NUMERICAL EXPERIMENTS WITH A MULTIPLE GRID AND A PRECONDITIONED LANCZOS TYPE METHOD

P. Wesseling and P. Sonneveld
Dept. of Mathematics, Delft University of Technology,
Julianalaan 132, 2600 AJ Delft, The Netherlands.

1. Introduction

Numerical experiments will be described with two methods for the numerical solution of difference schemes for general non-self-adjoint elliptic boundary value problems, namely a multiple grid method (MG method) and a preconditioned Lanczos type method (PIDR method). The interest of these relatively novel methods lies in the fact, that they seem potentially to be significantly more efficient than the methods that are currently popular. Our purpose is to investigate the efficiency of these methods by carrying out numerical experiments on two problems that are representative of the sort of problem that is encountered in engineering applications, namely the convection-diffusion equation, and the driven cavity problem for the Navier-Stokes equations.

2. Two testproblems

Our first testproblem is the convection-diffusion equation:

$$-(a_{ij}\phi_{,i})_{,j} + (b_i\phi)_{,i} + cu = f,\ x_i \in \Omega,$$
$$\phi\big|_{\partial\Omega} = g, \tag{2.1}$$

where Cartesian tensor notation has been used. The equation is assumed to be uniformly elliptic, i.e. there exist constants C_1, $C_2 > 0$ such that $C_1\xi_i\xi_i \le a_{ij}\xi_i\xi_j \le C_2\xi_i\xi_i$, $\forall\ \xi_i \in \mathbb{R}$. Our numerical experiments have been performed for the following special case:

$$-\varepsilon\phi_{,ii} + x_1\phi_{,1} = f = 2\varepsilon(x_1+x_2-x_ix_i) + x_1x_2(1-2x_1)(1-x_2),\ g = 0 \tag{2.2}$$

in two dimensions, with $\Omega = (0,1) \times (0,1)$. The exact solution is: $\phi = (x_1-x_1^2)(y_1-y_1^2)$. Keeping engineering applications in mind, special attention has been paid to the situation $\varepsilon << 1$. For simplicity we have restricted ourselves to Dirichlet boundary conditions, but this restriction is not necessary for the applicability of the MR and PIDR methods.

Our second testproblem is the vorticity-streamfunction formulation of the Navier-Stokes equations in two dimensions:

$$\psi_{,ii} = \omega, \qquad x_i \in \Omega,$$

$$u_i\omega_{,i} = Re^{-1}\omega_{,ii}\ , \quad u_1 = \psi_{,2}\ , \quad u_2 = -\psi_{,1}\ , \qquad x_i \in \Omega, \tag{2.3}$$

$$\psi\big|_{\partial\Omega} = 0\ , \qquad \psi_{,n}\big|_{\partial\Omega} = g,$$

with ψ the streamfunction, ω the vorticity, Re the Reynoldsnumber, and $\psi_{,n}$ the normal derivative of ψ.
Furthermore, $\Omega = (0,1) \times (0,1)$, and $g = 1$ for $x_2 = 1$, $g = 0$ elsewhere. This is the so-called driven-cavity problem.

Finite difference discretizations are chosen as follows. An equidistant computational grid Ω^ℓ with stepsize h is defined by:

$$\Omega^\ell \equiv \{(x_1,x_2)\big|\ x_i = m_i h,\ m_i = 1(1)2^\ell+1,\ h = (2+2^\ell)^{-1} \tag{2.4}$$

Note that Ω^ℓ lies a distance h within $\partial\Omega$. This is because the boundary conditions are substituted in the difference scheme. A more general stepsize may be chosen just as well for the PIDR method, but the programming of the MG method would become much more complicated. Forward and backward difference operators Δ_i and ∇_i are defined by:

$$(\Delta_1\phi)_{ij} \equiv (\phi_{i+1,j}-\phi_{ij})/h,\ (\nabla_1\phi)_{ij} \equiv (\phi_{ij}-\phi_{i-1,j})/h, \tag{2.5}$$

and similarly for Δ_2 and ∇_2; the subscripts i,j here indicate a point of Ω^ℓ in the usual way.

Equation (2.2) is discretized by means of the Il'in scheme [16] (twice rediscovered [4,7]):

$$-\gamma\Delta_1\nabla_1\phi - \varepsilon\nabla_2\Delta_2\phi + \tfrac{1}{2}x_1(\Delta_1+\nabla_1)\phi = f\ , \tag{2.6}$$

with $\gamma \equiv x_1 h \quad \coth(x_1 h/\varepsilon)$. The Il'in scheme has certain advantages when $\varepsilon << 1$ over a straightforward central or upwind difference scheme. These advantages do not interest us here; for us (2.6) is merely an example of a system on which to test the efficiency of the MR and PIDR methods. Near the boundary (2.6) is modified by substitution of the boundary conditions in the equation.

Equation (2.3) is discretized by means of central differences:

$$\begin{aligned} &\Delta_i\nabla_i\psi = h^{-2}\bar{\omega}\ , \\ &\tfrac{1}{2}u_i(\Delta_i+\nabla_i)\bar{\omega} = Re^{-1}\Delta_i\nabla_i\bar{\omega}\ , \end{aligned} \tag{2.7}$$

with

$$u_1 = \tfrac{1}{2}(\Delta_2+\nabla_2)\psi, \quad u_2 = -\tfrac{1}{2}(\Delta_1+\nabla_1)\psi.$$

The boundary contitions are:

$$\bar{\omega}_w + \tfrac{1}{2}\bar{\omega}_{w+1} - 3\psi_{w+1} = 3hg, \quad \psi_w = 0. \tag{2.8}$$

Here $\bar{\omega} = h^2\omega$. This rescaling of ω is introduced in order to get coefficients of O(1)

in (2.8). For the derivation of this well-known approximation, see e.g. [21]. The subscript w indicates a point of $\partial\Omega$, the subscript w+1 indicates its nearest neighbour in Ω^ℓ in the direction of the normal.

The nonlinear system (2.7) has to be solved by some iterative method. We have chosen the Newton method:

$$\begin{aligned}
&\Delta_i\nabla_i\psi = h^{-2}\bar{\omega} \ , \\
&\tfrac{1}{2}\hat{u}_i(\Delta_i+\nabla_i)\bar{\omega} + \tfrac{1}{2}u_i(\Delta_i+\nabla_i)\hat{\bar{\omega}} = \mathrm{Re}^{-1}\Delta_i\nabla_i\hat{\bar{\omega}} + \tfrac{1}{2}\hat{u}_i(\Delta_i+\nabla_i)\hat{\bar{\omega}}, \qquad (2.9)\\
&\bar{\omega}_w + \tfrac{1}{2}\bar{\omega}_{w+1} - 3\psi_{w+1} = 3hg, \quad \psi_w = 0,
\end{aligned}$$

with $\hat{}$ indicating values obtained from the preceding iteration. We do not claim that the use of the Newton method is always advisable; for us (2.9) is merely an interesting system on which to test the MG and PIDR methods.

