Direct Elliptic Equation Solvers with Low Memory Requirements

MARK R. SCHOEBERL

Naval Research Laboratory, Washington, DC 20375

(Manuscript received 16 January 1980, in final form 12 April 1980)

ABSTRACT

Several simple modifications of the Lindzen-Kuo Gaussian elimination algorithm for solving elliptic differential equations are derived. These modifications greatly decrease the auxiliary memory requirements with only some increase in computation, thus making this method suitable for solving general elliptic equations over large grids.

1. Introduction

The solution of elliptic differential equations, such as Poisson's equation, is a necessity in many areas of geophysics. The numerical methods used for solving these equations fall into two broad categories: iterative solvers and direct solvers. With iterative solvers solutions are obtained by iterative correction of an initial guess solution until a suitable error level is reached. These solvers can be quite fast for some problems and have low auxiliary memory requirements but require an unknown number of operations to achieve the desired results. Examples of popular iterative methods are SOR (successive overrelaxation) and ADI (alternate direction implicit), which are described in most textbooks on numerical methods, and ICCG [incomplete Cholesky-conjugate gradient (Kershaw, 1978)].

Direct solvers obtain a solution in a fixed number of operations, but they usually require large auxiliary memory requirements. The advantage of direct solvers for some problems is that in a region where diagonal dominance is weak they produce quick, accurate solutions where iterative solvers converge very slowly. For separable differential equations, direct solvers may also have relatively low memory requirements (Rosmond and Faulkner, 1976; Bank and Rose, 1977). However, this class of equations is extremely restrictive.

For nonseparable equations the Gaussian elimination algorithm discussed by Lindzen and Kuo (1969, hereafter referred to as LK) provides accurate solutions but has very large auxiliary memory requirements. It is simple to code and appears stable for a wide class of elliptic problems. More efficient algorithms exist for Gaussian elimination such as nested dissection (George, 1973, 1976) or Cuthill-McKee (Lin and Sherman, 1976) but so far these techniques have been applied to positive definite, symmetric matrices and appear to be difficult to code. Madala (1978) has developed a fast direct solver by blocking the error propagation scheme of Hirota *et al.* (1970). Madala's method (SEVP) requires a factor of 5 less auxiliary memory than LK, but it is also somewhat difficult to code.

In this paper we will show how the LK method may be modified to reduce the memory requirements substantially on an $N \times M$ grid ($M \geq N$). Two variations of the LK method will be presented—S1 and S2. The first is suitable for large grids where $M \geq 100$, while the second is suitable for smaller arrays. A hybrid method for moderately sized grids is also presented. The S1 method requires $2\sqrt{M} \times (N+1)N$ auxiliary memory with a factor of 2 asymptotic increase in computation. The S2 method requires $M(N+1)N/10$ auxiliary memory. This should be compared to SEVP which has memory requirements of $N^2M/5$, LK with $N(N+1)M$ and Crout reduction (Cholesky's method), with $2N^2M$.

2. Computational algorithm

a. *The Lindzen-Kuo algorithm*

In order to introduce the new direct solvers it is necessary to review the LK method.

We consider the differential equation

$$A\frac{\partial^2\psi}{\partial z^2} + B\frac{\partial^2\psi}{\partial y^2} + C\frac{\partial\psi}{\partial z} + D\frac{\partial\psi}{\partial y} + E\psi = F,$$

where A, B, C, D, E and F are functions of z and y. Using second-order differencing we may write this equation as

$$\bar{\mathbf{R}}_j\psi_{j+1} + \bar{\mathbf{S}}_j\psi_j + \bar{\mathbf{Q}}_j\psi_{j-1} = \bar{\mathbf{F}}_j, \qquad (1)$$

