_j^{(i)}-r_{ij}q_i$
    end;
end;
```

This may be converted into an incomplete Gram–Schmidt by including immediately after the formation of r_{ij} and within the inner loop the rejection test

$$\text{if } r_{ij}<\psi/\|a_i^{(1)}\|_E \text{ then } r_{ij}=0;$$

where ψ is a rejection parameter similar to that used by Jennings and Malik for incomplete Choleski factorization [17]. If $\psi=0$, all nonzero off-diagonal elements in R are retained, Q is orthogonal and the Gram–Schmidt process is "complete." If $\psi=1$ all nonzero off-diagonal elements in R are rejected, thus making R diagonal. Normal use of the algorithm is therefore with $0<\psi<1$.

It is possible to prove that $a^{(i)}$ cannot be null (and hence incomplete Gram–Schmidt decomposition cannot break down) as follows:

Consider that r_{11} and r_{22} are nonzero permitting the vector $a_3^{(3)}$ to be computed for a case where $n=4$. It is possible to link vectors available at the time when $a_3^{(3)}$ is formed by the matrix equation

$$A=[q_1 \quad q_2 \quad a_3^{(3)} \quad a_4]\begin{bmatrix} r_{11} & r_{12} & r_{13} & \\ & r_{22} & r_{23} & \\ & & 1 & \\ & & & 1 \end{bmatrix}=\bar{Q}\bar{R} \quad \text{(say)} \tag{5.1}$$

where $\bar{R}$ is nonsingular.

Let

$$y=\bar{R}^{-1}e_3 \tag{5.2}$$

where e_3 is the third column of the unit matrix. Since y must be nonnull and A^TA is positive definite,

$$(a_3^{(3)})^Ta_3^{(3)}=e_3^T\bar{Q}^T\bar{Q}e_3=y^TA^TAy>0. \tag{5.3}$$

Similar results hold for $j \neq 3$, thus permitting the proof to be completed by induction.

In the case where A is sparse, rejection of small elements in R has the effect of curtailing fill-in in R. There is a corresponding reduction in the number of vector orthogonalization operations in the orthogonal decomposition and there is likely to be a reduction in the number of nonzero elements in Q. This may be illustrated by considering

$$(5.4) \qquad A=\begin{bmatrix} 1 & 0.1 & \\ 1 & & \\ & 1 & -1 \end{bmatrix}, \qquad A^TA=\begin{bmatrix} 2 & 0.1 & \\ 0.1 & 1.01 & -1 \\ & -1 & 1 \end{bmatrix}.$$

If $\psi > 0.07071$, then element r_{12} is rejected, which means that column 2 of A is not orthogonalized with respect to column 1. Hence not only is $r_{12}=0$ but also q_{22} and q_{23} do not fill-in, although fill-in of q_{13} still takes place unless $\psi > 0.995$, i.e.,

$$(5.5) \qquad \bar{Q}=\begin{bmatrix} 0.7071 & 0.0995 & 0.0990 \\ 0.7071 & & \\ & 0.9950 & -0.0099 \end{bmatrix}, \qquad \bar{R}=\begin{bmatrix} 1.4142 & & \\ & 1.0050 & -0.9950 \\ & & 0.0995 \end{bmatrix},$$

$$\bar{R}^T\bar{R}=\begin{bmatrix} 2 & & \\ & 1.01 & -1 \\ & -1 & 1 \end{bmatrix}.$$

After completion of orthogonal decomposition the iterative phase of the algorithm proceeds as for the incomplete Choleski method.

6. Incomplete Givens rotations. An alternative method of orthogonal decomposition is to apply successive Givens rotations to A in such a way that all the elements below the diagonal in A are eliminated. One such rotation can be written in the form

$$(6.1) \qquad Q^{(k)}A^{(k)}=A^{(k+1)}$$

where $Q^{(k)}$ is an $m \times m$ matrix specifying the rotation. The decomposition is complete when

$$(6.2) \qquad A^{(l)}=\begin{bmatrix} R \\ 0 \end{bmatrix}$$

in which case

$$(6.3) \qquad Q^{(l-1)} \cdots Q^{(2)}Q^{(1)}A=\begin{bmatrix} R \\ 0 \end{bmatrix}.$$

Hence

$$(6.4) \qquad A^TA=[R^T\,O]\begin{bmatrix} R \\ 0 \end{bmatrix}=R^TR$$

which shows that the computed R matrix is identical (except for rounding effects) to the upper triangular Choleski matrix associated with A^TA provided that the angles of rotation have been chosen such that the diagonal elements of R are positive.

Since the decomposition process involves elimination of individual elements in A, the most obvious method of converting this into an incomplete method is to avoid performing the rotation to eliminate any particular element if it is small compared with other elements. The criterion that has been chosen is that elimination is not performed if

$$a_{ij}^{(k)} < \psi \|a_j^{(k)}\|_E \tag{6.5}$$

where $a_j^{(k)}$ is the jth column of $A^{(k)}$ and ψ is the rejection parameter. Thus if $\psi = 0$, the decomposition will be of complete Givens form and if $\psi \geqq 1$, all the elements will be rejected. It is not satisfactory, however, just to ignore $a_{ij}^{(k)}$ if it is to be rejected, since with $\psi \geqq 1$ this would yield a null matrix for R. Instead the element is retained, but shifted onto a new row which is separate from the other nonzero elements in row i. By doing this $\|a_j\|_E$ is preserved so giving

$$\|r_j\|_E = \|a_j\|_E. \tag{6.6}$$

The correction matrix, relating the incomplete Choleski matrix R to the original matrix A through

$$R^TR = A^TA + C \tag{6.7}$$

must have zero diagonal elements. Hence the sum of its eigenvalues is zero. This is in contrast to the corresponding correction matrix for the Jennings and Malik ICCG method where C has diagonal entries sufficient to ensure that it has no negative eigenvalues.