Naturally a comparison with other methods would be of interest. A discussion of other methods is deferred to a later chapter. One method will be described here, namely the method called LAD method in [22]. This method is defined as follows:

$$\begin{aligned}
&\mathrm{Re}^{-1}\Delta_i\nabla_i\bar{\omega} = \tfrac{1}{2}\hat{u}_i(\Delta_i+\nabla_i)\hat{\bar{\omega}} \ , \\
&\Delta_i\nabla_i\psi = h^{-2}\bar{\omega} \ , \qquad (2.10)\\
&\bar{\omega}_w = r(-\tfrac{1}{2}\bar{\omega}_{w+1}+3\psi_{w+1}+3hg) + (1-r)\hat{\bar{\omega}}_w \ ,
\end{aligned}$$

with r a relaxation factor for which a good value has to be found by trial and error. Hence, the problem is reduced to two Poisson equations on a rectangle, which can be solved very efficiently with a fast Poisson solver.

Because of its great speed one is tempted to also use a fast Poisson solver for (2.6). This can be done iteratively as follows:

$$\varepsilon\Delta_i\nabla_i\phi = (\varepsilon-\gamma)\Delta_1\nabla_1\hat{\phi} + \tfrac{1}{2}x_1(\Delta_1+\nabla_1)\hat{\phi} - f \ . \qquad (2.11)$$

This method will be referred to as the FP method.

The iteration processes defined by (2.9), (2.10) and (2.11) will sometimes be referred to as outer iterations; the MG and PIDR methods are also of an iterative nature, and will sometimes be referred to as inner iterations.

3. A multiple grid method

For the intuitive background of multiple grid methods we refer to [6]. We start with a formal description of a specific multiple grid method which we shall call MG method. Then we discuss the relation between the MG method and other multiple grid methods, and make a few historical remarks.

In addition to the computational grid Ω^ℓ a hierarchy of computational grids Ω^k,

$k < \ell$ is defined by replacing ℓ by k in (2.4). The simultaneous use of such computational grids Ω^k is the reason for the appellation "multi-level methods" (cf. [6]), "multigrid methods" (cf. [12]) or "multiple grid methods" (cf. [19]). The d-dimensional vector of unknowns in each point of Ω^k is denoted by u^k, with d the number of unknowns in the partial differential equations that are to be solved. For example, for equations (2.6) or (2.9) d = 1 or d = 2 respectively. The set of functions U^k is defined by:

$$U^k \equiv \{u^k : \Omega^k \to \mathbb{R}^d\}$$

Furthermore, a prolongation operator $p^k : U^{k-1} \to U^k$ is defined by linear interpolation:

$$\begin{aligned}
(p^k u^{k-1})_{2i,2j} &= u^{k-1}_{ij} \ , \\
(p^k u^{k-1})_{2i+1,2j} &= \tfrac{1}{2}(u^{k-1}_{ij} + u^{k-1}_{i+1,j}) \ , \\
(p^k u^{k-1})_{2i,2j+1} &= \tfrac{1}{2}(u^{k-1}_{ij} + u^{k-1}_{i,j+1}) \ , \\
(p^k u^{k-1})_{2i+1,2j+1} &= \tfrac{1}{4}(u^{k-1}_{ij} + u^{k-1}_{i+1,j} + u^{k-1}_{i,j+1} + u^{k-1}_{i+1,j+1}) .
\end{aligned} \tag{3.2}$$

The restriction operator $r^k : U^k \to U^{k-1}$ is defined to be the adjoint of p^k:

$$\begin{aligned}
(r^k u^k)_{ij} = {} & \tfrac{1}{4} u^k_{2i,2j} + \tfrac{1}{8}(u^k_{2i+1,2j} + u^k_{2i-1,2j} + u^k_{2i,2j+1} + u^k_{2i,2j-1}) + \\
& \tfrac{1}{16}(u^k_{2i+1,2j+1} + u^k_{2i-1,2j+1} + u^k_{2i+1,2j-1} + u^k_{2i-1,2j-1}) .
\end{aligned} \tag{3.3}$$

The linear algebraic system to be solved (e.g. (2.6) or (2.9)) is denoted by:

$$A^\ell u^\ell = f^\ell \ . \tag{3.4}$$

The matrix A^ℓ is approximated on coarser grids by $A^{\ell-1}$, $A^{\ell-2}$,..., defined as follows:

$$A^{k-1} \equiv r^k A^k p^k \ , \quad k = \ell, \ell-1, \ldots \tag{3.5}$$

With each matrix A^k an approximate LU-decomposition $L^k U^k$ is associated; L^k and U^k will be defined in section 5.

The MG method is an iterative method. One iteration is defined as follows, in quasi-Algol:

Algorithm 3.1

$e^\ell := f^\ell - A^\ell u^\ell$;

for k:=ℓ(-1)2 do $e^{k-1} := r^k e^k$;

$v^1 := (L'U')^{-1} e^1$;

for k:=2(1)ℓ-1 do

```
begin v^k:=p^k v^{k-1};
      e^k:=e^k-A^k v^k;
      v^k:=v^k+(L^k U^k)^{-1} e^k
end;
v^ℓ:=p^ℓ v^{ℓ-1};
u^ℓ:=u^ℓ+v^ℓ;
e^ℓ:=f^ℓ-A^ℓ u^ℓ;
u^ℓ:=u^ℓ+(L^ℓ U^ℓ)^{-1} e^ℓ;
```

We proceed to estimate the computational complexity (number of operations) needed for this method. Every operation from the set {+,-,*,/,sqrt} is counted is one operation. Computation of pointers etc. is neglected. The computational complexity thus defined is easy to determine and is independent of the machine and the skill of the programmer, but gives only a rough indication of the actual computer time. However, for the comparison of methods this is a very useful quantity, and we feel that this kind of information should be provided more often.

It turns out that if A^ℓ results from a difference scheme based upon a 5-point or a 9-point difference molecule, then the structure of A^k, $k = \ell-1,\ell-2,\ldots$ corresponds to a 9-point molecule. One arrives at the following computational complexity:

	d = 1	d = 2
$A^k:=r^{k+1}A^{k+1}p^{k+1}$	254,278	1016,1112
L^k and U^k	25,41	246,406
$v^k:=(L^kU^k)^{-1}e^k$	16,20	58,72
$e^k:=r^{k+1}e^{k+1}$	18,18	36,36
$v^k:=p^kv^{k-1}$	1.5,1.5	3,3
$e^k:=e^k-A^kv^k$	15,27	60,108
$u^k:=u^k+v^k$	1,1	2,2

Table 3.1 Computational complexity for portions of MG-algorithm, divided by $(2^k+1)^2$.