where

$$R_j^{i,k} = (A_{i,j}/\Delta z^2 + C_{i,j}/2\Delta z)\delta_{i,k},$$

$$Q_j^{i,k} = (A_{i,j}/\Delta z^2 - C_{i,j}/2\Delta z)\delta_{i,k},$$

$$S_j^{i,k} = (E_{i,j} - 2A_{i,j}/\Delta z^2 - 2B_{i,j}/\Delta y^2)\delta_{i,k} + (B_{i,j}/\Delta y^2 + D_{i,j}/2\Delta y)\delta_{i,k-1} + (B_{i,j}/\Delta y^2 - D_{i,j}/2\Delta y)\delta_{i,k+1},$$

and $z = j\Delta z$, $y = i\Delta y$ and $\delta_{i,k}$ is the Kronecker delta function $\bar{\mathbf{R}}$, $\bar{\mathbf{S}}$ and $\bar{\mathbf{Q}}$ are matrices; $\boldsymbol{\psi}$ and $\mathbf{F}$ are vectors.

If we assume that

$$\boldsymbol{\psi}_{j-1} = \bar{\boldsymbol{\Omega}}_j\boldsymbol{\psi}_j + \boldsymbol{\Gamma}_j. \tag{2}$$

Then using Eq. (1) we may derive the following recursion relations for $\boldsymbol{\Omega}$ and $\boldsymbol{\Gamma}$:

$$\bar{\boldsymbol{\Omega}}_{j+1} = -(\bar{\mathbf{Q}}_j\bar{\boldsymbol{\Omega}}_j + \bar{\mathbf{S}}_j)^{-1}\bar{\mathbf{R}}_j, \tag{3a}$$

$$\boldsymbol{\Gamma}_{j+1} = (\bar{\mathbf{Q}}_j\bar{\boldsymbol{\Omega}}_j + \bar{\mathbf{S}}_j)^{-1}(\mathbf{F}_j - \bar{\mathbf{Q}}_j\boldsymbol{\Gamma}_j). \tag{3b}$$

The LK method starts at the bottom boundary ($j = 0$) and generates $\boldsymbol{\Omega}_1$ and $\boldsymbol{\Gamma}_1$ from the boundary conditions. Eqs. (3a) and (3b) are then used to produce $\bar{\boldsymbol{\Omega}}$'s and $\bar{\boldsymbol{\Gamma}}$'s for each level j until the upper boundary is reached. The Ω's and Γ's are stored in auxiliary memory. To obtain the solution, the upper boundary condition along with Eq. (2) is used to generate $\bar{\boldsymbol{\psi}}_M$. Subsequent application of Eq. (2) produces $\bar{\boldsymbol{\psi}}_j$ at each level. The total auxiliary memory required is $N(N+1)M$, where $N \leq M$.

The LK method is essentially a block form of the tridiagonal Gaussian elimination algorithm which can be applied to both real and complex differential equations. Higher order equations in both a differential and difference sense can be treated using the same basic technique by modifying Eq. (2). Dirichlet, Neumann or mixed boundary conditions are easily treated in the j direction but periodic boundary conditions are difficult to impose. All types of boundaries are allowed in the i direction. The speed of the LK method depends critically on the matrix inversion required in Eq. (3) since M inversions are required to compute the solution.

LK can be broken with two steps: set-up and solution. For LK the set-up corresponds to producing the $\bar{\boldsymbol{\Omega}}_j$ and matrices by Eq. (3a). The solution step is the application of Eqs. (2) and (3b). Provided only F changes between successive applications of the method, the $\bar{\boldsymbol{\Omega}}_j$ matrices may be stored and $\boldsymbol{\Gamma}_j$ vectors can be regenerated by modifying Eq. (3b) to

$$\boldsymbol{\Gamma}_{j+1} = -\bar{\boldsymbol{\Omega}}_{j+1}\bar{\mathbf{R}}_j^{-1}(\mathbf{F}_j - \bar{\mathbf{Q}}_j\boldsymbol{\Gamma}_j).$$

Since $\bar{\mathbf{R}}_j$ is diagonal, no actual matrix inversions are required. This procedure increases the algorithm speed by a factor of 10.