The matrix $(A^{(k)})^TA^{(k)}$ can only be singular if a vector q can be found for which $v = A^{(k)}q$ is null. By examining the conditions under which this may occur for the case when a row of $A^{(k-1)}$ has been split to form $A^{(k)}$ it can be shown that the split matrix can only give a null v if $A^{(k-1)}$ can also. Hence by induction the splitting process cannot cause any modified matrix $(A^{(k)})^TA^{(k)}$ to be singular.

It can also be shown that the eigenvalues of $(A^{(k)})^TA^{(k)}$ need not be improved by splitting. An example where it may not be beneficial is

$$A = \begin{bmatrix} 1 & \\ 2 & -2 \\ 1 & 1 \\ & 1 \end{bmatrix}, \qquad A^TA = \begin{bmatrix} 6 & -3 \\ -3 & 6 \end{bmatrix}. \tag{6.8}$$

For if the rejection parameter is chosen so that row 3 is split, the modified matrix is

$$\hat{A} = \begin{bmatrix} 1 & \\ 2 & -2 \\ 1 & \\ & 1 \\ & 1 \end{bmatrix}, \qquad \hat{A}^T\hat{A} = \begin{bmatrix} 6 & -4 \\ -4 & 6 \end{bmatrix} \tag{6.9}$$

giving eigenvalues of 2 and 10 as opposed to 3 and 9 for the unsplit matrix. Such a case is likely, however, to be the exception rather than the rule. For the matrix A in (5.4), element a_{12} is rejected if $\psi > 0.0995$ thus giving modified matrices

$$\hat{A} = \begin{bmatrix} 1 & & \\ 1 & & \\ & 1 & -1 \\ & 0.1 & \end{bmatrix}, \qquad \hat{A}^T\hat{A} = \begin{bmatrix} 2 & & \\ & 1.01 & -1 \\ & -1 & 1 \end{bmatrix} \tag{6.10}$$

and the R matrix as in (5.5).

7. Comparison of methods. An algorithm is available for the Jennings and Malik ICCG method in sparse code [18]. This consists of two phases, the incomplete factorization and the CG preconditioned iteration. No sparse matrix implementation for the complete orthogonal decomposition has been devised. The purpose of the following tests is to investigate whether the amount of computation in the iterative phase tends to be sensitive to the way in which R is computed and hence whether it might be profitable to execute a sparse matrix implementation of one of these methods taking into account that such an implementation is likely to be more complex than the ICCG method of Jennings and Malik. Therefore both orthogonal decomposition procedures were programmed in full matrix code and the resulting R matrices converted to a sparse store for the *CG* preconditioned iteration. Because of the storage requirement for the orthogonal decompositions it was not possible to compare results for large order problems. However, since most of the nonzero multiplications occur in the iterative phase of the analysis and the amount of computation in this phase is proportional to the product of the number of iterations multiplied by the total number of nonzero elements in both A^TA and L (or R), and the storage space requirement is also dependent on the latter, it is useful to see if these methods give different convergence rates for similar numbers of nonzero elements in L or R.

Analyses of the four structures shown in Fig. 1 were taken as test cases. The number of nonzero elements on each row of A was no more than six for the portal frames (see Fig. 2) or four for the pin-jointed space frame. In each case analyses were carried out with different values of ψ using ICCG, incomplete Gram–Schmidt and incomplete Givens. However, since ψ values for the different methods are not directly comparable, the number of nonzero multiplications for complete solution was compared with the number of nonzero elements in L or R (Fig. 3). The number of multiplications required to form A^TA was not included since this was performed for

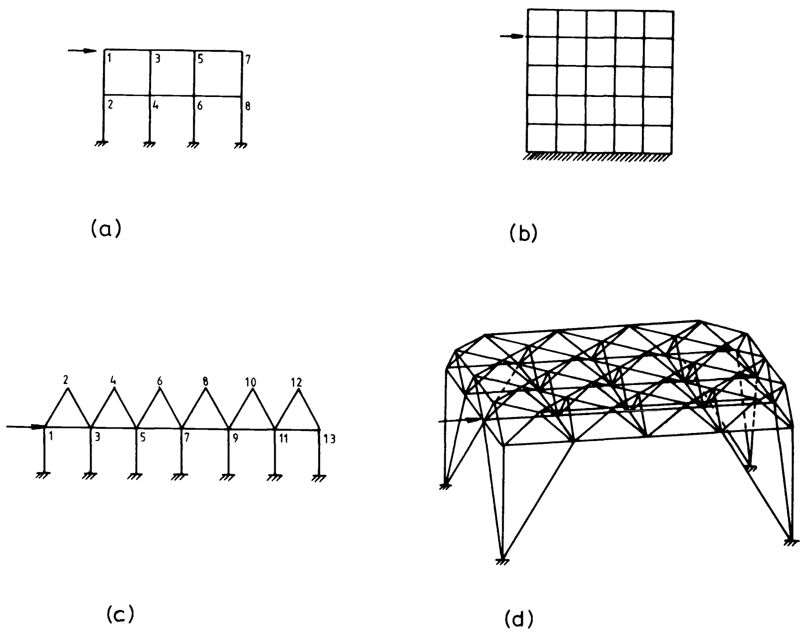


FIG. 1. *Structural frames analysed.* (a) *A* 3 *bay* 2 *storey portal frame* ($n = 24$). (b) *A* 5 *bay* 5 *storey portal frame* ($n = 90$). (c) *A* 6 *bay* 1 *storey portal frame with pitched rooves* ($n = 39$). (d) *A* 3×5 *bay double layer pin-jointed space frame roof structure* ($n = 153$).