The figure before the comma refers to the case that A^k (in rows 2,3 and 6) or A^{k+1} (in row 1) has 5-point structure, the figure after the comma corresponds to a 9-point structure (in our examples A has 5-point structure, A^k, $k < \ell$ have 9-point structures). We recall that d is the number of unknowns in the given differential equation. In the computation of A^k only terms proportional to $(2^k+1)^2$ have been taken into account. In addition there is a (negligible) constant amount of work.

The computation of A^k, L^k and U^k has to be performed only once for a given problem

(3.4); the associated computational work will be called "preliminary work", whereas the work to be performed for each iteration will be called "iteration work". For the complete MG method we then find:

	ℓ	2	3	4	5
d = 1	p.w.	$3.3.10^3$	$1.2.10^4$	$4.2.10^4$	$1.5.10^5$
	i.w.	$1.6.10^3$	$6.0.10^3$	$2.2.10^4$	$8.1.10^4$
d = 2	p.w.	$1.9.10^4$	$6.9.10^4$	$2.4.10^5$	$8.5.10^5$
	i.w.	$5.6.10^3$	$2.1.10^4$	$7.8.10^4$	$3.1.10^5$

Table 3.2 Computational complexity for MG-method. p.w. = preliminary work, i.w. = iteration work.

With the approximation $(2^k+1)^2 \cong 4^k$ one may deduce:

$$\begin{aligned} d = 1: &\quad \text{preliminary work} \tilde{=} 126.4^\ell, \\ &\quad \text{iteration work} \tilde{=} 72.4^\ell. \\ d = 2: &\quad \text{preliminary work} \tilde{=} 728.4^\ell. \\ &\quad \text{iteration work} \tilde{=} 259.4^\ell. \end{aligned} \tag{3.6}$$

Hence, 1 MG-iteration has a computational complexity of $O(4^\ell)$. As we will see, the distinguishing feature of multiple grid methods is that their rate of convergence is independent of the mesh-size of the computational grid, i.e. independent of ℓ. Requiring a precision of $O(4^{-\ell})$ (of the same order as the discretization error) then results in an overall computational complexity of $O(\ell 4^\ell)$.

The MG method described here is but one example of a multiple grid method; many different multiple grid methods are possible and have been proposed. The above algorithm is a variant of a multiple grid method used by Frederickson [10] for the Poisson equation. Frederickson did not use approximate LU decomposition, but an approximate inverse. We have tried this also, but in our applications approximate LU was significantly faster (roughly by a factor 2). The other multiple grid methods that we know of do not use approximate LU decomposition or an approximate inverse, but some form of relaxation iteration, such as Jacobi- or line-iteration. Except [12] the other methods define A^k, $k > \ell$ not by (3.5) but by a discretization of the differential equations. Also, p^k and r^k are often different: for p^k higher order interpolation may be used, and r^k may simply be given by canonical injection: $(r^k u^k)_{ij} = u^k_{2i,2j}$. The freedom one has in these matters is both a blessing and a curse: by making careful decisions ([6] contains much useful advice) one may obtain a very efficient method for a given problem, but the rate of convergence may also be disappointing if wrong decisions are made. We feel that in this respect algorithm (3.1) has the advantage, that it is completely defined and that it has worked without fail for the cases that we have tried. Furthermore, for a closely related algorithm

it has been proven [26] that for (2.1) in its full generality with Ω a square the computational complexity is $O(\ell 4^{\ell})$.

The intuitive ideas underlying multiple grid methods can already be found in the work of Southwell [24]. A multiple grid method was first proposed by Fedorenko [8]. A^k is defined by discretization, p^k by cubic interpolation and r^k by canonical injection. For the Poisson equation on a rectangle Fedorenko has shown in [9] that the asymptotic computational complexity is $O(\ell 4^{\ell})$. This was generalized to (2.1) on a rectangle by Bakhvalov [3]. In the same period a two-grid method has been proposed by Wachspress [25]. Apparently these methods did not find widespread use at the time although encouraging results were described in [25]. The present period of widespread interest in multiple grid methods was inaugurated by Brandt's pioneering paper [5]. His multiple grid methods are of the same type as the method of Fedorenko. For the Poisson equation a multiple grid method with A^k defined by (3.5) has been developed by Frederickson [10], who proves $O(\ell 4^{\ell})$ asymptotic computational complexity. The generalization to (2.1) on a rectangle is given in [26].

The multiple grid methods have been discussed here as linear equation solvers. They can also be used as special nonlinear equation solvers (see e.g. [6,13]), eliminating the need for the Newton-iterations (2.9). Furthermore, they can be used to construct adaptive discretizations [6]. These aspects fall outside the scope of the present paper.

4. A preconditioned Lanczos type method

A Lanczos type method for solving a linear system of equations is a method which is based essentially on the bi-orthogonalisation method of Lanczos finding the characteristic polynomial of a matrix. Lanczos type methods are of interest for the solution of equations with large sparse matrices, because they only involve the matrix in computing matrix-vector products.

A Lanczos type method is in principle finite, however, due to roundoff, this is not at all true in practice. If the matrix satisfies certain requirements, then Lanczos type methods can be constructed which behave very much like iterative methods and have quite competitive rates of convergence. An example is the conjugate gradient method (c.g. method) for symmetric positive definite systems. For this method it has been proven that the number of iterations necessary for gaining one decimal digit is:

$$\nu_{dig} \leq \tfrac{1}{2} \sqrt{\text{cond}(A)}\, \ln(10) \qquad (4.1)$$

with cond(A) the spectral condition number of the matrix, i.e. cond(A) = λ max/λ min. (see for example [1]). For the matrix of the 5-point discretization of Poisson's equation, this is the same as for the successive overrelaxation method with optimal relaxation factor. The spectral condition number is related to the order of the differential equation and to the stepsize:

$$\text{cond}(A) = \alpha.h^{-2k} \tag{4.2}$$

for a 2k-th order operator.
In a D-dimensional domain, h is related to the total number of equations by $N = \beta h^{-D}$, resulting in:

$$\nu_{dig} = O(N^{k/D}) \tag{4.3}$$

This means for a second order elliptic equation on a domain in $\mathbb{R}^2$, that the computational complexity for a fixed accuracy is:

$$w_{f\ dig} = O(N^{1.5}), \tag{4.4}$$

since each c.g step has computational complexity O(N).

One can try to speed up the convergence of a Lanczos type method by replacing A by a suitably chosen matrix $\tilde{A}$, which has a smaller spectral condition number. This is called preconditioning. This can be done by premultiplication by some suitable matrix C:

$$Ax = b \rightarrow CAx = Cb \leftrightarrow \tilde{A}x = \tilde{b}.$$

"Suitable" means here, that the computational work for the construction of C and calculating CAx must be of the same order as the work for calculating Ax.