b. Error propagation in the Lindzen-Kuo algorithm

It is of interest to ask how round-off error is treated in the generation of the $\bar{\boldsymbol{\Omega}}_j$, $\bar{\boldsymbol{\Gamma}}_j$ arrays and the solution vectors $\boldsymbol{\Psi}_j$. For simplicity take S, Q and R as scalars independent of j. These scalers might be defined as $S = \max\|\bar{\mathbf{S}}_j\|$, for example. For large grids ($\Omega_j \approx \Omega_{j+1}$) we obtain from Eq. (3a)

$$\Omega \approx [-S + (S^2 - 4Q/R)^{1/2}]/2Q, \tag{4}$$

where we have taken S, Q and R to be scalars with constant values over the grid. If an error ϵ is introduced at level j then

$$\Omega_{j+1} = \Omega_{j+1}^e(1 + \epsilon\Omega_{j+1}^e Q/R), \tag{5}$$

where Ω^e is the value of Ω without error. The scheme becomes unstable if $\Omega_{j+1}^e Q/R > 1$ which for Q, $R = 1$ occurs for $S > -2$. This is the limit for which Eq. (4) is valid and also the limit for diagonal dominance. That is, LK is unstable for a non-diagonally dominent system and Ω_j grows with j. However, the practical limit on the stability of the LK scheme is not as severe as suggested by Eqs. (5) and (6). For example, if Eq. (1) is non-diagonally dominant within small regions the overall error accumulation will still be within tolerable limits.

c. Block storage memory reduction (S1)

The first scheme to reduce the memory requirements for LK is called S1. The j grid is divided into blocks of some size (IBLK) as indicated in Fig. 1 and we begin as before generating $\bar{\boldsymbol{\Omega}}_j$ and $\bar{\boldsymbol{\Gamma}}_j$ starting with the lower boundary and moving to the point $M - \text{IBLK} + 1$ where computations are halted. $\bar{\boldsymbol{\Omega}}_j$ and $\bar{\boldsymbol{\Gamma}}_j$ are stored only at points K*IBLK + 1 where K is the block number. Next we start at the upper boundary and generate a new set of $\bar{\boldsymbol{\Omega}}_j'$ and $\bar{\boldsymbol{\Gamma}}_j'$ from

$$\left.\begin{aligned}\bar{\boldsymbol{\Omega}}_{j-1}' &= -(\bar{\mathbf{R}}_j\bar{\boldsymbol{\Omega}}_j' + \bar{\mathbf{S}}_j)^{-1}\bar{\mathbf{Q}}_j \\ \bar{\boldsymbol{\Gamma}}_{j-1}' &= (\bar{\mathbf{R}}_j\bar{\boldsymbol{\Omega}}_j' + \bar{\mathbf{S}}_j)^{-1}(\mathbf{F}_j - \bar{\mathbf{R}}_j\boldsymbol{\Gamma}_j')\end{aligned}\right\}. \tag{6}$$

These equations are derived by assuming

$$\boldsymbol{\psi}_{j+1} = \bar{\boldsymbol{\Omega}}_j'\boldsymbol{\psi}_j + \boldsymbol{\Gamma}_j'. \tag{7}$$

The matrices $\bar{\boldsymbol{\Omega}}_j'$ and vectors $\boldsymbol{\Gamma}_j'$ are stored until the level $M - \text{IBLK}$ is reached. Then Eqs. (7) and (2) may be used to find $\boldsymbol{\psi}_j$. The resulting equation is

$$\boldsymbol{\psi}_j = (1 - \bar{\boldsymbol{\Omega}}_{j+1}\bar{\boldsymbol{\Omega}}_j')(\bar{\boldsymbol{\Omega}}_j\boldsymbol{\Gamma}_j' + \boldsymbol{\Gamma}_j). \tag{8}$$