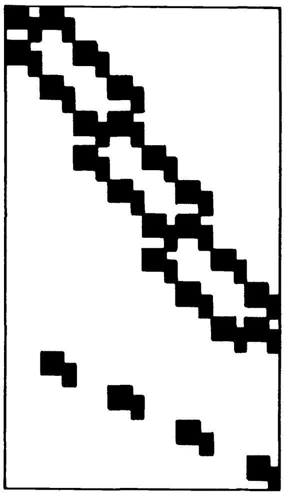


FIG. 2. *Positions of nonzero elements in A* (42×24) *for example* (a).

each of the methods. The right-hand points pertain to $\psi=0$ and so indicate the different number of nonzero multiplications in the "complete" methods. Here the ICCG method has a low value because these small examples give a fast solution using complete Choleski factorization. On the left-hand side ψ is large and so, with only a few nonzero elements in R the performance of all three methods tends to be similar. For larger values

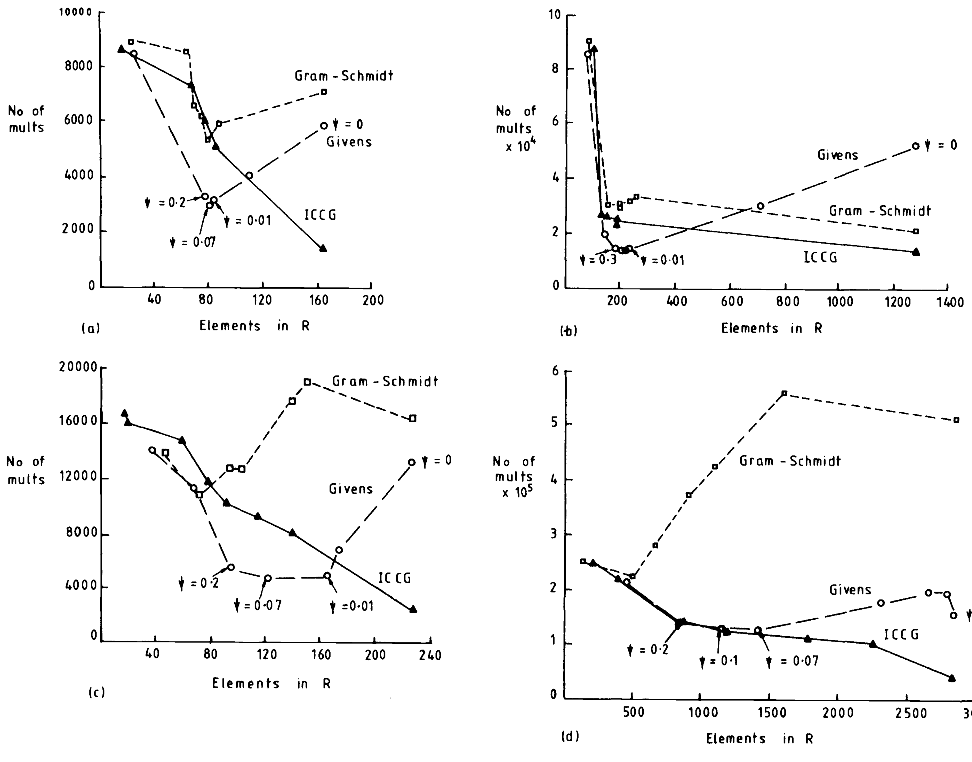


FIG. 3. *Numbers of nonzero multiplications for each of the four frames.* (a) 3×2 *portal.* (b) 5×5 *portal.* (c) 6×1 *pitched portal.* (d) 3×5 *space frame roof.*

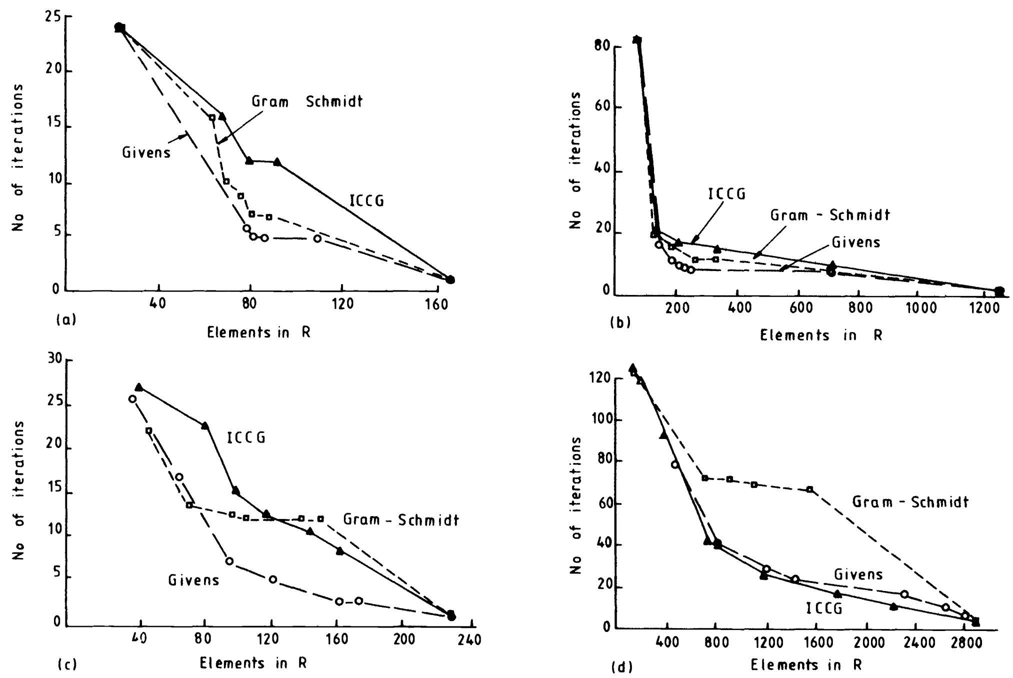


FIG. 4. *Numbers of iterations for each of the four frames.* (a) 3 ×2 *portal.* (b) 5 ×5 *portal.* (c) 6 ×1 *pitched portal.* (d) 3 ×5 *space frame roof.*

of ψ the Givens method performs best and is itself most economical at about $\psi = 0.1$. Since, when ψ is large, the number of nonzero multiplications required for the decomposition phase is small, the better performance of Givens method is due to generally better convergence rates (Fig. 4).