In section 5 we shall describe a simple but very powerful example of preconditioning, the so called incomplete LU decomposition. For the Poisson equation and a special type of preconditioning it has been proven [11], that this preconditioning method reduces the spectral condition number to its square root. Other variants also transform the spectrum of A in such way that the convergence is substantially faster then is guarantueed by (4.1).

We will now describe a Lanczos type method that is suitable for non symmetric systems. The following theorem, proven in [23], demonstrates the principle on which the algorithm is based:

Theorem 4.1 1) Let A be an N * N matrix;
2) Let P be a proper subspace of $\mathbb{R}^N$;
3) Let f be any vector in $\mathbb{R}^N$;
4) Let G_o be the Krylov space associated with f and A, i.e.:

$$G_o = \bigoplus_{n\geq 0} A^n f;$$

5) Let the sequence of spaces G_n be defined by

$$G_n = (1-\alpha A)\{G_{n-1} \cap P\}\ ,\quad n \geq 1.$$

where $\{\alpha_n\}$ is a sequence of real numbers, $\alpha_n \neq 0$.
Then $M \leq N$ exists such that

$$G_n \equiv G_M \subset P\ ,\quad \forall\ n \geq M.$$

In fact, G_n is a proper subspace of G_{n-1} for all n, until G_n becomes itself a subspace of P. In general G_M will be the null-space, unless P contains an eigenvector of A, which is not very likely.

An algorithm can be based on theorem 4.1 as follows. Let the system to be solved be given by Ax = b. If preconditioning is used it is assumed that A has already been preconditioned. A sequence $\{x_n\}$ is constructed iteratively such that $f_n = Ax_n - b \subset G_k$; we try to make k increase as fast as possible with n. Because the dimension of G_k decreases as k increases this method has been called induced dimension reduction (IDR) method by the second author [23]. When preconditioning is used the method is called PIDR method. The algorithm is defined as follows. Let x_0, $p \in \mathbb{R}^N$ be given.

Algorithm 4.1 (IDR method)

$n:=\omega_n:=0$; $f_n:=Ax_n-b$, $dg_n:=dy_n:=0$;

for n:=n+1 do

begin $s_n:=f_{n-1}+\omega_{n-1}dg_{n-1}$; $t_n:=As_n$;

if n=1 V n is even then $\tilde{\alpha}_n:=(t_n,s_n)/(t_n,t_n)$

else $\tilde{\alpha}_n:=\tilde{\alpha}_{n-1}$;

$dx_n:=\omega_{n-1}dy_{n-1}-\tilde{\alpha}_n s_n$;

$df_n:=\omega_{n-1}dg_{n-1}-\tilde{\alpha}_n t_n$;

$x_n:=x_{n-1}+dx_n$; $f_n:=f_{n-1}+df_n$;

if n is even then

begin $dg_n:=dg_{n-1}$; $dy_n:=dy_{n-1}$

end else

begin $dg_n:=df_n$; $dy_n:=dx_n$

end;

$\omega_n:=-(p,f_n)/(p,dg_n)$

end of IDR method;

The relation between $\tilde{\alpha}_n$ in algorithm 4.1 and α_n in theorem 4.1 is: $\tilde{\alpha}_{2n} = \tilde{\alpha}_{2n+1} = \tilde{\alpha}_n$.

It will be shown that algorithm 4.1 is finite. Define $P = \{x \in \mathbb{R}^N \mid (p,x) = 0\}$. It is easy to show that $f_n = (1-\tilde{\alpha}_n A)s_{n-1}$, so that theorem 4.1 is applicable:

$$s_n \in G_r \cap P \rightarrow f_n \in G_{r+1}. \tag{4.5}$$

By induction we will show:

$$f_{2n},\ f_{2n+1} \in G_n. \tag{4.6}$$

Obviously (4.6) holds for n = 0. Suppose (4.6) holds for some n > 0.

Then $s_{2n+2} \in G_n \cap P$, and with (4.5): $f_{2n+2} \in G_{n+1}$. Similarly, $f_{2n+3} \in G_{n+1}$, so that (4.6) is established for general n. From theorem 4.1 it follows that

$$f_{2n},\ f_{2n+1} \in G_M\ ,\quad n \geq M.$$

and in general $G_M = \{0\}$, so that the algorithm terminates when n = M.

Usually $p = f_0$ is a good choice (the best if A is symmetric positive definite). Because of rounding errors the algorithm is not finite in practice. The strategy for choosing $\{\alpha_n\}$ is such as to make $||(1-\alpha_n A)s_n||$ as small as possible.

The algorithm breaks down when P contains eigenvectors of A, but this is highly unlikely. More serious is the possibility of (almost) zero division when computing ω_n. For A symmetric positive definite is has been shown [23] that this does not occur, and that the following relation holds between the IDR method and the classical c.g. method:

$$f_{2n} = (1-\tilde{\alpha}_2 A)(1-\tilde{\alpha}_4 A)(1-\tilde{\alpha}_6 A)\ldots(1-\tilde{\alpha}_{2n} A)r_n \qquad (4.7)$$

with r_n the n^{th} residual of the c.q. method. In the numerical experiments reported here breakdown did not occur.

The computational complexity of one iteration with the PIDR method is easily determined as follows. Matrix * vector, scalar * vector, vector $\pm$ vector, (vector, vector) each take respectively W(A), 1, 1, 2 operations per unknown, with W(A) the number of operations per unknown for the preconditioned matrix-vector multiplication; W(A) will be further discussed in section 5. The number of unknowns is $(2^\ell+1)^2$ or $2(2^\ell+1)^2$ respectively for the convection-diffusion and the Navier-Stokes problem. One arrives at the following computational complexity per unknown for the PIDR method:

$$\begin{aligned} &\text{p.w.} = W(LU) + W(A) + 1, \\ &\text{i.w.} = W(A) + 14. \end{aligned} \qquad (4.8)$$

with p.w. the preliminary work and i.w. the work per iteration. W(LU) is the computational complexity per unknown of the incomplete LU-decomposition; this will be discussed in section 5.

An additional test problem was solved with the PIDR method, namely the Navier-Stokes problem (2.9) with $\bar{\omega}$ eliminated, i.e. $\bar{\omega}$ everywhere replaced by $h^2\Delta_i\nabla_i\psi$. This will be called the fourth order formulation. The three test cases for the PIDR method will be denoted by PIDR(1), PIDR(2) and PIDR(3), corresponding to the convection-diffusion problem, the ω-ψ formulation and the fourth order formulation of the Navier-Stokes problem respectively.

