8 MULTIGRID ALGORITHMS

8.1. Introduction

The order in which the grids are visited is called the multigrid schedule. Several schedules will be discussed. All multigrid algorithms are variants of what may be called the basic multigrid algorithm. This basic algorithm is nonlinear, and contains linear multigrid as a special case.

The most elegant description of the basic multigrid algorithm is by means of a recursive formulation. FORTRAN does not allow recursion, thus we also present a non-recursive formulation. This can be done in many ways, and various flow diagrams have been presented in the literature. If, however, one constructs a structure diagram not many possibilities remain, and a well structured non-recursive algorithm containing only one goto statement results. The decision whether to go to a finer or to a coarser grid is taken in one place only.

8.2. The basic two-grid algorithm

Preliminaries

Let a sequence \( \{G^k : k = 1, 2, \ldots, K\} \) of increasingly finer grids be given. Let \( U^k \) be the set of grid functions \( G^k \rightarrow \mathbb{R} \) on \( G^k \); a grid function \( u^k \in U^k \) stands for \( m \) functions in the case where we want to solve a set of equations for \( m \) unknowns. Let there be given transfer operators \( P^k : U^{k-1} \rightarrow U^k \) (prolongation) and \( R^k : U^{k-1} \rightarrow U^k \) (restriction). Let the problem to be solved on \( G^k \) be denoted by

\[
L^k(u^k) = b^k \tag{8.2.1}
\]

The operator \( L^k \) may be linear or non-linear. Let on every grid a smoothing algorithm be defined, denoted by \( S(u, v, f, v, k) \). \( S \) changes an initial guess \( u^k \) into an improved approximation \( v^k \) with right-hand side \( f^k \) by \( \nu \) iterations with a suitable smoothing method. The use of the same symbol \( u^k \) for the sol-
The basic two-grid algorithm

The nonlinear two-grid algorithm

Let us first assume that we have only two grids $G^k$ and $G^{k-1}$. The following algorithm is a generalization of the linear two-grid algorithm discussed in Section 2.3. Let some approximations $\tilde{u}^k$ of the solution on $G^k$ be given. How $\tilde{u}^k$ may be obtained will be discussed later. The non-linear two-grid algorithm is defined as follows. Let $f^k = b^k$.

\begin{verbatim}
Subroutine TG ($\tilde{u}$, $u$, $f$, $k$)
  comment nonlinear two-grid algorithm
  begin
    (1) $S(\tilde{u}, u, f, \cdot, k)$
    (2) $r^k = f^k - L^k(u^k)$
    (3) Choose $\tilde{u}^{k-1}$, $s_{k-1}$
    (4) $f^{k-1} = L^{k-1}(\tilde{u}^{k-1}) + s_{k-1}R^{k-1}r^k$
    (5) $S(\tilde{u}, u, f, \cdot, k - 1)$
    (6) $u^k = u^k + (1/s_{k-1})P^k(u^{k-1} - \tilde{u}^{k-1})$
    (7) $S(u, u, f, \cdot, k)$
  end of TG
\end{verbatim}

A call of TG gives us one two-grid iteration. The following program performs $ntg$ two-grid iterations:

Choose $\tilde{u}^k$

$f^k = b^k$

for $i = 1$ step 1 until $ntg$ do
  TG ($\tilde{u}$, $u$, $f$, $k$)
  $\tilde{u} = u$
od

Discussion

Subroutine TG is a straightforward implementation of the basic multigrid principles discussed in Chapter 2, but there are a few subtleties involved.
We proceed with a discussion of subroutine TG. Statement (1) represents νk smoothing iterations (pre-smoothing), starting from an initial guess \( \tilde{u}^k \). In (2) the residual \( r^k \) is computed; \( r^k \) is going to steer the coarse grid correction. Because ‘short wavelength accuracy’ already achieved in \( u^k \) must not get lost, \( u^k \) is to be kept, and a correction \( \delta u^k \) (containing ‘long wavelength information’) is to be added to \( u^k \). In the non-linear case, \( r^k \) cannot be taken for the right-hand side of the problem for \( \delta u^k \); \( L(\delta u^k) = r^k \) might not even have a solution. For the same reason, \( R^k r^k \) cannot be the right-hand side for the coarse grid problem on \( G^{k-1} \). Instead, it is added in (4) to \( L^{k-1} (\tilde{u}^{k-1}) \), with \( \tilde{u}^{k-1} \) an approximation to the solution of (1) in some sense (e.g. \( P^k \tilde{u}^{k-1} = \) solution of equation (8.2.1)). Obviously, \( L^{k-1} (u^{k-1}) = L^k (\tilde{u}^{k-1}) \) has a solution, and if \( R^{k-1} r^k \) is not too large, then \( L^{k-1} (u^{k-1}) = L^k (\tilde{u}^{k-1}) + R^{k-1} r^k \) can also be solved, which is done in statement (5) (exactly or approximately).

\( R^{k-1} r^k \) will be small when \( \tilde{u}^k \) is close to the solution of equation (8.2.1), i.e. when the algorithm is close to convergence. In order to cope with situations where \( R^{k-1} r^k \) is not small enough, the parameter \( s_{k-1} \) is introduced. By choosing \( s_{k-1} \) small enough one can bring \( f^{k-1} \) arbitraly close to \( L^{k-1} (\tilde{u}^{k-1}) \). Hence, solvability of \( L^{k-1} (u^{k-1}) = f^{k-1} \) can be ensured. Furthermore, in bifurcation problems, \( u^{k-1} \) can be kept on the same branch as \( \tilde{u}^{k-1} \) by means of \( s_{k-1} \). In (6) the coarse grid correction is added to \( u^k \). Omission of the factor \( 1/s_k \) would mean that only part of the coarse grid correction is added to \( u^k \), which amounts to damping of the coarse grid correction; this would slow down convergence. Finally, statement (7) represents \( \mu_k \) smoothing iterations (post-smoothing).

### The linear two-grid algorithm

It is instructive to see what happens when \( L^k \) is linear. It is reasonable to assume that then \( L^{k-1} \) is also linear. Furthermore, let us assume that the smoothing method is linear, that is to say, statement (5) is equivalent to

\[
  u^{k-1} = \tilde{u}^{k-1} + B^{k-1}(f^{k-1} - L^{k-1} \tilde{u}^{k-1})
\]

(8.2.2)

with \( B^{k-1} \) some linear operator. With \( f^{k-1} \) from statement (4) this gives

\[
  u^{k-1} = \tilde{u}^{k-1} + s_{k-1} B^{k-1} R^{k-1} r^k
\]

(8.2.3)

Statement (6) gives

\[
  u^k = u^k + P^k B^{k-1} R^{k-1} r^k
\]

(8.2.4)

and we see that the coarse grid correction \( P^k B^{k-1} R^{k-1} r^k \) is independent of the choice of \( s_{k-1} \) and \( \tilde{u}^{k-1} \) in the linear case. Hence, we may as well choose
sk−1 = 1 and ůk−1 = 0 in the linear case. This gives us the following linear two-grid algorithm.

**Subroutine LTG (ū, u, f, k)**

*comment* linear two-grid algorithm

*begin*

\[ S(ū, u, f, v, k) \]

\[ r^k = f^k - L^k u^k \]

\[ f^{k-1} = R^{k-1} r^k \]

\[ ů^{k-1} = 0 \]

\[ S(ū, u, f, v, k - 1) \]

\[ u^k = u^k + P^k ů^{k-1} \]

\[ S(u, u, f, v, k) \]

*end of LTG*

**Choice of ůk−1 and sk−1**

There are several possibilities for the choice of ůk−1. One possibility is

\[ ů^{k-1} = \tilde{R}^{k-1} u^k \]  \hspace{1cm} (8.2.5)

where \( \tilde{R}^{k-1} \) is a restriction operator which may or may not be the same as \( R^{k-1} \).

With the choice \( sk−1 = 1 \) this gives us the first non-linear multigrid algorithm that has appeared, the FAS (full approximation storage) algorithm proposed by Brandt (1977). The more general algorithm embodied in subroutine TG, containing the parameter \( sk−1 \) and leaving the choice of ůk−1 open, has been proposed by Hackbusch (1981, 1982, 1985). In principle it is possible to keep ůk−1 fixed, provided it is sufficiently close to the solution of \( L^{k-1} (u^{k-1}) = b^{k-1} \). This decreases the cost per iteration, since \( L^{k-1} (ū^{k-1}) \) needs to be evaluated only once, but the rate of convergence may be slower than with ůh−1 defined by (5). We will not discuss this variant. Another choice of ůk−1 is provided by nested iteration, which will be discussed later.

Hackbusch (1981, 1982, 1985) gives the following guidelines for the choice of ůk−1 and the parameter sk−1. Let the non-linear equation \( L^{k-1} (u^{k-1}) = f^{k-1} \) be solvable for \( \| f^{k-1} \| < \rho_{k-1} \). Let \( \| L^{k-1} (ū^{k-1}) \| < \rho_{k-1}/2 \). Choose sk−1 such that \( \| sk−1 R^{k-1} ŋ^k \| < \rho_{k-1}/2 \), for example:

\[ sk−1 = \frac{1}{2} \rho_{k−1} / \| R^{k−1} ŋ^k \|. \]  \hspace{1cm} (8.2.6)

Then \( \| f^{k-1} \| < \rho_{k-1} \), so that the coarse grid problem has a solution.
8.3. The basic multigrid algorithm

The recursive non-linear multigrid algorithm

The basic multigrid algorithm follows from the two-grid algorithm by replacing the coarse grid solution statement (statement (5) in subroutine TG) by $\gamma_k$ multigrid iterations. This leads to

```
Subroutine MG1 ($\tilde{u}, u, f, k, \gamma$)
  comment recursive non-linear multigrid algorithm
  begin
    if ($k \text{ eq } 1$) then
      $S(\tilde{u}, u, f, \cdot, k)$
    else
      $S(\tilde{u}, u, f, \nu, k)$
      $r^k = f^k - L^k(u^k)$
      Choose $\tilde{u}^{k-1}, s_{k-1}$
      $f^{k-1} = L^{k-1}(\tilde{u}^{k-1}) + s_{k-1}R^{k-1}r^k$
      for $i = 1$ step 1 until $\gamma_k$ do
        MG1 ($\tilde{u}, u, f, k-1, \gamma$)
        $\tilde{u}^{k-1} = u^{k-1}$
      od
      $u^k = u^k + (1/s_{k-1})P^k(u^{k-1} - \tilde{u}^{k-1})$
      $S(u, u, f, \mu, k)$
    endif
  end of MG1
```

After our discussion of the two-grid algorithm, this algorithm is self-explanatory. According to our discussion of the choice of $\tilde{u}^{k-1}$ in the preceding section, statement (7) could be deleted or replaced by something else.