For larger problems involving more fill-in the advantage of using direct Choleski factorization disappears and the efficiency of ICCG is best at intermediate values of ψ (in the interval 0.02 to 0.2). From the above results (albeit limited in scope) it would seem that a form of incomplete Givens orthogonal decomposition might be competitive with ICCG for larger problems provided that suitable software can be developed to perform the decomposition efficiently in some form of packed store. It is worthy of note that the standard Givens orthogonal decomposition method has been implemented in a sparse store by George and Heath [19], so this could form a starting point for consideration of sparse matrix implementations of incomplete Givens orthogonal decomposition.

8. Conclusions. Two incomplete orthogonal decomposition methods have been investigated. As with the Jennings and Malik ICCG method, it can be shown that neither of the methods can fail due to loss of the positive definite property during the determination of the incomplete Choleski factor. In view of the different rejection criteria, some numerical comparisons for simple examples were made by plotting the numbers of iterations and the numbers of nonzero arithmetic operations for each of the methods against the numbers of nonzero elements in the incomplete Choleski matrix. From this limited number of comparisons it can be deduced that the Givens method is worthy of further consideration if an efficient sparse incomplete factorization can be devised. It would be useful to make comparisons with the *LU* decomposition method of Saunders.

REFERENCES

[1] J. H. AGYRIS AND S. KELSEY, *Energy Theorems in Structural Analysis*, Butterworths, London, 1960.

[2] W. M. JENKINS, *Matrix and Digital Computer Methods in Structural Analysis*, McGraw-Hill, London, 1969.

[3] K. H. M. BRAY, P. C. L. CROXTON AND L. H. MARTIN, *Matrix Analysis of Structures*, Edward Arnold, London, 1976.

[4] ALAN JENNINGS AND G. M. MALIK, *The solution of sparse linear equations by the conjugate gradient method*, Int. J. Num. Meth. Engng, 12 (1978), pp. 141–158.

[5] J. W. SHELDON, *On the numerical solution of elliptic difference equations*, Math. Tables Aids. Comput., 12 (1955), pp. 174–186.

[6] D. J. EVANS, *The use of preconditioning in iterative methods for solving linear equations with symmetric positive definite matrices*, J. Inst. Maths. Applics., 4 (1967), pp. 295–314.

[7] AKE BJORCK, *Methods for sparse least squares problems*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 177–199.

[8] M. T. HEATH, *Numerical methods for large sparse linear least squares problems*, this Journal, 5 (1984), pp. 497–513.

[9] AKE BJORCK, SSOR *preconditioning methods for sparse least squares problems*, in Proc. Computer Science and Statistics: 12th Annual Symposium on the Interface, J. F. Gentleman, ed., Univ. Waterloo, Ontario, 1979, pp. 21–25.

[10] M. A. SAUNDERS, *Sparse least squares by conjugate gradients: A comparison of preconditioning methods*, in Proc. Computer Science and Statistics: 12th Annual Symposium on the Interface, J. F. Gentleman, ed., Univ. Waterloo, Ontario, 1979, pp. 15–20.

[11] ALAN JENNINGS, *Development of an* ICCG *algorithm for large sparse systems*, in Preconditioning Techniques in Numerical Solution of Partial Differential Equations, D. J. Evans, ed., Gordon and Breach, New York, 1982, pp. 425–538.

[12] ———, *Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method*, J. Inst. Maths. Applics., 20 (1977), pp. 61–72.

[13] D. S. KERSHAW, *The incomplete Choleski-conjugate gradient method for iterative solution of systems of linear equations*, J. Comp. Phys., 26 (1978), pp. 42–65.

[14] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[15] T. A. MANTEUFFEL, *Shifted incomplete Choleski factorization*, in Sparse Matrix Proceedings 1978, I. S. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[16] ALAN JENNINGS AND G. M. MALIK, *Partial elimination*, J. Inst. Maths. Applics., 20 (1977), pp. 307–316.

[17] M. A. AJIZ, *Incomplete methods for computer structural analysis*, Ph.D. Dissertation, Queens University, Belfast, 1982.

[18] M. A. AJIZ AND ALAN JENNINGS, *A robust* ICCG *algorithm*, Int. J. Num. Meth. Engng., 20 (1984), pp. 949-966.

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

SIAM J. SCI. STAT. COMPUT.
Vol. 5, No. 4, December 1984

SOLUTION OF SPARSE UNDERDETERMINED SYSTEMS OF LINEAR EQUATIONS*

ALAN GEORGE,† MICHAEL T. HEATH‡ AND ESMOND NG†

Abstract. The problem of computing the minimum-norm solution to a sparse, underdetermined system of linear equations is considered. An algorithm is developed which is based on orthogonal factorization. The algorithm is generalized to cope with systems which are rank deficient, inconsistent, or require column updating.