5. Approximate LU decomposition

An algorithm to decompose an N * N matrix A into the product of a lower triangular matrix L and an upper triangular matrix U is the following (in quasi-Algol):

Algorithm 5.1

$A^0 := A$;

for r:=1(1)N do

begin $a^r_{rr} := \mathrm{sqrt}(a^{r-1}_{rr})$;

for $j \geq r$ do $a^r := a^{r-1}_{rj} / a^r_{rr}$;

for $i > r$ do $a^r_{ir} := a^{r-1}_{ir} / a^r_{rr}$;

for $i > r \wedge j > r$ do

$a^r_{ij} := a^{r-1}_{ij} - a^r_{ir} a^r_{rj}$

end LU decomposition;

We have $\ell_{ij} = a^N_{ij}$, $j \leq i$ and $u_{ij} = a^N_{ij}$, $j \geq i$. This special version of LU decomposition has $\ell_{ii} = u_{ii}$. It only works as long as $a^{r-1}_{rr} > 0$.

If A is a bandmatrix, i.e. $a_{ij} = 0$ for $|i-j| > b$ (the bandwidth), then L and U share this property, which results in the following computational complexity c.c. of algorithm 5.1:

$$\text{c.c.} \cong N(2b^2+b) \tag{5.1}$$

For the difference scheme (2.6) $b = O(N^{\frac{1}{2}})$, which results in c.c. $= O(N^2)$. A disadvantage of this algorithm is that sparsity of A inside the bandwidth is not used. Approximate LU decomposition does not have this weakness. A nonzero pattern P is defined, usually consisting of the nonzero pattern of A plus perhaps only a few more entries. Elements of L and U outside P are per definition zero and need not be computed. Of course we no longer have $A = LU$. It turns out that P can be chosen such that Lanczos-type methods for $\tilde{A} \equiv L^{-1}AU^{-1}$ have a much faster convergence than for A; see for example [11,18] for the case that A is symmetric positive definite.

An imcomplete LU decomposition algorithm is the following. Note the resemblance to algorithm 5.1.

Algorithm 5.2

$A^0 := A$;

for r:=1(1)N do

begin $a^r_{rr} := \mathrm{sqrt}(a^{r-1}_{rr})$; (5.2)

for $j > r \wedge (r,j) \in P$ do $a^r_{rj} := a^{r-1}_{rj} / a^r_{rr}$; (5.3)

for $i > r \wedge (i,r) \in P$ do $a^r_{ir} := a^{r-1}_{ir} / a^r_{rr}$; (5.4)

for $(i,j) \in P \wedge i > r \wedge j > r \wedge (i,r) \in P \wedge (r,j) \in P$ do

$a^r_{ij} := a^{r-1}_{ij} - a^r_{ir} a^r_{rj}$

end incomplete LU decomposition;

In another version the neglected quantities $-a^r_{ir} a^r_{rj}$, $(i,j) \notin P$ are added to the

diagonal:

Algorithm 5.3 As algorithm 5.2, with the last for-statement replaced by:

```
for i>r ∧j>r ∧(i,r)∈P ∧(r,j)∈P do
begin quant:=-a^r_ir a^r_rj;
      if (i,j)∈P then a^r_ij:=a^{r-1}_ij+quant                    (5.5)
      else a^r_ii:=a^r_ii+quant
end;
```

For algorithm 5.3 it has been shown [11]that for the Poisson equation cond $(\tilde{A})$ = $O(\sqrt{\text{cond }(A)})$ with cond the spectral condition number.

For the MG method algorithm 2 was used. The nonzero pattern P was chosen as follows. If A corresponds to a 5-point difference molecule (as on the finest grid) then P corresponds to the following 7-point difference molecule: $\begin{matrix} & * & * \\ * & * & * \\ * & * & \end{matrix}$ If A corresponds to a 9-point molecule (as on the coarser grids) then P corresponds to the same difference molecule. For the Navier-Stokes problem there are two unknowns, and the matrix is four times as large. The matrix may be thought to consist of four equal parts, the first two related to the first equation of (2.9) and the second two to the second equation of (2.9), and the first part of each pair corresponding to ψ, the second part corresponding to $\bar{\omega}$. For each of these four parts P is the same as for the convection-diffusion equation.

For PIDR(2) algorithm 5.3 was used as preconditioning. For PIDR(3) algorithm 5.2 was used, while algorithm 5.3 did not always work, because of the occurrence of negative diagonal elements. The computational complexity of algorithm 5.3 is easily determined. Define

$$\sigma_i^+ \equiv \sum_{j>i,(i,j)\in P'} 1 \qquad (5.6)$$

i.e. σ_i^+ is the number of non-zero elements to the right of the diagonal in the i^{th} row. Let P be symmetric. The following table gives a breakdown of the computational complexity (c.c.) for the r^{th} elimination step.

statement	c.c.
(5.2)	1
(5.3)	σ_r^+
(5.4)	σ_r^+
(5.5)	$2(\sigma_r^+)^2$

Table 5.1 Computational complexity of r^{th} elimination step of algorithm 5.3.

Hence the total computational complexity of algorithm 5.3 is:

$$\text{c.c. algorithm 5.3} = \sum_{r=1}^{N} [1+2\sigma_r^+(1+\sigma_r^+)] . \tag{5.7}$$

A general formula for the c.c. of algorithm 5.2 is not easy to find. But for a given P the c.c. is easily determined; results are given in table 3.1.

The following table gives W(A) and W(LU) (which occur in (4.8)).

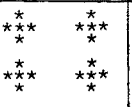
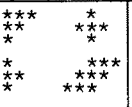
	P(A)	P(LU)	alg.	W(LU)	W(A)	p.w.	i.w.
1			5.3	41	28	70	42
2			5.3	106	48	155	62
3			5.2	201	72	274	86

Table 5.2 W(LU), W(A) p.w. and i.w. for PIDR(α).

Table 5.2 gives the c.c. per unknown of the construction of L and U, a preconditioned matrix-vector multiplication, the preliminary work p.w. and the work per iteration i.w. For $\alpha = 1,2,3$ the number of unknowns is, respectively: $(2^\ell+1)^2$, $2(2^\ell+1)^2$, $(2^\ell+1)^2$.