The stored values of $\bar{\boldsymbol{\Omega}}_j'$ and $\boldsymbol{\Gamma}_j'$ along with Eq. (7) may then be used to determine $\boldsymbol{\psi}_j$ within the rest of the block as in Fig. 1. Using the values of $\bar{\boldsymbol{\Omega}}_j'$ and $\boldsymbol{\Gamma}_j'$ at the block boundary we may start again at the next block generating $\bar{\boldsymbol{\Omega}}_j'$ and $\boldsymbol{\Gamma}_j'$. Since there is no need to retain $\bar{\boldsymbol{\Omega}}_j'$ and $\boldsymbol{\Gamma}_j'$ from the previous block we use the same auxiliary storage for the new

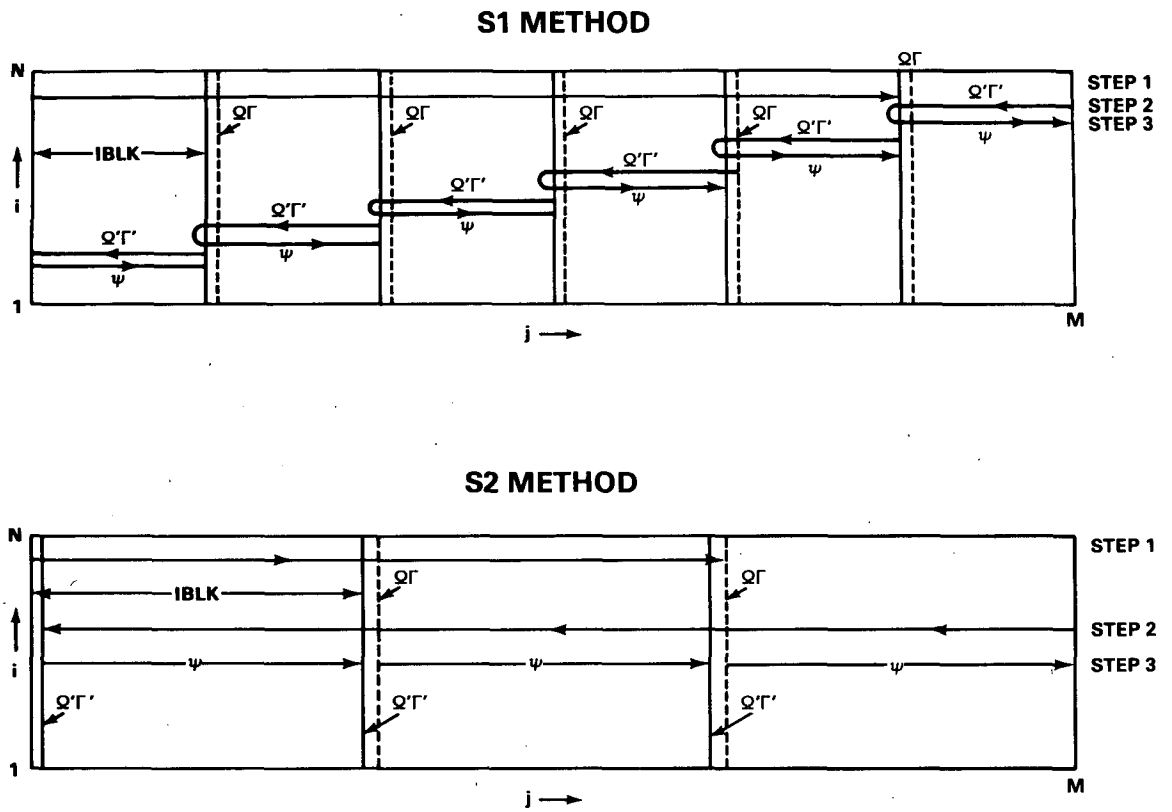

FIG. 1. Computational diagrams showing the sweep direction for the S1 and S2 methods. See text for discussion.

values of $\bar{\Omega}_j'$ and Γ_j'. Using Eq. (8) at the next block boundary the process described above is repeated over for each block.