The following program carries out nmg multigrid iterations, starting on the finest grid $G^K$:

```
Program 1:
  Choose $\tilde{u}^K$
  $f^K = b^K$
  for $i = 1$ step 1 until nmg do
    MG1 ($\tilde{u}, u, f, K, \gamma$)
    $\tilde{u}^K = u^K$
  od
```
The recursive linear multigrid algorithm

The linear multigrid algorithm follows easily from the linear two-grid algorithm LTG:

**Subroutine LMG (\( \tilde{u}, u, f, k \))**

**comment** recursive linear multigrid algorithm

**begin**

if \( k = 1 \) then

\( S(\tilde{u}, u, f, \cdot, k) \)

else

\( S(\tilde{u}, u, f, v, k) \)

\[ r^k = f^k - L^k u^k \]

\[ f^{k-1} = R^{k-1} r^k \]

\( \tilde{u}^{k-1} = 0 \)

for \( i = 1 \) step 1 until \( \gamma_k \) do

\( LMG (\tilde{u}, u, f, k - 1) \)

\( \tilde{u}^{k-1} = u^{k-1} \)

od

\[ u^k = u^k + P^k u^{k-1} \]

\( S(u, u, f, \cdot, k) \)

endif

**end** LMG

Multigrid schedules

The order in which the grids are visited is called the multigrid schedule or multigrid cycle. If the parameters \( \gamma_k, k = 1, 2, \ldots, K - 1 \) are fixed in advance we have a fixed schedule; if \( \gamma_k \) depends on intermediate computational results we have an adaptive schedule. Figure 8.3.1 shows the order in which the grids are visited with \( \gamma_k = 1 \) and \( \gamma_k = 2, k = 1, 2, \ldots, K - 1 \), in the case \( K = 4 \). A dot represents a smoothing operation. Because of the shape of these diagrams, these schedules are called the V-, W- and sawtooth cycles, respectively. The sawtooth cycle is a special case of the V-cycle, in which smoothing before coarse grid correction (pre-smoothing) is deleted. A schedule intermediate between these two cycles is the F-cycle. In this cycle coarse grid correction takes place by means of one F-cycle followed by one V-cycle. Figure 8.3.2 gives a diagram for the F-cycle, with \( K = 5 \).
Recursive algorithm for V-, F- and W-cycle

A version of subroutine MG1 for the V-, W- and F-cycles is as follows. The parameter $\gamma$ is now an integer instead of an integer array.

*Subroutine* MG2 ($\bar{u}, u, f, k, \gamma$)

*comment* nonlinear multigrid algorithm V-, W- or F-cycle

*begin*

*if* ($k \text{ eq } 1$) *then*

$S(\bar{u}, u, f, \cdot, k)$

*if* (cycle eq $F$) *then* $\gamma = 1$ *endif*

*else*

*A*

*for* i = 1 *step* 1 *until* $\gamma$ *do*

MG2 ($\bar{u}, u, f, k - 1, \gamma$)

$\bar{u}^{k-1} = u^{k-1}$

*od*

*B*

*if* ($k \text{ eq } K \text{ and cycle eq } F$) *then* $\gamma = 2$ *endif*

*endif*

*end MG2*
The basic multigrid algorithm

Here \( A \) and \( B \) represent statements (2) to (5) and (8) and (9) in subroutine MG1. The following program carries out \( nmg \) V-, W- or F-cycles.

Program 2:

Choose \( \tilde{u}^K \)

\[
f^K = b^K
\]

if (cycle eq W or cycle eq F) then \( \gamma = 2 \) else \( \gamma = 1 \)

for \( i = 1 \) step 1 until \( nmg \) do

MG2 (\( \tilde{u}, u, f, K, \gamma \))

\( \tilde{u}^K = u^K \)

od

Adaptive schedule

An example of an adaptive strategy is the following. Suppose we do not carry out a fixed number of multigrid iterations on level \( G^k \), but wish to continue to carry out multigrid interactions, until the problem on \( G^k \) is solved to within a specified accuracy. Let the accuracy requirement be

\[
\| L^k(u^k) - f^k \| \leq \varepsilon_k = \delta s_k \| L^{k+1}(u^{k+1}) - f^{k+1} \|
\]  

(8.3.1)

with \( \delta \in (0, 1) \) a parameter.

At first sight, a more natural definition of \( \varepsilon_k \) would seem to be \( \varepsilon_k = \delta \| f^k \| \). Since \( f^k \) does not, however, go to zero on convergence, this would lead to skipping of coarse grid correction when \( u^{k+1} \) approaches convergence. Analysis of the linear case leads naturally to condition (8.3.1). An adaptive multigrid schedule with criterion (8.3.1) is implemented in the following algorithm. In order to make the algorithm finite, the maximum number of multigrid iterations allowed is \( \gamma \).

Subroutine MG3 (\( \tilde{u}, u, f, k \))

comment recursive nonlinear multigrid algorithm with adaptive schedule

begin

if \( (k \text{ eq 1}) \) then

\[ S(\tilde{u}, u, f, \cdot, k) \]

else

\[ A \]

(1)

\[
t_{k-1} = \| r^k \| - \varepsilon_k
\]

\[
\varepsilon_{k-1} = \delta s_{k-1} \| r^k \|
\]

\[ n_{k-1} = \gamma \]
while \((t_{k-1} > 0 \text{ and } n_{k-1} \geq 0)\)
\[
\text{MG3 } (\tilde{u}, u, f, k - 1) \\quad \tilde{u}^{k-1} = u^{k-1} \\
n_{k-1} = n_{k-1} - 1 \\
t_{k-1} = \| L^{-1}(\tilde{u}^{k-1}) - f^{k-1} \| - \varepsilon_{k-1}
\]
\]
end
endif
end

MG3

Here \(A\) and \(B\) stand for the same groups of statements as in subroutine \(MG2\). The purpose of statement (1) is to allow the possibility that the required accuracy is already reached by pre-smoothing on \(G^k\), so that coarse grid correction can be skipped. The following program solves the problem on \(G^k\) within a specified tolerance, using the adaptive subroutine \(MG3\):

**Program 3:**

Choose \(\tilde{u}^k\)

\[
f^k = b^k; \quad \varepsilon^k = \text{tol} \ast \| b^k\|; \quad t^k = \| L^k(\tilde{u}^k) - b^k \| - \varepsilon^k
\]

\(n = \text{nmg}\)

while \((t^k > 0 \text{ and } n \geq 0)\) do
\[
\text{MG3}(\tilde{u}, u, f, K) \\quad \tilde{u}^K = u^K \\
n = n - 1 \\
t^K = \| L^K(\tilde{u}^K) - b^K \| - \varepsilon^K
\]
\]
end

The number of iterations is limited by \(\text{nmg}\).

**Storage requirements**

Let the finest grid \(G^K\) be either of the vertex-centred type given by (5.1.1) or of the cell-centred type given by (5.1.2). Let in both cases \(n_a = n_a^{(K)} = m_a \cdot 2^K\). Let the coarse grids \(G^k, k = K - 1, K - 2, \ldots, 1\) be constructed by successive doubling of the mesh-sizes \(h_a\) (standard coarsening). Hence, the number of grid-points \(N_k\) of \(G^k\) is

\[
N_k = \prod_{\alpha=1}^{d} (1 + m_\alpha \cdot 2^k) = M2^{kd}
\] (8.3.2)
The basic multigrid algorithm

in the vertex-centred case, with

\[ M = \prod_{\alpha=1}^{d} m_{\alpha}, \]

and

\[ N_k = M2^{kd} \quad (8.3.3) \]

in the cell-centred case. In order to be able to solve efficiently on the coarsest grid \( G^1 \) it is desirable that \( m_{\alpha} \) is small. Henceforth, we will not distinguish between the vertex-centred and the cell-centred case, and assume that \( N_k \) is given by (8.3.3).

It is to be expected that the amount of storage required for the computations that take place on \( G^k \) is given by \( c_1 N_k \), with \( c_1 \) some constant independent of \( k \). Then the total amount of storage required is given by

\[ c_1 \sum_{k=1}^{K} N_k \approx \frac{2^d}{2^d - 1} c_1 N_K \quad (8.3.4) \]

Hence, as compared to single grid solutions on \( G^K \) with the smoothing method selected, the use of multigrid increases the storage required by a factor of \( 2^d/(2^d - 1) \), which is 4/3 in two and 8/7 in three dimensions, so that the additional storage requirement posed by multigrid seems modest.

Next, suppose that semi-coarsening (cf. Section 7.3) is used for the construction of the coarse grids \( G^k, k < K \). Assume that in one coordinate direction the mesh-size is the same on all grids. Then

\[ N_k = M2^{K+k(d-1)} \quad (8.3.5) \]

and the total amount of storage required is given by

\[ c_1 \sum_{k=1}^{K} N_k \approx \frac{2^{d-1}}{2^{d-1} - 1} c_1 N_K \quad (8.3.6) \]

Now the total amount of storage required by multigrid compared with single grid solution on \( G^K \) increases by a factor 2 in two and 4/3 in three dimensions. Hence, in two dimensions the storage cost associated with semi-coarsening multigrid is not negligible.

