Chapter 4

Implementation

The preceding chapter was devoted to the development of several multigrid schemes. We now turn to the practical issues of writing multigrid programs and determining whether they work. This will lead us to issues such as data structures, complexity, predictive tools, diagnostic tools, and performance.

Complexity

Writing multigrid programs can be both fun and challenging. The experience of many practitioners suggests that such programs should be highly modular. This allows them to evolve from simple relaxation programs and makes them much easier to check and debug. Also, the various components of the program (for example, relaxation, interpolation, and restriction subroutines) can be replaced individually.

Choosing a manageable data structure for a multigrid program is essential. Modern programming languages are replete with devices that make data management easy. For example, in most languages, one can declare a *structure* that groups together all the associated information for each grid level. In a structured language, for instance, a V-cycle could be written along the lines of the following *pseudocode*:

```
declare structure:
     grid =  { double Ddim_array  f  %% the right hand side
               double Ddim_array  v  %% the current approximation }

declare Grid: array of structure grid

for j = 0 to coarsest - 1
    Grid[j].v <- relax(Grid[j].v, Grid[j].f, num_sweeps_down)
    Grid[j+1].f  <- restrict(Grid[j].f - apply_operator(Grid[j].v))
endfor

Grid[coarsest].v = direct_solve(Grid[coarsest].v, Grid[coarsest].f)

for j = coarsest-1 to 1
    Grid[j].v <- Grid[j].v + interpolate(Grid[j+1].v)
    Grid[j].v <- relax(Grid[j].v, Grid[j].f, num_sweeps_down)
endfor
```

The routines *relax, restrict, apply_operator, interpolate*, and *direct_solve* take the appropriate *Ddim_arrays*, *v* and *f*, for the specified grid level and perform the appropriate operations. We do not describe this type of data management in any further detail, as the advances in these languages occur so rapidly that any discussion would soon be outdated!

We describe a data structure for a simpler FORTRAN-like language. Multigrid codes "grew up" in such an environment and many people learn to write multigrid codes using *MATLAB* or a similar prototyping language with more restrictive data structures. With these languages, there seems to be general agreement that the solutions and right-side vectors on the various grids should be stored contiguously in single arrays. Complicating factors such as irregular domains or local fine-grid patches might require an exception to this practice. However, single arrays are advisable for the regular grids discussed in this chapter.

We begin by considering a four-level V-cycle applied to a one-dimensional problem with $n = 16$ points. A typical data structure is shown in Fig. 4.1. It is instructive to note how the data structure changes as the V-cycle progresses. Each grid needs two arrays: one to hold the current approximations on each grid and one to hold the right-side vectors on each grid. Because boundary values must also be stored, the coarsest grid involves three grid points (one interior and two boundary points). In general, the ℓth coarsest grid involves $2^\ell + 1$ points.

Initially, the entire solution array $\mathbf{v}$ may be set to zero, which will be the initial guess on each grid. The right-side array $\mathbf{f}$ will also be set to zero, except for the values on the finest grid, which are known at the outset.

As the V-cycle "descends" into coarser grids, relaxation fills the segment of the solution array corresponding to each grid. At the same time, the residual vectors $\mathbf{f}$ fill the right-side array corresponding to the next coarsest grid. As the V-cycle "ascends" through finer grids, the right-side array does not change. However, the solution array is overwritten by additional relaxations on each level. Notice that when a new approximation is computed on one level, the approximation on the previous level is zeroed out. This provides a zero initial guess on each level in case another V-cycle is performed.

We now turn to the important questions of complexity. How much do the multigrid schemes cost in terms of storage and computation? The storage question is easier to answer. Consider a d-dimensional grid with n^d points. (Actually, for Dirichlet boundary conditions, there will be $(n-1)^d$ interior points and, as we will see later, for Neumann boundary conditions, there will be $(n+1)^d$ unknown points.) For simplicity, suppose n is a power of 2. We have just seen that two arrays must be stored on each level. The finest grid, Ω^h, requires $2n^d$ storage locations; Ω^{2h} requires 2^{-d} times as much storage as Ω^h; Ω^{4h} requires 4^{-d} times as much storage as Ω^h; in general, Ω^{ph} requires p^{-d} times as much storage as Ω^h. Adding these terms and using the sum of the geometric series as an upper bound gives

$$\text{Storage} = 2n^d\{1 + 2^{-d} + 2^{-2d} + \cdots + 2^{-nd}\} < \frac{2n^d}{1-2^{-d}}.$$

In particular, for a one-dimensional problem ($d = 1$), the storage requirement is less than twice that of the fine-grid problem alone. For problems in two or more dimensions, the requirement drops to less than $\frac{4}{3}$ of the fine-grid problem alone (Exercise 3). Thus, the storage costs of multigrid algorithms decrease relatively as the dimension of the problem increases.

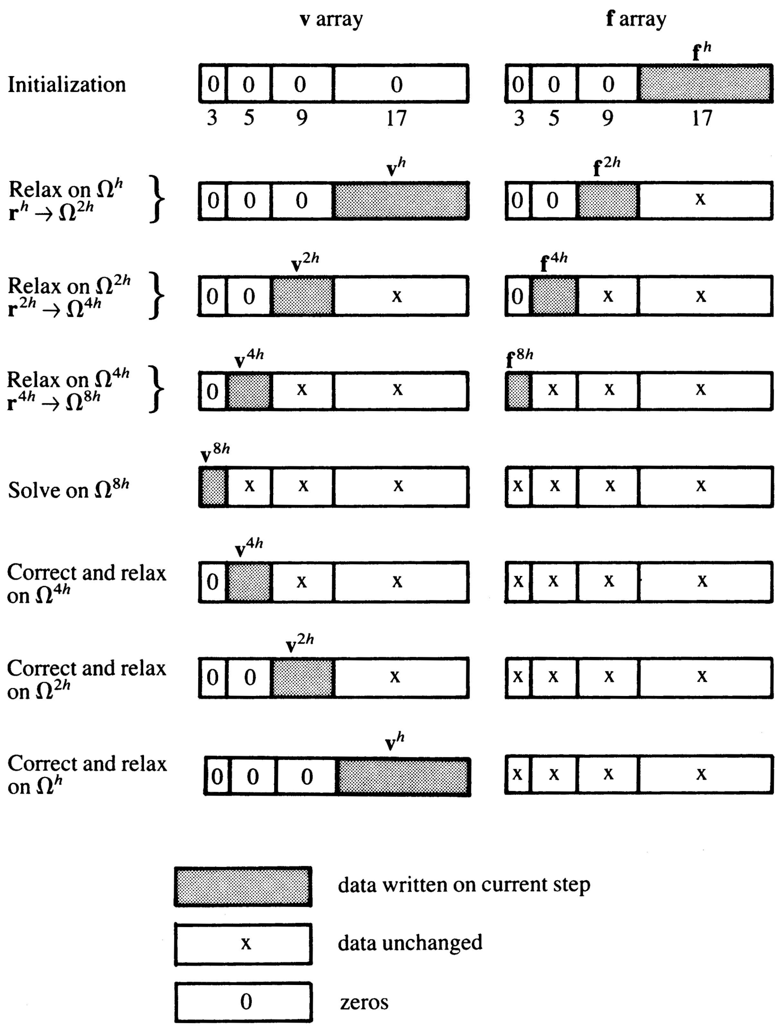

Figure 4.1: *Illustration of the course of a four-level (n = 16) V-cycle showing changes in the data arrays. The* **v** *and* **f** *arrays hold the solution vectors and right-side vectors, respectively, in the four grids.*

We may use similar reasoning to estimate the computational cost of multigrid methods. It is convenient to measure these costs in terms of a *work unit* (WU), which is the cost of performing one relaxation sweep on the finest grid. It is customary to neglect the cost of intergrid transfer operations, which typically amounts to 10–20% of the cost of the entire cycle.

