MULTIGRID AND CONJUGATE GRADIENT METHODS AS CONVERGENCE ACCELERATION TECHNIQUES

P. Sonneveld and P. Wesseling

(Delft University of Technology)

and

P.M. de Zeeuw

(Centre for Mathematics and Computer Science, Amsterdam)

ABSTRACT

Multigrid and conjugate gradient type techniques for the acceleration of iterative methods are discussed. A detailed discussion is given of incomplete factorizations. The theoretical background of the classical conjugate gradient method and preconditioning is briefly reviewed. A conjugate gradient type method for non-symmetric-positive-definite systems is presented. Multigrid methods are discussed, and two portable, autonomous computer codes are introduced. Multigrid treatment of convection-diffusion entails special difficulties, and ways to overcome these are outlined. Numerical experiments on a set of test problems are reported. Efficiency and robustness of several conjugate gradient and multigrid methods are compared and discussed.

1. INTRODUCTION

Finite difference and element discretizations of partial differential equations give rise to large sparse systems of equations, which in this paper will be assumed to have been linearized. In practice, the number of unknowns can be quite large, and solution methods must exploit the sparsity and the structure of the system. This can be done with direct methods, using sparse matrix techniques, or by iterative methods, which will be considered here.

Classical iterative methods, a review of which is given by Young (1971), are relatively simple to implement, but converge slowly for large problems. In recent years conjugate gradient (CG) and multigrid (MG) methods have been drawing increased attention as powerful and rather general techniques to accelerate the convergence of iterative methods. Our aim is to discuss recent progress that has been made with these techniques. A new CG method for systems that are not symmetric positive definite will be presented.

Both CG and MG have a wider use and significance than just being acceleration techniques. But the present viewpoint makes it possible to grasp the main principles in a simple manner, and furthermore, it brings out the main similarities and differences of the two methods.

The authors beg forgiveness for focussing on developments with which they are especially familiar.

2. ITERATIVE METHODS AND INCOMPLETE FACTORIZATIONS

The problem to be solved is a linear algebraic system denoted as

$$Ay = b. \tag{2.1}$$

Stationary iterative methods for the solution of (2.1) can usually be written as

$$y^{n+1} = y^n + B(b-Ay^n). \tag{2.2}$$

For example, for the Gauss-Seidel method (GS) we have $B = (D+L)^{-1}$, and for the successive overrelaxation method (SOR) we have $B = \omega(D+\omega L)^{-1}$, with L, D and U defined by

$$A = L + D + U, \tag{2.3}$$

where L and U are the lower and upper triangular parts of A, and $D = \text{diag}(A)$.

For the error $e^n = y^n - y$ we find from (2.2)

$$e^n = (I-BA)^n e^0. \tag{2.4}$$

For special cases the spectral radius $\rho(I-BA)$ is known. For example, if A is the familiar 5-point finite difference discretization of Laplace's equation on the unit square with Dirichlet boundary conditions and mesh-size h in both directions, we have

$$\text{GS: } \rho(I-BA) = \cos^2\pi h = 1 - \pi^2h^2 + O(h^4), \tag{2.5}$$

$$\text{SOR: } \rho(I-BA) = \frac{1-\sin\pi h}{1+\sin\pi h} = 1 - 2\pi h + O(h^2). \tag{2.6}$$

Let us define the computational cost W of an algorithm to be the number of operations from the set $\{+,-,*,/\}$. In practice computer time will depend on W, but also on the programming language, the skill of the programmer, the frequency of the occurrence of indirect addressing, type of machine etc., but still, W defined above is a convenient yardstick for measuring computational complexity.

Let N ($=h^{-2}$ in two dimensions) be the number of unknowns. Then the cost of one GS or SOR iteration is $W = O(N)$. The required number of iterations n for a desired residual or error reduction ε follows from

$$(\rho(I-BA))^n \leqslant \varepsilon. \tag{2.7}$$

From (2.5) and (2.6) we find, if ε is fixed (independent of h) that $n = O(N)$, $O(N^{\frac{1}{2}})$, for GS and SOR respectively, resulting in the following estimates for the total computational cost:

$$\text{GS: } W = O(N^2), \tag{2.8}$$

$$\text{SOR: } W = O(N^{3/2}). \tag{2.9}$$

These results are found to hold in practice not only for the Poisson equation, but for elliptic equations in general, as long as A is an M-matrix. In a general situation, the estimate for SOR may be optimistic, because the optimal ω is not known.

Obviously, the best one may hope to achieve is $W = O(N)$. There exist MG methods with this property, as has been shown rigorously for general elliptic equations. For CG no theoretical results with the same degree of generality as for MG are available. For the Poisson equation Gustaffson (1978) has proved for a certain preconditioned CG method that $W = O(N^{5/4})$. Practical experience indicates that this holds for a wider class of equations.

Before further discussing MG and CG we first introduce a number of iterative methods that have special significance in the context of MG and CG.

Red-black Gauss-Seidel (RBGS) relaxation is Gauss-Seidel relaxation with a certain ordering of the grid-points. These are divided in red and black points in a checkerboard fashion. First the points of one colour are relaxed simultaneously, then the points of the other colour.

Horizontal zebra (HZ) relaxation is Gauss-Seidel relaxation by horizontal lines, taking first the odd and then the even lines, assuming that the boundary lines are odd.

Alternating zebra (AZ) relaxation is Gauss-Seidel relaxation by lines, taking successively the odd, even horizontal, odd, even vertical lines.

Hackbusch (1980) and Foerster et al. (1981) have shown that RBGS and AZ in combination with MG result in efficient iterative methods.

Incomplete LU (ILU) (or incomplete Crout, or incomplete Cholesky) decompositions have been introduced as preconditionings for CG by Meijerink and van der Vorst (1977), and as smoothing processes for MG by Wesseling and Sonneveld (1980). ILU has been found useful in transonic flow computations, cf. van der Wees et al. (1983), Nowak and Wesseling (1983). More recently, incomplete line LU (ILLU) or incomplete block factorizations have been proposed by Underwood (1976), Concus, Golub and Meurant (1982), Axelsson (1983) and Meijerink, see Kettler (1982), Meijerink (1983).

For completeness we will give a description of ILU and ILLU decompositions. Let P be a set of 2-tuples representing a matrix sparsity pattern. Then a class of ILU decompositions of the matrix A can be defined as follows. L and U are lower and upper triangular matrices satisfying

$$\ell_{ij} = 0,\ (i,j) \notin P;\ u_{ij} = 0,\ (i,j) \notin P;$$
$$(LU)_{ij} = a_{ij},\ (i,j) \in P. \qquad (2.10)$$

Note that L and U now have a different meaning than in equation (2.3). The ILU decomposition may be made unique by requiring, for example,

$$\ell_{ii} = 1. \qquad (2.11)$$

In many cases ILU-decompositions can be computed simply by means of (incomplete) Crout formulae. For example, assume that the given problem (2.1) is a discretization of a partial differential equation on an m*n grid, and let the grid-points be enumerated as in Fig. 2.1.

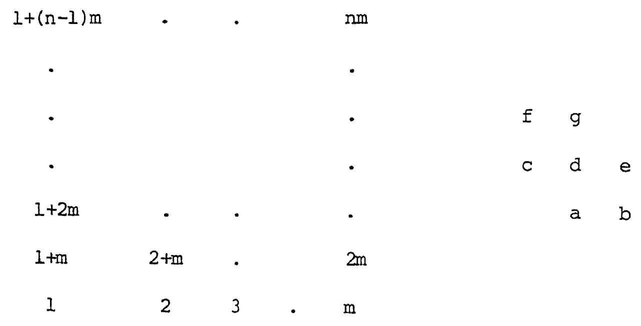


Fig. 2.1 Enumeration of computational grid-points, and difference molecule

Let A be a 7-point discretization of a second order elliptic partial differential equation with the difference molecule abcdefg of Fig. 2.1 (the atoms b and f are needed if a mixed derivative is present). Then the sparsity pattern of A is:

$$\{(i,i-m),(i,i-m+1),(i,i-1),(i,i),(i,i+1),(i,i+m-1),(i,i+m)\}.$$

For brevity the following notation is introduced:

$$a_i = a_{i,i-m};\ b_i = a_{i,i-m+1};\ c_i = a_{i,i-1};\ d_i = a_{ii};$$
$$e_i = a_{i,i+1};\ f_i = a_{i,i+m-1};\ g_i = a_{i,i+m}. \qquad (2.12)$$

Let the sparsity pattern P of L and U be chosen identical to that of A, and let the elements of L and U be called α_i, β_i, γ_i, δ_i, ε_i, ζ_i, η_i. The locations of these elements are identical to those of $a_i, b_i, \ldots, g_i$, respectively. The diagonal of L is specified to be unity; δ_i are the elements of diag(U). Then L and U can be conveniently computed by means of the following Crout formulae:

$$\begin{aligned}
&\alpha_i = a_i/\delta_{i-m}, \qquad \beta_i = (b_i - \alpha_i\varepsilon_{i-m})/\delta_{i-m+1},\\
&\gamma_i = (c_i - \alpha_i\zeta_{i-m})/\delta_{i-1}, \quad \delta_i = d_i - \gamma_i\varepsilon_{i-1} - \beta_i\zeta_{i-m+1} - \alpha_i\eta_{i-m},\\
&\varepsilon_i = e_i - \beta_i\eta_{i-m+1}, \quad \zeta_i = f_i - \gamma_i\eta_{i-1}, \quad \eta_i = g_i.
\end{aligned} \tag{2.13}$$

Quantities that are not defined because their subscript is outside the range [1,nm] are to be replaced by zero. This is but one example of an ILU-decomposition of A. Other possibilities are described, for example, by Meijerink and van der Vorst (1981).

Sometimes it pays to add certain neglected entries (compared to the full LU-decomposition) to the diagonal element or to other non-neglected entries in the same row. Then we no longer have

$$(LU)_{ij} = a_{ij}, \quad (i,j) \in P. \tag{2.14}$$

For details see Axelsson (1982). We will not go into this here.

An ILU decomposition can be used in an iterative method by choosing $B = (LU)^{-1}$ in (2.2), obtaining

$$LUy^{n+1} = b + (LU-A)y^n. \tag{2.15}$$

The cost of one iteration can be reduced by means of the following simple device. With L and U computed by means of (2.13) we have

$$LU = A + C. \tag{2.16}$$

The only non-zero elements of C are given by

$$c_{i,i-m+2} = \beta_i\varepsilon_{i-m+1}, \quad c_{i,i+m-2} = \gamma_i\zeta_{i-1}. \tag{2.17}$$

With (2.16), (2.14) becomes

$$LUy^{n+1} = b + Cy^n \tag{2.18}$$

which is cheaper than (2.15), because C is more sparse than A.

It is easily verified that the construction of L and U according to (2.13) takes 17 flops (floating point operations) per grid-point. L and U are stored in place of A, and if C is generated, no extra storage beyond that for A is needed.

The solution of $LUy = q$ is obtained by back-substitution:

$$\begin{aligned} y_i &:= q_i - \gamma_i y_{i-1} - \beta_i y_{i-m+1} - \alpha_i y_{i-m}, \\ y_i &:= (y_i - \varepsilon_i y_{i+1} - \zeta_i y_{i+m-1} - \eta_i y_{i+m})/\delta_i. \end{aligned} \tag{2.19}$$

Hence, the solution for y^{n+1}, the computation of $b+Cy^n$ (generating C) and the execution of one iteration require 13, 6 and 19 flops per grid-point, respectively.

We will not discuss existence of ILU decompositions. Meijerink and van der Vorst (1977) prove existence for M-matrices, but often ILU is applied successfully to more general matrices.

ILLU decomposition can be described as follows. With the computational grid and the finite difference molecule of Fig. 2.1 the matrix A has the following structure:

$$A = \begin{vmatrix} B_1 & U_1 & & & & \\ L_2 & B_2 & U_2 & & & \\ & L_3 & B_3 & U_3 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & & L_n & B_n \end{vmatrix} \tag{2.20}$$

with L_i, B_i and U_i $m \times m$ matrices; B_i are triangular matrices; L_i and U_i are lower and upper triangular, with sparsity patterns $\{(j,j-1),(j,j)\}$ and $\{(j,j),(j,j+1)\}$ respectively. We try to find a matrix D such that

$$A = (L+D)D^{-1}(D+U), \tag{2.21}$$

where

$$L = \begin{vmatrix} 0 & & & & & \\ L_2 & 0 & & & & \\ & L_3 & 0 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & L_n & 0 \end{vmatrix}, \quad U = \begin{vmatrix} 0 & U_1 & & & & \\ & 0 & U_2 & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & 0 & U_{n-1} \\ & & & & & 0 \end{vmatrix},$$

$$D = \begin{vmatrix} D_1 & & & & \\ & D_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & D_n \end{vmatrix}.$$

We call (2.21) a line LU decomposition of A, because the blocks in L, D and U correspond to (in our case horizontal) lines of the computational grid. Given the decomposition (2.21), solving (2.1) is just as simple as with a classical LU decomposition. Equation (2.21) can be rewritten as

$$A = L + D + U + LD^{-1}U. \tag{2.22}$$

One finds that $LD^{-1}U$ is the following block-diagonal matrix:

$$LD^{-1}U = \begin{vmatrix} 0 & & & & \\ & L_2D_1^{-1}U_1 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & L_nD_{n-1}^{-1}U_{n-1} \end{vmatrix}. \tag{2.23}$$

From (2.22) and (2.23) we deduce the following algorithm for the computation of D:

$$D_1 = B_1, \; D_i = B_i - L_iD_{i-1}^{-1}U_{i-1}, \; i = 2,3,\ldots,n. \tag{2.24}$$

The matrix D_i^{-1} is full, which causes the cost of a line LU decomposition to be $O(nm^3)$, as for standard LU-decomposition. An incomplete line LU decomposition is obtained if we replace $L_i D_{i-1}^{-1} U_{i-1}$ by its tridiagonal part. Thus, algorithm (2.24) is replaced by:

$$\tilde{D}_1 = B_1,\ \tilde{D}_i = B_i - \text{tridiag}(L_i \tilde{D}_{i-1}^{-1} U_{i-1}),\quad i = 2,3,\ldots,n. \tag{2.25}$$

The ILLU decomposition of A is now defined to be

$$A = (L+\tilde{D})\tilde{D}^{-1}(\tilde{D}+U) + E, \tag{2.26}$$

with E the error matrix, and $\tilde{D}$ the block diagonal matrix with blocks $\tilde{D}_i$.