Key words. linear equations, underdetermined systems, minimum-norm solution, sparse least squares, *LQ* factorization

1. Introduction.

This paper is concerned with solving systems of linear equations of the form

$$Ax=b, \tag{1.1}$$

where A is an $m\times n$ matrix, b is a known vector of dimension m, and x is the desired solution vector of dimension n. Our particular interest is in problems which are underdetermined ($m<n$), and for which the matrix A is large and sparse. If an underdetermined system is consistent, then it has infinitely many solutions. In this case we seek the solution x having minimum Euclidean norm. If the system is inconsistent, which can happen only if $rank(A)<m$, we seek the vector x of minimum norm which minimizes the residual norm

$$\|b-Ax\|_2.$$

A survey of methods for solving consistent underdetermined systems is given in [2]. In the present paper we adapt one of those methods, *LQ* factorization, to solve sparse problems and generalize it to include problems which may not be consistent.

In a series of papers ([4], [6], [1]) a collection of algorithms is developed for solving sparse linear least squares problems by *QR* factorization using Givens rotations. The basic algorithm of [4] is intended for overdetermined problems ($m>n$) having full column rank, which is the most commonly occurring case in practice. Subsequent generalizations ([6], [1]) extend this basic technique to solve more complicated problems having rank degeneracy, equality constraints, or updating (modifying a previous solution to incorporate new information). Due to the particular nature of the underlying sparse technique, these generalizations are of more than academic interest, even for problems that at first do not appear to require the extended capabilities. In particular, since the factorization scheme is based on the nonzero structure of A^TA, it is essential to sparsity preservation that any relatively dense rows of the input matrix be withheld from the sparse orthogonal factorization and their effect be subsequently

* Received by the editors October 25, 1982, and in revised form December 8, 1983. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc. and by the Canadian Natural Sciences and Engineering Research Council under Grant A8111. This paper was typeset at Oak Ridge National Laboratory using an Autologic APS-μ5 phototypesetter. Final copy was produced on July 2, 1984.

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

‡ Mathematics and Statistics Research Section, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830.

incorporated into the solution by updating. This in turn means that even if a problem has full rank, the sparse and dense portions, when treated separately, may not have.

Because of their ability to solve least squares problems of arbitrary rank, the QR-based algorithms of [6] and [1] can also solve underdetermined linear systems. The two factorizations LQ and QR lead to significantly different algorithms, however. We will discuss the relative merits of the two approaches after details of the LQ-based algorithm have been given. The sparse LQ factorization is obtained, in effect, by applying the QR factorization algorithm of [4] to A^T. Thus it is now dense *columns* of A which must be withheld from the factorization process and subsequently considered as updates. In summary, an underdetermined sparse system having a few dense rows can be solved by LQ without updating or by QR with row updating. An underdetermined sparse system having a few dense columns can be solved by QR without updating or by LQ with column updating. If both dense rows and dense columns are present, then either method will require updating.

Although the basic factorization algorithm of [4] produces a sparse triangular factor L or R, the orthogonal factor Q is in general not particularly sparse and is therefore not stored. Instead, the Givens rotations which triangularize A are simply discarded as they are used. This presents no problem in solving (1.1) by the QR approach, since the right-hand side vector b can be processed by the Givens rotations simultaneously with A, and Q is not needed subsequently in computing the final solution. In the standard, full rank implementation of LQ (see [2]), where

$$A=[L \quad O]Q$$

with L a nonsingular, lower triangular matrix of order m and

$$Q=\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}$$

an orthogonal matrix of order n partitioned conformally, the minimum-norm solution to (1.1) is given by

$$x=Q_1^TL^{-1}b.$$

Gill and Murray [5] and Saunders [9] have observed that this apparent need for Q can be circumvented by computing instead

$$x=A^TL^{-T}L^{-1}b, \tag{1.2}$$

in effect re-creating the necessary part of Q by taking

$$Q_1=L^{-1}A,$$

with surprisingly good numerical results. Paige [8] has analyzed and explained this good numerical behavior. The approach represented by (1.2) is a natural one for the problems in which we are interested, since we cannot retain Q. Thus, the algorithms developed in the present paper can be regarded as generalizations of (1.2) to solve sparse underdetermined systems of linear equations which may not be consistent and which may require column updating.