First consider a V-cycle with one relaxation sweep on each level ($\nu_1 = \nu_2 = 1$). Each level is visited twice and grid Ω^{ph} requires p^{-d} work units. Adding these costs and again using the geometric series for an upper bound gives

$$\text{V-cycle computation cost} = 2\left\{1 + 2^{-d} + 2^{-2d} + \cdots + 2^{-nd}\right\} < \frac{2}{1-2^{-d}}\,\text{WU}.$$

A single V-cycle costs about 4 WUs for a one-dimensional ($d = 1$) problem, about $\frac{8}{3}$ WUs for $d = 2$, and $\frac{16}{7}$ WUs for $d = 3$ (Exercise 4).

With a slight modification, we can find the computational cost for an FMG cycle. Assume again that one relaxation sweep is done on each level ($\nu_0 = \nu_1 = \nu_2 = 1$). As just shown, a full V-cycle beginning from Ω^h costs about $2(1 - 2^{-d})^{-1}$ WUs. A V-cycle beginning from Ω^{2h} costs 2^{-d} of a full V-cycle. In general, a V-cycle beginning from Ω^{ph} costs p^{-d} of a full V-cycle. Adding these costs gives us

$$\begin{aligned}&\text{FMG computation cost}\\ &\quad= \left(\frac{2}{1-2^{-d}}\right)\left(1 + 2^{-d} + 2^{-2d} + \cdots + 2^{-nd}\right) < \frac{2}{(1-2^{-d})^2}\,\text{WU}.\end{aligned}$$

An FMG cycle costs 8 WUs for a one-dimensional problem, about $\frac{7}{2}$ WUs for $d = 2$, and $\frac{5}{2}$ WU for $d = 3$ (Exercise 5).

As expected, a single FMG cycle costs more than a single V-cycle, although the discrepancy is less for higher-dimensional problems. We really need to know how many V-cycles and FMG cycles are needed to obtain satisfactory results. This begs the fundamental question: how well do these multigrid cycling schemes work?

Predictive Tools: Local Mode Analysis

The previous section dealt with the practical considerations of implementing multigrid algorithms. However, it is a common experience to have a multigrid code that runs, but does not work! Indeed, it can often be puzzling to know what to expect in terms of efficiency and accuracy. The remainder of this chapter presents some practical tools for determining whether an algorithm is working properly. First, we deal with tools for predicting the convergence rates that can be expected from the basic relaxation methods applied to standard problems.

Recall from Chapter 2 that the asymptotic convergence factor of a relaxation scheme is the spectral radius (the largest eigenvalue magnitude) of the corresponding iteration matrix. We also defined the smoothing factor as the convergence factor associated with the oscillatory modes only. Because eigenvalue calculations can be difficult, this approach to finding convergence factors is limited to fairly simple iterations applied primarily to model problems.

We now present a more versatile approach for approximating convergence and smoothing factors called *local mode analysis* (or *normal mode analysis* or *Fourier analysis*). The goal of this section is rather modest: we show how to apply the basic procedure to some prototype problems and then point the way to more advanced calculations. In its full generality, local mode analysis can be applied to general operators and to a wide class of relaxation schemes on one or more levels. With this generality, local mode analysis is a powerful predictive tool that can be used to compare multigrid performance with theoretical expectations.

The original proponent of local mode analysis was Achi Brandt, who expressed its significance by saying that

> ...the main importance of the smoothing factor is that it separates the design of the interior relaxation from all other algorithmic questions. Moreover, it sets an ideal figure against which the performance of the full algorithm can later be judged. [4]

Local mode analysis begins with the assumption that relaxation is a local process: each unknown is updated using information from nearby neighbors. Because it is a local process, it is argued that boundaries and boundary conditions can be neglected if we are considering a few relaxation sweeps at interior points. For this reason, the finite domain of the problem is replaced by an infinite domain.

As before, we are interested in how a particular relaxation scheme acts on the errors in an approximation. Assume that relaxation is a linear process and denote the associated matrix by R. Let $\mathbf{e}^{(m)}$ denote the algebraic error at the mth step of relaxation. Recall (Chapter 2) that the error itself evolves under the action of R:

$$\mathbf{e}^{(m+1)} = R\mathbf{e}^{(m)}.$$

The approach of local mode analysis is to assume that the error consists of Fourier modes and to determine how relaxation acts on those modes. The Fourier modes we encountered in Chapter 2 have the form $w_j = \sin(\frac{jk\pi}{n})$, where the wavenumber k is an integer between 1 and n. This means that the term $\theta = \frac{k\pi}{n}$ runs roughly from 0 to π. With the new assumption of an infinite domain (no boundaries or boundary conditions to satisfy), the Fourier modes need not be restricted to discrete wavenumbers. Instead, we consider modes of the form $w_j = e^{\iota j\theta}$, where the wavenumber θ can take on any value in the interval $(-\pi, \pi]$. (For the remainder of the chapter, we let $\iota = \sqrt{-1}$ to avoid confusing i with the grid indices.) Notice that the mode corresponding to a particular θ has a wavelength of $\frac{2\pi h}{|\theta|}$; values of $|\theta|$ near zero correspond to low-frequency waves; value of $|\theta|$ near π correspond to high-frequency waves. The choice of a complex exponential makes computations much easier and accounts for both sine and cosine terms.

An important point should be mentioned here. Local mode analysis is not completely rigorous unless the Fourier modes are eigenvectors of the relaxation matrix, which is not generally the case. However, the analysis is useful for the high frequency modes of the error, which *do* tend to resemble the eigenvectors of the relaxation matrix very closely. For this reason, local mode analysis is used for a smoothing analysis of the high frequency modes.

With these ground rules, we are ready to apply the method. We begin with one-dimensional problems and assume that the error at the mth step of relaxation at the jth grid point consists of a single mode of the form

$$e_j^m = A(m)e^{\iota j\theta}, \quad \text{where} \quad -\pi < \theta \leq \pi. \tag{4.1}$$

The goal is to determine how the amplitude of the mode, $A(m)$, changes with each relaxation sweep. In each case we consider, the amplitudes at successive steps are related by an expression of the form

$$A(m+1) = G(\theta)A(m).$$

The function G that describes how the error amplitudes evolve is called the *amplification factor*. For convergence of the method, we must have $|G(\theta)| < 1$ for all θ. As we have seen, relaxation is used in multigrid to eliminate the oscillatory modes of the error. Therefore, the quantity of interest is really the *smoothing factor*, which is found by restricting the amplification factor, $G(\theta)$, to the oscillatory modes $\frac{\pi}{2} \leq |\theta| \leq \pi$. Specifically, we define the smoothing factor as

$$\mu = \max_{\frac{\pi}{2} \leq |\theta| \leq \pi} |G(\theta)|.$$

This is the factor by which we can expect the oscillatory modes to be damped (at worst) with each relaxation sweep. With these definitions, it is best to proceed by example.

