JOURNAL OF COMPUTATIONAL PHYSICS **68**, 474–484 (1987)

A 3-D Poisson Solver Based on Conjugate Gradients Compared to Standard Iterative Methods and Its Performance on Vector Computers

H. KAPITZA AND D. EPPEL

Forschungszentrum Geesthacht,
Postfach 1160,
D-2054 Geesthacht, Federal Republic of Germany

Received September 30, 1985; revised March 14, 1986

A conjugate gradient method for solving a 3-D Poisson equation in Cartesian unequally spaced coordinates is tested in concurrence to standard iterative methods. It is found that the tested algorithm is far superior to Red-Black-SOR with optimal parameter. In the conjugate gradient method no relaxation parameter is needed, and there are no restrictions on the number of gridpoints in the three directions. The iteration routine is vectorizable to a large extent by the compiler of a CYBER 205 without any special preparations. Utilizing some special features of vector computers it is completely vectorizable with only minor changes in the code. © 1987 Academic Press, Inc.

1. INTRODUCTION

Many numerical models in fluid dynamics require the solution of a Helmholtz equation for pressure p,

$$\Delta p + \mathbf{C} \cdot \nabla p = F \tag{1}$$

where $\mathbf{C}$ and F are space-dependent vector and scalar functions defined by the specific problem. The discrete forms of the first and second derivatives with respect to coordinate x appearing in the operators of Eq. (1) are

$$\frac{\partial p}{\partial x} = \frac{h_i^2 p_{i+1} - (h_i^2 - h_{i+1}^2) p_i - h_{i+1}^2 p_{i-1}}{h_i h_{i+1}(h_i + h_{i+1})}, \tag{2}$$

$$\frac{\partial^2 p}{\partial x^2} = \frac{2[h_i p_{i+1} - (h_i + h_{i+1}) p_i + h_{i+1} p_{i-1}]}{h_i h_{i+1}(h_i + h_{i+1})}, \tag{3}$$

where h_i is the distance in x direction between p_{i-1} and p_i. Both derivatives are defined at the same point as p_i. Using Eqs. (2) and (3) and similar expressions for y and z Eq. (1) leads to a system of linear equations:

$$A p_d = f_d \tag{4}$$

with the $N \times N$ coefficient matrix A. p_d and f_d are the discrete representations of p and F. For three-dimensional problems, N may become very large ($\sim 10^4 - 10^5$).

Numerical mathematics offers a variety of algorithms to solve Eq. (4). Elimination techniques are not suitable for the range of N expected for three-dimensional problems. Very efficient direct solvers utilize fast Fourier transform (FFT). This restricts the number of gridpoints to powers of 2 (see Wilhelmson and Ericksen [23]). Another class of methods are iterative techniques, see for instance the fundamental textbooks of Varga [22], Young [24], Ames [1], or Golub and van Loan [11]. The conjugate gradient technique was first seen as a generalization of elimination techniques since it has the property of finite termination after N steps (Hesteness and Stiefel [13]). So, for instance, Gaussian elimination is a special conjugate gradient technique prescribing the unit vectors in R^N as direction vectors for a new step. In recent years, a revival of the conjugate gradient technique as an iterative scheme has taken place since by appropriate preconditioning of the matrix the residual may be forced to small values in the very first steps. Nevertheless, disregarding rounding errors the algorithm is still finite. For conjugate gradient techniques see for instance Daniel [8], Reid [18], Concus, Golub, and O'Leary [7], Kershaw [15], Axelsson [2], Young and Jea [25], Khosla and Rubin [16]. For preconditioning see, for example, Dupont, Kendall and Rachford [9], Meijerink and van der Vorst [17], Gustafsson [12], Glowinski, Periaux, and Pironneau [10], van der Vorst [21], Behie and Vinsome [6], Behie, Collin, and Forsyth, Jr. [3]. Some comparisons have been performed with multigrid techniques (Behie and Forsyth [4, 5]) and with direct solvers (Taylor, Hirsh, and Nadworny [20]) which show very promising results.

In the following sections the algorithms are presented and their different behaviour when applied to a model problem is shown. In the Appendix a method is described which permits acceleration of the code on vector computers like the CDC CYBER 205.