We will now show how $\tilde{D}$ and $\tilde{D}^{-1}$ may be computed. Consider tridiag $(L_i \tilde{D}_{i-1}^{-1} U_{i-1})$, or, temporarily dropping the subscript, tridiag $(L\tilde{D}^{-1}U)$. Let the elements of $\tilde{D}^{-1}$ be s_{ij}; we shall see shortly how to compute them. The elements t_{ij} of tridiag$(L\tilde{D}^{-1}U)$ can be computed as follows:

$$\begin{aligned}
&\sigma_{-1} = \ell_{i,i+1}s_{i+1,i-1} + \ell_{ii}s_{i,i-1},\quad \sigma_0 = \ell_{i,i+1}s_{i+1,i} + \ell_{ii}s_{ii},\\
&\sigma_1 = \ell_{i,i+1}s_{i+1,i+1} + \ell_{ii}s_{i,i+1},\quad \sigma_2 = \ell_{i,i+1}s_{i+1,i+2} + \ell_{ii}s_{i,i+2},\\
&t_{i,i-1} = \sigma_{-1}u_{i-1,i-1} + \sigma_0 u_{i,i-1},\quad t_{ii} = \sigma_0 u_{ii} + \sigma_1 u_{i+1,i},\\
&t_{i,i+1} = \sigma_1 u_{i+1,i+1} + \sigma_2 u_{i+2,i+1}.
\end{aligned} \tag{2.27}$$

The inverse of a tridiagonal matrix can be determined as follows. Let

$$T = \begin{vmatrix} a_1 & c_1 & & & & \\ b_2 & a_2 & c_2 & & & \\ & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & b_{m-1} & a_{m-1} & c_{m-1} \\ & & & & b_m & a_m \end{vmatrix}.$$

Let the triangular factorization of T be

$$T = (L+I)D^{-1}(I+U), \tag{2.28}$$

where L, D and U are not to be confused with the matrices occurring in (2.20). The only non-zero elements of L, D and U are $\ell_{i,i-1}$, d_{ii} and $u_{i,i+1}$, respectively. Call these elements ℓ_i, d_i, u_i for brevity. They can be computed by means of the following recursion formulae:

$$d_1^{-1} = a_1,\ u_1 = c_1 d_1, \text{ for } i > 1:$$

$$\ell_i = b_i d_{i-1},\ d_i^{-1} = a_i - \ell_i d_{i-1}^{-1} u_{i-1}, \tag{2.29}$$

$$u_i = c_i d_i.$$

The elements of T^{-1} can be calculated as follows. From (2.28) we have

$$T^{-1} = (I-U)^{-1}\ D(L+I)^{-1}. \tag{2.30}$$

Let λ_{ij} be the elements of $(L+I)^{-1}$. By requiring $(L+I)^{-1}(L+I) = I$ and proceeding row by row, we find the following recursion formulae:

$$\lambda_{ij} = 0,\ j > i;\ \lambda_{ii} = 1;\ \lambda_{ij} = -\ \ell_{j+1}\lambda_{i,j+1},\ j < i. \tag{2.31}$$

Similarly, by requiring $(U+I)(U+I)^{-1} = I$ and proceeding column by column, we find for the elements μ_{ij} of $(I+U)^{-1}$:

$$\mu_{ij} = 0,\ j < 1;\ \mu_{ii} = 1;\ \mu_{ij} = -\ u_i\mu_{i+1,j},\ j > i. \tag{2.32}$$

Using (2.30) we find for the elements s_{ij} of T^{-1}:

$$s_{mm} = d_{mm}, \tag{2.33}$$

$$\begin{aligned} s_{kk} &= \sum_{i=k}^{m} \mu_{ki} d_{ii} \lambda_{ik} = d_{kk} + \sum_{i=k+1}^{m} u_k \mu_{k+1,i} d_{ii} \ell_{k+1} \lambda_{i,k+1} \\ &= d_{kk} + u_k \ell_{k+1} s_{k+1,k+1}, \end{aligned} \tag{2.34}$$

$$\begin{aligned} s_{k,k-j} &= \sum_{i=k}^{m} \mu_{ki} d_{ii} \lambda_{i,k-j} = -\sum_{i=k}^{m} \mu_{ki} d_{ii} \ell_{k-j+1} \lambda_{i,k-j+1} \\ &= -\ \ell_{k-j+1} s_{k,k-j+1}, \end{aligned} \tag{2.35}$$

$$s_{k-j,k} = \sum_{i=k}^{m} \mu_{k-j,i} d_{ii} \lambda_{ik} = - \sum_{i=k}^{m} u_{k-j} \mu_{k-j+1,i} d_{ii} \lambda_{ik}$$
$$= - u_{k-j} s_{k-j+1,k}. \qquad (2.36)$$

This completes our description of the computation of tridiag $(L_i \tilde{D}_{i-1} U_{i-1})$.

The complete algorithm for the computation of the ILLU decomposition (2.26) can be summarized as follows. We compute $\tilde{D}$ and its triangular decomposition.

$D_1 := B_1$;

for $i = 2,3,\ldots,n$ do (i) - (iv):

(i) Compute the triangular decomposition of $\tilde{D}_{i-1}$ according to (2.29);

(ii) Compute the five main diagonals of $\tilde{D}_{i-1}^{-1}$ according to (2.33) - (2.36);

(iii) Compute tridiag $(L_i \tilde{D}_{i-1}^{-1} U_{i-1})$ according to (2.27);

(iv) Compute $\tilde{D}_i$ with (2.25);

Finally, compute the triangular decomposition of $\tilde{D}_n$ according to (2.29).

The number of flops required is given by:

Step (i): 5m; step (ii): 7m; step (iii): 21m; step (iv): 3m.

Hence, the total cost of computing $\tilde{D}$ and its triangular decomposition is 36mn. Storage to the extent of 3mn reals is needed for the triangular decomposition of $\tilde{D}$.

When using ILLU, the iterative method (2.2) becomes:

$$r := b - Ay^n, \qquad (2.37)$$

$$(L+\tilde{D})\tilde{D}^{-1}(\tilde{D}+U)y^{n+1} = r, \qquad (2.38)$$

$$y^{n+1} := y^{n+1} + y^n. \qquad (2.39)$$

Equation (2.38) is solved as follows:

$$(L+\tilde{D})y^{n+1} = r, \qquad (2.40)$$

$$r := \tilde{D}y^{n+1}, \qquad (2.41)$$

$$(\tilde{D}+L)y^{n+1} = r. \qquad (2.42)$$

With the block partitioning used before, and with y_i^{n+1} and r_i denoting m-dimensional vectors corresponding to the i-th block, equation (2.40) is solved as follows:

$$\tilde{D}_1 y_1^{n+1} = r_1, \; \tilde{D}_i y_i^{n+1} = r_i - L_{i-1} y_i^{n+1}, \quad i > 1. \tag{2.43}$$

Equation (2.42) is solved in similar fashion. The solution of an $m \times m$ tridiagonal system, with triangular decomposition available, takes 5m flops. The cost of the right side of (2.43) is 4m (for the 7-point difference molecule assumed here). The total cost of (2.40), and of (2.42) as well, is therefore 9mn, so that the cost of (2.38) is 23mn. The cost of (2.37) is 14mn. The total cost of one ILLU iteration is therefore 37mn.

For other ILLU variants, see Concus et al. (1982) and Meijerink (1983), who prove existence of ILLU decompositions for M-matrices. For remarks on vectorization, see Meijerink (1983), Meurant (1983), and Hemker, Wesseling and de Zeeuw (1983).

For future reference we note that the cost of RBGS is 12mn flops, assuming a 7-point difference molecule. In HZ, tridiagonal system solving takes 5mn flops, assuming that the necessary triangular decompositions have been computed beforehand, at a cost of 5mn flops, respectively. Residue evaluation takes 8mn flops, so that the total cost of one iteration with HZ is 13mn flops. For AZ, these figures should be doubled.

For RBGS, AZ, ILU and ILLU rate of convergence estimates are not available in the literature, but the number of iterations required certainly increases as the grid is refined. Therefore the computational cost of these methods is $O(N^{\alpha})$ with $\alpha > 1$. In the following sections we will discuss how the convergence of iterative methods such as those just discussed can be accelerated with CG or MG methods.

3. CONJUGATE GRADIENT METHODS

For an introduction to CG (and Chebyshev) acceleration of iterative methods, see Hageman and Young (1981). Within the confines of this paper we can only give a brief discussion.

When A in (2.1) is large and sparse it is attractive, because of efficiency and simplicity, to use A only as a multiplier. This means that we can build polynomials in A. At the start of the iterations the only special vectors available are b and the residue $r^0 = b - Ay^0$, with y^0 the starting iterand. A rather general form of possible algorithms would be

$$y^{n+1} = y^n + \alpha_n p^n, \tag{3.1}$$

$$p^n = \theta_n(A) r^0 + \tilde{\theta}_n(A) b. \tag{3.2}$$

Here θ_n and $\tilde{\theta}_n$ are polynomials, whose degree is increased by one at each iteration. At present, only the case where one chooses $\tilde{\theta}_n \equiv 0$ seems to have been investigated. It may not be worthwhile to allow $\tilde{\theta}_n \not\equiv 0$. For example, it seems reasonable to require that the sequence $\{y^{n+1}-y^n\} = \{\alpha_n p^n\}$ is identical for the following two cases:

case 1 : $Ay = b$, starting iterand y^0

case 2 : $Ay = \bar{b}$, starting iterand $\bar{y}^0$

with $\bar{b} = b + A(\bar{y}^0 - y^0)$. With overbars referring to case 2, we have

$$\bar{\alpha}_n \bar{p}^n = \bar{\alpha}_n \bar{\theta}_n(A)\bar{r}^0 + \bar{\alpha}_n \bar{\tilde{\theta}}_n(A)\bar{b}. \tag{3.3}$$

Since $\bar{r}^0 = r^0$ we can have $\bar{\alpha}_n \bar{p}^n = \alpha_n p^n$ for all b only if $\tilde{\theta}_n \equiv 0$, $\bar{\tilde{\theta}}_n \equiv 0$. Assuming henceforth $\tilde{\theta}_n \equiv 0$, we have

$$\begin{aligned} r^{n+1} &= b - Ay^{n+1} = b - Ay^n - \alpha_n Ap^n \\ &= r^n - A\alpha_n \theta_n(A) r^0 = \text{(induction)} \\ &= r^0 - A\{\alpha_n\theta_n(A) + \alpha_{n-1}\theta_{n-1}(A) + \ldots + \alpha_0\theta_0(A)\}r^0 \\ &= \phi_{n+1}(A)r^0, \end{aligned} \tag{3.4}$$

where ϕ_{n+1} is a polynomial of degree n+1 with the following property:

$$\phi_n(0) = 1. \tag{3.5}$$

Because of (3.4) we would like to choose ϕ_n such that $||\phi_n(A)r^0||$ is minimized, under the constraint (3.5). For SPD (symmetric positive definite) A this aim is achieved by CG methods. Let us define

$$\Pi_n^1 = \{\psi_n | \psi_n(0) = 1, \ \ \psi_n \text{ is polynomial of degree} \leq n\}. \tag{3.6}$$

Then we want to construct $\phi_n \in \Pi_n^1$ such that

$$||\phi_n(A)r^0|| \leq ||\psi_n(A)r^0||, \qquad \forall\, \psi_n \in \Pi_n^1. \tag{3.7}$$

If we choose the following norm:

$$||r||^2 \equiv r^T A^{-1} r, \tag{3.8}$$

then the following CG method solves (3.7):

$$\begin{aligned}
&p^{-1} = 0,\ r^0 = b - Ay^0,\\
&p^n = r^n + \beta_n p^{n-1},\ \beta_n = \rho_n/\rho_{n-1},\ \rho_n = r^{n^T} r^n,\\
&y^{n+1} = y^n + \alpha_n p^n,\ \alpha_n = \rho_n/\sigma_n,\ \sigma_n = p^{n^T} Ap^n,\\
&r^{n+1} = r^n - \alpha_n Ap^n.
\end{aligned} \tag{3.9}$$

For a proof see for example Hageman and Young (1981). The name of the method derives from the fact that the search vectors are conjugate:

$$p^{k^T} Ap^n = 0, \quad k = 0,1,2,\dots,n-1. \tag{3.10}$$

By making different choices for the norm $||\cdot||$ in (3.7), different CG methods are obtained.

For many practical applications the restriction of CG to SPD systems is a severe drawback. Several ways to generalize CG have been proposed, but at the moment it is not yet clear what are the best CG variants for non symmetric or indefinite systems. We present a promising new method.

First, we rewrite the CG method (3.9) in terms of the polynomials ϕ_n and θ_n introduced before. One easily obtains:

$$\theta_{-1} \equiv 0, \quad \phi_0 \equiv 1, \tag{3.11a}$$

$$\theta_n = \phi_n + \beta_n \theta_{n-1}, \tag{3.11b}$$

$$\phi_{n+1} = \phi_n - \alpha_n \psi \theta_n, \tag{3.11c}$$

with ψ the polynomial $\psi(\tau) = \tau$,

$$\begin{aligned}
&\beta_n = \rho_n/\rho_{n-1},\quad \alpha_n = \rho_n/\sigma_n,\quad \rho_n = (\phi_n,\phi_n),\\
&\sigma_n = (\phi_n,\psi\phi_n),
\end{aligned} \tag{3.12}$$

where the bilinear form (.,.) is defined by

$$(\phi,\theta) = r^{o^T}\phi(A^T)\theta(A)r^o. \tag{3.13}$$

We will now abandon the assumption that A is SPD, so that the algorithm no longer minimizes the residual in the sense of (3.7); $\|\cdot\|$ no longer has the properties of a norm. We replace (3.13) by

$$(\phi,\theta) = \tilde{r}^{o^T}\phi(A)\theta(A)r^o, \tag{3.14}$$

with $\tilde{r}_o$ a vector to be chosen. In general, this is not an inner product. Then for arbitrary A, (.,.) has the following properties:

$$(\phi,\theta) = (\theta,\phi), \tag{3.15}$$

$$(\phi,\zeta\theta) = (\zeta\phi,\theta), \tag{3.16}$$

for every triple of polynomials ϕ, θ, ζ. The following theorem suggests that the algorithm (3.11) might still be of use for solving Ay = b:

Theorem 3.1 The algorithm defined by (3.11), (3.12) and (3.14) has the following property:

$$(\phi_{n+1},\phi_k) = 0,\ k < n+1;\ (\theta_n,\psi\theta_k) = 0,\ \ k < n. \tag{3.17}$$

Hence, if A happens to be such that (.,.) is an inner product, then the residual lies in a subspace the dimension of which is reduced by one at each iteration, just as for the classical CG method.