The auxiliary memory requirement A for this scheme is given by

$$N(N + 1)(M/\text{IBLK} - 1 + \text{IBLK}) = A,$$

which is minimized when

$$\text{IBLK} = M^{1/2}.$$

The auxiliary memory requirement is thus given by

$$A = (2M^{1/2} - 1)N(N + 1).$$

Since additional matrix inversions are required to compute $\bar{\Omega}'$ and Γ', the computational load is increased. However, the increase is never greater than a factor of 2 and may be as low as 1.5 (2 blocks).

d. Block storage with back substitution

For small arrays ($M < 100$) further memory reduction is possible by utilizing Eq. (1). In this method, the procedure described in the last section is used except that Ω' and Γ' arrays are not stored in memory. At the end of the block, ψ_j is computed using Eq. (8). ψ_{j+1} is then computed using $\bar{\Omega}_j'$ and Γ_j' and Eq. (1) is used to compute the value of ψ' in the rest of the block. The procedure is illustrated in Fig. 1.

The limit on the size of IBLK is now given by the error growth involved in application of Eq. (1). If error ϵ_j is introduced at level j, it grows at least by

$$|\epsilon_{j+1}| \leqslant S\epsilon_j/R$$

for each step. For diagonal systems ($S/R \gg 1$) this exponential growth of the truncation error limits the applications of Eq. (1) to a small number of steps. Since $\epsilon_j \sim 10^{-6}$ on 4-byte word machines, IBLK must be less than 5 for Poisson's equation if $\epsilon_{\max} \leqslant 10^{-3}$. For 8-byte word calculations $\epsilon_j \sim 10^{-16}$ so IBLK must be less than 10 for $\epsilon_{\max} \leqslant 10^{-8}$. If these error limits can be tolerated.

$$A = \frac{M}{10} N(N + 1) \quad \text{(8-byte words)},$$

$$A = \frac{MN}{5} (N + 1) \quad \text{(4-byte words)}.$$

The break-even point for 4-byte word calculations between the method described in the last section and the one described above (S2) is $M \sim 100$. That is, if $M \geqslant 100$, the S1 method should be used, and if $M \leqslant 100$, S2 should be used.

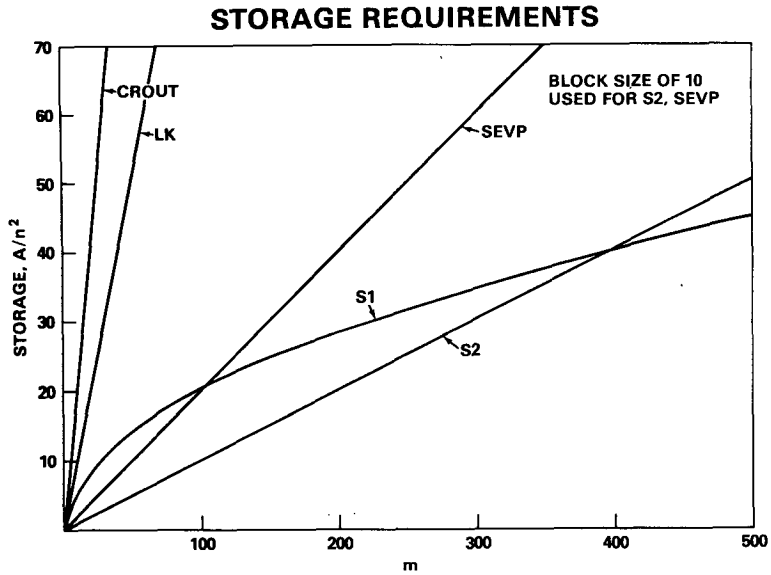

FIG. 2. Auxiliary memory requirements for several direct solvers. A is the total auxiliary requirement for an $N \times M$ grid. Block size of 10 is used for SEVP and S2. Block size of S1 is $\sqrt{M}$.

e. A hybrid method

Further auxiliary storage reduction may be obtained by combining the S1 and S2 methods. For example, every other $\bar{\Omega}_j'\Gamma_j'$ is retained so that Eq. (1) is applied where $\bar{\Omega}_j'$ and Γ_j' have not been stored and Eq. (7) is applied where they have been stored. Because $\bar{\Omega}'$ is generally less than 1, the error introduced and amplified by application of Eq. (1) is reduced by application of Eq. (7). The auxiliary requirement for the hybrid method is

$$A = 2N(N + 1)[(M/R)^{1/2} - 1],$$

where R is the fraction of time Eq. (1) is used instead of Eq. (12). For many problems R can be ½, ⅓ or ¼.