2. Algorithms

One step linear iterative methods for the solution of the general problem

$$Bx = f \tag{5}$$

can be written as

$$x^{(n+1)} = Gx^{(n)} + k, \qquad n = 0, 1, 2, \ldots, \tag{6}$$

with arbitrary $x^{(0)}$. An equivalent formulation uses a nonsingular splitting matrix Q,

$$\begin{aligned} x^{(n+1)} &= x^{(n)} - Q^{-1}(Bx^{(n)} - f) \\ &= x^{(n)} - Q^{-1}r^{(n)} \end{aligned} \tag{7}$$

with the residual

$$r^{(n)} = Bx^{(n)} - f, \tag{8}$$

Q is connected with G and k by

$$\begin{aligned} G &= I - Q^{-1}B, \\ k &= Q^{-1}f. \end{aligned} \tag{9}$$

Different iterative schemes result from different choices of Q. Equation (6) shows that for $G=0$ the solution is obtained in one step. By Eq. (9) this is equivalent to $Q=B$. So, efficient iterative schemes are obtained by choosing Q similar to B but easily invertible.

2.1. *Gauss–Seidel* (GS)

Here, B is decomposed into lower triangular matrix B_L, diagonal matrix D and upper triangular matrix B_U. Both triangular matrices have zero diagonal elements.

$$B = B_L + D + B_U. \tag{10}$$

Taking

$$Q = D + B_L \tag{11}$$

results in the iterative formula

$$x^{(n+1)} = -(D + B_L)^{-1} B_U x^{(n)} + (D + B_L)^{-1} f. \tag{12}$$

2.2. *Successive-Over-Relaxation* (SOR)

GS is generalized to SOR by introducing a relaxation parameter ω which weights the diagonal. From the decomposition of B,

$$B = \frac{D}{\omega} + B_L - \frac{1-\omega}{\omega} D + B_U, \tag{13}$$

Q is chosen to be

$$Q = \frac{D}{\omega} + B_L \tag{14}$$

leading to the SOR iteration formula

$$x^{(n+1)} = \left(\frac{D}{\omega} + B_L\right)^{-1} \left(\frac{1-\omega}{\omega} D - B_U\right) x^{(n)} + \left(\frac{D}{\omega} + B_L\right)^{-1} f. \tag{15}$$

The parameter ω can be chosen to speed up convergence. Often, ω must be adjusted empirically.

2.3. *Red-Black*-SOR (RBSOR)

Both GS and SOR are not vectorizable directly. A vectorizable version of SOR is found by partitioning the whole field into red and black points similar to a checkerboard. Red points are surrounded (in normal direction) purely by black points and vice versa. Applying Eq. (15) first on all red points and afterwards on all black points, a fully vectorizable algorithm is obtained referred to as RBSOR.

2.4. *Idealized Generalized Conjugate Gradient* (IGCG)

Details of the theory for this method may be taken from Young and Jea [25] or from Kapitza and Eppel [14]. Here only the basic algorithm and its specific formulation will be given.

For a given symmetric and positive definite matrix Y the ORTHOMIN(s) version of IGCG can be written as

$$\begin{aligned} x^{(n+1)} &= x^{(n)} + \lambda_n p^{(n)}, \\ \lambda_n &= \frac{(Y\delta^{(n)}, Q^{-1}Bp^{(n)})}{(YQ^{-1}Bp^{(n)}, Q^{-1}Bp^{(n)})}, \\ \delta^{(n)} &= Q^{-1}r^{(n)} \\ &= Q^{-1}(f - Bx^{(n)}), \\ p^{(0)} &= \delta^{(0)}, \\ p^{(n} &= \delta^{(n)} + \sum_{i=1}^{s} \alpha_{n,n-i} p^{(n-i)}, \\ \alpha_{n,i} &= -\frac{(YQ^{-1}B\delta^{(n)}, Q^{-1}Bp^{(i)})}{(YQ^{-1}Bp^{(i)}, Q^{-1}Bp^{(i)})}, \end{aligned} \tag{16}$$

where (a, b) is the dot product of vectors a and b, and s is the number of old direction vectors p used to determine the new one $(1 \leqslant s \leqslant n)$. As splitting matrix Q an incomplete Crout factorization is chosen

$$Q = L_1 U_D = B + E, \tag{17}$$

where L_1 and U_D are a lower triangular matrix with unit diagonal and an upper triangular matrix with diagonal D, respectively. Incomplete factorization retains the sparse structure of B by explicitly setting all elements of L_1 and U_D to zero which are also zero in B. So multiplying L_1 with U_D does not only give B but some additional error matrix E.