Proof of theorem 3.1 Obviously

$$\theta_o = \phi_o. \tag{3.18}$$

With (3.11c), (3.12), (3.16), (3.18):

$$(\phi_1,\phi_o) = (\phi_o,\phi_o) - \alpha_o(\psi\theta_o,\phi_o) = 0. \tag{3.19}$$

Using (3.11b,c):

$$\begin{aligned}(\theta_1,\psi\theta_o) &= (\phi_1,\psi\theta_o) + \beta_1(\theta_o,\psi\theta_o) = \\ &(\phi_1,\phi_o-\phi_1)/\alpha_o + \beta_1\sigma_o = -\rho_1/\alpha_o + \beta_1\sigma_o = 0.\end{aligned} \tag{3.20}$$

Similarly,

$$\begin{aligned}(\phi_2,\phi_0) &= (\phi_1,\phi_0) - \alpha_1(\psi\theta_1,\phi_0) = -\alpha_1(\theta_1,\psi\theta_0) = 0,\\ (\phi_2,\phi_1) &= (\phi_1,\phi_1) = \alpha_1(\psi\theta_1,\phi_1)\\ &= \rho_1 - \alpha_1(\psi\theta_1,\theta_1-\beta_1\theta_0) = \rho_1 - \alpha_1\sigma_1 = 0.\end{aligned} \tag{3.21}$$

This establishes the validity of (3.17) for n = 1. Proceeding by induction, for $k < n$,

$$(\phi_{n+1},\theta_k) = (\phi_n,\theta_k) - \alpha_n(\psi\theta_n,\theta_k) = (\phi_n,\theta_k). \tag{3.22}$$

From (3.11b) it follows that there exist constants c_{kj} such that

$$\theta_k = \sum_{j=1}^{k} c_{kj}\,\phi_j. \tag{3.23}$$

Hence, with (3.22) and the induction hypothesis,

$$(\phi_{n+1},\theta_k) = (\phi_n, \sum_{j=1}^{k} c_{kj}\phi_j) = 0, \quad k < n. \tag{3.24}$$

Furthermore

$$\begin{aligned}(\phi_{n+1},\theta_n) &= (\phi_n,\theta_n) - \alpha_n(\psi\theta_n,\theta_n)\\ &= (\phi_n,\phi_n) + \beta_n(\phi_n,\theta_{n-1}) - \alpha_n\sigma_n\\ &= \rho_n + \beta_n(\phi_n, \sum_{j=1}^{n-1} c_{n-1,j}\phi_j) - \rho_n = 0.\end{aligned} \tag{3.25}$$

It follows that

$$(\phi_{n+1},\phi_k) = (\phi_{n+1},\theta_k) - \beta_k(\phi_{n+1},\theta_{k-1}) = 0, \quad k \leqslant n, \tag{3.26}$$

establishing the first part of the induction hypothesis.

For $k \leqslant n$ we have

$$\begin{aligned}(\theta_{n+1},\psi\theta_k) &= (\phi_{n+1},\psi\theta_k) + \beta_{n+1}(\theta_n,\psi\theta_k)\\ &= (\phi_{n+1},\phi_k-\phi_{k+1})/\alpha_k + \beta_{n+1}\sigma_n\delta_{nk}\\ &= (-\rho_{n+1}/\alpha_n + \beta_{n+1}\sigma_n)\delta_{nk} = 0.\end{aligned}$$

with δ_{nk} the Kronecker delta. This completes the proof.

The algorithm (3.11) - (3.13) can be put in a form suitable for computation as follows. Define

$$\begin{aligned}&r^n = \phi^n(A)r^0,\ p^n = \theta_n(A)r^0,\\ &\tilde{r}^n = \phi_n(A^T)\tilde{r}^0,\ \tilde{p}^n = \theta_n(A^T)\tilde{r}^0,\end{aligned} \tag{3.27}$$

with r^0 the starting residue and $\tilde{r}^0$ some vector to be specified by the user of the algorithm. Then we have according to (3.12) and (3.13)

$$\rho_n = \tilde{r}^{n^T} r^n,\quad \sigma_n = \tilde{p}^{n^T} Ap^n, \tag{3.28}$$

and we obtain the following algorithm:

$$\begin{aligned}&r^0 = b - Ay^0,\quad \text{choose } \tilde{r}^0,\ p^{-1} = \tilde{p}^{-1} = 0,\\ &p^n = r^n + \beta_n p^{n-1},\\ &\tilde{p}^n = \tilde{r}^n + \beta_n \tilde{p}^{n-1},\\ &r^{n+1} = r^n - \alpha_n Ap^n,\\ &\tilde{r}^{n+1} = \tilde{r}^n - \alpha_n A^T\tilde{p}^n,\\ &y^{n+1} = y^n + \alpha_n p^n,\\ &\alpha_n = \rho_n/\sigma_n,\ \beta_0 = 0,\ \beta_n = \rho_n/\rho_{n-1},\ \rho_n = \tilde{r}^{n^T} r^n,\\ &\sigma_n = \tilde{p}^{n^T} Ap^n.\end{aligned} \tag{3.29}$$

The vectors r^n, $\tilde{r}^n$, p^n, $\tilde{p}^n$ satisfy (3.27). According to theorem 3.1 we have

$$\tilde{r}^{n^T} r^k = 0, \quad k < n, \tag{3.30a}$$

$$\tilde{p}^{n^T} A p^k = 0, \quad k < n. \tag{3.30b}$$

According to (3.30b) the sets $\{\tilde{p}^k\}$ and $\{p^k\}$ are conjugate with respect to A, which is why the algorithm is called the bi-CG method. It has first been proposed by Fletcher (1976).

The bi-CG method can be accelerated appreciably (roughly by a factor 2), by the following stratagem. The idea is to construct an algorithm for which the residue is $\phi_n(A)^2 r^0$ instead of $\phi_n(A) r^0$, which turns out to be possible at hardly any extra cost, and eliminates the need to work with A^T. If bi-CG converges, $\phi_n(A)$ will be a contraction, and $\phi_n(A)^2$ will be smaller than $\phi_n(A)$. A suitable algorithm is obtained by squaring (3.11). We call the resulting method the CGS (conjugate gradients squared) method. From (3.11) we obtain

$$\begin{aligned} \theta_n^2 &= \phi_n^2 + \beta_n^2 \theta_{n-1}^2 + 2\beta_n \phi_n \theta_{n-1}, \\ \phi_{n+1}^2 &= \phi_n^2 + \alpha_n^2 \psi^2 \theta_n^2 - 2\alpha_n \phi_n \psi \phi_n. \end{aligned} \tag{3.31}$$

Using (3.11b),

$$\begin{aligned} \theta_n \phi_n &= \phi_n^2 + \beta_n \phi_n \theta_{n-1}, \\ \theta_n^2 &= \theta_n \phi_n + \beta_n (\phi_n \theta_{n-1} + \beta_n \theta_{n-1}^2), \\ \phi_{n+1} \theta_n &= \phi_n \theta_n - \alpha_n \psi \theta_n^2, \\ \phi_{n+1}^2 &= \phi_n^2 - \alpha_n \psi (\phi_n \theta_n + \phi_{n+1} \theta_n), \end{aligned} \tag{3.32}$$

with α_n, β_n given by (3.12), where ρ_n and σ_n can now be evaluated as follows:

$$\rho_n (\phi_n, \phi_n) = (1, \phi_n^2),$$

$$\sigma_n = (\phi_n, \psi\phi_n) = (1, \psi\phi_n^2).$$

This is transformed into a workable algorithm with the aid of the following vectors:

$$f^n = \phi_n(A)^2 r^0, \quad g^n = \theta_n(A)^2 r^0, \qquad h^n = \phi_n(A)\theta_{n-1}(A) r^0. \tag{3.33}$$

Equations (3.32) are equivalent to (u corresponds to $\theta_n\phi_n$):

<u>CGS method</u>:

$$\begin{aligned} & f^0 = b - Ay^0, \quad g^{-1} = h^0 = 0, \\ & u = f^n + \beta_n h^n, \\ & g^n = u + \beta_n(\beta_n g^{n-1} + h^n), \\ & h^{n+1} = u - \alpha_n A g^n, \\ & y^{n+1} = y^n + \alpha_n(u + h^{n+1}), \\ & f^{n+1} = f^n - \alpha_n A(u + h^{n+1}), \end{aligned} \tag{3.34}$$

where we have used that $y^{n-1}-y^n$ follows directly from the difference in the residues $f^{n+1}-f^n$. In (3.34) we have

$$\alpha_n = \rho_n/\sigma_n, \quad \beta_0 = 0, \quad \beta_n = \rho_n/\rho_{n-1}, \qquad \rho_n = \tilde{r}^{0^T} f^n, \quad \sigma_n = \tilde{r}^{0^T} A g^n. \tag{3.35}$$

We usually choose

$$\tilde{r}^0 = b - Ay^0. \tag{3.36}$$

The cost of CGS is about the same as the cost of bi-CG. The correspondence between bi-CG and CGS is that the residues after n iterations are $\phi_n(A)r^0$ and $\phi_n(A)^2 r^0$, respectively.

Another type of method that seems promising for the indefinite case is Chebyshev iteration, for example the version proposed by Manteuffel

(1977, 1978). This method works well if certain parameters related to the spectrum of A can be estimated accurately. An important advantage of CGS is that no parameters need be estimated. A thorough comparison between CGS and Chebyshev iteration has not yet been made.

Bi-CG and CGS are but two examples of extensions of CG to non-SPD systems. We will not review other extensions that have been proposed, but restrict ourselves to mentioning the publications of Concus and Golub (1976), Vinsome (1976), Widlund (1978) and Axelsson (1980).

4. CONJUGATE GRADIENT ACCELERATION OF ITERATIVE METHODS: PRECONDITIONING

Until further notice A is assumed to be SPD. For a stationary iterative method (2.2) it follows from (2.4), that

$$e^n = \psi_n(BA)e^0, \quad \psi_n(x) = (1-x)^n \in \Pi_1^n. \tag{4.1}$$

Assuming that B is SPD we can write

$$B = E^T E. \tag{4.2}$$

For arbitrary powers of $E^T EA$ we have

$$(E^T EA)^k = E^T (EAE^T)^k E^{-T}, \tag{4.3}$$

so that (4.1) can be rewritten as

$$E^{-T} e^n = \psi_n(EAE^T)E^{-T} e^0. \tag{4.4}$$

If we apply CG not to (2.1) but to the following preconditioned version:

$$(EAE^T)(E^{-T} y) = Eb \tag{4.5}$$

then in (4.4) ψ_n is replaced by ϕ_n satisfying the optimality condition (3.7), so that we may say that CG accelerates (2.2). Of course it is equally true that CG is accelerated by preconditioning.

We will now study the rate of convergence that can be obtained with preconditioned CG. In the SPD case the rate of convergence of CG methods can be estimated in an elegant way cf. Axelsson (1977). From (3.4), (3.7) and (3.8) it follows that

$$||r^n||^2 = \min_{\psi \in \Pi_1^n} r^{0^T} \psi(A)^2 A^{-1} r^0, \tag{4.6}$$

$\Rightarrow$ choosing $\|z\|^2 = z^T A^{-1} z$.

Let the set of eigenvalues of A be

$$Sp(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_N\}, \tag{4.7}$$

with corresponding eigenvectors $x_1, x_2, \ldots, x_N$ satisfying $x_i^T x_j = \delta_{ij}$. Let

$$r^o = \sum_{i=1}^{N} \xi_i x_i, \tag{4.8}$$

then

$$\begin{aligned} ||r^n||^2 &= \min_{\psi \in \Pi_1^n} \sum_{i=1}^{N} \xi_i^2 \psi(\lambda_i)^2/\lambda_i \\ &\leqslant \min_{\psi \in \Pi_1^n} \max_{\lambda \in Sp(A)} \psi(\lambda)^2 \sum_{i=1}^{N} \xi_i^2/\lambda_i \\ &= ||r^o|| \min_{\psi \in \Pi_I^n} \max_{\lambda \in Sp(A)} \psi(\lambda)^2. \end{aligned} \tag{4.9}$$

Rate of convergence estimates are obtained by making a choice for $\psi(\lambda)$. For example,

$$\begin{aligned} \psi(\lambda) &= T_n(z)/T_n\left(\frac{\bar{\lambda}+\underline{\lambda}}{\bar{\lambda}-\underline{\lambda}}\right), \\ z &= (\bar{\lambda}+\underline{\lambda}-2\lambda)/(\bar{\lambda}-\underline{\lambda}), \end{aligned} \tag{4.10}$$

with T_n the Chebyshev polynomial of degree n, and $\bar{\lambda}$, $\underline{\lambda}$ the largest and smallest eigenvalue of A. Because

$$\max_{|z| \leqslant 1} T_n(z)^2 = 1,$$

we obtain

$$||r^n||^2/||r^o||^2 \leqslant 1/T_n\left(\frac{\bar{\lambda}+\underline{\lambda}}{\bar{\lambda}-\underline{\lambda}}\right)^2. \tag{4.11}$$

A well-known property of Chebyshev polynomials is

$$1/T_n\left(\frac{\bar\lambda+\underline\lambda}{\bar\lambda-\underline\lambda}\right)^2 \leqslant 4\left\{\frac{1-(\underline\lambda/\bar\lambda)^{\frac12}}{1+(\underline\lambda/\bar\lambda)^{\frac12}}\right\}^{2n}. \tag{4.12}$$

For $|z| < 1$ the following holds:

$$\left(\frac{1-z}{1+z}\right)^n = \exp\left\{-2n(z+\frac{z^3}{3}+\frac{z^5}{5}+\ldots)\right\} \leqslant e^{-2nz}. \tag{4.13}$$

Using (4.13) in (4.12) and noting that $\bar\lambda/\underline\lambda = \mathrm{cond}_2(A)$ we obtain

$$||r^n||/||r^0|| \leqslant 2e^{-2n\ \mathrm{cond}_2(A)^{-\frac12}}.$$

Requiring a residue reduction ε the required number of iterations n is

$$n \geqslant \frac12\left|\ln\frac{\varepsilon}{2}\right|\mathrm{cond}_2\,(A)^{\frac12}. \tag{4.14}$$

For discretizations of second order elliptic equations we usually have

$$\mathrm{cond}_2(A) = O(1/h^2), \tag{4.15}$$

so that

$$W = O(N^{3/2}), \tag{4.16}$$

as for SOR. In practice CG tends to be somewhat more expensive than SOR, but it is parameter free, and if for SOR the optimal overrelaxation factor is not accurately known, CG is faster.