Computational work

We will estimate the computational work of one iteration with the fixed schedule algorithm MG2. A close approximation of the computational work \( w_k \) to be performed on \( G_k \) will be \( w_k = c_2 N_k \), assuming the number of pre- and post-smoothings \( v_k \) and \( \mu_k \) are independent of \( k \), and that the operators
Multigrid algorithms

$L^k$ are of similar complexity (for example, in the linear case, $L^k$ are matrices of equal sparsity). More precisely, let us define $w_k$ to be all computing work involved in MG2 $(u, u, f, k)$, except the recursive call of MG2. Let $W_k$ be all work involved in MG2 $(u, u, f, k)$. Let $\gamma_k = \gamma, k = 2, 3, \ldots, K - 1$, in subroutine MG2 (e.g., the V- or W-cycles). Assume standard coarsening. Then

$$W_k = c_2 M^2 k^d + \gamma W_{k-1} \quad (8.3.7)$$

One may write

$$W_K = c_2 M^2 k^d (1 + \gamma (2^{-d} + \gamma (2^{-2d} + \ldots + \gamma 2^{(1-k)d})) \ldots)) = c_2 N_K (1 + \tilde{\gamma} + \tilde{\gamma}^2 + \ldots + \tilde{\gamma}^{K-1}) \quad (8.3.8)$$

with $\tilde{\gamma} = \gamma/2^d$. Here we have assumed $W_1 = c_2 M \gamma 2^d$. This may be inaccurate, since $W_1$ does not depend on $\gamma$ in reality, and, moreover, often a solution close to machine accuracy is required on $G^1$, for example when the problem is singular (e.g. with Neumann boundary conditions.) Since $W_1$ is small anyway, this inaccuracy is, however, of no consequence. From (8.3.8) it follows that

$$\tilde{W}_K = (1 - \tilde{\gamma}^K)/(1 - \tilde{\gamma}), \quad \tilde{\gamma} \neq 1$$

$$\tilde{W}_K = K, \quad \tilde{\gamma} = 1 \quad (8.3.9)$$

where $\tilde{W}_K = W_K/(c_2 N_K)$. If $\tilde{\gamma} < 1$ one may write

$$\tilde{W}_K < \tilde{W} = 1/(1 - \tilde{\gamma}) \quad (8.3.10)$$

The following conclusions may be drawn from (8.3.8), (8.3.9) and (8.3.10). $\tilde{W}_K$ is the ratio of multigrid work and work on the finest grid. The bulk of the work on the finest grid usually consists of smoothing. Hence, $\tilde{W}_K - 1$ is a measure of the additional work required to accelerate smoothing on the finest grid $G^K$ by means of multigrid.

If $\tilde{\gamma} \geq 1$ the work $W_K$ is superlinear in the number of unknowns $N_K$, because from (8.3.8) it follows that

$$W_K > (c_2 N_K/\tilde{\gamma}) \tilde{\gamma}^K = (c_2 N_K/\tilde{\gamma})(N_K/M)^{\ln \tilde{\gamma}/d \ln 2} \quad (8.3.11)$$

Hence, if $\tilde{\gamma} > 1$ $W_K$ is superlinear in $N_K$. If $\tilde{\gamma} = 1$ equation (8.3.8) gives

$$W_K = c_2 N_K K = c_2 N_K \ln (N_K/M)/d \ln 2 \quad (8.3.12)$$

again showing superlinearity of $W_K$. If $\tilde{\gamma} < 1$ equation (8.3.10) gives

$$W_K < c_2 N_K/(1 - \tilde{\gamma}) \quad (8.3.13)$$
so that $W_K$ is linear in $N_K$. It is furthermore significant that the constant of proportionality $c_2/(1 - \tilde{\gamma})$ is small. This because $c_2$ is just a little greater than the work per grid point of the smoothing method, which is supposed to be a simple iterative method (if not, multigrid is not applied in an appropriate way). Since an (perhaps the main) attractive feature of multigrid is the possibility to realize linear computational complexity with small constant of proportionality, one chooses $\tilde{\gamma} < 1$, or $\gamma < 2^d$. In practice it is usually found that $\gamma > 2$ does not result in significantly faster convergence. The rapid growth of $W_K$ with $\gamma$ means that it is advantageous to choose $\gamma \leq 2$, which is why the V- and W-cycles are widely used.

The computational cost of the F-cycle may be estimated as follows. In Figure 8.3.3 the diagram of the F-cycle has been redrawn, distinguishing between the work that is done on $G^k$ preceding coarse grid correction (pre-work, statements $A$ in subroutine MG2) and after coarse grid correction (post-work, statements $B$ in subroutine MG2). The amount of pre- and post-work together is $c_2 M 2^{kd}$, as before. It follows from the diagram, that on $G^k$ the cost of pre- and post-work is incurred $j_k$ times, with $j_k = K - k + 1$, $k = 2, 3, \ldots, K$, and $j_1 = K - 1$. For convenience we redefine $j_1 = K$, bearing our earlier remarks on the inaccuracy and unimportance of the estimate of the work on $G^1$ in mind. One obtains

$$W_K = c_2 M \sum_{k=1}^{K} (K - k + 1) 2^{kd}$$

(8.3.14)

We have

$$\sum_{k=1}^{K} k 2^{kd} = \frac{2^{(K+1)d}}{(2^d - 1)^2} [K(2^d - 1) - 1] + \frac{2^d}{(2^d - 1)^2}$$

(8.3.15)

as is checked easily. It follows that

$$W_K = c_2 M (2^{d(K+2)} + K + 1 - K 2^d)/(2^d - 1)^2$$

Figure 8.3.3  F-cycle (○ pre-work, ● post-work).
Table 8.3.1. Values of $\tilde{W}$, standard coarsening

<table>
<thead>
<tr>
<th>$d$</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V-cycle</td>
<td>$4/3$</td>
<td>$8/7$</td>
</tr>
<tr>
<td>F-cycle</td>
<td>$16/9$</td>
<td>$64/49$</td>
</tr>
<tr>
<td>W-cycle</td>
<td>$2$</td>
<td>$4/3$</td>
</tr>
<tr>
<td>$\gamma = 3$</td>
<td>$4$</td>
<td>$8/5$</td>
</tr>
</tbody>
</table>

so that

$$\tilde{W}_K < \tilde{W} = 1/(1 - 2^{-d})^2 \quad (8.3.16)$$

Table 8.3.1 gives $\tilde{W}$ as given by (8.3.10) and (8.3.16) for a number of cases. The ratio of multigrid over single grid work is seen to be not large, especially in three dimensions. The F-cycle is not much cheaper than the W-cycle. In three dimensions the cost of the V-, F- and W-cycles is almost the same.

Suppose next that semi-coarsening is used. Assume that in one coordinate direction the mesh-size is the same on all grids. The number of grid-points $N_k$ of $G^k$ is given by (8.3.5). With $\gamma_k = \gamma$, $k = 2, 3, \ldots, K - 1$ we obtain

$$W_k = c_2 M 2^{K+k(d-1)} + \gamma W_{k-1} \quad (8.3.17)$$

Hence $W_K$ is given by (8.3.8) and $\tilde{W}$ by (8.3.10) with $\tilde{\gamma} = \gamma/2^{d-1}$. For the F-cycle we obtain

$$W_K = c_2 M 2^K \sum_{k=1}^{K} (K-k+1)2^{k(d-1)} \quad (8.3.18)$$

Hence

$$W_K < \tilde{W} = 1/(1 - 2^{1-d})^2$$

Table 8.3.2. Values of $\tilde{W}$, semi-coarsening

<table>
<thead>
<tr>
<th>$d$</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V-cycle</td>
<td>2</td>
<td>4/3</td>
</tr>
<tr>
<td>F-cycle</td>
<td>4</td>
<td>16/9</td>
</tr>
<tr>
<td>W-cycle</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>$\gamma = 3$</td>
<td>-</td>
<td>4</td>
</tr>
</tbody>
</table>
Table 8.3.2 gives $\tilde{W}$ for a number of cases. In two dimensions $\gamma = 2$ or 3 is not useful, because $\tilde{\gamma} \geq 1$. It may happen that the rate of convergence of the V-cycle is not independent of the mesh-size, for example if a singular perturbation problem is being solved (e.g. convection–diffusion problem with $\varepsilon \ll 1$), or when the solution contains singularities. With the W-cycle we have $\tilde{\gamma} = 1$ with semi-coarsening, hence $\tilde{W}_k = K$. In practice, $K$ is usually not greater than 6 or 7, so that the W-cycle is still affordable. The F-cycle may be more efficient.

**Work units**

The ideal computing method to approximate the behaviour of a given physical problem involves an amount of computing work that is proportional to the number and size of the physical changes that are modeled. This has been put forward as the 'golden rule of computation' by Brandt (1982). As has been emphasized by Brandt in a number of publications, e.g. Brandt (1977, 1977a, 1980, 1982), this involves not only the choice of methods to solve (8.2.1), but also the choice of the mathematical model and its discretization. The discretization and solution processes should be intertwined, leading to *adaptive discretization*. We shall not discuss adaptive methods here, but regard (8.2.1) as given. A practical measure of the minimum computing work to solve (8.2.1) is as follows. Let us define one *work unit* (WU) as the amount of computing work required to evaluate the residual $L^K(u^K) - b^K$ of Equation (8.2.1) on the finest grid $G^K$. Then it is to be expected that (8.2.1) cannot be solved at a cost less than a few WU, and one should be content if this is realized. Many publications show that this goal can indeed be achieved with multigrid for significant physical problems, for example in computational fluid dynamics. In practice the work involved in smoothing is by far the dominant part of the total work. One may, therefore, also define one work unit, following Brandt (1977), as the work involved in one smoothing iteration on the finest grid $G^K$. This agrees more or less with the first definition only if the smoothing algorithm is simple and cheap. As was already mentioned, if this is not the case multigrid is not applied in an appropriate way. One smoothing iteration on $G^K$ then adds $2^d(k-K)$ WU to the total work. It is a good habit, followed by many authors, to publish convergence histories in terms of work units. This facilitates comparisons between methods, and helps in developing and improving multigrid codes.

**8.4. Nested iteration**

**The algorithm**

*Nested iteration*, also called *full multigrid* (FMG, Brandt (1980, 1982)) is based on the following idea. When no *a priori* information about the solution
is available to assist in the choice of the initial guess $\tilde{u}^K$ on the finest grid $G^K$, it is obviously wasteful to start the computation on the finest grid, as is done by subroutines MG$i$, $i = 1, 2, 3$ of the preceding section. With an unfortunate choice of $\tilde{u}^K$, the algorithm might even diverge for a nonlinear problem. Computing on the coarse grids is so much cheaper, thus it is better to use the coarse grids to provide an informed guess for $\tilde{u}^K$. At the same time, this gives us a choice for $\tilde{u}^k, k < K$. Nested iteration is defined by the following algorithm.

```
Program 1
  comment nested iteration algorithm
  Choose $\tilde{u}^1$
  (1) $S(\tilde{u}, \tilde{u}, f, \cdot, 1)$
  for k = 2 step 1 until K do
    (2) $\tilde{u}^k = \tilde{P}^k \tilde{u}^{k-1}$
        for i = 1 step 1 until $\gamma_k$ do
          (3) MG ($\tilde{u}, u, f, k$)
          (4) $\tilde{u}^k = u^k$
        od
  od
```

Of course, the value of $\gamma_k$ inside MG may be different from $\tilde{\gamma}_k$.