The matrix Y is chosen in a way to simplify the inner products in Eqs. (16),

$$Y = U_D^T D^{-1} U_D \tag{18}$$

where the superscript T denotes transposition. So we can now formulate the special form of the IGCG-method,

$$\begin{aligned}
x^{(n+1)} &= x^{(n)} + \lambda_n p^{(n)}, \\
\lambda_n &= \frac{(U_D \delta^{(n)}, D^{-1} L_1^{-1} B p^{(n)})}{(D^{-1} L_1^{-1} B p^{(n)}, L_1^{-1} B p^{(n)})}, \\
U_D \delta^{(n)} &= U_D \delta^{(n-1)} - \lambda_{n-1} L_1^{-1} B p^{(n-1)}, \\
p^{(0)} &= \delta^{(0)}, \\
p^{(n)} &= \delta^{(n)} + \sum_{i=1}^{s} \alpha_{n,n-i} p^{(n-i)}, \\
\alpha_{n,i} &= -\frac{(D^{-1} L_1^{-1} B \delta^{(n)}, L_1^{-1} B p^{(i)})}{(D^{-1} L_1^{-1} B p^{(i)}, L_1^{-1} B p^{(i)})}.
\end{aligned} \tag{19}$$

3. Model Problem

To compare the behaviour of the different methods for a fixed **C** (see below) a Gaussian distribution with tunable amplitude, peak location, and variances,

$$p = A \exp[-a(x-x_0)^2 - b(y-y_0)^2 - c(z-z_0)^2] \tag{20}$$

is chosen as analytical solution and inserted into Eq. (1) to determine the function F. The domain is taken to be $[0, 10] \times [0, 10] \times [0, 1]$ to reflect the relative geometrical scale in further applications. The grid was chosen equidistant in x and y direction, but with a variable gridspacing in z direction to allow for a higher resolution in the boundary layer. Three different vertical grids are used with 10, 20, and 26 points, respectively. They are depicted in Fig. 1 together with the corresponding x scale.

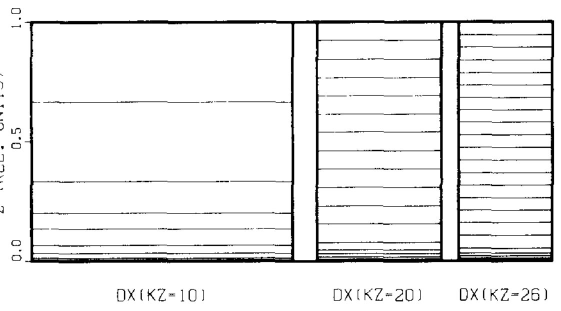


FIG. 1. Three vertical grid resolutions (10, 20, and 26 levels) together with the horizontal resolution DX (length of abscissa).

Boundary conditions are prescribed either as Dirichlet type or Neumann type. Two cases are considered here:

(a) Neumann at $z=0$, Dirichlet elsewhere: This case simulates an application in atmospheric modeling, where the departure of pressure from its hydrostatic value must be computed assuming this departure being small (or in fact zero) far outside the central part of the domain.

(b) Dirichlet at $z=1$, Neumann elsewhere: This case is used to test the behaviour of the algorithm when confronted with many Neumann conditions. The code is constructed such that at least one boundary must be of Dirichlet type.

The parameters in Eq. (20) were chosen such that the peak was not too sharp to be properly resolved by the grid:

$$\begin{aligned} &A=1, \\ &a=b=0.01, \qquad c=0.1, \\ &x_0=y_0=5, \qquad z_0=0.5. \end{aligned} \tag{21}$$

With a view to atmospheric application where the solution domain is chosen such that the lowest z coordinate plane coincides with the ground, the vector $\mathbf{C}$ in Eq. (1) may be estimated to have values of order 0.01 so that asymmetries entering by these gradient terms are considerably smaller than asymmetries caused by unequally spaced z-gridlines. So $\mathbf{C}$ was kept constant,

$$\mathbf{C}=(0.01, 0.01, 0.01). \tag{22}$$

There are two different choices of initial values taken: The first starts from zero, the second from $0.99 \times x_s$ (solution). The latter one is more adequate for real applications, since there the result of the last time step would be used as initial guess, which should not differ very much from the result. Convergence is assumed when the normalized residual ε falls below some prescribed value, in this case

$$\varepsilon=\frac{\|r^{(n)}\|}{\|r^{(0)}\|}<10^{-6}, \tag{23}$$

where

$$\|x\|=\sqrt{(x, x)} \tag{24}$$

is the Euclidean norm.