The efficiency of CG by itself is not very impressive, but the interest of CG derives from the possibility of convergence acceleration by preconditioning. Rewriting the CG algorithm (3.9) for (4.5) one obtains

$$\begin{aligned}
&p^{-1} = 0,\ r^0 = Eb - EAy^0,\\
&p^n = r^n + \beta_n p^{n-1},\ \beta_n = \rho_n/\rho_{n-1},\ \rho_n = r^{n^T} r^n,\\
&E^{-T}y^{n+1} = E^{-T}y^n = \alpha_n p^n,\ \alpha_n = \rho_n/\sigma_n,\ \sigma_n = p^{n^T} EAE^T p_n,\\
&r^{n+1} = r^n - \alpha_n EAE^T p^n.
\end{aligned} \tag{4.17}$$

Replacing $E^T p$ by p and redefining $r = b - Ay$ this can be rewritten as follows:

Preconditioned CG algorithm:

$$
\begin{aligned}
&p^{-1} = 0,\ r^0 = b - Ay^0\\
&p^n = E^T E r^n + \beta_n p^{n-1},\ \beta_n = \rho_n/\rho_{n-1},\ \rho_n = r^{n^T} E^T E r^n,\\
&y^{n+1} = y^n + \alpha_n p^n,\ \alpha_n = \rho_n/\sigma_n,\ \sigma_n = p^{n^T} A p^n, \qquad (4.18)\\
&r^{n+1} = r^n - \alpha_n A p^n.
\end{aligned}
$$

It has been found by Meijerink and van der Vorst (1977), that an effective preconditioning is obtained with incomplete Cholesky decomposition, given by

$$LL^T = A + C, \qquad (4.19)$$

the symmetric (Cholesky) variant of ILU decomposition discussed in section 2. We choose $E = L^{-1}$ in (4.18). The eigenvalue distributions of $L^{-1}AL^{-T}$ and A are compared for a few examples by Meijerink and van der Vorst (1977) and Kershaw (1978); it is found that $\mathrm{cond}_2(L^{-1}AL^{-T}) << \mathrm{cond}_2(A)$. For a full explanation of the acceleration effect of preconditioning not only the condition number but the eigenvalue distribution should be taken into account, but there is no general theory available concerning the influence of preconditioning on the eigenvalue distribution or even the condition number. For a special case, the 5-point discretization of the Poisson equation, Gustafsson (1978) shows that preconditioning with a certain type of ("modified") incomplete LL^T decomposition results in

$$\mathrm{cond}_2(L^{-1}AL^{-T}) = O(1/h), \qquad (4.20)$$

so that according to (4.14) the required number of iterations is $O(h^{-\frac{1}{2}})$, resulting in a computational cost of $O(N^{5/4})$. This result seems to hold approximately quite generally for CG with preconditioning by approximate decomposition. One finds that the number of iterations required increases slowly as the grid is refined. The modified incomplete LL^T decomposition seems in general to provide a somewhat better preconditioning than the version described here.

In the preconditioned CG algorithm (4.18) the matrix is needed only for multiplication with r^n. If one does not want to form E or E^{-1}

explicitly, but wants to define E implicitly by means of the iterative method (2.2), one can obtain Br^n from

$$Br^n = y^* - y^n, \tag{4.21}$$

with y^* the result of one iteration (2.2), starting with y^n.

A preconditioned version of CGS can be obtained as follows. Application of CGS to the following preconditioned version of (2.1)

$$BAy = Bb \tag{4.22}$$

results in an algorithm given by (3.34), (3.35) with A and b replaced by BA and Bb. By replacing $B^{-1}f$ by f we obtain:

<u>Preconditioned CGS algorithms</u>:

$$\begin{aligned}
&f^0 = b - Ay^0,\ g^{-1} = h^0 = 0,\\
&u = Bf^n + \beta_n h^n,\\
&g^n = u + \beta_n(\beta_n g^{n-1} + h^n),\\
&h^{n+1} = u - \alpha_n BAg^n,\\
&y^{n+1} = y^n + \alpha_n(u + h^{n+1}),\\
&f^{n+1} = f^n - A\alpha_n(u + h^{n+1}),
\end{aligned} \tag{4.23}$$

with

$$\alpha_n = \rho_n/\sigma_n,\ \beta_0 = 0,\ \beta_n = \rho_n/\rho_{n-1},$$

$$\sigma_n = \tilde{r}^{0^T} Bf^n,\ \sigma_n = \tilde{r}^{0^T} BAg^n.$$

If B is not explicitly available, as for instance when the iterative method to be accelerated is a MG method, then Bf^n and BAg^n can be obtained as follows. Carry out an iteration with the method to be accelerated (2.2), with starting iterand y^n:

$$y^* = y^n + B(b - Ay^n) = y^n + Bf^n. \tag{4.24}$$

Next, carry out an iteration with starting iterand g^n and right-hand-side $b = 0$:

$$g^* = g^n - BAg^n. \tag{4.25}$$

It follows that

$$Bf^n = y^* - y^n, \; BAg^n = g^n - g^*. \tag{4.26}$$

In this way one may try to use CG or CGS to accelerate the convergence of any iterative method, with the restriction that for CG the matrix B must be symmetric. Kettler (1982) has used CG to accelerate MG. To make B symmetric, he used incomplete LL^T decomposition for smoothing, and the V-cycle multigrid schedule (see the next section). One might say that CG accelerates MG which accelerates incomplete LL^T. Behie and Forsyth (1983) have used Orthomin, a non-symmetric CG variant (Vinsome 1976) to accelerate MG using the sawtooth cycle.

For CGS applied to a general system there is no guarantee that convergence will be rapid, but a rule of thumb is that a good rate of convergence may be expected with ILU and ILLU preconditioning if A satisfies

$$a_{ii} \geqslant - \sum_{j=1} a_{ij}, \; a_{ij} \leqslant 0, \; j = i. \tag{4.27}$$

(This makes A an M-matrix).

In order to obtain a rough idea of the computational cost of CGS we count flops (per grid-point) in (4.23). Preconditioning takes place with ILU or ILLU. Assume that multiplication with B or BA takes place using (4.24)-(4.26). Using in this case the explicitly available matrix B one can obtain slightly lower operation counts than those obtained below, but we will neglect this possibility here. Note that in (4.24) the residue $b - Ay^n = f^n$ is already available, so that $(b+Cy^n)$ or (2.37) need not be carried out for ILU or ILLU, respectively. Using the ILU and ILLU operation counts of section 2, we find that the cost of y^* is 13 flops (ILU) or 23 flops (ILLU). Similarly, the cost of g^* is found to be 18 flops (ILU) or 36 flops (ILLU). We assume that A has 7 non-zero elements per row. Hence, multiplication with A takes 18 flops. In addition to matrix multiplications, CGS needs 18 flops, as is easily seen from (4.23), including of course the cost of α_n, β_n. Therefore the total cost of one preconditioned CGS iteration is 60 flops (ILU) or 88 flops (ILLU).

5. MULTIGRID ACCELERATION OF ITERATIVE METHODS

The basic ideas of MG methods are quite general and have a wide range of application. They can be used not only to accelerate iterative methods, but also, for example, to formulate novel ways to solve non-linear problems, or to devise algorithms that construct adaptive discretizations. The volume edited by Hackbusch and Trottenberg (1982) represents a useful survey.

We restrict ourselves here to the one aspect of MG mentioned in the title of this chapter. This makes it possible to simplify MG, and to distinguish situations where its effectiveness is guaranteed. The significance of MG as accelerating technique derives from the fact that, in principle, a computational complexity of O(N) can be achieved, with N the number of unknowns. This has been proved rigorously by Fedorenko (1964) for a finite difference approximation of the Poisson equation and by Bakhvalov (1966), Hackbusch (1980), Wesseling (1980) for finite difference approximations to general second order elliptic partial differential equations. For a survey of MG rate of convergence theory, including finite element discretizations, see Hackbusch (1982). These general theories result in O(N) but pessimistic, and fortunately unrealistic, computational complexity estimates. The papers by Brandt (1977) and Hackbusch (1978) showed the great potential of MG for practical applications. The theoretical work just mentioned assumes a W-cycle. More recently, work on rate of convergence theory for the V-cycle has appeared, such as Musy (1982), Maitre and Musy (1983), McCormick (1983), Braess and Hackbusch (1983). The terms V- and W-cycle are explained for example by Stüben and Trottenberg (1982). These terms refer to the MG schedule, i.e. the switching strategy between the grids. For special equations, notably Poisson's equation, work on realistic rate of convergence predictions is underway, see for example Braess (1981, 1982), Stüben and Trottenberg (1982). We will not discuss these theoretical aspects here.

Equation (2.1) is assumed to represent a discretization of a partial differential equation. If the basic iterative method (2.2) converges, it usually (but not always) has the property, exploited by MG, that the non-smooth part of error and residue is annihilated rapidly, whereas it takes many iterations to get rid of the smooth part. A precise definition of smoothness will be given shortly. The fundamental MG idea is to approximate the problem with smooth error and residue on coarser grids. In the MG context (2.2) is called a smoothing process.

One way of discriminating between smooth and non-smooth parts of grid functions is by means of Fourier analysis, as proposed by Brandt (1977). Let the computational grid G associated with the discretization (2.1) be defined by (we restrict ourselves for simplicity to two-dimensional problems):

$$G := \{(x_1,x_2) \mid x_i = 0, h_i, 2h_i, \ldots, m_ih_i\}. \tag{5.1}$$

Any grid-function $e : G \to \mathbb{R}$ can be represented by a Fourier series as follows:

$$e_{mn} = \sum_{s=-m_1/2}^{m_1/2} \sum_{t=-m_2/2}^{m_2/2} c_{st}\exp(im\theta_s + in\phi_t). \tag{5.2}$$

with

$$\theta_s = (2s-1)\pi/(m_1+1),\ \phi_t = (2t-1)\pi/(m_2+1), \tag{5.3}$$

where we have assumed that m_i is even. By e_{mn} we mean the value of the grid-function e in the grid-point with coordinates (mh_1,nh_2). Let $\bar{G}$ be a coarse grid with step-size doubled, i.e.

$$\bar{G} := \{(x_1,x_2) \mid x_i = 0,2h_i,4h_i,\ldots,\tfrac{1}{2}m_i 2h_i\}. \tag{5.4}$$

Then we call those Fourier components that cannot be represented without aliasing on $\bar{G}$ non-smooth. That is, the set of non-smooth Fourier components is given by

$$\begin{aligned}&\{\exp(im\theta+in\phi) \mid (\theta,\phi) \in F\}\ ,\\ &F := \{(\theta,\phi) \mid -\pi \leq \theta,\phi \leq \pi,\ |\theta| \geq \pi/2 \text{ and/or } |\phi| \geq \pi/2\},\end{aligned} \tag{5.5}$$

where for convenience we do not restrict (θ,ϕ) to the discrete set occurring in (5.3).

For periodic boundary conditions and constant coefficients in the differential equation, many iterative processes (but not for example RBGS and AZ) have the property that, if the error e before iteration is given by (5.2), then the error $\bar{e}$ after iteration is given by

$$\bar{e}_{mn} = \sum_{s=-m_1/2}^{m_1/2} \sum_{t=-m_2/2}^{m_2/2} \bar{c}_{st}\exp(im\theta_s+in\phi_t)\ , \tag{5.6}$$

with

$$\bar{c}_{st} = \rho(\theta_s,\phi_t)c_{st}. \tag{5.7}$$

The annihilation of the non-smooth part of the error can be measured by the quantity defined below (Brandt (1977)):

Definition 5.1 The Fourier smoothing factor is

$$\rho_F := \sup_{(\theta,\phi)\in F} |\rho(\theta,\phi)|.$$

Note that ρ_F does not depend on the mesh-size h_i, in contrast to the rate of convergence of (2.2). A large catalogue of Fourier smoothing factors for various equations and smoothing processes has been compiled by Kettler (1982). It turns out that simple point-wise smoothing processes, such as damped Jacobi or Gauss-Seidel relaxation, have a good smoothing factor (i.e. ρ_F well below 1) for Poisson's equation, but not for equations with strong coupling in a certain direction, such as the convection-diffusion equation at high Péclet number, or anisotropic diffusion problems. In such cases more robust smoothing processes are called for, such as block Gauss-Seidel relaxation, AZ, ILU or ILLU (AZ does not work for convection-diffusion problems with upwind differences).

As noted before, Fourier smoothing analysis as just described assumes periodic boundary conditions and constant coefficients in the differential equation. The Fourier smoothing factor may be expected to be a good indicator of the quality of a smoothing process in more general circumstances, provided the coefficients vary smoothly, and provided the influence of perturbations of the boundary conditions attenuates as one moves into the interior of the region. However, MG is applied successfully to problems with discontinuous coefficients and problems where perturbations of the boundary conditions are felt in the interior, such as convection-diffusion and anisotropic diffusion problems. Apart from these limitations, Fourier smoothing analysis has the disadvantage, that the performance of the coarse grid corrections in the no-man's land between the smooth and non-smooth parts of the error is not taken into account. A different type of smoothing analysis that does not suffer these disadvantages is as follows.

Let the sets of grid-functions $G \to \mathbb{R}$ and $\bar{G} \to \mathbb{R}$ be defined by Y and $\bar{Y}$ respectively. Let the coarse grid approximation of (2.1) be given by

$$\bar{A}\bar{y} = \bar{b}. \tag{5.8}$$

Furthermore, let there be given a prolongation operator P and a restriction operator R:

$$P : \bar{Y} \to Y, \; R : Y \to \bar{Y}. \tag{5.9}$$

A two-grid method for the acceleration of the iterative method (2.2) can be formulated as follows. Let y^j be the current iterand, and let $y^{j+\frac{1}{2}}$ be the result of applying a coarse grid correction to y^j:

$$y^{j+\frac{1}{2}} = y^j - P\bar{A}^{-1}R(Ay^j-b), \tag{5.10}$$

where we assume for the time being that the coarse grid problem is solved exactly. For the residue $r^j := Ay^j-b$ we find:

$$r^{j+\frac{1}{2}} = (I-AP\bar{A}^{-1}R)r^j. \tag{5.11}$$

We now make the following choice for $\bar{A}$, called

Galerkin approximation:

$$\bar{A} = RAP. \tag{5.12}$$

Then it follows from (5.11) that

$$r^{j+\frac{1}{2}} \in \mathrm{Ker}(R), \tag{5.13}$$

as noted by Hemker (1982) and McCormick (1982). In other words, $r^{j+\frac{1}{2}} \perp \mathrm{Ker}^{\perp}(R)$, which justifies the appellation "Galerkin approximation" for (5.12). Following Hemker (1982A) we will relate the concept of smoothness to the kernel and range of R and P.

Definition 5.2 The set of R-smooth grid-functions is $\mathrm{Ker}^{\perp}(R)$.
Whether the grid-functions just defined are also what one would call physically smooth or smooth in the sense of the Fourier analysis presented above, depends on the choice made for R.