3. Comparison with other elliptic solvers

The two important factors which determine the utility of an elliptic equation solver are storage requirements and speed. Fig. 2 compares the storage requirements for S1, S2 and the methods SEVP,[1] Crout reduction and LK for an $M \times M$ grid. Nested dissection and sparse elimination may have auxiliary requirements as low as $7M^2/2$ and $O(M^2 \log_2 M)$, respectively (Eisenstat *et al.*, 1978), but so far have only been applied to symmetric matrices. The figure shows that S1 and SEVP are competitive in the small grid range but S2 is clearly superior. For large grids, S1 is superior to all methods compared here.

[1] SEVP memory requirements were calculated assuming double precision words. On machines with 60-bit words SEVP requirements are half that shown.

For timing, we have compared the same methods and SOR solving Poisson's equation with random forcing. This problem is not a very stringent test of capability of the direct solvers, but because the convergence properties of SOR vary so greatly with the diagonal dominance of the elliptic equation we felt it was reasonable "best case" problem for the

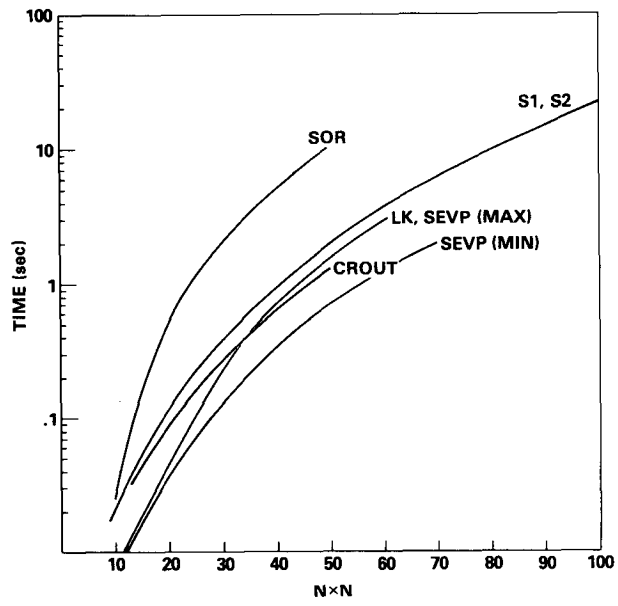

FIG. 3. Execution time is seconds for several elliptic equation solvers over an $N \times N$ grid. All solvers except SEVP use single precision, 4-byte words. SEVP is most efficient using 8-byte words.

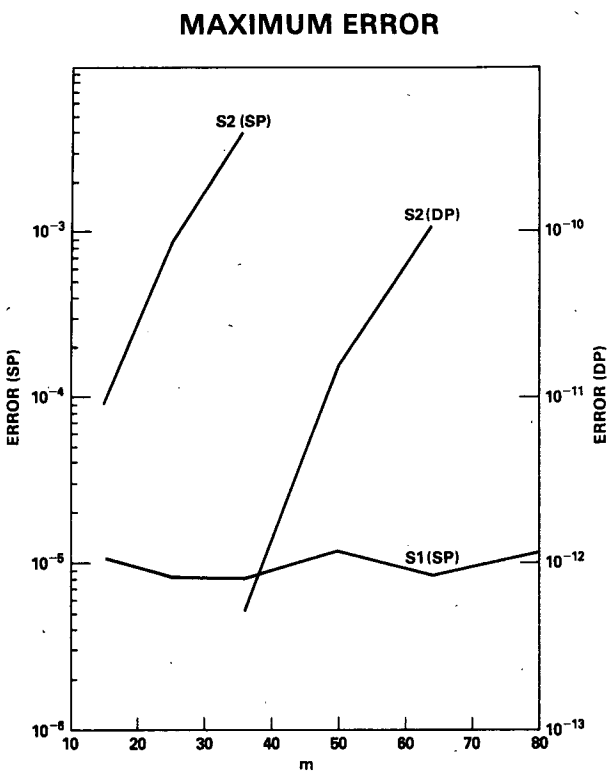

FIG. 4. Error for S1 and S2 solvers as a function of block size which is $\sqrt{M}$. SP stands for single precision; left axis gives the relative error. DP stands for double precision; right axis gives the relative error.