From Eqs. (19) follows that s previous direction vectors $p^{(i)}$ are necessary to estimate a new $p^{(n)}$. Some experiments were made with varying s. Table I shows the number of iterations and work units (definition see below) for the $20 \times 20 \times 20$ domain case (a). It is seen that $s=1$ gives the minimum number of iterations and work units. So, for all computations s is set to 1, i.e., only one old direction vector must be stored.

TABLE I

Number of Old Direction Vectors s, Number of Iterations N_{It}, and Work Units WU for a $20 \times 20 \times 20$ Domain with Initial Guess $x^{(0)} = 0$ and Boundary Type (a)

s	N_{It}	WU
1	14	325
2	26	658
3	21	588
4	19	578
5	19	620
6	19	659
7	19	695
8	20	771

4. Results

Figure 2 shows the behaviour of GS, RBSOR, and IGCG when applied to a $20 \times 20 \times 20$ domain for cases (a) and (b). ε is taken as ordinate, while the abscissa is in work units, which are defined as the number of floating point multiplications and divisions per unknown, for clearness divided by 1000. Initial value was $0.99 \times x_s$ (solution).

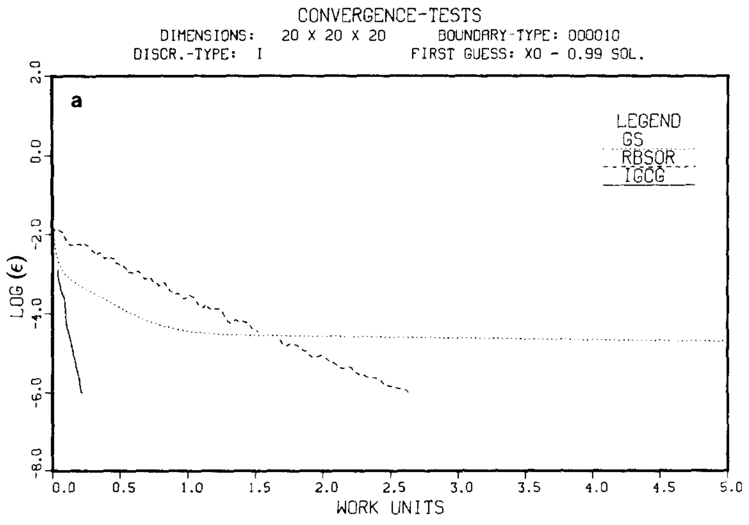
Both Figs. 2a and b show the rapid convergence of GS in the first few iterations followed by a plateau-like behaviour. This is due to the rapid smoothing of high frequencies of the residual, while the low frequencies are treated very poorly (see Stüben and Trottenberg [19]).

RBSOR with optimally chosen parameter shows considerably better results. The convergence criterion is fulfilled within the admitted range of work units. But, as can be seen on the graph, the residual is not guaranteed to get smaller in every iteration step. The curve shows some sort of oscillation.

IGCG, now, beats RBSOR by a factor of about 10. There is no oscillation because it may be shown that the residual must always get smaller (see Young and Jea [25]). The difference between cases (a) and (b) is about 10 % for RBSOR and none for IGCG.

All other problems solved with the three methods show similar results, therefore they are not displayed as a graph but sampled in Table II (excluding GS since convergence never happened). As seen from Table II in nearly all cases IGCG is faster than RBSOR by a factor of at least 10. Looking for a functional relation of the form

$$\text{Work} \sim N^a, \tag{25}$$



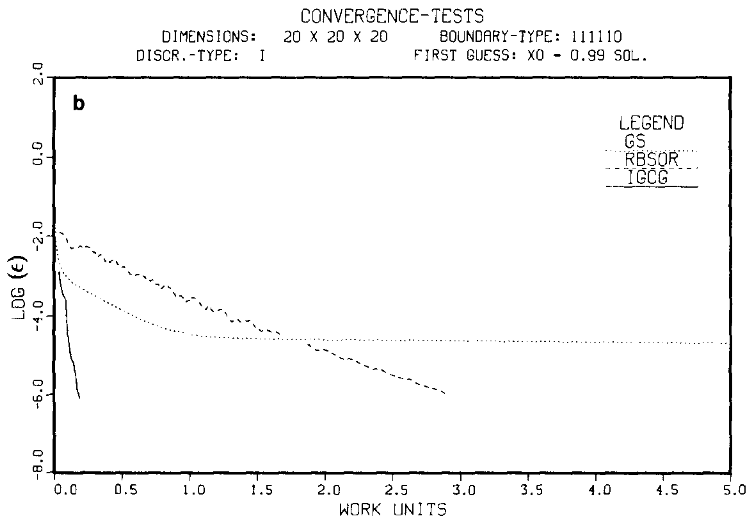


FIG. 2. Normalized residual ε as fuction of work for cases (a) and (b).