It remains to annihilate the non-smooth part of r, and this is done in the second part of the two-grid iteration, called smoothing. This is done with an iterative method of type (2.2):

$$y^{j+1} = y^{j+\frac{1}{2}} + B(b-Ay^{j+\frac{1}{2}}), \tag{5.14}$$

and we find:

$$r^{j+1} = (I-AB)r^{j+\frac{1}{2}}. \tag{5.15}$$

The projection operator on Ker(R) is given by $I-R^T(RR^T)^{-1}R$, and we may conclude from (5.13) and (5.15) that

$$||r^{j+1}|| \leq ||(I-AB)(I-R^T(RR^T)^{-1}R)|| \; ||r^{j+\frac{1}{2}}||. \tag{5.16}$$

This leads us to the following definition:

Definition 5.3 The R-smoothing factor of the smoothing process (2.2) is

$$\rho_R := ||(I-AB)(I-R^T(RR^T)^{-1}R)||.$$

Whether ρ_R will be approximately equal to ρ_F depends on whether $\mathrm{Ker}^{\perp}(R)$ approximately equals the space spanned by the Fourier components (5.5), and on the applicability of Fourier smoothing analysis. An advantage of Fourier smoothing analysis is that ρ_F is usually easier to compute than ρ_R.

We now take the dual viewpoint of considering the error instead of the residue, and define:

Definition 5.4 The set of P-smooth grid-functions is Range(P).

Let the error e^j be defined by $e^j := y^j - y$. Then it follows from (5.10) that

$$e^{j+\frac{1}{2}} = (I - P\bar{A}^{-1}RA)e^j. \tag{5.17}$$

A streamlined reasoning is obtained if we now assume that smoothing precedes coarse grid correction, so that (5.14) is replaced by

$$y^j = y^{j-\frac{1}{2}} + B(b - Ay^{j-\frac{1}{2}}). \tag{5.18}$$

Let $e^j = e_1^j + e_2^j$ with $e_1^j \in \text{Range}(P)$, $e_2^j \in \text{Range}^{\perp}(P)$. Again choosing $\bar{A}$ according to (5.12) we see that

$$(I - P\bar{A}^{-1}RA)e_1^j = 0, \tag{5.19}$$

(write $e_1^j = P\hat{e}$ for some $\hat{e}$), so that

$$e^{j+\frac{1}{2}} = (I - P\bar{A}^{-1}RA)e_2^j. \tag{5.20}$$

In general $e^{j+\frac{1}{2}}$ will be small only if e_2^j is small, which motivates the following definition:

Definition 5.5 The P-smoothing factor of the smoothing process (2.2) is

$$\rho_P := ||(I-P)(P^TP)^{-1}P^T)(I-BA)||$$

(Note that the projection operator on $\text{Range}^{\perp}(P)$ is given by $I - P(P^TP)^{-1}P^T$.) In the special case that $R = P^T$ we have that the sets of R-smooth and of P-smooth grid-functions are identical, since $\text{Ker}^{\perp}(R) = \text{Range}\ (R^T)$, but ρ_R and ρ_P will in general not be identical. The quantity ρ_P is defined and studied by McCormick (1982)

The two-grid method defined by (5.10) and (5.14) or (5.18) can be regarded as an acceleration technique for the iterative method (2.2). The striking efficiency of MG methods is due to the fact that there exist simple iteration methods of type (2.2) for which ρ_F, ρ_R and ρ_P

are well below 1 for a large class of problems, independent of the mesh-size.

Prolongation and restriction operators can be chosen in various ways. Examples of prolongations are (denoting grid-functions in $\bar{Y}$ by an overbar): 9-point prolongation:

$$\begin{aligned}
&(P\bar{y})_{2p,2q} = \bar{y}_{pq}, \quad (P\bar{y})_{2p+1,2q} = \tfrac{1}{2}(\bar{y}_{pq}+\bar{y}_{p+1,q}) \\
&(P\bar{y})_{2p,2q+1} = \tfrac{1}{2}(\bar{y}_{pq}+\bar{y}_{p,q+1}), \\
&(P\bar{y})_{2p+1,2q+1} = \tfrac{1}{2}\{(P\bar{y})_{2p+1,2q} + (P\bar{y})_{2p,2q+1}\}.
\end{aligned} \tag{5.21}$$

7-point prolongation: as 9-point prolongation, except

$$(P\bar{y})_{2p+1,2q+1} = \tfrac{1}{2}(\bar{y}_{p+1,q}+\bar{y}_{p,q+1}). \tag{5.22}$$

Examples of restrictions are:

Injection:

$$(Ry)_{pq} = y_{2p,2q}, \tag{5.23}$$

9-point restriction or full weighting:

$$\begin{aligned}
(Ry)_{pq} = y_{2p,2q} &+ \tfrac{1}{2}(y_{2p+1,2q}+y_{2p,2q+1}+y_{2p-1,2q}+y_{2p,2q-1}) + \\
&+ \frac{1}{4}(y_{2p+1,2q+1}+y_{2p-1,2q+1}+y_{2p+1,2q-1}+y_{2p-1,2q-1}),
\end{aligned} \tag{5.24}$$

7-point restriction:

$$\begin{aligned}
(Ry)_{pq} = y_{2p,2q}&+\tfrac{1}{2}(y_{2p+1,2q}+y_{2p,2q+1}+y_{2p-1,2q} \\
&+y_{2p,2q-1}+y_{2p+1,2q-1}+y_{2p-1,2q+1}).
\end{aligned} \tag{5.25}$$

We call (5.24), (5.25) 9-point or 7-point restriction because a weighted average is taken of 9 or 7 grid-function values, and we call (5.21), (5.22) 9-point or 7-point prolongation, because they are closely related to (5.24) and (5.25) respectively: with P, R according to (5.21), (5.24) or according to (5.22), (5.25) we have

$$R = P^T, \tag{5.26}$$

i.e. as matrices, P and R are adjoint.

In Fig. 5.1 we define a 5-point, 7-point and 9-point difference molecule. With a 5-point or a 7-point molecule, 7-point prolongation and restriction can be used. With a 9-point molecule, 9-point prolongation and restriction is more accurate. With a 7-point molecule one can construct finite difference approximations to any second order partial differential equation in two dimensions, including mixed derivatives. A 7-point molecule is also obtained with finite elements, using Courant triangulation (cf. Fig. 5.1). If one desires exactly symmetric numerical solutions to symmetric problems a 9-point molecule should be used, an example being symmetric flow around a symmetric airfoil.

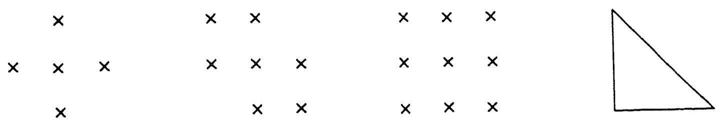


Fig. 5.1 Difference molecules: 5-point, 7-point, 9-point. Courant triangle.

In a loose sense, P is accurate if, given that $\bar{y}$ is a good discrete approximation to the exact solution of the differential equation, $P\bar{y}$ is also a good approximation. For prolongations based on linear interpolation, such as (5.21) and (5.22), this is certainly the case when the exact solution is smooth. An important case when the exact solution is not smooth occurs when the coefficients of the differential equation are discontinuous. In that case matrix-dependent prolongation should be used. In order to define this type of prolongation we use the grid-point enumeration of Fig. 2.1. A grid-point with coordinates (ph_1, qh_2) has the number $1+p+qm$. Indicating elements of the matrix A by a_{ij} in the usual way corresponding with this enumeration, we define:

Matrix-dependent prolongation:

$$(P\bar{y})_{2p,2q} = \bar{y}_{pq}, \tag{5.27a}$$

$$(P\bar{y})_{2p+1,2q} = (A_{i,i-1}\bar{y}_{pq} + A_{i,i+1}\bar{y}_{p+1,q})/(A_{i,i-1} + A_{i,i+1}), \tag{5.27b}$$

$$(P\bar{y})_{2p,2q+1} = (A_{j,j-m}\bar{y}_{pq} + A_{j,j+m}\bar{y}_{p,q+1})/(A_{j,j-m} + A_{j,j+m}), \tag{5.27c}$$

$$(P\bar{y})_{2p+1,2q+1} = -\sum_{j\neq k} A_{kj}y_j/A_{kk}, \tag{5.27d}$$

where $i = 2+2p+2qm$, $j = 1+2p+(2q+1)m$, $k = 2+2p+(2q+1)m$. In (5.27d) y-values obtained with (5.27a-c) are used. It is possible to use the right-hand-side in (5.27d); sometimes this enhances the rate of convergence. A matrix dependent restriction is obtained with (5.26). Matrix-dependent prolongations of this and related type have been proposed by Alcouffe, Brandt, Dendy and Painter (1981), Kettler (1980, 1982), Kettler and Meijerink (1981).

The coarse grid problem (5.10) is not solved exactly of course, but approximately. In the MGD-family of MG codes we do this with one two-grid iteration employing an additional coarser grid with doubled mesh-size, and so on recursively, until the coarsest grid (usually a 3×3 grid) is reached, where a few iterations (usually one) are performed according to (2.2). Smoothing is the costliest part of the algorithm. Therefore we choose to let coarse grid correction precede smoothing (i.e. we have (5.10), (5.14)), so that the first time that smoothing takes place on the finest grid we already have a first approximation available. The resulting MG method is said to be of sawtooth type, because its schedule is represented in a natural way by the schematic of Fig. 5.2, which is a sawtooth curve.

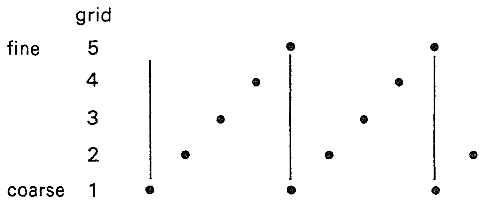


Fig. 5.2 Sawtooth multigrid schedule. A dot represents a smoothing step.

Various more general MG schedules have been described, see for example Brandt (1977), Stüben and Trottenberg (1982). Some comparative experiments are described in Wesseling (1982A). The sawtooth schedule is the simplest possible MG schedule. One may wonder whether such a simple fixed schedule can handle a sufficiently large variety of cases. Experience indicates that the answer is affirmative, see e.g. the experiments carried out by Wesseling and Sonneveld (1980), Kettler (1982), Wesseling (1982A,B), Hemker, Kettler, Wesseling and de Zeeuw (1983), McCarthy (1983). In transonic potential flow computation an MGD-type method has proved reliable, see Nowak and Wesseling (1983). We think that with an effective smoother and accurate coarse grid approximation, a simple MG schedule suffices for linear problems.

The sawtooth schedule can be programmed in a simple way without using recursion. Let the computational grids employed be denoted by $G^1, G^2, \ldots, G^\ell$, with G^1 the coarsest and G^ℓ the finest grid. Let a superscript k indicate grid-functions and operators on G^k. Let one application of the smoothing process (5.14) be executed by a subroutine SMOOTHING (y,b,k). Then a quasi-FORTRAN outline of MG algorithms using the sawtooth schedule is given by:

```
C     MULTIGRID PROGRAM, SAWTOOTH SCHEDULE
C     INITIAL GUESS IS $y^\ell = 0$
      $r^\ell = b^\ell$
      DO 10 k = ℓ-1(-1)1
      CALL RESTRICTION (r,k)        $r^k = R^{k+1}r^{k+1}$
   10 CONTINUE
C     START OF maxit MULTIGRID ITERATIONS
      DO 50 n = 1(1) maxit
      IF (n.EQ.1) GO TO 30
      CALL RESIDUE (r,b,y,ℓ)        $r^\ell = b^\ell - A^\ell y^\ell$
      DO 20 k = ℓ-1(-1)1
      CALL RESTRICTION (r,k)        $r^k = R^{k+1}r^{k+1}$
   20 CONTINUE
   30 $y^1 = 0$
      CALL SMOOTHING (u,r,1)        $y^1 = y^1 + B^1(r^1 - Ay^1)$
      DO 40 k = 2(1)ℓ-1
      CALL PROLONGATION (y,y,k)     $y^k = P^k y^{k-1}$
      CALL SMOOTHING (y,r,k)        $y^k = y^k + B^k(r^k - A^k y^k)$
   40 CONTINUE
      CALL PROLONGATION (v,y,ℓ)     $v^\ell = P^\ell y^{\ell-1}$
      $y^\ell = y^\ell + v^\ell$
      CALL SMOOTHING (y,b,ℓ)        $y^\ell = y^\ell + B^\ell(b - A^\ell y^\ell)$
   50 CONTINUE
```

Based on this algorithm, the MGD family of codes is being developed. Two portable FORTRAN codes have been implemented, called MGD1 and MGD5. They can be obtained by sending a magnetic tape to the second author. In these codes, prolongation and restriction are of 7-point type. For smoothing MGD1 and MGD5 use ILU and ILLU, respectively. Versions MGD1V and MGD5V have been designed for auto-vectorization on vector computers, such as the CYBER-205 and the CRAY-1, without sacrificing much on sequential machines. They are easily changed to versions MGD1S and MGD5S, which are slightly faster on sequential machines. More details, and CPU-time measurements on CYBER-170, CYBER-205 and CRAY-1 can be found in Hemker et al. (1983, 1983), Hemker and de Zeeuw (1984). Extensive tests of MGD1 have been carried out by McCarthy (1983). Other MG-software that is generally available is the collection of multigrid solution modules MGOO, see Foerster and Witsch (1982).

In order to facilitate comparison with other methods, especially CGS, for which we only have a research code in another programming language, we will estimate the cost in flops per finest grid-point. The cost of one MGD1 iteration is 30 flops per finest grid-point, see Wesseling (1982B). In MGD1 the cost of a smoothing step on one grid is 19 flops, for MGD5 it is 37 flops per grid-point, as shown in section 2.

The number of grid-points on all grids taken together is about 4/3 times the number of grid-points of the finest grid. Hence the total smoothing work for one MG iteration for MGD1 or MGD5 is about 25 or 49 flops per finest grid-point, respectively. Since the only difference between the two codes is the smoothing process, we estimate that the cost of one MGD5 iteration is 30-25+49=54 flops per finest grid-point. The measured CPU-time ratio on a CYBER-170 is 1.6.

Some design considerations concerning the MGD codes can be found in Wesseling (1982B). These codes have been constructed such that they are perceived by the user just like any other code for solving linear systems of algebraic equations. The user has only to give the matrix and the right-hand-side in a prescribed data structure. The matrix should have a sparsity pattern corresponding to a 7-point finite difference discretization. The user remains unaware of the underlying multigrid algorithm, and cannot make any choices or decisions, since the code is completely autonomous (black box MG, cf. Dendy (1982)).