Example: One-dimensional problems. Consider the one-dimensional model problem

$$-u''(x) + c(x)u(x) = f(x).$$

Letting v_j be the approximation to $u(x_j)$, we discretize the problem with the usual second-order finite-difference approximations and apply weighted Jacobi relaxation. This results in the familiar Jacobi updating step

$$v_j^{m+1} = \frac{\omega}{2 + h^2 c_j}(v_{j-1}^m + v_{j+1}^m + h^2 f_j) + (1-\omega)v_j^m, \tag{4.2}$$

where $c_j = c(x_j)$. Knowing that the error, $e_j = u(x_j) - v_j$, is also governed by the same weighted Jacobi relaxation, we can write the updating step for the error at the jth grid point as (Exercise 6)

$$e_j^{m+1} = \frac{\omega}{2 + h^2 c_j}(e_{j+1}^m + e_{j-1}^m) + (1-\omega)e_j^m. \tag{4.3}$$

Assume now that the error consists of a mode of the form (4.1) and substitute it into (4.3). Letting $c_j = 0$ for the moment, we have

$$A(m+1)e^{\iota j\theta} = \frac{\omega}{2}\left(A(m)\underbrace{(e^{\iota(j+1)\theta} + e^{\iota(j-1)\theta})}_{2e^{\iota j\theta}\cos\theta}\right) + (1-\omega)A(m)e^{\iota j\theta}.$$

As indicated, the Euler formula for $\cos\theta$ allows for some simplification. Collecting terms now leads to

$$A(m+1)e^{\iota j\theta} = A(m)(1 - \omega\underbrace{(1-\cos\theta)}_{2\sin^2(\theta/2)})e^{\iota j\theta}.$$

Canceling the common term $e^{\iota j\theta}$ and using the indicated trigonometric identity, we can write the following relationship between successive amplitudes:

$$A(m+1) = \left(1 - 2\omega\sin^2\left(\frac{\theta}{2}\right)\right)A(m) \equiv G(\theta)A(m), \quad \text{where} \quad -\pi < \theta \le \pi.$$

The amplification factor $G(\theta) = 1 - 2\omega\sin^2(\theta/2)$ appears naturally in this calculation and it should look familiar. In this case, we have just reproduced the eigenvalue calculation for the weighted Jacobi iteration matrix (see Chapter 2); that is, if we make the substitution $\theta_k = \frac{\pi k}{n}$, then $G(\theta_k)$ is just the kth eigenvalue of the Jacobi iteration matrix. As we know, $|G(\theta)| < 1$ provided $0 < \omega \le 1$; with $\omega = \frac{2}{3}$, we have the optimal smoothing factor

$$\mu = G\left(\frac{\pi}{2}\right) = |G(\pm\pi)| = \frac{1}{3}.$$

A similar calculation can be done with Gauss–Seidel. The updating step for the error at the jth grid point now appears as (Exercise 7)

$$e_j^{m+1} = \frac{e_{j-1}^{m+1} + e_{j+1}^m}{2 + c_j h^2}. \tag{4.4}$$

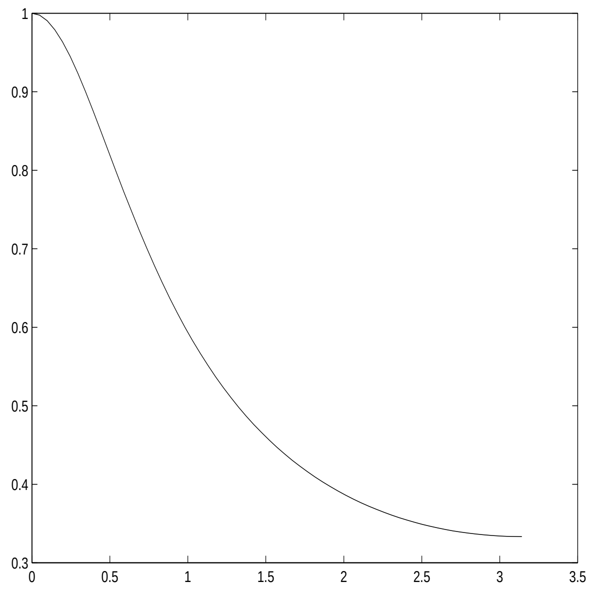

Figure 4.2: *The amplification factor,* $|G(\theta)|$*, for the Gauss–Seidel method applied to the one-dimensional problem* $-u''(x) = f(x)$*. The graph is symmetric about* $\theta = 0$*. The smoothing factor is* $\mu = |G(\frac{\pi}{2})| = \frac{1}{\sqrt{5}} = 0.45$.

Note that because we sweep across the grid from left to right, the previous ($(j-1)$st) component has already been updated. Again we assume that the errors have the form (4.1) and substitute. Assuming for the moment that $c_j = 0$, we find that the amplitudes are related by (Exercise 7)

$$A(m+1) = \frac{e^{\iota\theta}}{2 - e^{-\iota\theta}} A(m) \equiv G(\theta)A(m), \quad \text{where} \quad -\pi < \theta \leq \pi.$$

To find the smoothing factor from the complex amplification factor, it is easiest to plot $|G(\theta)|$, as shown in Fig. 4.2. A bit of analysis reveals that

$$\mu = \left| G\left(\frac{\pi}{2}\right) \right| = \frac{1}{\sqrt{5}} = 0.45.$$

A subtle point could be made here. The amplification factor, $G(\theta)$, gives the (complex) eigenvalues of the Gauss–Seidel iteration matrix, not on a bounded domain with specified boundary conditions, but on an infinite domain. This calculation differs from the eigenvalue calculation of Chapter 2, in which the eigenvalues for a bounded domain were found to be real. For this reason, the amplification factor gives only an estimate of the smoothing factors for a bounded domain problem. ◇◇

We can use the above example to illustrate how local mode analysis works with a variable coefficient operator. Suppose that $c(x) > 0$ on the domain. To avoid working with a different amplification factor at every grid point, the practice is to "freeze" the coefficient, $c(x)$, at a representative value $c_0 = c(\xi)$, for some ξ in the domain (often the minimum or maximum value of $c(x)$ on the domain). With the weighted Jacobi iteration, the amplification factor now appears as

$$G(\theta) = 1 - \omega \left(1 - \frac{2}{2 + c_0 h^2} \cos\theta\right).$$

The idea is to find the value of c_0, over all possible $c(\xi)$, that gives the worst (most pessimistic) smoothing factor. Occasionally, this calculation can be done analytically; more typically, it is done numerically by choosing several different possible values of c_0. We can rewrite this amplification factor as

$$G(\theta) = G_0(\theta) - \frac{c_0 \omega h^2}{2} \cos(\theta),$$

where $G_0(\theta)$ is the amplification factor for the case that $c(x) = 0$. In this form, we see that the effect of the variable coefficient is insignificant unless c_0 is comparable to h^{-2}. There is a more general principle at work here: usually the lower order terms of the operator can be neglected with impunity in local mode analysis.