iterative solver. Note that iterative solvers such as SSOR and ICCG, and the direct solvers such as block cyclic reduction would probably give better results for this particular problem, but we chose SOR because it has wide usage. Fig. 3 shows the results of these tests. The computer used was a Texas Instruments ASC with two vector pipes. Since this type of computer does not "vectorize" the recursive statements, iterative solvers like SOR are not most efficient as the timing comparisons indicate. All methods were found to be superior to SOR. S1 and S2 were found to be slightly slower than LK because of the larger number of matrix inversions required. The time required for SEVP has a range depending on block size which is a function of the diagonality of the elliptic equation. The actual time for SEVP will fall between the LK curve and the SEVP curve for smaller blocks. Like LK, if only F changes between consecutive problems, SEVP can be a factor of 10 faster for the second solution.

Fig. 4 shows the maximum relative error obtained using S1 and S2. The block size for S1 and S2 are the same in this case. Note that the relative error for S2 increases exponentially as the block increases in size. This confirms the results of the error analysis given in Section 2d. S1 error is independent of M since the method used to compute the solution is always stable.

4. Summary

Block storage memory reduction and block storage with back substitution appear to greatly reduce the memory requirements for the LK method (block Gaussian elimination) with only some increase in computational overhead. As a consequence these methods should make the solution of general elliptic equations over a large grid by direct methods feasible.

Acknowledgments. The author would like to acknowledge helpful discussions and comments by Drs. R. V. Madala, R. S. Lindzen and the reviewers. The back substitution method (Section 2d) was originally suggested by Ed Schneider and communicated to the author by R. S. Lindzen. Support for this research was provided by Office of Naval Research.

REFERENCES

Bank, R. E., and D. J. Rose, 1977: Marching algorithms for elliptic boundary value problems. I. Constant coefficient case. *SIAM J. Numer. Anal.*, **14**, 792–828.

Eisenstat, S. G., M. H. Schultz, and A. H. Sherman, 1978: Software for sparse Gaussian elimination with limited core storage. *SIAM Sparse Matrix Proceedings*, 135–153.

George, A., 1973: Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, **10**, 345–363.

——, 1976: Numerical experiments using dissection methods to solve n by n grid problems. *SIAM J. Numer. Anal.*, **14**, 161–179.

Hirota, I., T. Tokioka and M. Nishigucyi, 1970: A direct solution of Poisson's equation by generalized sweep-out method. *J. Meteor. Soc. Japan*, **48**, 161–167.

Kershaw, D. S., 1978: The complete Cholesky-conjugate gradient method for iterative solution of systems of linear equations. *J. Comput. Phys.*, **23**, 43–65.

Lindzen, R. S., and H. L. Kuo, 1969: A reliable method for the numerical integration of a large class of ordinary and partial differential equations. *Mon. Wea. Rev.*, **97**, 732–734.

Lin, W., and A. H. Sherman, 1976: Comparative analysis of the Cuthill-McKee and reverse Cuthill-McKee ordering algorithms for sparse matrices. *SIAM J. Numer. Anal.*, **13**, 198–213.

Madala, R. V., 1978: An efficient direct solver for separable and non-separable elliptic equations. *Mon. Wea. Rev.*, **106**, 1735–1741.

Rosmond, T. E., and F. D. Faulkner, 1976: Direct solution of elliptic equations by block cyclic reduction and factorization. *Mon. Wea. Rev.*, **104**, 641–649.