**Choice of prolongation operator**

The prolongation operator $\tilde{P}^k$ does not need to be identical to $P^k$. In fact, there may be good reason to choose it differently. As will be discussed in Section 8.6, it is often advisable to choose $\tilde{P}^k$ such that

$$m\bar{p} > m_c$$

(8.4.1)

where $m\bar{p}$ is the order of the prolongation operator as defined in Section 5.3, and $m_c$ is the order of consistency of the discretizations $L^K$, here assumed to be the same on all grids. Often $m_c = 2$ (second-order schemes). Then (8.4.1) implies that $\tilde{P}^k$ is exact for second-order polynomials.

Note that nested iteration provides $\tilde{u}^k$; this is an alternative to (8.2.5).

As will be discussed in the next section, if MG converges well then the nested iteration algorithm results in a $u^K$ which differs from the solution of (8.2.1) by an amount of the order of the truncation error. If one desires, the accuracy of $u^K$ may be improved further by following the nested iteration algorithm with a few more multigrid iterations.
Nested iteration

Computational cost of nested iteration

Let $\tilde{\gamma}_k = \hat{\gamma}$, $k = 2, 3, \ldots, K$, in the nested iteration algorithm, let $W_k$ be the work involved in MG $(\tilde{u}, u, f, k)$, and assume for simplicity that the (negligible) work on $G^1$ equals $W_1$. Then the computational work $W_{ni}$ of the nested iteration algorithm, neglecting the cost of $\tilde{P}^k$, is given by

$$W_{ni} = \sum_{k=1}^{K} \tilde{\gamma} W_k$$  \hspace{1cm} (8.4.2)

Assume inside MG $\gamma_k = \gamma$, $k = 2, 3, \ldots, K$ and let $\tilde{\gamma} = \gamma/2^d < 1$. Note that $\gamma$ and $\tilde{\gamma}$ may be different. Then it follows from (8.3.10) that

$$W_{ni} < \tilde{\gamma} \frac{c_2}{1 - \tilde{\gamma}} \sum_{k=1}^{K} N_k \approx \frac{c_2 \tilde{\gamma}}{(1 - \tilde{\gamma})(1 - 2^{-d})} N_K$$  \hspace{1cm} (8.4.3)

Defining a work unit as $1 \text{WU} = c_2 N_K$, i.e. approximately the work of $(\nu + \mu)$ smoothing iterations on the finest grid, the cost of a nested iteration is

$$W_{ni} = \tilde{\gamma} / [(1 - \tilde{\gamma})(1 - 2^{-d})] \text{ WU}$$  \hspace{1cm} (8.4.4)

Table 8.4.1 gives the number of work units required for nested iteration for a number of cases. The cost of nested iteration is seen to be just a few work units. Hence the fundamental property, which makes multigrid methods so attractive: multigrid methods can solve many problems to within truncation error at a cost of $cN$ arithmetic operations. Here $N$ is the number of unknowns, and $c$ is a constant which depends on the problem and on the multigrid method (choice of smoothing method and of the parameters $\nu_k, \mu_k, \gamma_k$). If the cost of the residual $b^K - L^K (u^K)$ is $dN$, then $c$ need not be larger than a small multiple of $d$. Other numerical methods for elliptic equations require $O(N^\alpha)$ operations with $\alpha > 1$, achieving $O(N \ln N)$ only in special cases (e.g. separable equations). A class of methods which is competitive with multigrid for linear problems in practice are preconditioned conjugate gradient methods. Practice and theory (for special cases) indicate that these require $O(N^\alpha)$ operations, with $\alpha = 5/4$ in two and $\alpha = 9/8$ in three dimensions. Comparisons will be given later.

<table>
<thead>
<tr>
<th>$d$</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>16/9</td>
<td>64/49</td>
</tr>
<tr>
<td>1</td>
<td>8/3</td>
<td>48/21</td>
</tr>
</tbody>
</table>
8.5. Rate of convergence of the multigrid algorithm

Preliminaries

For a full treatment of multigrid convergence theory, see Hackbusch (1985). See also Mandel et al. (1987). Here only an elementary introduction is presented, following the framework developed by Hackbusch (1985).

The problem to be solved

\[ L^K u^K = b^K \]  

is assumed to be linear. Two-grid convergence theory has been discussed in Section 6.5. We will extend this to multiple grids. \( \| \cdot \| \) will denote the Euclidean norm.

The smoothing and approximation properties

The smoothing method is assumed to be linear and of the type discussed in Section 4.1, with iteration matrix \( S^k \) on grid \( G^k \), \( k = 2, 3, \ldots, K \). It is assumed that on \( G^1 \) exact solution takes place. The smoothing and approximation properties are defined as follows, cf. Definitions 6.5.1 and 6.5.2.

Definition 8.5.1. Smoothing property. \( S^k \) has the smoothing property if there exist a constant \( C_S \) and a function \( \eta(\nu) \) independent of \( h_k \) such that

\[ \| L^K (S^K)^\nu \| \leq C_S h_k^{2m} \eta(\nu), \quad \eta(\nu) \to 0 \text{ for } \nu \to \infty \]  

(8.5.2)

where \( 2m \) is the order of the partial differential equation to be solved.

Definition 8.5.2. Approximation property. The approximation property holds if there exists a constant \( C_A \) independent of \( h_k \) such that

\[ \| (L^K)^{-1} - P^k (L^{k-1})^{-1} R^{k-1} \| \leq C_A h_k^{2m} \]  

(8.5.3)

where \( 2m \) is the order of the differential equation to be solved.

The multigrid iteration matrix

The multigrid algorithm is defined by subroutine LMG of Section 8.3. Let \( \nu_k = \nu, \mu_k = \mu \) and \( \gamma_k = \gamma \) be independent of \( k \). The error \( e^K \) is defined as \( e^K = u^K - (L^K)^{-1} f^K \). The error \( e^K_0 \) and \( e^K_1 \) before and after execution of LMG \( (\tilde{u}, u, f, k) \) satisfies

\[ e^K_1 = Q^K(\nu, \mu) e^K_0 \]  

(8.5.4)
with $Q^k$ the $k$-grid iteration matrix. $Q^k$ is given by:

**Theorem 8.5.1.** The iteration matrix $Q^k(\mu, \nu)$ of LMG $(\tilde{u}, u, f, k)$ satisfies

$$Q^2(\mu, \nu) = \tilde{Q}^2(\mu, \nu) \quad (8.5.5a)$$

$$Q^k(\mu, \nu) = \tilde{Q}^k(\mu, \nu) + (S^k)^\nu P^k(\mathbf{Q}^{k-1})^\gamma (L^{k-1})^{-1} R^{k-1} L^k (S^k)^\nu \quad (8.5.5b)$$

where

$$\tilde{Q}^k(\mu, \nu) = (S^k)^\nu (I - P^k (L^{k-1})^{-1} R^{k-1} L^k) (S^k)^\nu$$

is the iteration matrix of method LTG of Section 8.2.

**Proof.** From (6.5.11) it follows that $\tilde{Q}^k(\mu, \nu)$ is the iteration matrix of LTG $(\tilde{u}, u, f, k)$. Equation (8.5.5a) is obviously true. Equation (8.5.5b) is proved by induction. Let $e_0^{k+1}$, $e_{1/3}^{k+1}$, $e_2^{k+1}$ and $e_1^{k+1}$ be the error on $G^{k+1}$ before LMG $(\tilde{u}, u, f, k)$, after pre-smoothing, after coarse grid correction and after post-smoothing, respectively. We have

$$e_{1/3}^{k+1} = (S^{k+1})^\nu e_0^{k+1} \quad (8.5.6)$$

The coarse grid problem to be solved is

$$L^k u^k = -R^k L^{k+1} e_{1/3}^{k+1} \quad (8.5.7)$$

with initial guess $u^k = 0$. Hence the initial error $e_0^k$ equals minus the exact solution on $G^k$, i.e. $e_0^k = (L^k)^{-1} R^k L^{k+1} e_{1/3}^{k+1}$. After coarse grid correction the error on $G^k$ is $(Q^k)^\gamma e_0^k$. Hence the coarse grid correction is given by $[I - \gamma (Q^k)^\gamma] e_0^k$. Therefore

$$e_{2/3}^{k+1} = e_{1/3}^{k+1} + P^{k+1} [I - \gamma (Q^k)^\gamma] e_0^k \quad (8.5.8)$$

$$= [I - P^{k+1} (L^k)^{-1} R^k L^{k+1} + P^{k+1} (Q^k)^\gamma (L^k)^{-1} R^k L^{k+1}] e_{1/3}^{k+1}$$

Finally,

$$e_1^{k+1} = (S^{k+1})^\nu e_{2/3}^{k+1} \quad (8.5.9)$$

Combining (8.5.6), (8.5.8) and (8.5.9) gives (8.5.5b) with $k$ replaced by $k + 1$, which completes the proof. □

**Rate of convergence**

We will prove that the rate of convergence of LMG is independent of the mesh-size only for $\mu = 0$ (no post-smoothing). For the more general case, which is slightly more complicated, we refer to Hackbusch (1985).
Lemma 8.5.1. Let the smoothing property hold, and assume that there exists a constant $c_p$ independent of $k$ such that

$$\| P^k u^{k-1} \| \geq c_p^{-1} \| u^{k-1} \|, \quad \forall u^{k-1} \tag{8.5.10}$$

Then

$$\| (L^{k-1})^{-1} R^{k-1} L^k (S^k)^r \| \leq c_p (1 + \| \bar{Q}^k(0, \nu) \|) \tag{8.5.11}$$

Proof. It has been shown in Theorem 6.5.2 that, if $S^k$ has the smoothing property, then the smoothing method is convergent. Hence we can choose $\nu$ such that

$$\| (S^k)^r \| < 1 \tag{8.5.12}$$

Furthermore,

$$\| (L^{k-1})^{-1} R^{k-1} L^k (S^k)^r \| \leq c_p \| P^k (L^{k-1})^{-1} R^{k-1} L^k (S^k)^r \|$$

and

$$P^k (L^{k-1})^{-1} R^{k-1} L^k (S^k)^r = (S^k)^r - \{(L^k)^{-1} - P^k (L^{k-1})^{-1} R^{k-1}\} L^k (S^k)^r = (S^k)^r - \bar{Q}^k(0, \nu).$$

Using (8.5.10) and (8.5.12), (8.5.11) follows. \(\square\)

It will be necessary to study the following recursive inequality

$$\zeta_1 \leq \zeta, \quad \zeta_k \leq \zeta + C \zeta_{k-1}, \quad k \geq 2 \tag{8.5.13}$$

For this we have the following Lemma.