TABLE II

Relaxation Parameter ω, Number of Iterations N_{It} and Work Units WU of RBSOR and IGCG for Cases (a) and (b), for Different Domains and Different First Guesses $x^{(0)}$

				RBSOR		IGCG	
Domain	Case	$x^{(0)}$	ω	N_{It}	WU	N_{It}	WU
$10 \times 10 \times 10$	*a*	0.	1.89	120	1807	8	191
$10 \times 10 \times 10$	*b*	0.	1.89	176	2647	8	191
$20 \times 20 \times 20$	*a*	0.	1.94	274	4117	14	325
$20 \times 20 \times 20$	*b*	0.	1.94	308	4627	14	325
$20 \times 20 \times 20$	*a*	$0.99x_s$	1.94	176	2647	9	216
$20 \times 20 \times 20$	*b*	$0.99x_s$	1.94	194	2917	8	194
$26 \times 26 \times 26$	*a*	0.	1.95	346	5197	18	414
$30 \times 30 \times 20$	*a*	0.	1.95	276	4147	20	457

where N is the number of unknowns, we may derive from Table II,

$$\begin{aligned} a_{\text{RBSOR}} &\approx 1.33, \\ a_{\text{IGCG}} &\approx 1.22. \end{aligned} \tag{26}$$

This is to be compared to the analytically derivable numbers for SOR and GS

$$\begin{aligned} a_{\text{GS}} &\approx 2.00, \\ a_{\text{SOR}} &\approx 1.50 \end{aligned} \tag{27}$$

(see, e.g., Ames [1]).

5. Conclusions

It is shown that IGCG is about 10 times faster than RBSOR. Moreover, IGCG does not contain a relaxation parameter like RBSOR. Figure 3 shows that ω has to be known to at least two and preferably three digits to guarantee optimum performance of RBSOR, a knowledge which may be difficult to obtain in real applications. Another advantage of IGCG is its independence on specific dimension configurations for optimal performance contrary to FFT and multigrid.

Multigrid is reported to be optimal in the sense that $a \approx 1$ in Eq. (25), at least in 2-D cases (Stüben and Trottenberg [19]). We did not make any tests with multigrid solvers, but Behie and Forsyth [4, 5] reported that conjugate gradient methods are competitive up to dimensions of about $33 \times 33 \times 33$. Another advantage is perhaps the rather simple coding compared to multigrid codes.

In summary IGCG, apart from its superior speed, has several advantages compared to standard algorithms. It might be less efficient than the fastest Poisson

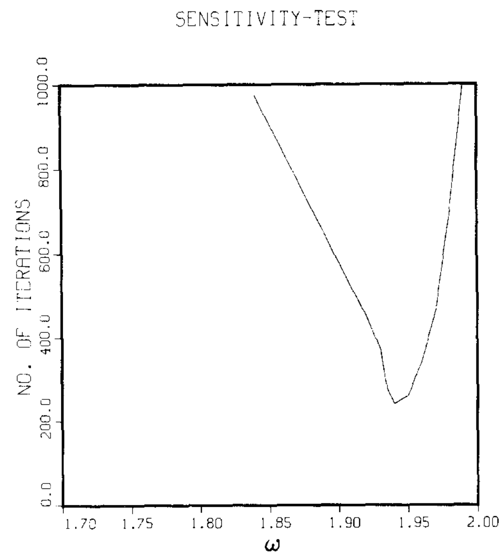


FIG. 3. Number of iterations for RBSOR as function of relaxation parameter ω.

solvers available, but it is easy to apply and offers the user any freedom in defining his domain.

The algorithm is coded for cartesian coordinates and, with regard to atmospheric application, also for terrain-following coordinates, both in 2- or 3-dimensional versions. Remarks on possible improvements of the code for vector computers are described in the appendix for special domain configurations.

APPENDIX: SOME REMARKS ON VECTORIZABILITY ON A CYBER 205

The code has been implemented on the CDC CYBER 205 at the Klimarechenzentrum (= climate computing center) in Hamburg. Each iterative sweep (Eqs. (19)) consists of

(a) multiplication of constants with vectors and adding the result to vectors,

(b) multiplication of matrices with vectors,

(c) scalar products of vectors,

(d) inverting of triangular matrices.