Local mode analysis can be extended easily to two or more dimensions. In two dimensions, the Fourier modes have the form

$$e_{jk}^{(m)} = A(m) e^{\iota(j\theta_1 + k\theta_2)}, \tag{4.5}$$

where $-\pi < \theta_1, \theta_2 \leq \pi$ are the wavenumbers in the x- and y-directions, respectively. Substituting this representation into the error updating step generally leads to an expression for the change in the amplitudes of the form

$$A(m+1) = G(\theta_1, \theta_2) A(m).$$

The amplification factor now depends on two wavenumbers. The smoothing factor is the maximum magnitude of the amplification factor over the oscillatory modes. As we see in Fig. 4.3, the oscillatory modes correspond to $\frac{\pi}{2} \leq |\theta_i| \leq \pi$ for either $i = 1$ or $i = 2$; that is,

$$\mu = \max_{\pi/2 \leq |\theta_i| \leq \pi} |G(\theta_1, \theta_2)|.$$

Example: Two-dimensional problems. Consider the model problem

$$u_{xx} + u_{yy} = f(x, y)$$

on a rectangular domain with a uniform grid in both directions. Applying the weighted Jacobi method, the error satisfies (Exercise 8)

$$e_{jk}^{(m+1)} = \frac{\omega}{4}\left(e_{j-1,k}^{(m)} + e_{j+1,k}^{(m)} + e_{j,k-1}^{(m)} + e_{j,k+1}^{(m)}\right) + (1-\omega) e_{jk}^{(m)}. \tag{4.6}$$

Substituting the Fourier modes (4.5) into the error updating equation, we find that (Exercise 8)

$$A(m+1) = \left[1 - \omega\left(\sin^2\left(\frac{\theta_1}{2}\right) + \sin^2\left(\frac{\theta_2}{2}\right)\right)\right] A(m) \equiv G(\theta_1, \theta_2) A(m).$$

Two views of the amplification factor are given in Fig. 4.4 for the case that $\omega = \frac{4}{5}$. In the left figure, each curve shows the variation of G over $0 \leq \theta_2 \leq \pi$ for fixed

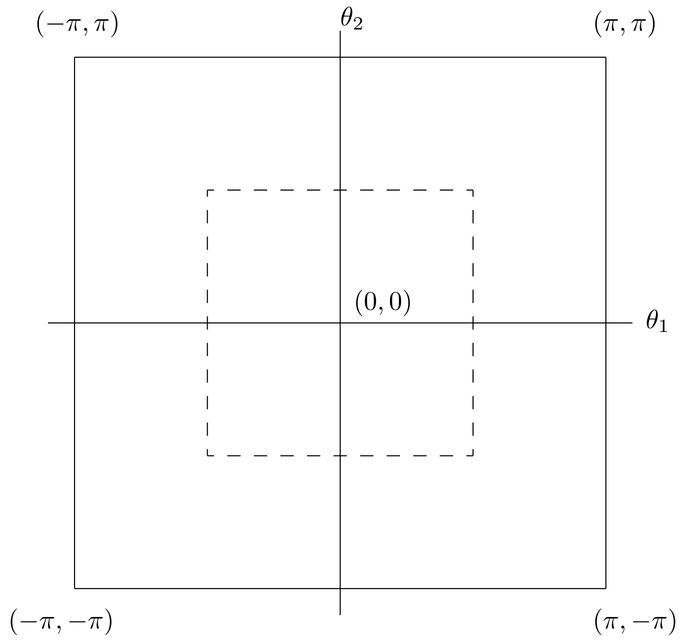

Figure 4.3: *The oscillatory modes in two dimensions correspond to the wavenumbers* $\frac{\pi}{2} \le |\theta_i| < \pi$ *for either* $i = 1$ *or* $i = 2$*; this is the region outside the dashed box.*

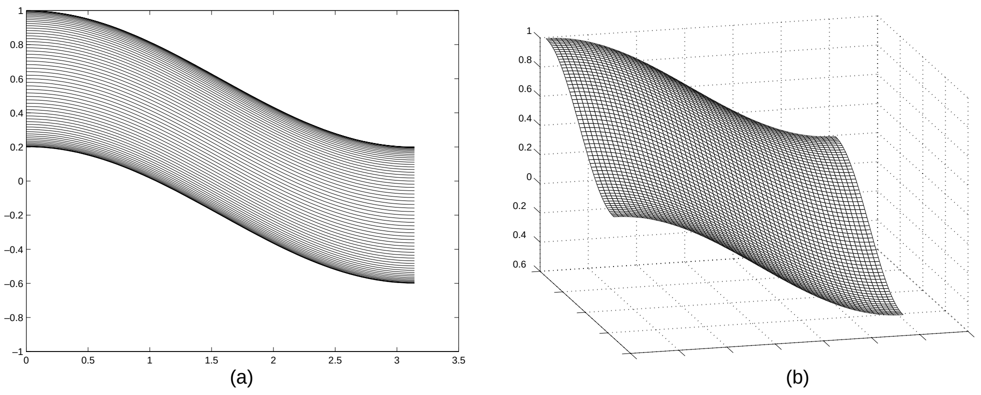

Figure 4.4: (a) *Amplification factor,* $G(\theta_1, \theta_2)$*, for the weighted Jacobi method applied to the model problem in two dimensions, shown as individual curves of fixed* θ_1 *(*$\theta_1 = 0$ *at the top and* $\theta_1 = \pi$ *at the bottom).* (b) *Same amplification factor shown as a surface over the region* $[0, \pi] \times [0, \pi]$*. The picture is symmetric about both the* θ_1*- and* θ_2*-axes.*

values of θ_1; the upper curve corresponds to $\theta_1 = 0$ and the lower curve corresponds to $\theta_2 = \pi$. Clearly, the amplification factor decreases in magnitude as the modes become more oscillatory. The right figure shows the same amplification factor as a

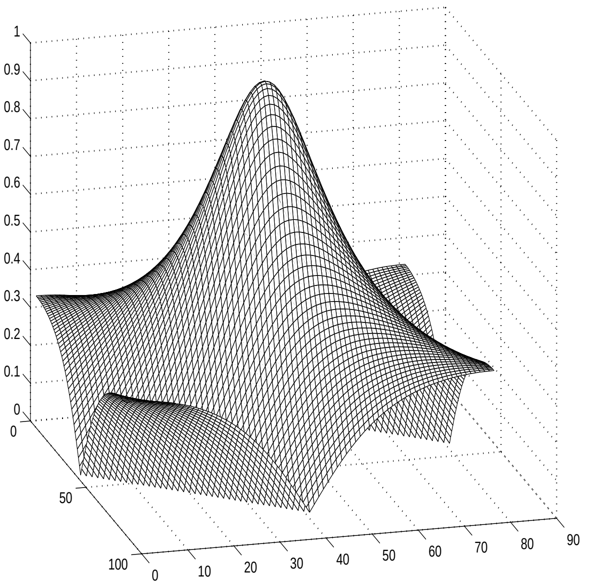

Figure 4.5: *Amplification factor, $|G(\theta_1, \theta_2)|$, for the Gauss–Seidel method applied to the model problem in two dimensions, shown as a surface over the region* $[-\pi, -\pi] \times [\pi, \pi]$.

surface over the region $[0, \pi] \times [0, \pi]$. (The surface is symmetric about both axes.) Some analysis or experimentation reveals that the best smoothing factor is obtained when $\omega = \frac{4}{5}$ and it is given by $\mu = |G(\pm\pi, \pm\pi)| = 0.6$ (Exercise 8). This means that if we use the Jacobi scheme with $\omega = \frac{4}{5}$, then we expect a reduction in the residual norm by approximately a factor of 0.6 per relaxation sweep. A V(2,1)-cycle, for example, should have a convergence factor of about $0.6^3 = 0.216$.