Lemma 8.5.2. Assume $C_\gamma > 1$. If

$$\gamma \geq 2, \quad \zeta \leq \bar{\zeta} = \frac{1}{\gamma} (\gamma C)^{-1/(\gamma-1)} \tag{8.5.14}$$

then any solution of (8.5.13) is bounded by

$$\zeta_k \leq z < 1 \tag{8.5.15}$$

where $z$ is related to $\zeta$ by

$$\zeta = z - Cz^\gamma \tag{8.5.16}$$
and $z$ satisfies

$$z \leq \frac{\gamma}{\gamma - 1} \; \zeta$$

(8.5.17)

**Proof.** We have $\zeta_k \leq z_k$, with $z_k$ defined by

$$z_1 = \zeta, \quad z_k = \zeta + C\zeta_{k-1}$$

(8.5.18)

Since $\{z_k\}$ is monotonically increasing, we have $z_k < z$, with $z$ the smallest solution of (8.5.16). Consider $f(z) = z - Cz^\gamma$. The maximum of $f(z)$ is reached in $z = z^* = (\gamma C)^{-1/(\gamma - 1)} < 1$, and $f(z^*) = \zeta$. For $\zeta \leq \zeta^*$ Equation (8.5.16) has a solution $z \leq z^* < 1$. We have

$$\zeta = z - Cz^\gamma \geq z - \frac{1}{\gamma} z = \frac{\gamma - 1}{\gamma} z,$$

which gives (8.5.17). $\blacksquare$

**Theorem 8.5.2. Rate of convergence of linear multigrid method.** Let the smoothing and approximation properties (8.5.2) and (8.5.3) hold. Assume $\gamma \geq 2$. Let $P^k$ satisfy (8.5.10) and

$$\| P^k u^{k-1} \| \leq C_p \| u^{k-1} \|, \quad C_p \text{ independent of } k.$$  

(8.5.19)

Let $\tilde{\nu} \in (0, 1)$ be given. Then there is a number $\tilde{v}$ independent of $K$ such that the iteration matrix $Q^K(0, \nu)$ defined by Theorem 8.5.1 satisfies

$$\| Q^K(\nu, 0) \| \leq \tilde{\zeta} < 1$$

(8.5.20)

if $\nu \geq \tilde{v}$.

**Proof.** $Q^K$ is defined by the recursion (8.5.5). According to Theorem 6.5.1 we have

$$\| \bar{Q}^K(\nu, 0) \| \leq C_s C_A \eta(\nu)$$

(8.5.21)

Choose a number $\tilde{\zeta} \in (0, \tilde{\zeta})$ with $\tilde{\zeta}$ satisfying (8.5.14) and a number $\tilde{v}$ such that

$$C_s C_A \eta(\nu) < \tilde{\zeta}, \quad \nu \geq \tilde{v}$$

(8.5.22)

and that (8.5.12) is satisfied for $\nu \geq \tilde{v}$.

From (8.5.5), (8.5.12), (8.5.19) and Lemma 8.5.1 it follows that

$$\zeta_k \leq \zeta + C_p \zeta_{k-1} c_p (1 + \zeta) \leq \zeta + C \zeta_{k-1}$$

(8.5.23)
with $C = 2C_p c_p$ and $\xi_k = \|Q^k(0, \nu)\|$. The recursion (8.5.23) has been analyzed in Lemma 8.5.2. It follows that

$$\xi_k \leq \frac{\gamma}{\gamma - 1} \xi < 1, \quad k = 2, 3, \ldots, K$$

(8.5.24)

If necessary, increase $\nu$ such that $\xi < \lfloor (\gamma - 1)/\gamma \rfloor \tilde{\xi}$. \Box

This theorem works only for $\gamma \geq 2$. Hence the V- and F-cycles are not included. For self-adjoint problems, a similar theory is available for the V-cycle (Hackbusch 1985, Mandel, et al. 1987), which naturally includes the F-cycle.

The difficult part of multigrid convergence theory is to establish the smoothing and approximation properties. (See the discussion in Section 6.5.)

Convergence theory for the non-linear multigrid algorithm MG is more difficult than for LMG, of course. Hackbusch (1985) gives a global outline of a non-linear theory. A detailed analysis has to depend strongly on the nature of the problem. Reusken (1988) and Hackbusch and Reusken (1989) give a complete analysis for the following class of differential equations in two dimensions

$$-(a_{\alpha\beta} u_{\alpha})_{,\beta} + g(u) = f$$

(8.5.25)

with $g$ non-linear, $g'(t) \geq 0$, $\forall t$.

In general it is difficult to say in advance how large $\nu$ should be. Practical experience shows that quite often with $\nu = 1, 2$ or $3$ one already has $\tilde{\xi} < 0.1$, even with the V-cycle. Defining a work unit to be the cost of one smoothing on the finest grid, it follows that quite often with multigrid methods the cost of gaining a decimal digit accuracy is just a few work units, independent of the mesh-size of the finest grid.

Exercise 8.5.1. Consider the one-dimensional case, and define $P^k$ by (5.3.1). Show that (8.5.10) and (8.5.19) are satisfied with $c_p = 1$, $C_p = (3/2)^{1/2}$.

8.6. Convergence of nested iteration

For a somewhat more extensive analysis of the convergence of nested iteration, see Hackbusch (1985), on which this section leans heavily.

Preliminaries

Let the (non-linear) differential equation to be solved be denoted by

$$L(u) = b$$

(8.6.1)
and let the discrete approximation on the grids $G^k$ be denoted by

$$L^k(u^k) = b^k \quad (8.6.2)$$

Define the global discretization error $\varepsilon^k$ by

$$\varepsilon^k = u^k - \{u\}^k \quad (8.6.3)$$

where $\{u\}^k$ indicates the trivial restriction of $u$ to $G^k$:

$$\{u\}^k = u(x_i) \quad (8.6.4)$$

Let the order of the discretization error of $L^k$ be $m$, $k = 1, 2, \ldots, K$, i.e.

$$\| \varepsilon^k \| \leq C2^{mk}, \quad k = 1, 2, \ldots, K \quad (8.6.5)$$

where $\| \cdot \|$ is a norm which is not necessarily Euclidean and which we do not specify further; $m$ depends on the choice of $\| \cdot \|$. In (8.6.5) we assume that $2^{-k}$ is proportional to the step sizes on $G^k$, i.e. the step sizes on $G^k$ are obtained from those on $G^{k-1}$ by halving.

**Recursion for the error of nested iteration**

Denote the result of statement (2) in the nested iteration Algorithm (paragraph 1 of Section 8.4) by $u_0^k$, and the result of Statement (4) by $u^k$. Let $\gamma$ be an upper bound for the contraction number of the multigrid algorithm, and let $\gamma_k = \gamma$, $k = 2, 3, \ldots, K$. Then

$$\| \tilde{u}^k - u^k \| \leq \gamma \| u_0^k - u^k \| \quad (8.6.6)$$

($u^k$ is the solution of (8.6.2)). We have, for $k = 2$,

$$\| u_0^2 - u^2 \| = \| \tilde{\Phi}^2 u^1 - u^2 \| = C(2) \quad (8.6.7)$$

defining

$$C(k) = \| \tilde{\Phi}^k u^{k-1} - u^k \| \quad (8.6.8)$$

Estimates for $C(k)$ will be provided later. It follows that

$$\| \tilde{u}^2 - u^2 \| \leq C(2) \gamma$$

(8.6.9)
For general \( k \),
\[
\| u_0^k - u^k \| = \| \tilde{P}^k u^{k-1} - u^k \|
\]
\[
= \| \tilde{P}^k u^{k-1} - u^k + \tilde{P}^k (u^{k-1} - u^{k-1}) \|
\]
\[
\leq C(k) + C_p \| u^{k-1} - u^{k-1} \| \tag{8.6.10}
\]
assuming that
\[
\| \tilde{P}^k \| \leq C_p, \quad k = 2, 3, \ldots, K \tag{8.6.11}
\]
Let \( \delta_k = \| \tilde{u}^k - u^k \| \), then (8.6.6), (8.6.9) and (8.6.10) give
\[
\delta_k \leq \delta \tilde{g}(C(k) + C_p \delta_k), \quad \delta_2 \leq C(2) \delta \tilde{g} \tag{8.6.12}
\]
Hence
\[
\delta_k \leq \delta \tilde{g} \sum_{k=2}^{K} C(k)(\delta \tilde{g} C_p)^{K-k} \tag{8.6.13}
\]
We will provide estimates of \( C(k) \) of the form
\[
C(k) \leq \tilde{C}_p 2^{-pk} \tag{8.6.14}
\]
Substitution in (8.6.13) gives
\[
\delta_k \leq \delta \tilde{g} \tilde{C}_p 2^{-pk} \sum_{k=0}^{K-2} r^k, \quad r = 2^p \delta \tilde{g} C_p \tag{8.6.15}
\]
Assume
\[
r = 2^p \delta \tilde{g} C_p < 1 \tag{8.6.16}
\]
Then (8.6.15) gives the following result
\[
\| \tilde{u}^K - u^K \| \leq \delta \tilde{g} \tilde{C}_p 2^{-pk}(1 - r) \tag{8.6.17}
\]
**Accuracy of prolongation in nested iteration**

We now estimate \( C(k) \). We want to compare \( \tilde{u}^k - u^k \) with the discretization error \( e^k \). Suppose we have the following asymptotic expansion for \( e^k \)
\[
e^k = \{ e_1 \}^k 2^{-mk} + \{ e_2 \}^k, \quad \{ e_2 \}^k = o(2^{-mk}) \tag{8.6.18}
\]
This will hold if the solution of (8.6.1) is sufficiently smooth, and if (8.6.5) is satisfied. One may write

\[ \tilde{\mathbf{u}}^k u^{k-1} - u^k = \tilde{\mathbf{P}}^k \{ u \}^{k-1} - \{ u \}^k + 2^{-m(k-1)} \tilde{\mathbf{P}}^k \{ e_1 \}^{k-1} \]

\[ - 2^{-mk} \{ e_1 \}^k + o(2^{-mk}) \]  

(8.6.19)

Assume

\[ \| \tilde{\mathbf{P}}^k \{ u \}^{k-1} - \{ u \}^k \| \leq C u 2^{-pk} \]  

(8.6.20)

and

\[ \tilde{\mathbf{P}} \{ e_1 \}^{k-1} = \{ e_1 \}^k + o(1) \]  

(8.6.21)

Then it follows from (8.6.19) that

\[ \| \tilde{\mathbf{P}}^k u^{k-1} - u^k \| \leq C u 2^{-pk} + (2^m - 1) \| \{ e_1 \}^k \| 2^{-mk} + o(2^{-mk}) \]  

(8.6.22)

The inequalities (8.6.11), (8.6.20) and (8.6.21) are discussed further in Exercise 8.6.1.