The only operation not auto-vectorizable by the compiler is (d). But a simple idea leads to considerable improvement of the performance of the whole code:

The triangular matrices L and U are the result of the incomplete factorization of B. The points in the vector x are ordered such that in the matrix B the largest elements are concentrated near the main diagonal. So, it is tempting to perform the incomplete factorization for the tridiagonal part of B only. This leads to triangular matrices L and U which have only one side band. Since this "truncated" form of L and U is a worse factorization than the original one, it is expected that the iteration count will increase. But this is more than compensated by the ability of the CYBER 205 to handle recursive operations with a lag of 1 by "STACKLIB-Routines." This procedure improved the performance by a factor of 2 for our test problem. However, it must be emphasized that this trick works only successfully if one direction of the 3-D domain has a much smaller space increment than the other ones (e.g., the z direction for atmospheric models should have this property) and if at the same time the points of the domain are ordered running fastest into this direction. This idea should be applicable to other vector machines. The documented FORTRAN program code is available from the authors.

ACKNOWLEDGMENT

We want to express our gratitude to Professor Dr. J. Häuser who influenced our work with many fruitful discussions.

REFERENCES

1. W. F. AMES, *Numerical Methods for Partial Differential Equations* (Academic Press, New York/San Francisco, 1977).
2. O. AXELSSON, *Linear Algebra Its Appl.* **29**, 1 (1980).
3. A. BEHIE, D. COLLINS, AND P. A. FORSYTH, JR., *Comput. Methods Appl. Mech. Eng.* **42**, 287 (1984).
4. A. BEHIE AND P. A. FORSYTH, JR., *Appl. Math. Comput.* **13**, 229 (1983).
5. A. BEHIE AND P. A. FORSYTH, JR., *IMA J. Num. Anal.* **3**, 41 (1983).
6. A. BEHIE AND P. K. W. VINSOME, *Soc. Petr. Eng. J.* **22**, 658 (1982).
7. P. CONCUS, G. H. GOLUB, AND D. P. O'LEARY, *Computing* **19**, 321 (1978).
8. J. W. DANIEL, *SIAM J. Numer. Anal.* **4**, 10 (1967).
9. T. DUPONT, R. P. KENDALL, AND H. H. RACHFORD, JR., *SIAM J. Numer. Anal.* **5**, 559 (1968).
10. R. GLOWINSKI, J. PERIAUX, AND O. PIRONNEAU, *Appl. Math. Modelling* **4**, 187 (1980).
11. G. H. GOLUB, C. F. VAN LOAN, *Matrix Computations* (North Oxford Academic, Oxford, 1983).
12. I. GUSTAFSSON, *BIT* **18**, 142 (1978).
13. M. R. HESTENES AND E. STIEFEL, *J. Res. Nat. Bur. Stand.* **49**, 409 (1952).
14. H. KAPITZA AND D. EPPEL, Forschungszentrum Geesthacht External Report No. 85/E/23, D-2054 Geesthacht, Federal Republic Germany, 1985.
15. D. S. KERSHAW, *J. Comput. Phys.* **26**, 43 (1978).
16. P. K. KHOSLA AND S. G. RUBIN, *Comput. Fluids* **9**, 109 (1981).
17. J. A. MEIJERINK AND H. A. VAN DER VORST, *Math. Comput.* **31**, 148 (1977).
18. J. K. REID, *SIAM J. Numer. Anal.* **9**, 325 (1972).
19. K. STÜBEN AND U. TROTTENBERG, in *Lecture Notes in Mathematics*, vol. 960, edited by A. Dold and B. Eckmann, (Springer, Berlin/Heidelberg/New York, 1982), p. 1.
20. T. D. TAYLOR, R. S. HIRSH, AND M. M. NADWORNY, *Comp. Fluids* **1**, 1 (1984).
21. H. A. VAN DER VORST, *J. Comput. Phys.* **44**, 1 (1981).
22. R. S. VARGA, *Matrix Iterative Analysis* (Prentice-Hall, Englewood Cliffs, N. J., 1962).
23. R. B. WILHELMSON AND J. H. ERICKSEN, *J. Comput. Phys.* **25**, 319 (1977).
24. D. M. YOUNG, *Iterative Solution of Large Linear Systems* (Academic Press, New York/San Francisco, 1971).
25. D. M. YOUNG AND K. C. JEA, *Linear Algebra Its Appl.* **34**, 159 (1980).