A similar calculation can be done for the Gauss–Seidel method applied to the model problem. The error updating equation is (Exercise 9)

$$e_{jk}^{(m+1)} = \frac{e_{j-1,k}^{(m+1)} + e_{j+1,k}^{(m)} + e_{j,k-1}^{(m+1)} + e_{j,k+1}^{(m)}}{4}. \tag{4.7}$$

Here we assume that the unknowns have a lexicographic ordering (in order of increasing j and k); thus, the unknowns preceding e_{jk} are updated and appear at the $(m+1)$st step. Once again, we substitute the modes given in (4.5). The amplification factor is most easily expressed in complex form as (Exercise 9)

$$G(\theta_1, \theta_2) = \frac{e^{\iota\theta_1} + e^{\iota\theta_2}}{4 - e^{-\iota\theta_1} - e^{-\iota\theta_2}}.$$

The magnitude of this function is plotted over the region $[-\pi, -\pi] \times [\pi, \pi]$ in Fig. 4.5.

Some computation is required to show that

$$|G(\theta_1, \theta_2)|^2 = \frac{1 + \cos\beta}{9 - 8\cos(\frac{\alpha}{2})\cos(\frac{\beta}{2}) + \cos\beta},$$

The Discrete L^2 Norm. Another norm that is particularly appropriate for measuring errors in numerical calculations is the *discrete* L^2 *norm.* If the vector $\mathbf{u}^h$ is associated with a d-dimensional domain with uniform grid spacing h, then its discrete L^2 norm is given by

$$\|\mathbf{u}^h\|_h = \left(h^d \sum_i (u_i^h)^2 \right)^{1/2},$$

which is the usual Euclidean vector norm, scaled by a factor (h^d) that depends on the geometry of the problem. This scaling factor is introduced to make the discrete L^2 norm an approximation to the continuous L^2 norm of a function $u(\mathbf{x})$, which is given by

$$\|u\|_2 = \left(\int_\Omega u(\mathbf{x}) d\mathbf{x} \right)^{1/2}.$$

For example, with $d = 1$, let $u(x) = x^{m/2}$, where $m > -1$ is an integer. Also, let $\Omega = [0,1]$ with grid spacing $h = \frac{1}{n}$. Then the associated vector is $u_i^h = x_i^{m/2} = (ih)^{m/2}$. The continuous L^2 norm is

$$\|u\|_2 = \left(\int_0^1 x^{m/2} \right)^{1/2} dx = \frac{1}{\sqrt{m+1}},$$

while the corresponding discrete L^2 norm is

$$\|\mathbf{u}^h\|_h = \left(h \sum_{i=1}^{n} ((ih)^{m/2})^2 \right)^{1/2} \overset{h \to 0}{=} \frac{1}{\sqrt{m+1}}.$$

In this case, the discrete L^2 norm approaches the continuous L^2 norm as $h \to 0$, which occurs only because of the scaling (see Exercise 18).

where $\alpha = \theta_1 + \theta_2$ and $\beta = \theta_1 - \theta_2$. Restricting the amplification factor to the oscillatory modes, a subtle analysis [25] reveals that the smoothing factor is given by

$$\mu = G\left(\frac{\pi}{2}, \cos^{-1}\left(\frac{4}{5}\right)\right) = \frac{1}{2}.$$

◇◇

These examples illustrate local mode analysis for relatively elementary problems. The same technique can be extended, usually with more computation and analysis, to anisotropic equations (for example, $\epsilon u_{xx} + u_{yy} = f$) and line relaxation, as discussed in Chapter 7. It can be used for more general operators (for example, convection-diffusion) and for systems of equations. It can also be applied to other relaxation methods with different orderings, some of which lead to new complications. For example, red-black relaxation has the property that Fourier modes become mixed in pairs (in one dimension) or in groups of four (in two dimensions). Thus, the amplification factor is replaced by an *amplification matrix*. The extension to the coarse-grid correction scheme [10] on two levels requires an analysis of interpolation and restriction in a Fourier setting, a subject discussed in the next chapter.

Diagnostic Tools

As with any numerical code, debugging can be the most difficult part of creating a successful program. For multigrid, this situation is exacerbated in two ways. First, the interactions between the various multigrid components are very subtle, and it can be difficult to determine which part of a code is defective. Even more insidious is the fact that an incorrectly implemented multigrid code can perform quite well—sometimes better than other solution methods! It is not uncommon for the beginning multigrid user to write a code that exhibits convergence factors in the 0.2–0.3 range for model problems, while proper tuning would improve the factors to something more like 0.05. The difficulty is convincing the user that 0.2–0.3 is not good enough. After all, this kind of performance solves the problem in very few cycles. But the danger is that performance that is below par for model problems might really expose itself as more complexities are introduced. Diagnostic tools can be used to detect defects in a multigrid code—or increase confidence in the observed results.

Achi Brandt has said that "the amount of computational work should be proportional to the amount of real physical changes in the computed system" and "stalling numerical processes must be wrong." These statements challenge us to develop codes that achieve the best possible multigrid performance. The following short list of diagnostic tools should help to achieve that goal. A systematic approach to writing multigrid codes is also given by Brandt in the section "Stages in Developing Fast Solvers" in his 1984 *Guide* [4].

Of course, no short list of debugging techniques can begin to cover all contingencies. What we provide here is a limited list of tricks and techniques that can be useful in evaluating a multigrid code; they often tell more about the *symptoms* of a defective code than the *causes*. Nevertheless, with increasing experience, they can guide the user to develop properly tuned codes that achieve multigrid's full potential.

- **Methodical Plan.** The process of testing and debugging a multigrid code should be planned and executed methodically. The code should be built in a modular way so that each component can be tested and integrated into the evolving code with confidence. It is best to test the algebraic solver first (for example, V-cycles); then the discretization can be tested, followed by the FMG solver, if it is to be used. In other words, the initial focus should be on ensuring that the basic cycling process solves the discrete system up to expectations. This solver can then be used to test discretization accuracy. Poor discretization, especially at boundaries, is often the source of multigrid inefficiency. Therefore, it is important to test the discretization, perhaps with another solver, when multigrid troubles persist. FMG requires an efficient V-cycle or W-cycle solver *and* an accurate discretization method. This means that FMG should be implemented and tested after the solver and the discretization components are verified (see last item).

- **Starting Simply.** This recommendation is perhaps the most obvious: it is always best to begin with basic methods applied to small, simple problems. Simpler cases expose troubles more clearly and make it easier to trace the sources. Test problems should consist of either a discrete system with a known solution or the simplest form of the desired PDE (usually this means constant coefficients, no convection, no nonlinearity, and trivial boundary conditions). A good sequence is to test the solver on the coarsest grid that your code accepts, then add one finer level to test the two-grid scheme thoroughly. One can then proceed progressively and methodically to larger problems. Once the simplest cases show expected performance, complexities can be added one at a time.

- **Exposing Trouble.** It is critical to start with simple problems so that potential difficulties are kept in reserve. At the same time, it is also important to avoid aspects of the problem that mask troubles. For example, reaction terms can produce strong enough diagonal dominance in the matrix that relaxation by itself is efficient. These terms should be eliminated in the initial tests if possible. Similarly, if the matrix arises from implicit time-stepping applied to a time-dependent problem, then a very large or even infinite time step should be taken in the initial tests.