2. Algorithms.

First we establish some more detailed notation with which to describe our algorithms. Since we wish to segregate the sparse and dense columns, we will take A in (1.1) to be $m \times (n+p)$, partitioned as

$$A=[B \quad C], \tag{2.1}$$

where B is $m\times n$ and sparse, while C is $m\times p$ and p is assumed to be small relative to m and n. If $p>0$ and

$$x=\begin{bmatrix} u \\ v \end{bmatrix}\begin{matrix} \}n \\ \}p \end{matrix}$$

is partitioned conformally with (2.1), then (1.1) becomes

$$Bu+Cv=b. \tag{2.2}$$

Applying the algorithm of [4] to the sparse matrix B^T yields a factorization of the form

$$B=[L \quad O]Q, \tag{2.3}$$

where L is a lower triangular matrix of order m and Q is an orthogonal matrix of order n. If B does not have full row rank, say $rank(B)=m-k$, then the factorization (2.3) takes the form

$$B=\begin{bmatrix} L & O \\ M & O \end{bmatrix}Q, \tag{2.4}$$

where L is now a lower triangular matrix of order $m-k$. (In arriving at (2.4) a numerical rank decision must be made, and row and column permutations may be required, but these permutations can be handled implicitly: see [6].) Again we assume that the rank deficiency k is small relative to m and n. Whatever the rank, we partition Q as

$$Q=\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}\begin{matrix} \}m-k \\ \}n-m+k \end{matrix}$$

and use the change of variable

$$\begin{bmatrix} y \\ z \end{bmatrix}=\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix}u,$$

so that

$$u=Q_1^Ty+Q_2^Tz.$$

If

$$B=\begin{bmatrix} B_1 \\ B_2 \end{bmatrix}\begin{matrix} \}m-k \\ \}k \end{matrix}$$

is permuted and partitioned conformally with (2.4), then

$$Q_1=L^{-1}B_1,$$

and this fact will be used to avoid storing Q. When $k>0$ we will also partition the right-hand side vector as

$$b=\begin{bmatrix} c \\ d \end{bmatrix}\begin{matrix} \}m-k \\ \}k \end{matrix} \quad .$$

The complexity of solving a linear system of the form (2.2) is characterized by the number p of dense columns, the rank deficiency k of B, and whether the system is consistent. A completely general algorithm for solving the most complex case — a rank-deficient, inconsistent problem which requires updating — is rather complicated and somewhat difficult to follow. We will therefore present a sequence of algorithms covering various cases, beginning with the most simple and leading up to the most complex. In this way the justification for each aspect of the final algorithm should become clear.

Case 1. $p = 0$, $k = 0$, consistent. Under the factorization (2.3), system (2.2) becomes

$$Bu=[L \quad O]Qu=[L \quad O]\begin{bmatrix} y \\ z \end{bmatrix}=b.$$

Now y is completely determined by the nonsingular triangular system $Ly=b$, and thus, since the orthogonal matrix Q does not change the norm, the minimum-norm solution occurs when $z=0$. Therefore the minimum-norm solution is given by

$$x=u=Q^T\begin{bmatrix} y \\ 0 \end{bmatrix}=Q_1^Ty=B^TL^{-T}L^{-1}b.$$

Hence we have the following algorithm for case 1:

ALGORITHM 1.

1. Factor B as in (2.3).
2. Solve $Ly=b$.
3. Solve $L^Tt=y$.
4. $x=u=B^Tt$.

Case 2. $p = 0$, $k > 0$, consistent. The factorization of B now has the form of (2.4), and the system (2.2) becomes

$$\begin{bmatrix} L & O \\ M & O \end{bmatrix}\begin{bmatrix} y \\ z \end{bmatrix}=\begin{bmatrix} c \\ d \end{bmatrix}.$$

Again $z=0$, and consistency implies that if $Ly=c$, then $My=d$. Therefore the minimum-norm solution is given by

$$x=Q_1^Ty=B_1^TL^{-T}L^{-1}c.$$

Hence we have the following algorithm for case 2:

ALGORITHM 2.

1. Factor B as in (2.4).
2. Solve $Ly=c$.
3. Solve $L^Tt=y$.
4. $x=u=B_1^Tt$.

Case 3. $p > 0$, $k = 0$, consistent. Under the factorization (2.3), system (2.2) now becomes

$$[L \quad O]\begin{bmatrix} y \\ z \end{bmatrix}+Cv=b. \tag{2.5}$$

Once again minimizing the norm requires that $z=0$, so that we may write (2.5) as

$$Ly+Cv=b. \tag{2.6}$$

Since L is nonsingular, y is completely determined by (2.6) if v is known. In particular,

$$y=w-Ev, \tag{2.7}$$

where E is an $m\times p$ matrix satisfying $LE=C$, and w is a vector satisfying $Lw=b$. Thus, minimizing $\|y\|^2+\|v\|^2$ subject to (2.7) is equivalent to the least squares problem

$$\min_v \left\| \begin{bmatrix} I \\ E \end{bmatrix} v - \begin{bmatrix} 0 \\ w \end{bmatrix} \right\|_2 . \tag{2.8}$$

By our assumption that p is relatively small, (2.8) can be solved by methods appropriate for small dense least squares problems (e.g., [3], [7]). In particular, if we use the factorization

$$\begin{bmatrix} I \\ E \end{bmatrix} = U \begin{bmatrix} R \\ O \end{bmatrix}, \tag{2.9}$$

with U an orthogonal matrix of order $m+p$ and R an upper triangular matrix of order p and partition U conformally as

$$U=\begin{bmatrix} U_{11} & U_{21} \\ U_{12} & U_{22} \end{bmatrix}$$

with U_{11} $p\times p$, then the solution v is given by the nonsingular triangular linear system $Rv=U_{21}^T w$ and the part of the residual in which we are interested is given by the projection $y=U_{22}U_{22}^T w$. Hence we have the following algorithm for case 3:

ALGORITHM 3.

1. Factor B as in (2.3).
2. Solve $LE=C$.
3. Solve $Lw=b$.
4. Compute the factorization (2.9).
5. Solve $Rv=U_{21}^T w$.
6. $y=U_{22}U_{22}^T w$.
7. Solve $L^T t=y$.
8. $u=B^T t$.
9. $x=\begin{bmatrix} u \\ v \end{bmatrix}$.

Case 4. $p>0$ **,** $k>0$ **, consistent.** Under the factorization (2.4), system (2.2) now becomes

$$\begin{bmatrix} L & O \\ M & O \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} + \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} v = \begin{bmatrix} c \\ d \end{bmatrix},$$

where we have partitioned C conformally with (2.4). Again we have $z=0$, and consistency implies that if $Ly+C_1v=c$, then $My+C_2v=d$. Thus the algorithm for this case is very similar to Algorithm 3.