Error after nested iteration

First assume

\[ m \bar{p} > m \]  

(8.6.23)

as announced in (8.4.1); \( m \bar{p} \) has been defined in Section 5.3. Then \( p > m \) in (8.6.22), cf Exercise 8.6.1. Furthermore, for reasonable norms, \( \| \{ e_1 \}^k \| \) is uniformly bounded in \( k \):

\[ \| \{ e_1 \}^k \| \leq C e = \sup \{ \| \{ e_1 \}^k \| : k = 2, 3, ..., \infty \} \]  

(8.6.24)

Then (8.6.22) may be rewritten as

\[ \| \tilde{\mathbf{P}}^k u^{k-1} - u^k \| \leq (2^m - 1) C e 2^{-mk} + o(2^{-mk}) \]  

(8.6.25)

Neglecting higher order terms, we have

\[ C(k) \leq (2^m - 1) C e 2^{-mk} \]  

(8.6.26)

Substitution in (8.6.14) gives

\[ \hat{C}_m = (2^m - 1) C e, \quad p = m \]  

(8.6.27)
so that (8.6.17) becomes

$$\| \hat{u}^K - u^K \| \leq \xi \eta (2^m - 1) C \varepsilon 2^{-mK} (1 - r)$$

(8.6.28)

Comparison with (8.6.18) and (8.6.24) gives us the following theorem, noting that (8.6.16) becomes

$$r = 2^m \xi \eta C_p < 1$$

(8.6.29)

**Theorem 8.6.1. Error after nested iteration.** If conditions (8.6.11), (8.6.18), (8.6.20), (8.6.23), (8.6.24) and (8.6.29) are fulfilled, then the error after nested iteration satisfies, neglecting higher order terms,

$$\| \hat{u}^K - u^K \| \leq D(\xi, \eta) \| e^K \|$$

(8.6.30)

where

$$D(\xi, \eta) = \xi \eta (2^m - 1) (1 - r)$$

(8.6.31)

This theorem says that after nested iteration the solution on $G^K$ is approximated within $D(\xi, \eta)$ times the discretization error. How large is $D(\xi, \eta)$? Assume $C_p < 2$, which is usually the case, cf. Exercise 8.6.1. Assume that $m = 2$ (second-order discretization). Then Condition (8.6.29) becomes

$$\xi \eta < 1/8$$

(8.6.32)

From (8.6.31) it follows with $m = 2$ that $D(\xi, \eta) \leq 1$ for $\xi \eta \leq 1/7$. Hence, if we want the error after nested iteration to be smaller than the discretization error $\eta$ should satisfy

$$\eta \geq -\ln 8/\ln \xi$$

(8.6.33)

Taking $\xi = 1/4$ as a typical value of the multigrid contraction number, $\eta = 2$ is sufficient. This shows that with nested iteration, multigrid gives the discrete solution within truncation error accuracy in a small number of work units, regardless of the mesh size.

**Less accurate prolongation**

For second-order accurate discretizations ($m = 2$) equation (8.6.23) implies that $\tilde{P}^k$ should be exact for polynomials of degree at least 2. For second-order differential equations, however, the multigrid method requires only prolongations that are exact for polynomials of degree 1 (i.e. $m_p = 2$, since usually $m_R \geq 1$, so that $m_p + m_R > 2$ is satisfied). We will now investigate the
Convergence of nested iteration

accuracy of the nested iteration result if $m = 2$. Again assuming $m = 2$, Equation (8.6.22) can be written as

$$\left\| \tilde{P}^k u^{k-1} - u^k \right\| \leq C_2 2^{-2k} + o(2^{-2k}) \quad (8.6.34)$$

so that (8.6.14) holds with $p = 2$, neglecting higher order terms. Assuming

$$r = 2^{2\tilde{r}} C_p < 1 \quad (8.6.35)$$

Equation (8.6.17) gives us the following theorem.

**Theorem 8.6.2. Error after nested iteration.** If $m = 2$ and if conditions (8.6.11), (8.6.20) (with $p = 2$) and (8.6.35) are satisfied and if $m = 2$ in (8.6.5) then the error after nested iteration satisfies, neglecting higher order terms,

$$\left\| \tilde{u}^k - u^K \right\| \leq \tilde{r} C_2 2^{-2K} (1 - r) \quad (8.6.36)$$

This theorem shows that with $m = 2$ after nested iteration the error is $O(2^{-2K})$, like the discretization error. Hence, it is also useful to apply nested iteration with $\tilde{P}^k = P^k$ (assuming $m = 2$), avoiding the use of a higher order prolongation operator. There is, however, now no guarantee that the iteration error will be smaller than the discretization error.

**Exercise 8.6.1.** Let the one-dimensional vertex-centred prolongation operator $\tilde{P}^k$ be defined by

$$[\tilde{P}^k] = \frac{1}{16} [-1 \quad 9 \quad 16 \quad 9 \quad -1] \quad (8.6.37)$$

Show that $m = 4$. Define $\| \cdot \|$ by

$$\| u^K \|^2 = 2^{-k} \sum_{j=0}^{2^k} (u(j))^2 \quad (8.6.38)$$

and show (cf. 8.6.11))

$$\| \tilde{P}^k \| \leq C_p = (41/32)^{1/2} \quad (8.6.39)$$

Show that (8.6.20) holds with

$$C u = 3\sqrt{2} \sup_{x \in 0} \left| \frac{d^4 u}{dx^4} \right|, \quad p = 4 \quad (8.6.40)$$

Show that this implies (8.6.21).
8.7. Non-recursive formulation of the basic multigrid algorithm

Structure diagram for fixed multigrid schedule

In FORTRAN, recursion is not allowed: a subroutine cannot call itself. The subroutines MG1, 2, 3 of Section 8.3 cannot, therefore, be implemented directly in FORTRAN. A non-recursive version will, therefore, be presented. At the same time, we will allow greater flexibility in the decision whether to go to a finer or to a coarser grid.

Various flow diagrams describing non-recursive multigrid algorithms have been published, for example in Brandt (1977) and Hackbusch (1985). In order to arrive at a well structured program, we begin by presenting a structure diagram. A structure diagram allows much less freedom in the design of the control structure of an algorithm than a flow diagram. We found basically only one way to represent the multigrid algorithm in a structure diagram (Wesseling 1988, 1990a). This structure diagram might, therefore, be called the canonical form of the basic multigrid algorithm. The structure diagram is given in Figure 8.7.1. This diagram is equivalent to Program 2 calling MG2 to do nmg multigrid iterations with finest grid \( G^K \) in Section 8.3. The schedule is fixed and includes the V-, W- and F-cycles. Parts A and B are specified after subroutine MG2 in Section 8.3. Care has been taken that the program also works as a single grid method for \( K = 1 \).

FORTRAN implementation of while clause

Apart from the while clause, the structure diagram of Figure 8.7.1 can be expressed directly in FORTRAN. A FORTRAN implementation of a while clause is as follows. Suppose we have the following program

\[
\text{while } (n(K) > 0) \text{ do}
\]

\[
\text{Statement 1}
\]

\[
n(K) = \cdots
\]

\[
\text{Statement 2}
\]

\[
\text{end}
\]

A FORTRAN version of this program is

\[
10 \text{ if } (n(K) > 0) \text{ then}
\]

\[
\text{Statement 1}
\]

\[
n(K) = \cdots
\]

\[
\text{Statement 2}
\]

\[
goto 10
\]

\[
\text{endif}
\]
The *goto* statement required for the FORTRAN version of the *while* clause is the only *goto* needed in the FORTRAN implementation of the structure diagram of Figure 8.7.1. This FORTRAN implementation is quite obvious, and will not be given.

**Structure diagram for adaptive multigrid schedule**

Figure 8.7.2 gives a structure diagram for a non-recursive version of Program 3 of Section 8.3, using subroutine MG3 with adaptive schedule. To ensure that the algorithm is finite, the number of iterations on $G^K$ is limited by $nmg$ and on $G^k$, $k < K$ by $\gamma$.

There is great similarity to the structure diagram for the fixed schedule. This is due to the fundamental nature of these structure diagrams. It is hard, if not impossible, to fit the algorithm into a significantly different structure diagram. The reason is that structure diagrams impose programming without *goto*. The flow diagrams of multigrid algorithms that have appeared show significant differences, even if they represent the same algorithm.