- **Fixed Point Property.** Relaxation should not alter the exact solution to the linear system: it should be a fixed point of the iteration. Thus, using the exact solution as an initial guess should yield a zero residual before *and* after the relaxation process. Furthermore, the coarse-grid problem takes the transferred residual as its right side, which means that its solution should also be zero. Because the coarse-grid problem uses zero as the initial guess, it is solved exactly, and the correction passed up to the finer grid is also zero. Therefore, neither relaxation nor coarse-grid correction should alter the exact solution.

  This property can be checked by creating a right-side vector corresponding to the known solution (of the linear system) and then using that solution as an initial guess. The relaxation module should be tested first, after which the V-cycle can be tested. If the output from either differs by more than machine precision from the input, there must be an error in the code.

- **Homogeneous Problem.** Applying a multigrid V-cycle code to a homogeneous problem has the advantage that both the norm of the residual and the norm of the error are computable and should decrease to zero (up to machine precision) at a steady rate; it may take eight to ten V-cycles for the steady rate to appear. The predictive mode analysis tools described above can be used to determine the factor by which the residual norm should decrease; it should tend to the asymptotic factor predicted by the smoothing analysis.

- **Zero Residuals.** A useful technique is to multiply the residual by zero just prior to transferring it to the coarse grid. As in the homogeneous problem, the coarse-grid problem now has a zero right side, so its solution is zero. Because the initial guess to the coarse-grid problem is zero, it is solved exactly, and the correction (also zero) is passed back to the fine grid. The utility of this test is that the only part of the code now affecting the approximation is relaxation on the fine grid. This means that the sequence of approximations generated by V(ν_1, ν_2)-cycles should be identical to the approximations generated using only relaxation; this is an easy property to test.

- **Residual Printout.** A good technique for monitoring performance is to print out the norm of the residual on each level after relaxation on both the descent and ascent legs of the V-cycle. Here again, it is important that the discrete L^2 norm be used, because its scaling makes the error norms comparable from one level to the next. The residuals should behave in a regular fashion: on each level, the sequence of norms should decline to machine zero at a steady rate as the cycling continues. The norm on each level should be smaller after post-relaxation (on the upward leg) than it was after pre-relaxation (on the downward leg). This ensures that the coarse-grid correction/relaxation tandem is working on each level.

  Because most multigrid codes are recursive in nature (although they may not actually use recursive calls), it is important to note that seeing an abnormal residual pattern on a given level does not necessarily mean the code is somehow wrong on that level. More frequently, the flaw exists on all levels, because all levels are treated by the same code. Indeed, the most common culprits are the intergrid transfer operators and boundary treatments. Monitoring the sequence of residuals, however, can be more helpful in ascertaining the presence of a problem than simply observing the overall convergence rate on the fine grid.

- **Error Graph.** Solver trouble that seems impervious to diagnosis can sometimes be resolved with a picture of the error. Surface plots of the algebraic error before and after relaxation on the fine grid can be extremely informative. Of course, knowledge of the error is required, so solving the homogeneous problem can be advantageous here. Is the error oscillatory after coarse-grid correction? Is it effectively smoothed by relaxation everywhere? Is there unusual behavior of the error near special features of the domain such as boundaries or interfaces?

- **Two-Level Cycles.** For any multigrid method to work, it is necessary that the two-level scheme (relaxation and exact coarse-grid correction) work. A useful technique is to test the two-level scheme. This may be done by replacing the recursive call for V-cycles with a direct or iterative solver on the coarse grid. If an iterative solver is used (many multigrid cycles on the coarse grid might actually be used here), the coarse-grid problem should be solved very accurately, possibly to machine precision. The correction can then be transferred to the fine grid and applied there.

  Another useful trick is to use two-level cycling between specific pairs of levels. In particular, if the residual printouts indicate an abnormal residual on a specific level, it is useful to perform two-level cycling between that level and the level below it (or between that level and the one above it). This may isolate exactly where the problem occurs.

- **Boundaries.** One of the most common sources of errors in multigrid programs is the incorrect treatment of boundary data. Generally, interpolation and restriction stencils must be altered at boundary points and neighbors of boundary points. Issues involved here are often very subtle and errors can be difficult to detect. Among the most common indicators is that V-cycles will show convergence factors at or near the predicted values for several V-cycles, only to have convergence slow down and eventually stall with continued cycling. The reason is that early in the process, errors at the boundaries have little effect, since the boundary is a lower-dimensional feature of the problem. However, as the cycling continues, the errors from the boundaries propagate into the interior, eventually infecting the entire domain.

  One of the most useful techniques in building or debugging a multigrid code is to separate the effects of the boundary entirely. This can be done by replacing the given boundary conditions by periodic conditions. A periodic boundary value problem should attain convergence factors near those predicted by mode analysis, because the assumptions underlying the Fourier analysis are more closely met. Once it is determined that periodic boundary conditions work properly, the actual boundary conditions may be applied. If the convergence factors degrade with continued cycling, then the treatment at the boundaries may be suspected. A useful technique is to apply extra relaxation sweeps at the boundaries. Often, the extra relaxations will overcome any boundary difficulties and the overall results will approach the ideal convergence factors. If this does not occur, then a very careful examination of how each piece of the code treats the boundaries is in order. Special attention should be paid to the intergrid transfer operators at and near the boundaries. Just as it is easy to obtain discretizations that are of lower order on the boundary than the interior, it is also easy to produce intergrid transfers that fail to be consistent in their treatment of boundary data.

- **Symmetry.** Matrices that arise in the discretization of self-adjoint problems are often symmetric. Such is the case for most of the matrices featured in this book. Many coarsening strategies preserve matrix symmetry when it is present, as does the so-called Galerkin scheme introduced in the next chapter. Inadvertent loss of symmetry often occurs at boundaries, especially at corners and other irregularities of the domain. This loss can lead to subtle difficulties that are hard to trace. If the fine- and coarse-grid matrices are supposed to be symmetric, then this should be tested. This is easily done by comparing entries i, j and j, i of the matrix on each grid.

- **Compatibility Conditions.** We have not yet discussed compatibility conditions, but they arise in the context of Neumann boundary conditions (among other settings), which we discuss in Chapter 7. A compatibility condition is one that must be enforced on the right-side vector to ensure that the problem is solvable. A common source of error is that, while great care may be taken to enforce a compatibility condition on the fine grid, it might not be enforced on the coarse grids. Sometimes, the compatibility condition is enforced automatically in the coarsening process. However, round-off errors may enter and compound themselves as coarsening proceeds, so it may be worthwhile to enforce the condition explicitly on all coarse grids.

- **Linear Performance by Nonlinear Codes.** We describe the FAS (Full Approximation Scheme) for nonlinear problems in Chapter 6. A useful technique for debugging an FAS code is to begin by applying it to a linear problem. FAS reduces to standard multigrid in this case, so an FAS code should exhibit standard performance on these problems. FAS codes should be written so the nonlinearity can be controlled by a parameter: setting the parameter to zero yields a linear problem, while increasing the parameter value strengthens the nonlinearity. As the nonlinearity increases, the performance of FAS should not, in general, degrade significantly (at least for reasonable values of the parameter).