6. Numerical experiments

For every computation, as initial iterand the gridfunction zero was chosen. The initial iterand for the inner iterations was the result of the preceding outer iteration. The inner iterations were terminated when, in the notation of eq. (3.4), $||A^\ell u^{\ell,\nu}-f^\ell|| \leq 10^{-6}||f^\ell||$, $u^{\ell,\nu}$ being the result of the ν^{th} and last inner iteration and $||.||$ the maximum norm. The rate of convergence was monitored by the quantity

$$\mu_\nu \equiv \{||A^\ell u^{\ell,\nu}-f^\ell||/||A^\ell u^{\ell,0}-f^\ell||\}^{1/\nu}. \tag{6.1}$$

It was found that μ_ν is a weak function of ν for the MG method, but varies appreciably with ν for the PIDR method (this is typical for this type of method: during a few iterations one may notice hardly any decrease in $||A^\ell u^{\ell,\nu}-f^\ell||$, and then an iteration comes along in which the residue is reduced appreciably; cf. c.g. methods). We have listed

$$\mu \equiv \{\mu_\nu \mid \nu \text{ the number of the last inner iteration}\}. \tag{6.2}$$

A reasonable estimate of the number of inner iterations needed for one extra decimal digit is: $\ln 0.1/\ln \mu$.

The rate of convergence of the outer iterations was monitored by means of

$||u^{\ell,\nu+1}-u^{\ell,\nu}||$, $u^{\ell,\nu}$ now being the result of the ν^{th} outer iteration. The Newton method (2.9) was always found to be quadratically convergent. The outer iterations (2.10) and (2.11) were not always convergent. When they converged the convergence seemed linear, with a somewhat variable value of $||u^{\ell,\nu+1}-u^{\ell,\nu}||/||u^{\ell,\nu}-u^{\ell,\nu-1}||$, with ν counting outer iterations. The quantity

$$\sigma_\nu \equiv \{||u^{\ell,\nu+1}-u^{\ell,\nu}||/||u^{\ell,1}-u^{\ell,0}||\}^{1/\nu} \tag{6.3}$$

was found to be almost independent of ν for $\nu \geq 10$ when there was convergence; therefore we have listed $\sigma \equiv \sigma_{10}$ as an indicator of the rate of convergence. This is not true for the LAD method, where we worked with $\sigma \equiv \sigma_{25}$; even $\nu = 25$ is sometimes not quite high enough for this method.

Obviously the computational complexity can be reduced by making the termination criterion dependent on the accuracy of the outer iterand, but we have not done this: our sole aim is to investigate the efficiency of the MG and PIDR methods as linear equation solvers.

For the convection-diffusion equation the following results were obtained:

ε	ℓ	2	3	4	5
1	MG	4, .030, 9.7'3	5, .052, 4.2'4	5, .061, 1.5'5	5, .063, 5.5'5
	PIDR(1)	4, .027, 6.0'3	6, .097, 2.6'4	11, .29, 1.5'5	18, .45, 9.0'5
	FP	5, .067, 2.0'3	5, .061, 1.5'4	5, .058, 6.3'4	5, .055, 2.6'5
0.1	MG	5, .032, 1.1'4	5, .055, 4.2'4	5, .054, 1.5'5	5, .054, 5.5'5
	PIDR(1)	4, .031, 6.0'3	7, .11, 2.9'4	11, .29, 1.5'5	20, .49, 9.9'5
	FP	66, .81, 2.6'4	46, .74, 1.4'5	36, .68, 4.6'5	31, .64, 1.7'6
0.01	MG	5, .043, 1.1'4	5, .040, 4.2'4	5, .062, 1.5'5	5, .052, 5.5'5
	PIDR(1)	4, .025, 6.0'3	4, .027, 1.9'4	6, .071, 9.3'4	8, .16, 4.4'5
	FP	divergent			

Table 6.1 Results for the convection-diffusion equation.

The first number of each entry is the number of iterations, the second number is μ (for MG and PIDR(1) or σ (for FP), and the third number is the computational complexity. For the fast Poisson solver the Fourier analysis cyclic reduction method (FACR) of Hockney [14,15] was chosen with 0 cyclic reductions. The computational complexity for one application of this fast Poisson solver is:

$$(2^\ell+1)^2(10+14.42\ \ln(2^\ell+1)) \tag{6.4}$$

(in the computations this fast Poisson solver was not actually used, but the MG or

PIDR(1) method, but (6.4) was used for the determination of the computational complexity). For the FP method only 10 iterations were carried out in order to determine σ; the listed number of iterations (say, n) is determined from $\sigma^n < 10^{-6}$.

As $\varepsilon \downarrow 0$ the system (2.6) tends to a triangular matrix, for which the approximate LU-decomposition becomes exact. This explains why the rate of convergence of the MG and especially the PIDR(1) methods becomes larger as $\varepsilon \downarrow 0$. This would not happen for (2.1) in general.

Table 6.1 confirms that the rate of convergence of the MG method is independent of the mesh-size (independent of ℓ), so that the computational complexity is $O(\ell 4^{\ell})$ for a precision of $O(4^{-\ell})$. This is independent of ε, so that for $\varepsilon \downarrow 0$, $\ell \to \infty$ this method is a clear winner. However, for $\ell = 5$ the advantage of MG is not yet very great, and because of the much easier programming we feel, that for moderately large problems of this type the PIDR(1) method is preferable over the MG method. For $\varepsilon = 1$ the FP method is clearly attractive, but as one would expect FP gets worse as $\varepsilon \downarrow 0$; already for $\varepsilon = 0.1$ the other methods are faster. FP may be improved by convergence acceleration techniques, but for the very small values of ε that one encounters in engineering practice it seems highly unlikely that FP can be made competitive.

The data of table 6.1 indicate an asymptotic computational complexity of PIDR(1) of $O(N^{\alpha})$ with $\alpha \in [1.25, 1.4]$ and N the number of unknowns; cf. the theoretical $O(N^{1.25})$ result for Poisson's equation, see section 7.

For the Navier-Stokes equations the following results were obtained for the MG-method:

Re	n\ℓ	2	3	4	5
10	1	5, .039, 4.7'4	5, .047, 1.7'5	5, .044, 6.3'5	5, .043, 2.4'6
	2	4, .032, 4.1'4	4, .038, 1.5'5	4, .044, 5.5'5	4, .045, 2.1'6
	3	2, .036, 3.0'4	2, .051, 1.1'5	1, .041, 3.2'5	1, .11, 1.2'6
	4	1, .051, 2.5'4	-	-	-
50	1	5, .039, 4.7'4	5, .047, 1.7'5	5, .044, 6.3'5	5, .043, 2.4'6
	2	6, .062, 5.3'4	5, .054, 1.7'5	5, .053, 6.3'5	4, .044, 2.1'6
	3	5, .065, 4.7'4	4, .075, 1.5'5	3, .051, 4.7'5	3, .067, 1.8'6
	4	2, .11, 3.0'4	1, .079, 0.9'5	1, .12, 3.2'5	1, .084, 1.2'6
100	1	5, .039, 4.7'4	5, .047, 1.7'5	5, .044, 6.3'5	5, .043, 2.4'6
	2	7, .12, 5.8'4	7, .13, 2.2'5	5, .079, 6.3'5	5, .072, 2.4'6
	3	7, .16, 5.8'4	5, .088, 1.7'5	3, .059, 4.7'5	4, .068, 2.1'6
	4	5, .15, 4.7'4	3, .099, 1.3'5	1, .078, 3.2'5	1, .085, 1.2'6
	5	2, .23, 3.0'4	1, .11, 0.9'5	-	-

Table 6.2 Results for the MG method applied to the Navier-Stokes problem.