ALGORITHM 4.

1. Factor B as in (2.4).
2. Solve $LE=C_1$.
3. Solve $Lw=c$.
4. Compute the factorization (2.9).
5. Solve $Rv=U_{21}^T w$.
6. $y=U_{22}U_{22}^T w$.
7. Solve $L^T t=y$.
8. $u=B_1^T t$.
9. $x=\begin{bmatrix} u \\ v \end{bmatrix}$.

Case 5. $p = 0$, $k > 0$, inconsistent. Define the residual vector

$$r=Ax-b.$$

Since we are no longer assuming $r=0$, we now seek the vector x of minimum norm which minimizes $\|r\|$. Under the factorization (2.4), the residual vector becomes

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix}=\begin{bmatrix} L & O \\ M & O \end{bmatrix}\begin{bmatrix} y \\ z \end{bmatrix}-\begin{bmatrix} c \\ d \end{bmatrix}, \tag{2.10}$$

where we have partitioned r conformally with (2.4). It is clear that $z=0$, and from the top equation of (2.10) we can solve for y in terms of r by using the system

$$Ly=c+r_1.$$

Thus, if we let H be a $k\times(m-k)$ matrix satisfying $L^TH^T=M^T$ and define the vector $e=d-Hc$, then we have from the bottom equation of (2.10)

$$[H \quad -I]\begin{bmatrix} r_1 \\ r_2 \end{bmatrix}=e. \tag{2.11}$$

Since we want to minimize $\|r\|$ subject to the constraint (2.11), we have reduced the original large sparse rank deficient problem to the solution of a much smaller ($k\times m$) full-rank minimum-norm problem which can be solved by methods appropriate for small dense underdetermined systems (see, e.g., [2]). One such method is to use the factorization

$$[H \quad -I]=[T \quad O]V=[T \quad O]\begin{bmatrix} V_1 \\ V_2 \end{bmatrix}, \tag{2.12}$$

where T is a lower triangular matrix of order k and V is an orthogonal matrix of order m partitioned conformally, so that

$$r=V_1^T T^{-1}e.$$

(Note that although V is a large matrix, we need only the much smaller $k\times m$ portion V_1; moreover, we can always avoid retaining V by using the same technique employed in (1.2).) Hence we have the following algorithm for case 5:

ALGORITHM 5.

1. Factor B as in (2.4).
2. Solve $L^T H^T = M^T$.
3. $e = d - Hc$.
4. Compute the factorization (2.12).
5. Solve $Ts = e$.
6. $r = V_1^T s$.
7. $q = c + r_1$.
8. Solve $Ly = q$.
9. Solve $L^T t = y$.
10. $x = u = B_1^T t$.

Case 6. $p > 0$, $k > 0$, **inconsistent.** The residual vector now has the form

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} L & O \\ M & O \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} + \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} v - \begin{bmatrix} c \\ d \end{bmatrix}. \tag{2.13}$$

Once again $z = 0$ and from the top equation of (2.13) we can solve for y in terms of r and v by using the system

$$Ly = c + r_1 - C_1 v. \tag{2.14}$$

If we let H and e be defined as in case 5 and in addition define the matrix $F = C_2 - HC_1$, then the bottom equation of (2.13) becomes

$$[H \quad -I] \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = e - Fv. \tag{2.15}$$

Once again we have a small, full-rank, minimum-norm problem for r in (2.15), but we do not yet know v. If we apply the orthogonal transformation (2.12), however, (2.15) becomes

$$[I \quad O] \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = f - Gv, \tag{2.16}$$

where $s = Vr$, and f and G satisfy $Tf = e$ and $TG = F$, respectively. It is now obvious from (2.16) that $\|s\|$, and hence, since the orthogonal transformation V does not change the norm, $\|r\|$ is minimized when v is chosen to minimize the residual

$$\min_v \|f - Gv\|_2. \tag{2.17}$$

Since G is $k \times p$, (2.17) is a very small least squares problem. Although (2.17) determines s (and hence r) uniquely, it does not determine a unique value for v unless G has full column rank. If the latter is not the case, then we must choose v among all solutions of (2.17) so as to minimize $\|y\|^2 + \|v\|^2$ in (2.14).

Any solution of (2.17) can be expressed in the form $v = \bar{v} + Z\beta$, where $\bar{v}$ is a particular solution to (2.17), Z is a matrix whose columns form a basis for the null space of G, and β is a vector of appropriate dimension. (Of course, if G has full column rank, then Z is not present, and $v = \bar{v}$.) Suitable $\bar{v}$ and Z can be conveniently computed by means of the singular value decomposition of G (see, e.g., [3] or [7]). To determine β we formally substitute $v = \bar{v} + Z\beta$ into (2.14). The resulting least squares problem analogous to (2.8) is

$$\min_{\beta} \left\| \begin{bmatrix} Z \\ EZ \end{bmatrix} \beta - \begin{bmatrix} -\bar{v} \\ w - E\bar{v} \end{bmatrix} \right\|_2, \tag{2.18}$$

where E and w are given by the linear systems $LE = C_1$ and $Lw = c + r_1$. The full-rank least squares problem (2.18) can be solved by an orthogonal-triangular factorization analogous to (2.9), with the $(m+p) \times (m+p)$ orthogonal matrix U partitioned similarly. The part of the residual in which we are interested is then given by the projection

$$y = U_{22}[U_{12}^T \;\; U_{22}^T] \begin{bmatrix} -\bar{v} \\ w - E\bar{v} \end{bmatrix}. \tag{2.19}$$

Hence we have the following algorithm for case 6:

ALGORITHM 6.