- **Solutions to the PDE.** A known solution to the underlying PDE can be very useful in assessing whether a multigrid code is working as it should. The first thing to consider is whether the solution computed from the multigrid code looks like a sampled version of the continuous solution. The first cursory examination can be qualitative: Does the overall shape of the computed solution resemble the exact solution? Are the peaks and valleys in the right places?

  If the qualitative comparison is good, more quantitative tests can be performed. First, the norm of the error (the difference between the sampled continuous solution and the approximation) should be monitored as a function of the number of V-cycles. The discrete L^2 norm is usually appropriate here. This norm should behave in very specific ways, depending on the accuracy of the discretization. If the known solution has no discretization error (for example, if second-order finite differences are used and the known solution is a second degree polynomial), then the error norm should be driven steadily to machine zero with continued V-cycles. Indeed, the rate at which it goes to zero should be about the same rate at which the norm of the residual declines, and it should reflect the predicted asymptotic convergence factor.

  On the other hand, if discretization error is present, then we expect the norm to stop decreasing after several V-cycles, and it may even grow slightly. This indicates (in a correctly working code) that we have reached the *level of discretization error* (roughly the difference between the continuous and discrete solutions, as discussed in Chapter 5). Here it is useful to solve the same problem repeatedly, with the same right side, on a sequence of grids with decreasing grid spacing h. If the discretization is, for example, $O(h^2)$ accurate in the discrete L^2 norm, then the error norms should decrease roughly by a factor of four each time h is halved. Naturally, the fit will not be perfect, but if the code is working correctly, the trend should be very apparent.

- **FMG Accuracy.** The basic components of an effective FMG scheme are an efficient V-cycle (or W-cycle) solver and an accurate discretization. The idea is to assess the discretization error for a given problem using a sequence of increasingly finer grids. This can be done by choosing a PDE with a known solution and solving each level in turn, starting from the coarsest. Each level should be solved very accurately, perhaps using many V-cycles, testing the residual norm to be sure it is small. The computed solution can then be compared on each level to the PDE solution, evaluated at the grid points, using the discrete L^2 norm. These comparisons yield discretization error estimates on each level. FMG can then be tested by comparing its computed approximation on each level to the PDE solution. The ratio of these estimates to the discretization error estimates should be close to one as the mesh size decreases. This signals that FMG is achieving accuracy to the level of discretization error. Properly tuning the FMG scheme by choosing the right number of V-cycles, pre-smoothing sweeps, and post-smoothing sweeps may be required to achieve this property.

Having discussed a variety of practical matters, it is now time to observe these ideas at work in some numerical experiments.

Numerical example. The first experiment deals with the one-dimensional model problem $A\mathbf{u} = \mathbf{0}$. The weighted Jacobi method with $\omega = \frac{2}{3}$ is applied to this problem on a grid with $n = 64$ points. The initial guess consists of two waves with wavenumbers $k = 3$ and $k = 10$. For purposes of illustration, we implemented a modified V-cycle algorithm. Called the *immediate replacement algorithm*, this version makes an error correction directly to the fine grid after every coarse-grid relaxation. In this way, it is possible to see the immediate effect of each coarse-grid correction on the full, fine-grid solution. Although this version is algebraically equivalent to the standard algorithm, it is impractical because it involves an inordinate number of intergrid transfers. Nevertheless, it is useful for demonstrating algorithm performance because it allows us to monitor the effects of individual coarse-grid operations on the error.

Figures 4.6(a, b) show the course of the algorithm by plotting the maximum norms of the error and the residual after each coarse-grid correction. The algorithm progresses from left to right in the direction of increasing work units (WUs). Notice that the data points are spaced nonuniformly due to the different amount of work done on each level.

Figure 4.6(a) illustrates five full V-cycles with $\nu_1 = \nu_2 = 1$ relaxation sweep on each grid. This computation requires roughly 15 WUs. In these 15 WUs, the initial error norm is reduced by about three orders of magnitude, giving an average convergence rate of about 0.2 decimal digits per WU. Figure 4.6(b) shows the result of performing V-cycles with $\nu_1 = \nu_2 = 2$ relaxation sweeps on each level. In this case, three V-cycles are done in 20 WUs, giving an average convergence rate of about 0.15 decimal digits per WU. In this case, it seems more efficient to relax only once on each level. For another problem, a different cycling strategy might be preferable. It is also possible to relax, say, twice on the "descent" phase of the V-cycle and only once on the "ascent" phase.

The curves of these two figures also show a lot of regular fine structure. Neither the error norm nor the residual norm decreases monotonically. The error curve decreases rapidly on the "descent" to the coarsest grid, while the residual curve decreases rapidly on the "ascent" to the finest grid. A detailed account of this fine structure should be left to multigrid aficionados! $\diamond\diamond$

The previous experiment indicates that the choice of cycling parameters (ν_0, ν_1, ν_2) for multigrid schemes may not be obvious. They are often chosen *a priori*, based on analysis or prior experimentation, and they remain fixed throughout the course of the algorithm. For certain problems, there are heuristic methods for changing the parameters adaptively [4].