The symbol n is the Newton-iteration number. The first number in each entry is the number of multiple grid iterations, the second number is μ (defined by 6.2). It is clear that the rate of convergence of the MG method is independent of ℓ, so that the computational complexity is $O(\ell 4^{\ell})$ for this problem also. The comparatively large values of μ for Re = 100, ℓ = 2 are probably due to the fact that for this case the matrix is not diagonally dominant, except for the first Newton iteration. In all cases we had $||u^{\ell,\nu}-u^{\ell,\nu-1}|| < 10^{-5}$, with ν the number of the last Newton iteration.

The following table gives the results for PIDR(2): the ω-ψ formulation of the Navier-Stokes problem solved with the PIDR method.

Re	n	2	3	4	
10	1	8, .16, 3.2'4	11, .25, 1.4'5	19, .48, 7.7'5	
	2	7, .16, 2.9'4	10, .28, 1.3'5	15, .47, 6.3'5	
	3	3, .15, 1.7'4	4, .26, 6.5'4	5, .45, 2.7'5	
	4	1, .20, 1.1'4	1, .42, 3.5'4	1, .49, 1.3'5	
50	1	8, .16, 3.2'4	11, .25, 1.4'5	19, .48, 7.7'5	
	2	10, .25, 3.9'4	13, .35, 1.6'5	21, .54, 8.4'5	
	3	9, .30, 3.6'4	11, .42, 1.4'5	14, .54, 5.9'5	
	4	5, .48, 2.3'4	2, .22, 4.5'4	1, .39, 1.3'5	
100	1	8, .16, 3.2'4	11, .25, 1.4'5	19, .48, 7.7'5	42, .72, 6.0'6
	2	13, .31, 4.8'4	18, .47, 2.1'5	26, .60, 1.0'6	51, .78, 7.2'6
	3	13, .38, 4.8'4	14, .44, 1.7'5	21, .61, 8.4'5	48, .83, 6.8'6
	4	9, .42, 3.6'4	11, .51, 1.4'5	-	27, .87, 4.0'6

Table 6.3 Results for PIDR(2).

In order to save computer time the outer iterations were sometimes terminated earlier than for table 6.2. For the same reason, for ℓ = 5 only the case Re = 100 was computed. The same holds for the following table, which gives the results for PIDR(3).

Apparently PIDR(3) is roughly a factor 2 fasterthan PIDR(2). For large problems MG (in this case $\ell > 4$) is faster than PIDR, as is to be expected. Just as in the case of the convection-diffusion problem we conclude that for medium-sized problems the PIDR methods seem preferable.

Re	n\ℓ	2	3	4	5
10	1	4, .01, 1.5'4	6, .083, 6.4'4	11, .28, 3.53'5	
	2	3, .008, 1.3'4	5, .077, 5.7'4	10, .30, 3.28'5	
	3	2, .007, 1.1'4	2, .035, 3.6'4	3, .17, 1.54'5	
50	1	4, .01 1.5'4	6, .083, 6.4'4	11, .28, 3.53'5	
	2	4, .025, 1.5'4	6, .104, 6.4'4	11, .30, 3.53'5	
	3	3, .022, 1.3'4	4, .059, '4	8, .33, 2.78'5	
	4	1, .029, '3	1, .012, 2.9'4	1, .05, 1.04'5	
100	1	4, .01, 1.5'4	6, .083, 6.4'4	11, .28, 3.53'5	33, .66, 3.4'6
	2	5, .037, 1.8'4	8, .154, 7.8'4	12, .31, 3.77'5	29, .64, 3.0'6
	3	4, .023, 1.5'4	6, .117, 6.4'4	10, .35, 3.28'5	27, .70, 2.8'6
	4	2, .019, 1.1'4	3, .125, 4.3'4	6, .45, 2.28'5	8, .59, 1.0'6

Table 6.4 Results for PIDR(3)

Some results with the LAD method (defined by eq. (2.10)) are given in the following table.

Re\ℓ	2	3	4
10	.276, 21, 3.5'4	.175, 26, 1.8'5	.092, 36, 1.0'6
20	-	.170, 27, 1.8'5	-
40	-	.089, 54, 3.7'5	-

Table 6.5 Results for LAD method.

The entries are, respectively, r (see eq. (2.10)), number of iterations, computational complexity. The values of r were the best we could find.
Each iteration takes two applications of the fast Poisson solver. We have measured $\sigma \equiv \sigma_{25}$ as defined in (6.3), and the number of iterations ν in table 6.5 follows from:

$$\sigma^{\nu}||u^{\ell,1}-u^{\ell,0}|| \leq 10^{-5}/(1-\sigma)$$

which more or less guarantees $||u^{\ell,\nu}-u^{\ell}|| \leq 10^{-5}$. This makes comparison with the MG results possible, because here the Newton iterations were terminated when $||u^{\ell,\nu}-u^{\ell,\nu-1}|| \leq 10^{-5}$, which for this quadratic process also guarantees that $||u^{\ell,\nu}-u^{\ell}|| \leq 10^{-5}$. With LAD, we were not able to get convergence for Re = 50 and Re = 100.

Clearly the LAD method is only competitive for Re $\lesssim$ 10. Roache [21] gives reasons to believe that LAD is more efficient than many methods currently in practical use.

The convergence difficulties for Re >> 1 are due to the ω-ψ splitting. In [28] Roache and Ellis therefore solve the fourth-order formulation by iterating with biharmonic solvers with asymptotic complexity $O(N^{3/2})$. Because no actual computational complexity is presented a direct comparison is not possible. The convergence of this method gets worse as $Re \to \infty$, which is not true for the MG and PIDR methods. We conclude that for Re >> 1 Newton-iteration combined with fast solvers for general elliptic non-self-adjoint systems, such as MG and PIDR, is preferable.

7. Discussion

There are a great many methods for solving partial differential equations by means of finite difference methods, and we have just highlighted two relatively novel ones. There is no general consensus on the relative merits of these methods, but a few global remarks can be made. In the following (non-exhaustive) table several important (classes of) methods are listed, together

Gauss elimination		$3N^2$
SOR	c.s.a.	$12.8\ N^{1.5} \ln N$
ADI/PR	c.s.a.	$75N(\ln N)^2$
FACR	c.s.a.	$(10+7.21 \ln N)N$
MG		$O(N \ln N)$
PCG	c.s.a.	$O(N^{1.25} \ln N)$
PCG	s.a.	?
PIDR		?