The use of coarse grid Galerkin approximation (CGGA) (5.12) greatly facilitates the realization of the design goals just mentioned, since by using (5.12) the algorithm can set up the coarse grid operators independently from the user, using as input only the fine grid matrix. This would be less easy to achieve with the popular alternative of coarse grid finite difference approximation, (CGFDA), in which the coarse grid matrices are finite difference approximations of the given differential equation, usually of the same type as the fine grid matrix. CGFDA has another disadvantage, namely, that the approximations obtained on the coarsest grids make little sense if the coefficients of the differential equation are sampled pointwise. This may lead to divergence; Wesseling (1982B) gives an example. Of course the user can avoid this by using suitably averaged values of the coefficients of the differential equation on the coarsest grids. Using CGGA leads automatically to an accurate type of averaging. A disadvantage of CGGA can be the cost, which in practice equals the cost of about two MG iterations with our codes. If the coefficients of the differential equation are not expensive it is cheaper to set up finite difference approximations. For a few experiments comparing CGFDA and CGGA and a few remarks on efficient programming of (5.12), see Wesseling (1982A,B). The total work for computing A^k, $k = \ell-1(-1)$ is found to be about 64 flops per finest grid point, for a 7-point finite difference approximation on the finest grid.

Additional preliminary work is required for setting up the incomplete decompositions before iteration starts. The total cost of preliminary work is equivalent to about 3 iterations for MGD1 and 2 iterations for MGD5, cf. Hemker and de Zeeuw (1984). This preliminary work is considerable in view of the fact that convergence is usually so rapid that only a few iterations are needed. This price buys robustness. For self-adjoint problems with smoothly varying coefficients of the same order of magnitude, one obtains good rates of convergence with pointwise relaxation processes for smoothing and CGFDA, which require little preliminary work. But with these MG-ingredients convergence will deteriorate if the problem is strongly anisotropic, or strongly non-self-adjoint (convection-diffusion at high Péclet number). Under these circumstances the MGD codes continue to converge fast, with some exceptions for MGD1.

Under what conditions may MG methods be expected to converge rapidly? A priori theoretical results are not available except for the Poisson equation, but a rule of thumb is that good smoothing processes can be found and rapid convergence may be expected if A satisfies (4.27). If (4.27) is strongly violated, deterioration of the rate of convergence may occur.

However, for convection-diffusion problems at high Péclet numbers MG methods have not performed well. This situation has improved only recently. In order to satisfy (4.27) upwind differencing must be used, or a sufficient amount of artifical viscosity must be added. Nevertheless, when (5.12) is used with 7-point or 9-point prolongation and restriction, the coarse grid matrices do not satisfy (4.27). This is illustrated by the transformation that an upwind difference undergoes by repeated application of (5.12) with 7-point prolongation and restriction:

$$\begin{array}{ccc} 0 & 0 & \\ -1 & 1 & 0 \\ & 0 & 0 \end{array} \qquad \begin{array}{ccc} -1 & 1 & \\ -5 & 4 & 1 \\ & -1 & 1 \end{array} \qquad \begin{array}{ccc} -5 & 5 & \\ -15 & 8 & 7 \\ & -5 & 5 \end{array} \qquad \begin{array}{ccc} -21 & 21 & \\ -51 & 16 & 35 \\ & -21 & 21 \end{array}$$

$$\begin{array}{ccc} -85 & 85 & \\ -187 & 32 & 155 \\ & -85 & 85 \end{array} \qquad \begin{array}{ccc} -341 & 341 & \\ -751 & 64 & 651 \\ & -341 & 341 \end{array} .$$

Scaling factors have been omitted. As the number of grids increases, the diagonal becomes weaker. Because the coarse grid matrices do not satisfy (4.27) the smoothing process does not perform well, and furthermore, the coarse grid solution may show wiggles. The situation becomes worse as the number of grids increases. Hence, convergence is not rapid or divergence occurs, unless the smoothing process is almost an exact solver on the finest grid. In that case, the bad coarse grid approximations are corrected on the finest grid and convergence is rapid; in fact the coarse grids are superfluous. ILLU has this property: ILLU-decomposition is almost exact for the convection-diffusion equation at high Péclet number with upwind differences. Therefore MGD5 works for convection-diffusion equations. For MGD1 cases of divergence have been found. If one does not want to use ILLU smoothing, an easy way out would seem to be not to use CGGA but CGFDA with upwind differences or artificial viscosity on all grids. But then convergence is found to become disappointingly slow. A very good way to handle the convection-diffusion equation turns out to be the use of matrix-dependent prolongation and restriction, with CGGA or CGFDA. With this prolongation and restriction CGGA leaves upwind differences invariant, so that the coarse grid matrices satisfy (4.27), and do not differ much from the coarse grid matrices obtained with upwind differencing and CGFDA. Good rates of convergence are obtained with smoothing processes less formidable than ILLU. For a more extensive treatment of the ideas just discussed and numerical experiments, see van Asselt (1982), de Zeeuw and van Asselt (1985), Hemker, Kettler, Wesseling and de Zeeuw (1983).

A rather different MG approach to the convection-diffusion equation, proposed by Brandt, is not to use upwind differencing, which has inherent anisotropic numerical viscosity, but to use isotropic artificial viscosity. This makes point-wise relaxation processes applicable for smoothing. The accuracy and probably also the rate of convergence is improved by what is called double discretization, which amounts to applying defect correction on every grid. For this approach see Brandt (1982) section 10.2. A disadvantage is that the method is especially designed for convection-diffusion problems, so that for other problems one would perhaps prefer other MG ingredients. No definitive results with this approach have been published as yet for the convection-diffusion equation.

The accuracy of upwind or artificial viscosity discretizations can also be improved by applying defect-correction on the finest grid only. See Hemker (1982) for an application of this idea to the convection-diffusion equation.

For ease of programming of MG methods it is very convenient if coarser grids can be obtained by mesh doubling. Therefore, the number of grid points of the finest grid in the x_i-direction should be given by $1+2^{\ell}(m_i-1)$, with m_i a small integer. Sometimes it is awkward to achieve this, for example when a system of partial differential equations is solved on a staggered grid. One can then change the number of grid points by either eliminating Dirichlet boundaries or not, or by increasing the number of discretized equations by adding artificial equations, for example the identity. This is called padding. Padding can also be used to make the shape of the computational region rectangular. Of course, the computational complexity is influenced unfavourably by padding.

6. NUMERICAL EXPERIMENTS

Realistic estimates of the performance in practice of CG and MG by purely theoretical means are possible only for very simple problems. Therefore, numerical experiments are necessary to obtain insight and confidence in the efficiency and robustness of a particular method. Numerical experiments can be used only to rule out methods that fail, not to guarantee good performance of a method for problems that have not yet been attempted. Nevertheless, one strives to build up confidence by carefully choosing test problems, trying to make them representative for large classes of problems, taking into account the nature of the mathematical models that occur in the field of application that one has in mind. For the development of CG and MG, in particular the subject areas of computational fluid dynamics, petroleum reservoir engineering and neutron diffusion are pace-setting. We will list here the most significant test problems, and discuss a few numerical results.

Only the case of a single second order elliptic equation in two dimensions is discussed, although the applicability of CG and MG is not restricted to this case. The general form of our problem then is, in Cartesian tensor notation,

$$-(a_{ij}\phi_{,j})_{,i} + (b_i u)_{,i} + cu = f. \qquad (6.1)$$

Important constant coefficient test problems are the following special cases of (6.1):

$$-(\varepsilon c^2+s^2)\phi_{,11} - 2(\varepsilon-1)sc\phi_{,12} - (\varepsilon s^2+c^2)\phi_{,22} = f \tag{6.2}$$

and

$$-\varepsilon\phi_{,ii} + c\phi_{,1} + s\phi_{,2} = f, \tag{6.3}$$

with $c = \cos\alpha$, $s = \sin\alpha$. Equation (6.2) is obtained by a coordinate rotation over an angle α for the anisotropic diffusion equation:

$$-\varepsilon\phi_{,11} - \phi_{,22} = f. \tag{6.4}$$

Equation (6.3) is the convection diffusion equation. Equation (6.2) is self-adjoint, and can be handled such that the matrix arising from discretization is SPD.

Problems with constant coefficients are thought to be representative of problems with smoothly varying coefficients. Of course, in the code to be tested the fact that the coefficients are constant should not be exploited. As pointed out by Curtiss (1981), one should keep in mind that for constant coefficient problems the spectrum of the matrix resulting from discretization can have very special properties, that are not present when the coefficients are variable. Therefore one should also carry out tests with variable coefficients, especially with CG, for which the properties of the spectrum are very important. For MG, constant coefficient test problems are often more demanding than variable coefficient problems, because it may happen that the smoothing process is not effective for certain combinations of ε and α. This fact goes easily unnoticed with variable coefficients, where the unfavourable values of ε and α perhaps occur only in a small part of the domain.

In petroleum reservoir engineering and neutron diffusion problems quite often equations with strongly discontinuous coefficients appear. For these equations (6.2) and (6.3) are not representative. Suitable test problems with strongly discontinuous coefficients have been proposed by Stone (1968) and Kershaw (1978); a definition of these test problems may also be found in Kettler (1982). In Kershaw's problem the domain is non-rectangular, but is a rectangular polygon. The matrix for both problems is SPD. For the parameter p in Stone's problem we choose p=5 (cf. Kettler (1982)).

The four test problems just mentioned, i.e. (6.2), (6.3), and the problems of Stone and Kershaw, are gaining acceptance among CG and MG practitioners as standard test problems. Given these test problems, the dilemma of robustness versus efficiency presents itself. Should one try to devise a single code to handle all problems (robustness), or develop codes that handle only a subset, but do so more efficiently than a robust code? This dilemma is not novel, and just as in other parts of numerical mathematics, we expect that both approaches will be fruitful, and no single "best" code will emerge.

For CG methods a natural subdivision of the problems presents itself, namely in self-adjoint and non-self-adjoint problems. The former lead to SPD matrices, to which the applicability of classical CG is restricted. In non-self-adjoint cases, a non-symmetric CG variant should be used, for example CGS. Of course, CGS can be used also for SPD matrices, at little extra cost compared with classical CG.

The robustness and efficiency of CG and MG are determined to a large extent by the preconditioning and the smoothing process respectively. Pointwise relaxation methods, such as RBGS, are easy to implement and require no preliminary work before iterations start, and are efficient for (6.2), (6.3) for $\varepsilon \approx 1$. But these methods fail for ε differing widely from 1, and for the problems of Stone and Kershaw. In these cases suitable block relaxation methods are called for. We have not yet found a case where ILLU fails. ILU is found to fail in certain cases where property (4.27) is violated. AZ may fail also when (4.27) holds, for convection-diffusion problems. These findings will be amplified in the sequel.

Property (4.27) is violated in the case (6.2) for certain combinations of ε and α, for which the coefficients of the mixed derivative are relatively large. However, in practical applications the mixed derivative coefficient is often small. When the mixed derivative is introduced by a non-orthogonal coordinate transformation its coefficient is usually small, because for accuracy reasons one prefers coordinate transformations that do not deviate much from orthogonality. In anisotropic diffusion problems there is usually a preferred direction, along which one aligns one of the coordinate axes, so that $sc = 0$, and no mixed derivative is present.

Property (4.27) is also violated for (6.3) when $\varepsilon < h/2$ and central differences are used. With upwind differences it can still be violated on the coarse grids, as discussed in the preceding section.

Apart from a robust smoothing process, an MG method for the problems of Stone and Kershaw needs matrix-dependent prolongation and restriction, because of the occurrence of discontinuous coefficients.

Numerical experiments with MG concerning special cases of (6.2) (notably Poisson's equation) have been reported by Brandt (1977), Hackbusch (1978), Nicolaides (1979), Foerster et al. (1981), Foerster and Witsch (1982), Kettler (1982), Wesseling (1982A,B), Hemker et al. (1983, 1983). We will not list here experiments with CG, of which there are many more. Hackbusch (1978) and Foerster and Witsch (1982) include examples of Poisson's equation in non-rectangular regions. In the last mentioned publication also an MG method specially designed for Poisson's equation is presented. Computing times are reported similar to those obtained with fast Poisson solvers using the fast Fourier transform and cyclic reduction, and about 15 times as fast as a certain CG method (ICCG, Meijerink and van der Vorst (1977)) on a 257×257 mesh. It is to be noted that ICCG is much more generally applicable than the MG method concerned, which uses RBGS smoothing. This method would fail for example for the problems of Stone and Kershaw, for which ICCG performs well.

For test problem (6.3) MG results have been reported by Wesseling and Sonneveld (1980), Hemker (1982), Wesseling (1982A,B), Hemker, Kettler, Wesseling and de Zeeuw (1983), de Zeeuw and van Asselt (1985). Because the discretization matrix is not SPD, classical CG cannot be applied. Chebyshev iteration has been used for this type of problem by Manteuffel (1977, 1978) and van der Vorst (1981). This method is not parameter free, unlike CGS.

We will present results for the general case of (6.2) and (6.3), letting α vary with intervals of 15^o, and choosing $\varepsilon << 1$, ε intermediate, and $\varepsilon = 1$. The following methods will be tested:

- MGD1 and MGD5, described in section 5;
- CGS1 and CGS5, the CGS method described in section 4 with ILU and ILLU preconditioning, respectively;
- MGHZ and MGAZ, which are MGD1 with ILU smoothing replaced by HZ and AZ smoothing, respectively.

For easy reference, in the following table we give the operation counts for the various methods, as determined before. For MGHZ and MGAZ the operation count is determined by noting that the work for MGD1 excluding smoothing is 5 flops per finest grid-point. The smoothing work with HZ and AZ is 4/3 times the work of a single grid iteration. Here we neglect certain savings that are possible because the residue is zero in half the number of grid-points after application of HZ and AZ. From the results reported by Hemker, Wesseling and de Zeeuw (1983) we deduce that both for PW and IW the measured CP-time ratio on a CYBER-170 is MGD1 : MGHZ = 1.24. From Hemker and de Zeeuw (1984) we deduce that on the same machine MGD5 : MGD1 = 1.10 for PW and MGD5 : MGD1 = 1.62 for IW. These figures are roughly consistent with table 6.1.

Table 6.1

Flops per (finest) grid-point for one iteration (IW) and preliminary work (PW)

	MGD1	MGD5	MGHZ	MGAZ	CGS1	CGS5
PW	87	114	69	74	17	29
IW	30	54	22	40	60	88

We have run test problems (6.2) and (6.3) on a uniform 65 × 65 computational grid in the unit square. The initial guess is given by $-\sin\pi x_1 \sin\pi x_2 + \sin 48\pi x_1 \sin 48\pi x_2$, and the boundary conditions, which are eliminated, are given by

$$\phi\big|_{\partial\Omega} = x_i x_i . \qquad (6.5)$$

For the MG methods, the termination criterion was that the ℓ_2-norm of the residue should be less than 10^{-10}, with a maximum of 10 iterations. The CG iterations were terminated after a residue reduction factor of 10^{-8} had been reached, with a maximum of 15 iterations.