**FORTRAN subroutine**

The great similarity of the two structure diagrams means that it is easy to join them in one structure diagram. We will not do this, because this makes the basic simplicity of the algorithm less visible. Instead, we give a FORTRAN subroutine which incorporates the two structure diagrams (cf. Khalil and Wesseling 1991).

```fortran
Subroutine MG(ut,u,b,K,cycle,nmg,tol)
  Nonlinear multigrid algorithm including V-, W-, F- and adaptive cycles.
  Problem to be solved: $L(u;K) = b(K)$ on grid $G(K)$.
    character cycle
    dimension ut(.),u(.),b(.)
  ut (input): initial approximation.
  u (output): current solution.
  b (input): right-hand-side on finest grid.
  K (input): number of finest grid.
  cycle (input): V, W, F or A; A gives adaptive cycle.
  nmg (input): fixed cycle: number of iterations.
  adaptive cycle: maximum number of iterations.
  tol (input): accuracy requirement for adaptive cycle:
    dimension f(.),r(.),n,eps,t(1:K)
  f: right-hand-sides
  r: residuals
```

196  *Multigrid algorithms*
Choose $\tilde{u}^K$, tol and $\delta$;

$f^K = b^K; \ k = K; \ n_K = nmg$

$\varepsilon_K = \text{tol} \cdot \| b^K \|; \ t_K = || L^K(\tilde{u}^K) - f^K || - \varepsilon_K$

while ($t_K > 0$ and $n_K \geq 0$)

Figure 8.7.2 Structure diagram of non-recursive multigrid algorithm with adaptive schedule.
Multigrid algorithms

n: counter of coarse grid iterations
c eps: tolerances for coarse grid solutions with adaptive cycles
t: t(k) < 0 implies coarse grid convergence within tolerance

logical go on, finer
if (cycle.eq.'A') then
  tol = ...
delta = ...
  eps(K) = tol*anorm(b(K))
  t(K) = anorm(L(ut;K) - b(K)) - eps(K)
gamma = ...
endif
endif
endif
f(K) = b(K)
K = K
n(K) = nmg
if (cycle.eq.'A') then
  go on = t(K).gt.0.and.n(K).ge.0
else
  go on = n(K).ge.0
endif
if (go on) then
  finer = n(k).eq.0.or.k.eq.1
if (cycle.eq.'A') then
  finer = finer.or.t(k).le.0
endif
if (finer) then
  if (k.eq.1) then
    S(ut,u,f,..,k) then
      if (cycle.eq.'F')
        gamma = 1
      endif
    endif
  endif
if (k.eq.K) then
  if (go on) then
    if (cycle.eq.'A') then
      finer = finer.or.t(k).le.0
    endif
    if (finer) then
      if (k.eq.1) then
        S(ut,u,f,..,k) then
          if (cycle.eq.'F')
            gamma = 1
          endif
        endif
      endif
      if (k.eq.K) then
      endif
    endif
  endif
endif
if (cycle.eq.'F')
  igamma = 2
else
  c go to finer grid
  k = k + 1
  B
endif
  n(k) = n(k) - 1
  ut(k) = u(k)
  if (cycle.eq.'A') then
    t(k) = anorm(L(ut;k) - f(k)) - eps(k)
  endif
  else
    c go to coarser grid
    A
    if (cycle.eq.'A') then
      t(k - 1) = anorm(r(k)) - eps(k)
      eps(k - 1) = delta*s(k - l)*anorm(r(k))
    endif
    k = k - 1
    n(k) = igamma
  endif
  goto 10
endif
return
end

After our discussion of the structure diagrams of Figures 8.7.1 and 8.7.2 no further explanation of subroutine MG is necessary.

Testing of multigrid software

A simple way to test whether a multigrid algorithm is functioning properly is to measure the residual before and after each smoothing operation, and before and after each visit to coarser grids. If a significant reduction of the size of the residual is not found, then the relevant part of the algorithm (smoothing or coarse grid correction) is not functioning properly. For simple test problems predictions by Fourier smoothing analysis and the contraction number of the multigrid method should be correlated. If the coarse grid problem is solved exactly (a situation usually approximately realized with the W-cycle) the multigrid contraction number should usually be approximately equal to the smoothing factor.
Local smoothing

It may, however, happen that for a well designed multigrid algorithm the contraction number is significantly worse than predicted by the smoothing factor. This may be caused by the fact that Fourier smoothing analysis is locally not applicable. The cause may be a local singularity in the solution. This occurs for example when the physical domain has a reentrant corner. The coordinate mapping from the physical domain onto the computational rectangle is singular at that point. It may well be that the smoothing method does not reduce the residual sufficiently in the neighbourhood of this singularity, a fact that does not remain undetected if the testing procedures recommended above are applied. The remedy is to apply additional local smoothing in a small number of points in the neighbourhood of the singularity. This procedure is recommended by Brandt (1982, 1988, 1989) and Bai and Brandt (1987), and justified theoretically by Stevenson (1990). This local smoothing is applied only to a small number of points, thus the computing work involved is negligible.

8.8. Remarks on software

Multigrid software development can be approached in various ways, two of which will be examined here.

The first approach is to develop general building blocks and diagnostic tools, which helps users to develop their own software for particular applications without having to start from scratch. Users will, therefore, need a basic knowledge of multigrid methods. Such software tools are described by Brandt and Ophir (1984).

The second approach is to develop autonomous (black box) programs, for which the user has to specify only the problem on the finest grid. A program or subroutine may be called autonomous if it does not require any additional input from the user apart from problem specification, consisting of the linear discrete system of equations to be solved and the right-hand side. The user does not need to know anything about multigrid methods. The subroutine is perceived by the user as if it were just another linear algebra solution method. This approach is adopted by the MGD codes (Wesseling 1982, Hemker et al. 1983, 1984, Hemker and de Zeeuw 1985, Sonneveld et al. 1985, 1986), which are available in the NAG library, and by the MGCS code (de Zeeuw 1990).

Of course, it is possible to steer a middle course between the two approaches just outlined, allowing or requiring the user to specify details about the multigrid method to be used, such as offering a selection of smoothing methods, for example. Programs developed in this vein are BOXMG (Dendy 1982, 1983, 1986), the MG00 series of codes (Foerster and Witsch 1981, 1982, Stüben et al. 1984) which is available in ELLPACK (Rice and Boisvert 1985), MUDPACK (Adams 1989, 1989a), and the PLTMG code (Bank 1981, 1981a, Bank and Sherman 1981). Except for PLTMG and MGD,
the user specifies the linear differential equation to be solved and the program generates a finite difference discretization. PLTMG generates adaptive finite element discretizations of non-linear equations, and therefore has a much wider scope than the other packages. As a consequence, it is not (meant to be) a solver as fast as the other methods.

By sacrificing generality for efficiency very fast multigrid methods can be obtained for special problems, such as the Poisson or the Helmholtz equation. In MG00 this can be done by setting certain parameters. A very fast multigrid code for the Poisson equation has been developed by Barkai and Brandt (1983). This is probably the fastest two-dimensional Poisson solver in existence.

If one wants to emulate a linear algebraic systems solver, with only the fine grid matrix and right-hand side supplied by the user, then the use of coarse grid Galerkin approximation (Section 6.2) is mandatory. Coarse grid Galerkin approximation is also required if the coefficients in the differential equations are discontinuous. Coarse grid Galerkin approximation is used in MGD, MGCS and BOXMG; the last two codes use operator-dependent transfer operators and are applicable to problems with discontinuous coefficients.

In an autonomous subroutine the method cannot be adapted to the problem, so that user expertise is not required. The method must, therefore, be very robust. If one of the smoothers that were found to be robust in Chapter 7 is used, the required degree of robustness is indeed obtained for linear problems.

Non-linear problems may be solved with multigrid codes for linear problems in various ways. The problem may be linearized and solved iteratively, for example by a Newton method. This works well as long as the Jacobian of the non-linear discrete problem is non-singular. It may well happen, however, that the given continuous problem has no Fréchet derivative. In that case the condition of the Jacobian deteriorates as the grid is refined, and the Newton method does not converge rapidly or not at all. An example of this situation will be given in Section 9.4. The non-linear multigrid method can be used safely and efficiently, because the global system is not linearized. A systematic way of applying numerical software outside the class of problems to which the software is directly applicable is the defect correction approach. Auzinger and Stetter (1982) and Böhmer et al. (1984) point out how this ties in with multigrid methods.

8.9. Comparison with conjugate gradient methods

Although the scope and applicability of multigrid principles are much broader, multigrid methods can be regarded as very efficient ways to solve linear systems arising from discretization of partial differential equations. As such multigrid can be viewed as a technique to accelerate the convergence of basic iterative methods (called smoothers in the multigrid context). Another
powerful technique to accelerate basic iterative methods for linear problems that also has come to fruition relatively recently is provided by conjugate gradient and related methods. In this section we will briefly introduce these methods, and compare them with multigrid. For an introduction to conjugate gradient acceleration of iterative methods, see Hageman and Young (1981) or Golub and Van Loan (1989).

**Conjugate gradient acceleration of basic iterative methods**

Consider the basic iterative method (4.1.3). According to (4.2.2) after \(n\) iterations the residual satisfies

\[
\mathbf{r}^n = \psi_n(\mathbf{A}\mathbf{M}^{-1})\mathbf{r}^0, \quad \psi_n(x) = (1 - x)^n
\]

Until further notice it is assumed that \(\mathbf{A}\) is symmetric positive definite. Let us also assume that \(\mathbf{M}^{-1}\) is symmetric positive definite, so that we may write

\[
\mathbf{M}^{-1} = \mathbf{E}^\mathsf{T}\mathbf{E}
\]

Since for arbitrary \(\mathbf{m}\) we have

\[
(\mathbf{A}\mathbf{E}^\mathsf{T}\mathbf{E})^k = \mathbf{E}^{-1}(\mathbf{E}\mathbf{A}\mathbf{E}^\mathsf{T})^k\mathbf{E}
\]

we can rewrite (8.9.1) as

\[
\mathbf{E}\mathbf{r}^n = \psi_n(\mathbf{E}\mathbf{A}\mathbf{E}^\mathsf{T})\mathbf{E}\mathbf{r}^0
\]

Let the linear system to be solved be denoted by

\[
\mathbf{A}\mathbf{y} = \mathbf{b}
\]

The conjugate gradient method will be applied to the following preconditioned system

\[
\mathbf{E}\mathbf{A}\mathbf{E}^\mathsf{T}(\mathbf{E}^{-1}\mathbf{y}) = \mathbf{E}\mathbf{b}
\]

The conjugate gradient algorithm that will be presented below has the following fundamental property

\[
\mathbf{E}\mathbf{r}^n = \phi_n(\mathbf{E}\mathbf{A}\mathbf{E}^\mathsf{T})\mathbf{E}\mathbf{r}^0
\]

with

\[
\phi_n = \arg \min \{ \| \phi_n(\mathbf{E}\mathbf{A}\mathbf{E}^\mathsf{T})\mathbf{E}\mathbf{r}^0 \| : \phi_n \in \Pi_n \}
\]

where the norm is defined by

\[
\| \mathbf{r} \| = \mathbf{r}^\mathsf{T}\mathbf{A}^{-1}\mathbf{r}
\]
Comparison with conjugate gradient methods

and the set $\Pi_n^1$ by

$$\Pi_n^1 = \{\theta_n: \theta_n \text{ is a polynomial of degree} \leq n \text{ and } \theta(0) = 1\} \quad (8.9.10)$$

Since $\psi_n$ in (8.9.4) belongs to $\Pi_n^1$, we see that the number of iterations required is likely to be reduced by application of the conjugate gradient method.