Table 7.1 Applicability and operations count of several solution methods

with some information about their applicability to (2.1) and their computational complexity. The region Ω is rectangular; the computational grid is assumed to have N points (in the preceding sections, $N = (2^{\ell}+1)^2$). The third column gives the computational complexity. For the iterative methods in table 7.1 it has been assumed that the iterations are terminated when the residue has been reduced by a factor 1/N, which is proportional to the discretization error. The second column gives restrictions (if any) on the method. No entry in column 2 means no restrictions;c means constant coefficients; s.a. means self-adjoint, i.e. $b_i = 0$. SOR is the successive overrelaxation method [27] with optimal ω, which is only known for special cases including the c.s.a. case. ADI/PR is the ADI method with optimal Peaceman-Rachford parameters [20], known in the c.s.a. case. FACR is the method mentioned in section 6. Strictly speaking c in eq. (2.1) should be zero, but the method can be generalized to non-zero constant c and even to the case where c and a_{ij} depend on x_1 or x_2; the computational complexity remains $O(N \ln N)$. For the MG method a proof is given in [26] that the computational

complexity is $O(N \ln N)$ for (2.1) with Ω a rectangle. Our present results indicate that also for the Navier-Stokes equations we have computational complexity = $O(N \ln N)$. This makes the MG-method the most general and for large N the most efficient method of the methods listed in table 7.1. PDG stands for the class of preconditioned conjugate gradient methods. These methods have recently seen much development, and are often very efficient, see for example the application of the ICCG-method [18] in [17]. In [11] a proof is given that the asymptotic computational complexity is $O(N^{1.25})$ for a method of PCG type applied to the Poisson equation on a square. This is not much more than $O(N \ln N)$, and practical experience will have to show whether multiple grid or PCG methods are to be preferred. PCG methods are easy to program, but for large N multiple grid methods might be significantly faster. It seems likely that in the future SOR and ADI will be superseded by PCG. Efforts are under way to develop methods similar to PCG for the non-self-adjoint case, and PIDR may be regarded as one such method; another example is to be found in [2]. At the moment a rigorous theoretical estimate of the operations count of the PIDR method is not available. Our numerical experiments indicate that the operations count is $O(N^{\alpha})$ with $\alpha \in [1.25, 1.40]$.

From our numerical experiments one may conclude that for large N (in our examples $N \sim 289$) MG is rather more efficient than PIDR. But PIDR is much easier to program, and the programming does not become more complicated when Ω has an arbitrary shape. There is every reason to believe that the multiple grid approach works for arbitrary regions, see e.g. [12], but the programming gets very complicated.

References

1. O.Axelsson: Solution of linear systems of equations: iterative methods. In: V.A. Barker (ed.): Sparse matrix techniques. Lecture Notes in Mathematics 572, 1977, Berlin etc., Springer-Verlag.
2. O. Axelsson, I. Gustafsson: A modified upwind scheme for convective transport equations and the use of a conjugate gradient method for the solution of non-symmetric systems of equations. J. Inst. Math. Appl. 23, 321-338, 1979.
3. N.S. Bakhvalov: On the convergence of a relaxation method with natural constraints on the elliptic operator. USSR Comp. Math. Math. Phys. 6 no. 5, 101-135, 1966.
4. K.E. Barrett: The numerical solution of singular-perturbation boundary-value problems. J. Mech. Appl. Math. 27, 57-68, 1974.
5. A. Brandt: Multi-level adaptive technique (MLAT) for fast numerical solution to boundary-value problems. Proc. 3rd Internat. Conf. on Numerical Methods in Fluid Mechanics, Paris, 1972. Lecture Notes in Physics 180, 82-89, Springer-Verlag 1973.
6. A. Brandt: Multi-level adaptive solutions to boundary value problems. Math. Comp. 31, 333-390, 1977.
7. J.C. Chien: A general finite difference formulation with application to the Navier-Stokes equations. Comp. Fl. 5, 15-31, 1977.
8. R.P. Fedorenko: A relaxation method for solving elliptic difference equations. USSR Comp. Math. Math. Phys. 1, 1092-1096, 1962.
9. R.P. Fedorenko: The speed of convergence of one iterative process USSR Comp. Math. Math. Phys. 4, no. 3, 227-235, 1964.

10. P.O. Frederickson: Fast approximate inversion of large sparse linear systems. Mathematics Report 7-75, 1975, Lakehead University.

11. I. Gustafsson: A class of first order factorization methods. BIT 18, 142-156, 1978.

12. W. Hackbusch: On the multi-grid method applied to difference equations. Computing 20, 291-306, 1978.

13. W. Hackbusch: On the fast solutions of nonlinear elliptic equations. Num. Math. 32, 83-95, 1979.

14. R.W. Hockney: A fast direct solution of Poisson's equation using Fourier analysis. J. Ass. Comput. Mach. 12, 95-113, 1965.

15. R.W. Hockney: The potential calculation and some applications. Methods in Comp. Phys. 9, 135, 1970.

16. A.M. Il'in: Differencing scheme for a differential equation with a small parameter affecting the highest derivative. Math. Notes Acad. Sc. USSR 6, 596-602, 1969.

17. D.S. Kershaw: The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. J. Comp. Phys. 26, 43-65, 1978.

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

19. R.A. Nicolaides: On multiple grid and related techniques for solving discrete elliptic systems. J. Comp. Phys. 19, 418-431, 1975.

20. D.W. Peaceman and H.H. Rachford Jr.: The numerical solution of parabolic and elliptic differential equations. JSIAM 3, 28-41, 1955.

21. P.J. Roache: Computational fluid dynamics. Hermosa Publishers, Albuquerque, 1972.

22. P.J. Roache: The LAD, NOS and Split NOS methods for the steady-state Navier-Stokes equations. Computers and Fluids 3, 179-196, 1975.

23. P. Sonneveld: The method of induced dimension reduction, an iterative solver for non-symmetric linear systems. Publication in preparation.

24. R.V.Southwell: Stress calculation in frameworks by the method of systematic relaxation of constraints. Proc. Roy. Soc. London A 151, 56-95, 1935.

25. E.L. Wachspress: Iterative solution of elliptic systems. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1966.

26. P. Wesseling: The rate of convergence of a multiple grid method. To appear in the Proceedings of the Biennial Conference on Numerical Analysis, Dundee, 1979, Lecture Notes in Mathematics, Springer-Verlag.

27. D.M. Young: Iterative methods for solving partial difference methods of elliptic type. Trans. Amer. Math. Soc. 76, 92-111, 1954.

28. P.J. Roache and M.A. Ellis: The BID method for the steady-state Navier-Stokes equations. Computers and Fluids 3, 305-321, 1975.