```
1. Factor $B$ as in (2.4).
2. Solve $L^T H^T = M^T$.
3. $e = d - Hc$.
4. Compute the factorization (2.12).
5. $F = C_2 - HC_1$.
6. Solve $TG = F$.
7. Solve $Tf = e$.
8. Compute a particular solution $\bar{v}$ to (2.17), along with the residual vector $s$
   and, if $G$ does not have full column rank, the null-space basis matrix $Z$.
9. $r = V_1^T s_1$.
10. Solve $LE = C_1$.
11. Solve $Lw = c + r_1$.
12. If $G$ has full column rank, set $U = I$, the identity matrix of order $m + p$;
    otherwise, let $U$ be the orthogonal matrix which triangularizes the matrix of
    (2.18).
13. Compute $y$ as in (2.19).
14. Solve $L^T t = y$.
15. $u = B_1^T t$.
16. $x = \begin{bmatrix} u \\ v \end{bmatrix}$.
```

3. Concluding remarks. We have developed an algorithm based on LQ factorization for computing the minimum-norm solution to a sparse underdetermined system of linear equations. We now compare this approach with another orthogonalization method, that based on QR factorization. Using the sparse Givens approach of [4], the nonzero structure of the triangular factor resulting from QR or LQ factorization is determined by the nonzero structure of $A^T A$ or AA^T, respectively. Thus either factorization may suffer more fill than the other, depending on the problem, and hence require more storage or computation. Generally speaking, however, the two factorizations should be about equal in cost for square or nearly square systems, but for more strongly underdetermined systems $(m \ll n)$, LQ factorization should require more computation because a larger proportion of the input matrix A is annihilated (Fig. 1).

Unfortunately, in order to compute the minimum-norm solution to an underdetermined problem of rank m by the algorithm of [6], the sparse QR factorization is fol-

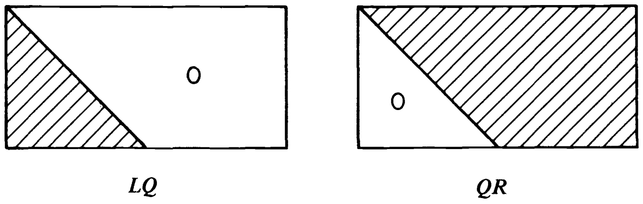


FIG. 1. *Triangular forms resulting from orthogonal factorization.*

lowed by solving a subsequent (generally dense) least squares problem of size $n \times (n-m)$. Thus, although the basic sparse factorization may be cheaper, the *QR* approach is impractical unless $n-m$ is relatively small. (In fairness it should be pointed out that for cases in which any particular solution of (1.1) will suffice, the *QR* algorithm does not suffer from this heavy restriction (see [6]).) The possibility of row or column updating further complicates any comparison of methods. Nevertheless, it seems apparent that the *LQ*-based algorithm is the method of choice for a broad range of underdetermined systems.

This conclusion is supported by the numerical results reported in Table 1. Three underdetermined test problems were generated by transposing three overdetermined problems from the Harwell sparse matrix test set and by letting the right hand side vectors have as components 1, 2, ..., m. All three problems are of full rank and require no updating. The times reported are in seconds on an IBM 4341 using the VS Fortran compiler and single precision floating point arithmetic. The storage reported is in words and includes storage for all pointers and other overhead required to solve the problem. As expected for problems having many more columns than rows, the time and storage required for computing the minimum-norm solution using the *QR* factorization are huge. Therefore, for a more direct comparison of the sparse orthogonal factorizations, we have also reported the time and storage required for computing a basic solution using the *QR* factorization. These figures show that for these three problems the *QR* factorization is somewhat faster than the *LQ* factorization, but requires significantly more storage. As for accuracy, the minimum-norm solutions computed by the two algorithms agree very well for the first two problems but differ appreciably for the third problem, which is more ill conditioned.

TABLE 1
Numerical test results

Problem	ASH219		ASH331		ABB313	
rows	85		104		176	
columns	219		331		313	
nonzeros	438		662		1557	
	store	time	store	time	store	time
LQ (min. sol.)	1642	1.96	2261	3.57	6821	4.03
QR (bas. sol.)	5944	1.26	10688	1.73	15310	3.13
QR (min. sol.)	34920	24.79	85705	94.79	50728	40.17

Acknowledgment. The authors are pleased to acknowledge the helpful insights and persistent prodding of Michael Saunders, which substantially improved the content and presentation of this paper.

REFERENCES

[1] Å. BJÖRCK, *A general updating algorithm for constrained linear least squares problems*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 394-402.

[2] R. E. CLINE AND R. J. PLEMMONS, *l_2-solutions to underdetermined linear systems*, SIAM Review, 18 (1976), pp. 92-106.

[3] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER AND G. W. STEWART, LINPACK *Users' Guide*, SIAM, Philadelphia, 1979.

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

[5] P. E. GILL AND W. MURRAY, *A numerically stable form of the simplex method*, Linear Algebra Appl., 7 (1973), pp. 99-138.

[6] M. T. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 223-237.

[7] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[8] C. C. PAIGE, *An error analysis of a method for solving matrix equations*, Math. Comp., 27 (1973), pp. 355-359.

[9] M. A. SAUNDERS, *Large-scale linear programming using the Cholesky factorization*, Report No. CS 252, Computer Science Dept., Stanford University, Stanford, CA, January 1972.