**Preconditioned conjugate gradient algorithm**

Application of the conjugate gradient method to the preconditioned system (8.9.6) leads to the following algorithm (for a derivation see, for example, Sonneveld et al. (1985)):

Choose $y^0$

\[
p^{-1} = 0, \quad r^0 = b - Ay^0, \quad \rho_0 = r^0 E^T E r^n
\]

for $n = 1, 2, \ldots,$ do

\[
\rho_n = r^n E^T E r^n, \quad \beta_n = \rho_n / \rho_{n-1}
\]

\[
p^n = E^T E r^n + \beta_n p^{n-1}
\]

\[
\sigma_n = p^n T A p^n, \quad \alpha_n = \rho_n / \sigma_n
\]

\[
y^{n+1} = y^n + \alpha_n p^n
\]

\[
r^{n+1} = r^n - \alpha_n A p^n
\]

od

There are other variants, corresponding to other choices of the norm in (8.9.8), which need not be discussed here.

Computation of $E^T E r^n$ is equivalent to carrying out an iteration with the basic iterative method (4.1.3) that is to be accelerated. Some further work is required for $A p^n$; the rest of the work is small. A conjugate gradient iteration therefore does not involve much more work than an iteration with the basic iterative method (4.1.3).

**Rate of convergence**

The rate of convergence of conjugate gradient methods can be estimated in an elegant way, cf. Axelsson (1977). It can be shown that from the fundamental property (8.9.8) it follows that

$$\|E r^n\|^2 = \|E r^0\|^2 = \min \{\max \{\psi(\lambda)^2: \lambda \in Sp(B)\}: \psi \in \Pi_n^1\} \quad (8.9.11)$$

where $Sp(B)$ is the set of eigenvalues of $B = E A E^T$. From this it may be shown that

$$\|E r^n\| / \|E r^0\| \leq 2 \exp\{-2n \ cond_2(B)^{-1/2}\} \quad (8.9.12)$$

with $\text{cond}_2(B)$ the condition number measured in the spectral norm.
It has been shown by Meijerink and Van der Vorst (1977) that an effective preconditioning is obtained by choosing $E = L^{-1}$ with

$$LL^T = A + N$$

(8.9.13)

which is the symmetric (Choleski) variant of incomplete LU factorization. It is found that in many cases $\text{cond}_2(L^{-1}AL^{-T}) < \text{cond}_2(A)$. For a full explanation of the acceleration effect of the conjugate gradient method, not just the condition number but the eigenvalue distribution should be taken into account, cf. Van der Sluis and Van der Vorst (1986, 1987). For a special case, the five-point discretization of the Laplace equation in two dimensions, Gustafsson (1978) shows that preconditioning with modified incomplete LLT factorization results in $\text{cond}_2(L^{-1}AL^{-T}) = O(h^{-1})$, so that according to (8.9.12) the computational cost is $O(N^{5/4})$ with $N$ the number of unknowns, which comes close to the $O(N)$ of multigrid methods. Theoretical estimates of $\text{cond}_2(L^{-1}AL^{-T})$ for more general cases are lacking, whereas for multigrid $O(N)$ complexity has been established for a large class of problems. It is surprising that, although the algorithm is much simpler, the rate of convergence of conjugate gradient methods is harder to estimate theoretically than for multigrid methods. Nevertheless, the result of $O(N^{5/4})$ computational complexity (and probably $O(N^{9/8})$ in three dimensions) seems to hold approximately quite generally for conjugate gradient methods preconditioned by approximate factorization.

Conjugate gradient acceleration of multigrid

The conjugate gradient method can be used to accelerate any iterative method, including multigrid methods. Care must be taken that the preconditioned system (8.9.6) is symmetric. This is easy to achieve if the multigrid iteration matrix $Q^K(\mu, \nu)$ is symmetric. From Theorem 8.5.1 it follows that this is the case if $\nu = \mu, R^{k-1} = (P^k)^* \text{ and } S^k = (S^k)^*$, i.e. the smoother must be symmetric. These conditions are easily satisfied, and choosing $E = \{Q^K(\mu, \mu)\}^{-1}$ in the preconditioned conjugate gradient algorithm gives us conjugate gradient acceleration of multigrid. If the multigrid algorithm is well designed and fits the problem it will converge fast, making conjugate gradient acceleration superfluous or even wasteful. If multigrid does not converge fast one may try to remedy this by improving the algorithm (for example, introducing additional local smoothing near singularities, or adapting the smoother to the problem), but if this is impossible because an autonomous (black box) multigrid code is used, or difficult because one cannot identify the cause of the trouble, then conjugate gradient acceleration is an easy and often very efficient way out. The reason for the often spectacular acceleration of a weakly convergent multigrid method by conjugate gradients is as follows. In the case of deterioration of multigrid convergence, quite often only a few eigenmodes are slow to converge. This means that $Sp(B): B = (Q^K)^{-1}A(Q^K)^{-1}$ will be
Comparison with conjugate gradient methods

highly clustered around just a few values, so that $\psi(\lambda)$ in (8.9.11) will be small on $Sp(B)$ for $n = n_0$ with $n_0$ the number of clusters, indicating that $n_0$ iterations will suffice. Numerical examples are given by Kettler (1982), who finds indeed that multigrid is much accelerated by the conjugate gradient method for some difficult test problems, using non-robust smoothers. Hence conjugate gradient acceleration may, if necessary, be used to improve the robustness of multigrid methods. Furthermore, Kettler (1982) finds the conjugate gradient method by itself, using as preconditioner the smoother used in multigrid, to be about equally efficient as multigrid on medium-sized grids ($50 \times 50$, say). As the number of unknowns increases multigrid becomes more efficient.

The non-symmetric case

Severe limitations of conjugate gradient methods are their restriction to linear systems with symmetric positive definite matrices. A number of conjugate gradient type methods have been proposed that are applicable to the non-symmetric case. Although no theoretical estimates are available, their rate of convergence is often satisfactory in practice. We will present one such method, namely CGS (conjugate gradients squared), described in Sonneveld et al. (1985, 1986) and Sonneveld (1989). Good convergence is expected if the eigenvalues of $A$ have positive real part, cf. the remarks on convergence in Sonneveld (1989).

As preconditioned system we choose

$$EAF(F^{-1}y) = b$$

(8.9.14)

The preconditioned CGS algorithm is given by

$$
\begin{align*}
    r^0 &= E(b - Ay^0), \quad r^0 = r^0 \\
    q^0 &= p^{-1} = 0, \quad p^{-1} = 1 \\
    \text{for } n = 0, 1, 2, \ldots, \text{do} & \\
    \rho_n &= \tilde{r}^0 r^n, \quad \beta_n = \rho_n/\rho_{n-1} \\
    u^n &= r^n + \beta_n q^n \\
    p^n &= u^n + \beta_n (q^n + \beta_n p^{n-1}) \\
    v^n &= EAFp^n \\
    \sigma_n &= \tilde{r}^0 u^n, \quad \alpha_n = \rho_n/\sigma_n \\
    q^{n+1} &= u^n - \alpha_n u^n \\
    v^n &= \alpha_n F(u^n + q^{n+1}) \\
    r^{n+1} &= r^n - EAv^n \\
    y^{n+1} &= y^n + v^n
\end{align*}
$$

od
In numerical experiments with convection–diffusion type test problems with ILU and IBLU preconditioning Sonneveld (1989) finds CGS to be more efficient than some other non-symmetric conjugate gradient type methods. With ILU one chooses for example $E = L^{-1}$, $F = U^{-1}D$ whereas with IBLU one may choose for example $E = (L + D)^{-1}$, $F = (U + D)^{-1}D$. Multigrid may be accelerated with CGS by choosing $E = Q^K(\mu, \nu)$, $F = I$.

**Comparison of conjugate gradient and multigrid methods**

Realistic estimates of the performance in practice of conjugate gradient and multigrid methods 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 tests 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 conjugate gradient and multigrid methods, in particular the subject areas of computational fluid dynamics, petroleum reservoir engineering and neutron diffusion are pace-setting.

Important constant coefficient test problems are (7.5.6) and (7.5.7). 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 conjugate gradient methods, for which the properties of the spectrum are very important. For multigrid methods, 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 $e$ and $\beta$. This fact goes easily unnoticed with variable coefficients, where the unfavourable values of $e$ and $\alpha$ 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 problems equations (7.5.6) and (7.5.7) 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
Comparison with conjugate gradient methods

symmetric positive definite. With vertex-centred multigrid, operator-dependent transfer operators have to be used, of course.

The four test problems just mentioned, i.e. (7.5.6), (7.5.7) and the problems of Stone and Kershaw, are gaining acceptance among conjugate gradient and multigrid 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.

Numerical experiments for the test problems of Stone and Kershaw and equations (7.5.6) and (7.5.7), comparing CGS and multigrid, are described by Sonneveld et al. (1985), using ILU and IBLU preconditioning and smoothing. As expected, the rate of convergence of multigrid is unaffected when the mesh size is decreased, whereas CGS slows down. On a $65 \times 65$ grid there is no great difference in efficiency. Another comparison of conjugate gradients and multigrid is presented by Dendy and Hyman (1981). Robustness and efficiency of conjugate gradient and multigrid methods are determined to a large extent by the preconditioning and the smoothing method respectively. The smoothing methods that were found to be robust on the basis of Fourier smoothing analysis in Chapter 7 suffice, also as preconditioners. It may be concluded that for medium-sized linear problems conjugate gradient methods are about equally efficient as multigrid in accelerating basic iterative methods. As such they are limited to linear problems, unlike multigrid. On the other hand, conjugate gradient methods are much easier to program, especially when the computational grid is non-rectangular.