For test problem (6.2), the discretization of the mixed derivative is as given in Fig. 6.1.

$$\begin{array}{ccc} \frac{1}{2} & -\frac{1}{2} & \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ & -\frac{1}{2} & \frac{1}{2} \end{array}$$

Fig. 6.1 Difference molecule for $-h^2\phi_{,12}$.

The following table specifies the problems that were treated. For problems 5, 6, upwind difference were used, for the other problems, central differences.

Table 6.2

Specification of test problems

Problem	1	2	3	4	5	6
Equation	(6.2)	(6.2)	(6.3)	(6.3)	(6.3)	(6.3)
ε	10^{-2}	10^{-8}	10^{-1}	$h/2$	10^{-3}	10^{-8}

In Figs. 6.2-6.5 we give a graphical representation of the number of iterations needed to reduce the ℓ_2-norm of the residue by a factor 10, or, roughly speaking, to gain a decimal figure in accuracy. This number is given by

$$N = n/\log_{10}\{||\text{initial residue}||/||\text{final residue}||\} \qquad (6.6)$$

where n is the number of iterations that were performed.

Computations were performed with α a multiple of 15°. If for a value of α no symbol appears for one of the methods, this means that more than 10 iterations are necessary to gain one decimal, or that the method diverges.

The results clearly show that the rate of convergence can strongly depend on α, and that performing experiments for just a few values of α can be misleading.

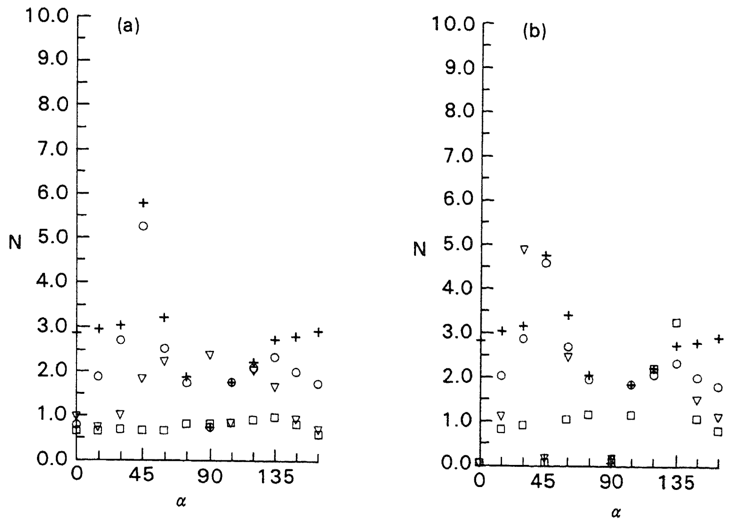


Fig. 6.2 Multigrid results for equation (6.2). (a): Problem 1. (b): Problem 2. ∇: MGD1; □: MGD5; +: MGHZ; O: MGAZ.

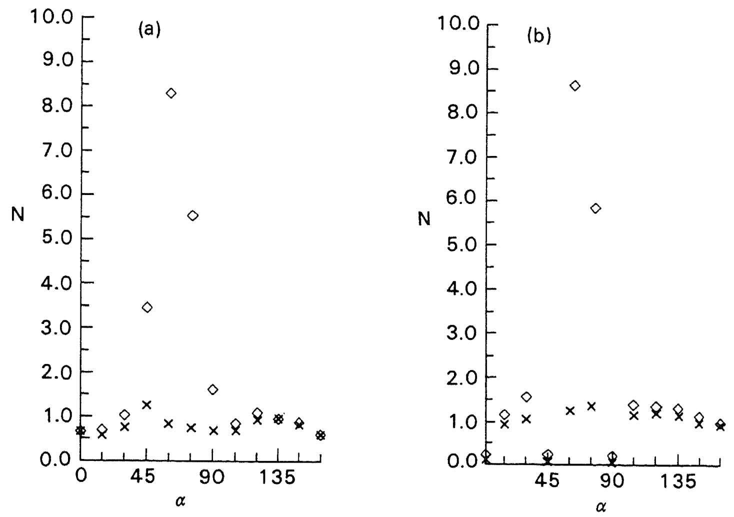


Fig. 6.3 CGS results for equation (6.2). (a): Problem 1; (b): Problem 2; ◊: CGS1; ×: CGS5.

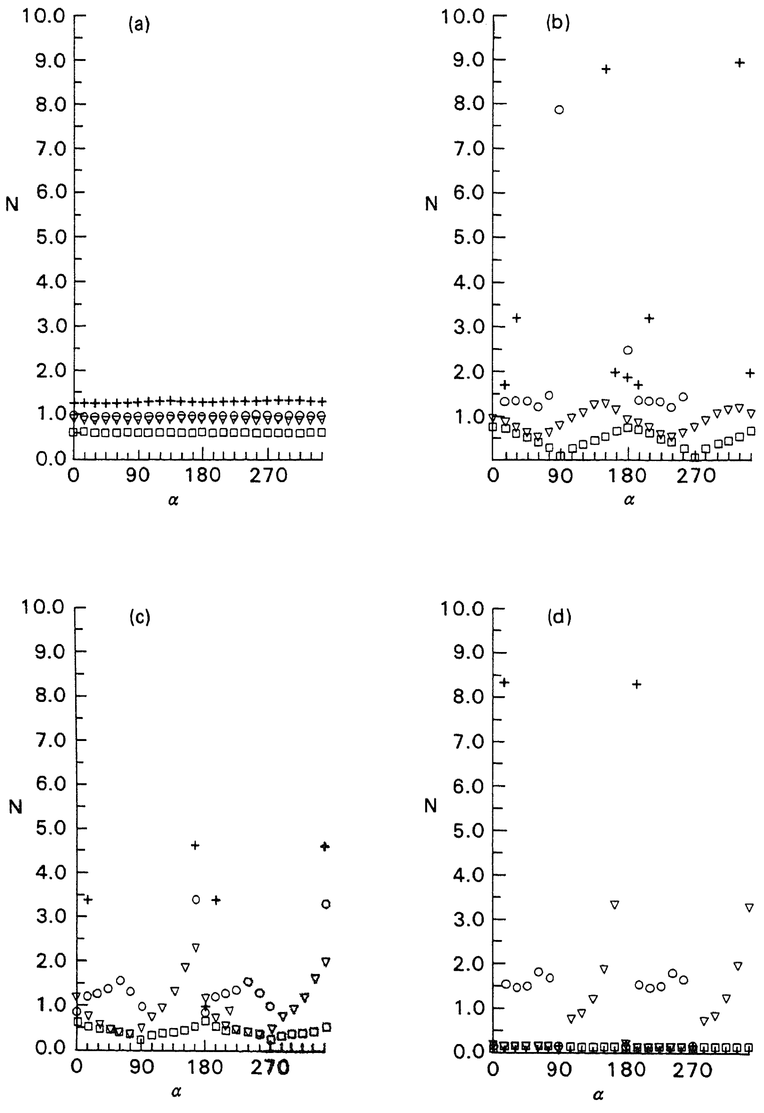


Fig. 6.4 Multigrid results for equation (6.3). (a)-(d): Problems 3-6, respectively. Symbols as in Fig. 6.2

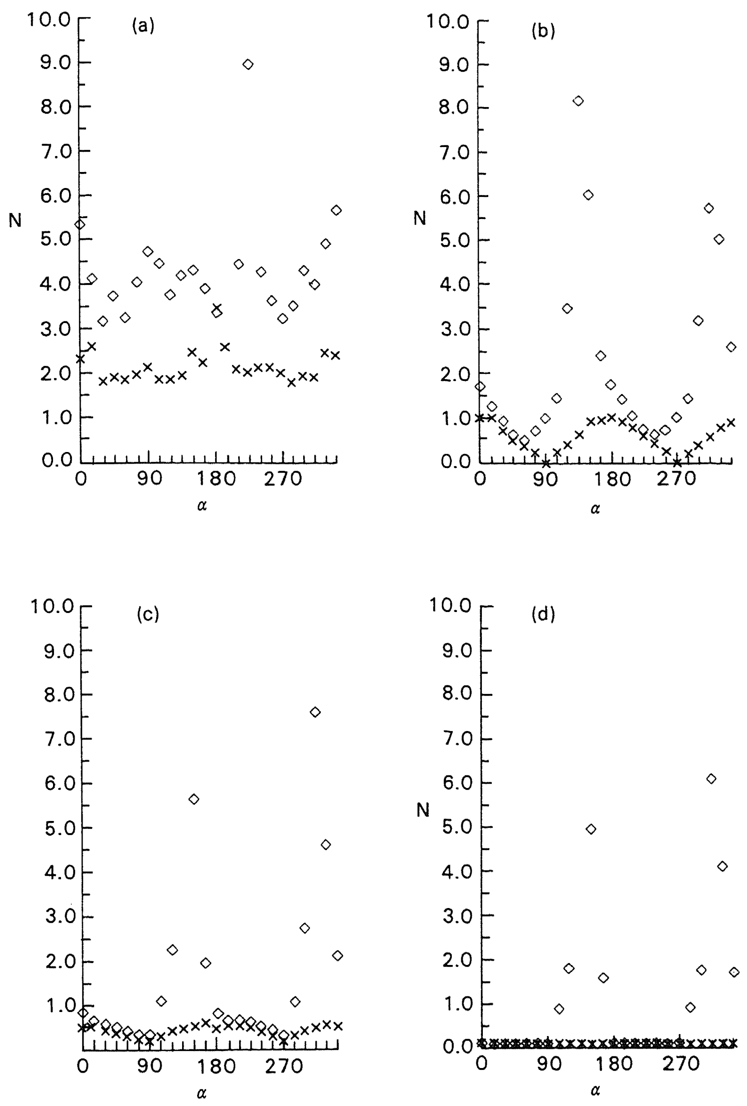


Fig. 6.5 CGS results for equation (6.3). (a)-(d): Problems 3-6, respectively. Symbols as in Fig. 6.3

Fig. 6.2 shows that for the anisotropic diffusion problem (6.2) MGD1 does not work well for α slightly different from 90°. This is predicted by smoothing analysis (Kettler (1982)), which does not explain, however, why it works for α precisely 90°. For this problem, MGD1 out-performs the other MG methods for α around 0° or 180°, taking table 6.1 into account. For general α, MGD5 is the best MG method. Fig. 6.3 shows that CGS1 and CGS5 behave much like MGD1 and MGD5, respectively. This means that in that case, when ILU or ILLU is a good smoother, it is a good preconditioner, and vice-versa. CGS5 is the most efficient method for this problem. Of course, with classical CG one would even be better off, and it is guaranteed to work (hence, CGS also), since the matrix is SPD.

Fig. 6.4 shows that MGHZ and MGAZ do not work well for equation (6.3). This is because HZ and AZ are ineffective smoothing processes for convection-diffusion problems. For small ε, ILLU is almost an exact solution method, and MG or CG acceleration is in fact not needed. MGD1 also works well. For a detailed discussion of the behaviour of the MGD codes for convection-diffusion problems we refer to the preceding section. Comparison of Figs. 6.4 and 6.5 shows that for this problem CGS is less effective than MG in accelerating ILU and ILLU. MGD5 is the most efficient method for problem (6.3). Nevertheless, CGS is a good acceleration method for these non-symmetric problems.

The rate of convergence of MG is found to be unaffected by mesh-refinement, with exceptions in the convection-diffusion case discussed in the preceding section. Table 6.3 gives some results for N as defined in equation (6.6) for CGS1 and CGS5, as the mesh-size is varied. The dependence on h is not clear-cut. As h decreases, the required number of iterations generally increases, but there are exceptions.

Table 6.3

Number of iterations per decimal figure for various mesh-sizes for CG methods

1/h		CGS1	CGS5		CGS1	CGS5
33	Problem 2	1.151	.857	Problem 5	.899	.369
65	$\alpha=120^\circ$	1.322	1.186	$\alpha=165^\circ$	1.967	.598
129		3.340	1.842		div	.233

In the case of problem 5, $\alpha=120^\circ$, like CGS1, MGD1 is found to diverge see Hemker, Kettler, Wesseling and de Zeeuw (1983). The explanation and the remedy has been given in the preceding section. In the case of CGS1 all we can say at present is, that apparently ILU is not a good preconditioning for this problem.

Next, we turn to the test-problems of Stone and Kershaw. Because the matrix is SPD, CGS will behave more or less like classical CG, which has been applied to these problems by several authors. Therefore CGS will not be used. The MGD1 and MGD5 codes are not applicable, because matrix-dependent prolongation and restriction is necessary. We show some of the results obtained by Kettler (1982). The MG methods used are similar to MGD1 and MGD5, but matrix-dependent prolongation and restriction is used, as defined by (5.27) and (5.26), and the sawtooth-cycle is replaced by the V-cycle. Fig. 6.6 gives $\log_{10}||\text{residue}||_2$ as a function of the estimated number of flops per finest grid-point. The ILU and ILLU iteration methods are accelerated by MG and classical CG. As explained in section 4, CG can be used to accelerate any iterative method (which corresponds to a symmetric preconditioned matrix), and Kettler (1982) has used CG to accelerate MG (MGCG). Of course, one could also try to use MG for acceleration of CG. Then one would expect fast convergence if CG is a good smoother; therefore one would have to tailor CG such that it works primarily on the non-smooth components of the error. We have not pursued this avenue.

The figures show no systematic trend. All methods converge very rapidly compared with older methods, see Kershaw (1978). CG is not inferior to MG in these tests, but when the grid is refined, CG would probably start lagging behind. The dimension of the computational grid was 31×31 for Stone and 51×51 for Kershaw. By padding (see the preceding section) this was increased to 33×33 and 57×57, respectively, in order to make construction of 4 or 3 coarse grids possible by mesh doubling. Padding was also used to fill in the L-shaped region in Kershaw's problem to a square, to facilitate MG programming. For more results, including the use of several other smoothing processes, see Kettler (1982).

Fig. 6.6 shows that CG acceleration of MG is effective when ILU smoothing is used, but with ILLU it does not help, although it does no harm either. CG and MG have also been applied to the test problems of Stone and Kershaw (and two other similar problems) by Behie and Forsyth (1983). They advocate acceleration of MG in its non-symmetric sawtooth variant by means of Orthomin (Vinsome (1976)), a CG-variant for non-symmetric problems.

7. CONCLUSIONS

Multigrid and conjugate gradient type techniques for the acceleration of iterative methods have been discussed. A detailed discussion has been given of incomplete factorizations (ILU and ILLU), which lend themselves especially well for MG or CG acceleration.

A brief review has been given of the theoretical background of classical CG and preconditioning. Classical CG methods are restricted to SPD matrices, but generalization is possible. One such generalized algorithm, called the CGS method, has been presented. Preconditioning of CG type methods has been discussed.

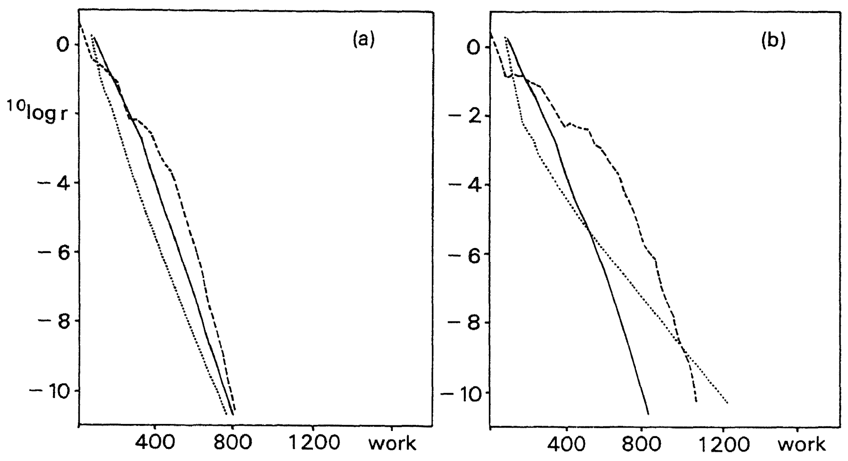
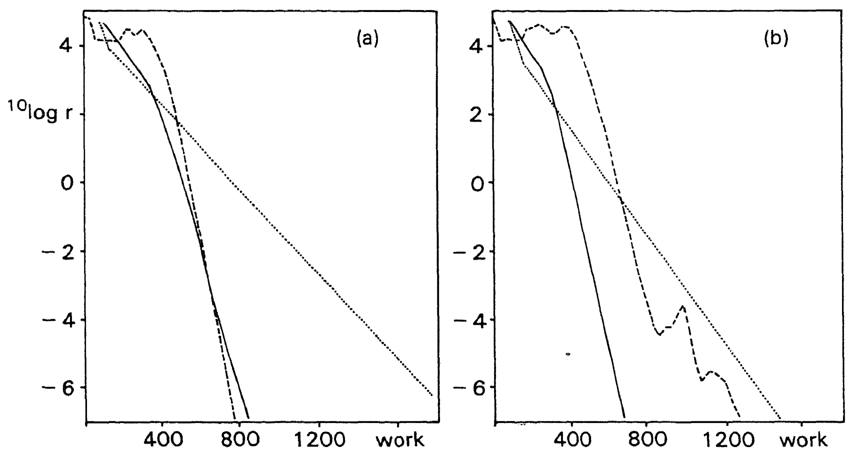


Fig. 6.6 Results for test problems of Stone (above) and Kershaw (below)
(a) : ILLU; (b) : ILU.
—— : MGCG; --- : CG; ... : MG.



Multigrid methods have been discussed within the framework of acceleration techniques. Various ways of looking at the smoothing factor have been discussed. Prolongation, restriction and coarse grid approximation methods have been reviewed. Two portable, autonomous multigrid codes, MGD1 and MGD5, have been introduced. MG treatment of convection-diffusion problems entails special difficulties, and ways to overcome these have been outlined.

For the general class of problems that we have treated no rate of convergence theory of practical utility is available. Therefore, numerical experiments are necessary for comparison and validation purposes. The choice of a suitable set of test problems has been discussed, and experiments described with several CG and MG methods, including a combination of MG and CG, both self-adjoint and non-self-adjoint problems, and problems with strongly discontinuous coefficients. These problems are of medium size, and roughly speaking, CG is about as efficient as MG, but as the mesh is refined, we would expect CG to lag behind. It should be remembered that CG is easier to program than MG.

Generally speaking, the use of incomplete factorizations leads to more robust and efficient methods than the use of line-relaxations with a zebra pattern, for the test problems considered. With incomplete line factorization (ILLU) one can handle all problems considered with a single code (MGD5 or CGS5), without requiring user-provided adaptations.

ACKNOWLEDGEMENT

The authors are indebted to Mr. W. Lioen for performing the calculations with the MGHZ and MGAZ methods.

REFERENCES

Alcouffe, R.E., Brandt, A., Dendy, Jr., J.E. and Painter, J.W. (1981) The multigrid method for the diffusion equation with strongly discontinuous coefficents. *SIAM J. Sci. Stat. Comp.*, **2**, pp. 430-454.

Asselt, E.J. van (1982) The multigrid method and artificial viscosity. In: Hackbusch and Trottenberg, pp. 313-326.

Axelsson, O. (1977) Solution of linear systems of equations: iterative methods. In: "Sparse Matrix Techniques", V.A. Barker (ed.), Lecture Notes in Math. 572, Springer-Verlag, Berlin, pp. 1-51.

Axelsson, O. (1980) Conjugate gradient type methods for unsymmetric and inconsistent systems of linear equations. *Lin. Algebra and its Applications*, **29**, pp.1-16.

Axelsson, O. (1982) Numerical Integration of Differential Equations and Large Linear Systems. In: J. Hinze (ed.), Proceedings, Bielefeld 1980. Lecture Notes in Mathematics 968, Springer-Verlag, Berlin, pp. 310-322.

Axelsson, O. (1983) A General Incomplete Block-Matrix Factorization Method. Report 8337, Catholic University, Nijmegen, The Netherlands.

Bakhvalov, N.S. (1966) On the convergence of a relaxation method with natural constraints on the elliptic operator. *USSR Comp. Math. Math. Phys.*, **6**, No. 5, pp. 101-135.

Behie, A. and Forsyth, Jr., P. (1983) Comparison of Fast Iterative Methods for Symmetric Systems. *IMA J. of Numer. Anal.*, **3**, pp. 41-63.

Braess, D. (1981) The contraction number of a multigrid method for solving the Poisson equation. *Numer. Math.*, **37**, pp. 387-404.

Braess, D. (1982) The convergence rate of a multigrid method with Gauss-Seidel relaxation for the Poisson equation. In: Hackbusch and Trottenberg, pp. 368-386.

Braess, D. and Hackbusch, W. (1983) A new convergence proof for the multigrid method including the V-cycle. *SIAM J. Num. Anal.*, **20**, pp. 967-975.

Brandt, A. (1977) A Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, **31**, 333-390.

Brandt, A. (1982) Guide to Multigrid Development. In: Hackbusch and Trottenberg, pp. 220-312.

Concus, P. and Golub, G.H. (1976) A generalized conjugate gradient method for nonsymmetric systems of linear equations. In: R. Glowinski and J.L. Lions (eds.), Proc. of the Second Int. Symposium on Computer Methods in Applied Sciences and Engineering, Paris, 1975. Lecture Notes in Economics and Mathematical Systems, 134, Springer-Verlag, Berlin.

Concus, P., Golub, G.H. and Meurant, G. (1982) Block Preconditioning for the Conjugate Gradient Method. Report LBL-14856, Lawrence Berkeley Laboratory, Un. of California.

Curtiss, A.R. (1981) On a property of some test equations for finite difference or finite element methods. *IMA J. Numer. Anal.*, **1**, pp. 369-375.

Dendy, Jr. J.E. (1982) Black Box Multigrid. *J. Comp. Phys.*, **48**, pp. 366-386.

Fedorenko, R.P. (1964) The speed of convergence of one iterative process. *USSR Comp. Math. Math. Phys.*, **4**, no. 3, pp. 227-235.

Fletcher, R. (1976) Conjugate gradient methods for indefinite systems. In: G.A. Watson (ed.): Numerical analysis. Proceedings, Dundee 1975, Lect. Notes in Math., 506, Springer-Verlag, Berlin, pp. 73-89.

Foerster, H., Stüben, K. and Trottenberg, U. (1981) Non-standard multigrid techniques using checkered relaxation and intermediate grids. In: M. Schultz (ed.): Elliptic Problem Solvers, Academic Press, New York, pp. 285-300.

Foerster, H. and Witsch, K. (1982) Multigrid software for the solution of elliptic problems on rectangular domains: MG00 (Release 1). In: Hackbusch and Trottenberg, pp. 427-461.

Gustafsson, I. (1978) A class of first order factorization methods. *BIT,* **18**, pp. 142-156.

Hackbusch, W. (1978) On the multigrid method applied to difference equations. *Computing,* **20**, pp. 291-306.

Hackbusch, W. (1980) Convergence of multigrid iterations applied to difference equations. *Math. Comp.,* **34**, pp. 425-440.

Hackbusch, W. and Trottenberg, U., eds. (1982) Multigrid Methods. Proceedings, Köln-Porz, 1981. Lecture Notes in Mathematics 960. Springer-Verlag, Berlin.

Hageman, L.A. and Young, D.M. (1981) Applied Iterative Methods. Academic Press, New York.

Hemker, P.W. (1982A) A note on defect correction processes with an approximate inverse of deficient rank. *J. Comp. Appl. Math.,* **8**, pp. 137-139.

Hemker, P.W. (1982) Mixed Defect Correction Iteration for the Accurate Solution of the Convection Diffusion Equation. In: Hackbusch and Trottenberg, pp. 485-501.

Hemker, P.W., Kettler, R., Wesseling, P. and de Zeeuw, P.M. (1983) Multigrid methods: development of fast solvers. *Appl. Math. and Comp.,* **13**, pp. 311-326.

Hemker, P.W., Wesseling, P. and de Zeeuw, P.M. (1983) A portable vector-code for autonomous multigrid modules. Report NW 154/83, Mathematical Centre, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands.

Hemker, P.W. and de Zeeuw, P.M. (1984) Some implementations of multi-grid linear systems solvers. In this volume.

Kershaw, D.S. (1978) The incomplete Choleski-conjugate gradient method for the iterative solution of systems of linear equations. *J. Comp. Phys.,* **26**, pp. 43-65.

Kettler, R. (1980) A study of the applicability of the multiple grid method in reservoir simulation. Part II. Master's thesis, Delft University of Technology, Nov. 1980.

Kettler, R. (1982) Analysis and Comparison of Relaxation Schemes in Robust Multigrid and Preconditioned Conjugate Gradient Methods. In: Hackbusch and Trottenberg, pp. 502-534.

Kettler, R. and Meijerink, J.A. (1981) A multigrid method and a combined multigrid-conjugate gradient method for elliptic problems with strongly discontinuous coefficients in general domains. Publication 604, Shell Research B.V., Kon. Shell Expl. and Prod. Lab., Rijswijk, The Netherlands.

Maitre, J.-F. and Musy, F. (1983) Méthodes multigrilles: opérateur associé et estimations du facteur de convergence; le cas du V-Cycle. C.R. Acad. Sc. Paris, 296, Série I, pp. 521-524.

Manteuffel, T.A. (1977) The Tchebychev Iteration for Nonsymmetric Linear Systems. *Numer. Math.*, **28**, pp. 307-327.

Manteuffel, T.A. (1978) Adaptive Procedure for Estimating Parameters for the Nonsymmetric Tchebychev. *Numer. Math.*, **31**, pp. 183-208.

McCarthy, G.J. (1983) Investigations into the Multigrid Code MGD1. Report AERE R 10889, Harwell, U.K.

McCormick, S.F. (1982) An algebraic interpretation of multigrid methods. *SIAM J. Numer. Anal.*, **19**, pp. 548-560.

McCormick, S.F. (1983) Multigrid Methods for Variational Problems: the V-cycle. *Math. and Comp. in Sim.*, **25**, pp. 63-65.

Meurant, G. (1983) Vector Preconditionings for the Conjugate Gradient Method. To be submitted to BIT. Private Communication.

Meijerink, J.A. and van der Vorst, H.A. (1977) An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, **31**, pp. 148-162.

Meijerink, J.A. (1983) Iterative Methods for the Solution of Linear Equations based on Incomplete Factorization of the Matrix. Publication 643, Shell Research B.V., Kon. Shell Expl. and Prod. Lab., Rijswijk, The Netherlands, July 1983.

Meijerink, J.A. and van der Vorst, H.A. (1981) Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J. Comp. Phys.*, **44**, pp. 134-155.

Musy, F. (1982) Sur les méthodes multigrilles: formalisation algébrique et démonstration de convergence. C.R. Acad. Sc. Paris 295, Serie I, pp. 471-474.

Nicolaides, R.A. (1979) On some theoretical and practical aspects of multigrid methods. *Math. Comp.*, **33**, pp. 933-952.

Nowak, Z. and Wesseling, P. (1983) Multigrid acceleration of an iterative method with application to transonic potential flow. In: INRIA, Proceedings Sixth International Conference on Computing Methods in Applied Sciences and Engineering, Versailles, France, Dec. 1983.

Stone, H.L. (1968) Iterative solution of implicit approximations of multidimensional partial differential equations, *SIAM J. Numer. Anal.*, **5**, pp. 530-558.

Stüben, K. and Trottenberg, U. (1982) Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications. In: Hackbusch and Trottenberg, pp. 1-176.

Underwood, R.R. (1976) An approximate factorization procedure based on the block Cholesky decomposition and its use with the conjugate gradient method. Report NEDO-11386, General Electric Co., Nuclear Energy Div., San Jose, CA.

Vinsome, P.K.W. (1976) ORTHOMIN, an iterative method for solving sparse sets of simultaneous linear equations. Society of Petroleum Engineers, paper SPE 5729.

Van der Vorst, H.A. (1981) Iterative Solution Methods for Certain Sparse Linear Systems with a Non-Symmetric Matrix Arising from PDE-Problems. *J. Comp. Phys.*, **44**, pp. 1-19.

Van der Wees, A.J., van de Vooren, J. and Meelker, J.H. (1983) Robust calculation of 3D transonic potential flow based on the nonlinear FAS multigrid method and incomplete LU-decomposition. AIAA paper 83-1950.

Wesseling, P. (1980) The rate of convergence of a multiple grid method. In: G.A. Watson (Ed.), Numerical Analysis. Proceedings, Dundee 1979. Lect. Notes in Math. 773, Springer-Verlag, Berlin, pp. 164-184.

Wesseling, P. (1982A) Theoretical and practical aspects of a multigrid method. *SIAM J. Sci. Stat. Comput.*, **4**, pp. 387-407.

Wesseling, P. (1982B) A robust and efficient multigrid method. In: Hackbusch and Trottenberg, pp. 614-630.

Wesseling, P. and Sonneveld, P. (1980) Numerical experiments with a multiple grid and a preconditioned Lanczos type method. In: R. Rautmann (ed.), Approximation methods for Navier-Stokes problems. Proceedings, Paderborn 1979. Lecture Notes in Math. 771, Springer-Verlag, pp. 543-562.

Widlund, O. (1978) A Lanczos method for a class of nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, **15**, pp. 801-812.

Young, D.M. (1971) Iterative solution of large linear systems. Academic Press, New York.

Zeeuw, P.M. de and van Asselt, E.J. (1985) The convergence rate of multi-level algorithms applied to the convection-diffusion equation. To appear, *SIAM J. Sci. Stat. Comp.*