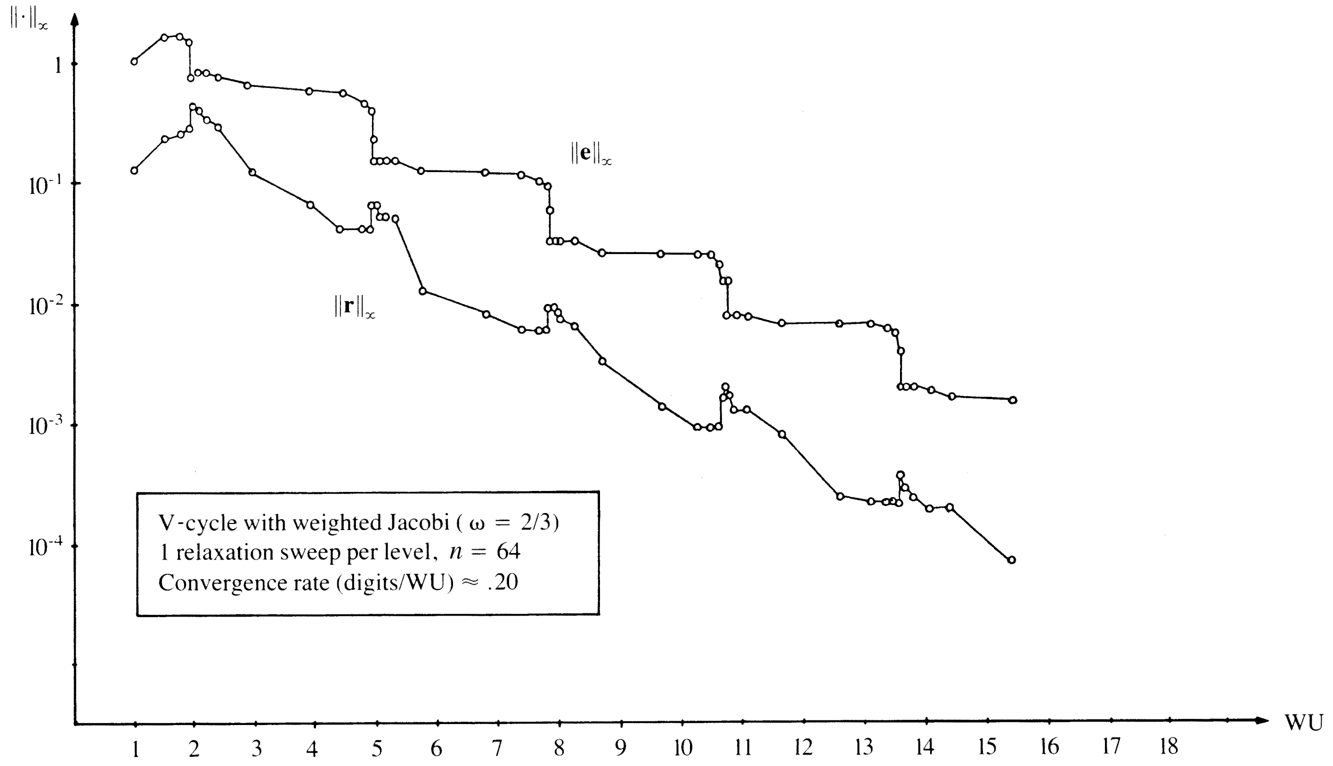

Figure 4.6: (a). *Immediate replacement version of the* V*-cycle scheme applied to the one-dimensional model problem with* $n = 64$ *points with* $\nu_1 = \nu_2 = 1$ *relaxation sweep on each level. The maximum norms of the error and the residual are plotted against WUs over the course of five* V*-cycles.*

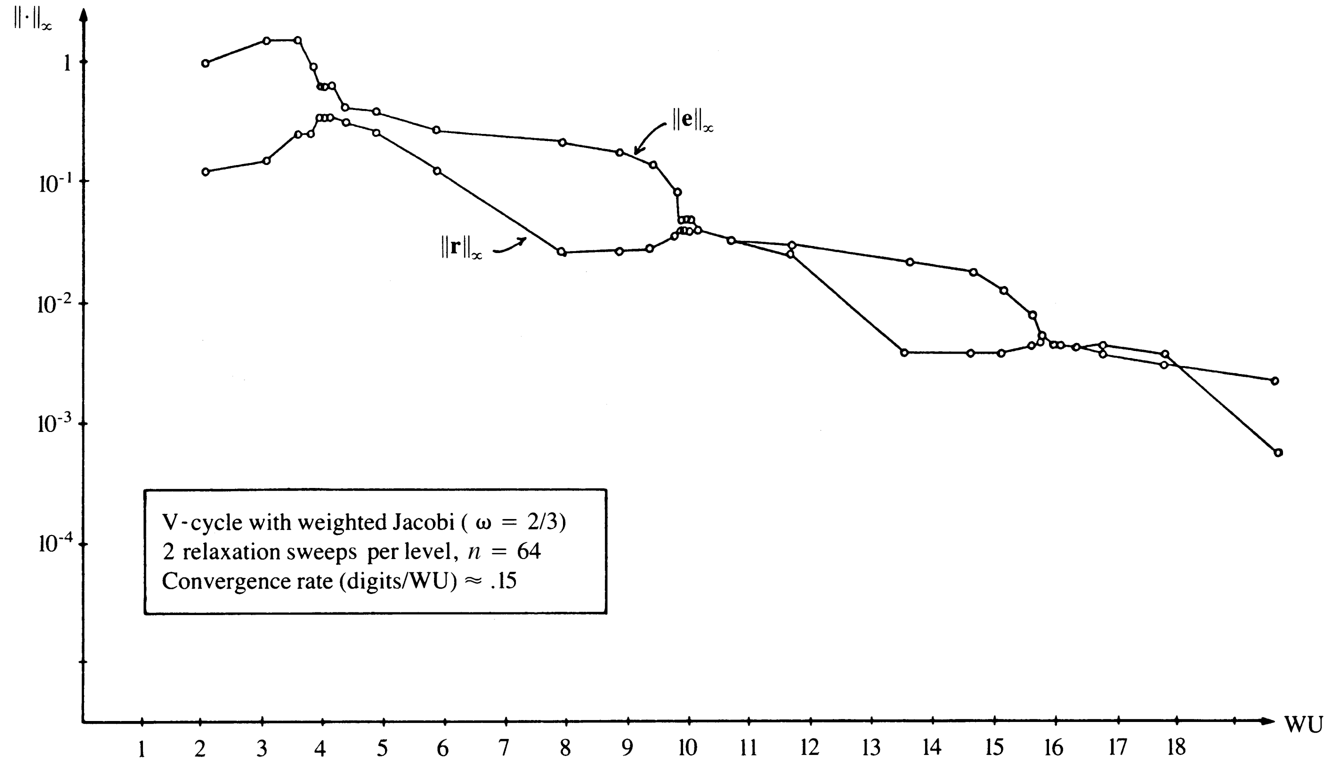

Figure 4.6, continued: (b). *Immediate replacement version of the* V*-cycle scheme applied to the one-dimensional model problem with* $n = 64$ *points with* $\nu_1 = \nu_2 = 2$ *relaxation sweeps on each level. The maximum norms of the error and the residual are plotted against WUs over the course of three* V*-cycles.*

We have devoted much attention to the one-dimensional model problem with the understanding that many of the algorithms, ideas, and results extend directly to higher dimensions. It is useful to mention a few issues that arise only in higher-dimensional problems. For example, the basic relaxation schemes have many more variations. In two dimensions, we can relax by points (updating one unknown at a time in one-dimensional problems) or by lines. In line relaxation, an entire row or column of the grid is updated at once, which generally requires the direct solution of a tridiagonal system for each line. Line relaxation permits additional options, determined by orderings. The lines may be swept forward, backward, or both (symmetric relaxation). The lines may be colored and relaxed alternately in a red-black fashion (often called *zebra relaxation*). Or the grid may be swept alternately by rows and then by columns, giving an *alternating direction* method. Line relaxation will be discussed further in Chapter 7.

These ideas are further generalized in three dimensions. Here we can relax by points, lines, or planes, with various choices for ordering, coloring, and direction. There are also ways to incorporate fast direct solvers for highly structured subproblems that may be imbedded within a relaxation sweep. Many of these possibilities have been analyzed; many more have been implemented. However, there is still room for more work and understanding.

Numerical example. We conclude this chapter with an extensive numerical experiment in which several multigrid methods are applied to the two-dimensional problem

$$\begin{aligned} -u_{xx} - u_{yy} &= 2\big[(1-6x^2)y^2(1-y^2) + (1-6y^2)x^2(1-x^2)\big] \quad \text{in} \quad \Omega, \\ u &= 0 \quad \text{on} \quad \partial\Omega, \end{aligned} \qquad (4.8)$$

where Ω is the unit square, $\{(x,y) : 0 < x < 1,\ 0 < y < 1\}$. Knowing that the analytical solution to this problem is

$$u(x,y) = (x^2 - x^4)(y^4 - y^2),$$

errors can be computed.

It should be mentioned in passing that the convergence properties of the basic relaxation methods carry over directly from one to two dimensions when they are applied to the model problem. Most importantly, weighted Jacobi and Gauss–Seidel retain the property that they smooth high-frequency Fourier modes effectively and leave low-frequency modes relatively unchanged. A guided eigenvalue calculation that leads to these conclusions is given in Exercise 12.

We first use red-black Gauss–Seidel relaxation in a V-cycle scheme on fine grids with $n = 16$, 32, 64, and 128 points in each direction (four separate experiments). Full weighting and linear interpolation are used. We let $\mathbf{e}$ be the vector with components $u(x_i) - v_i^h$ and compute its discrete L^2 norm of the error, $\|\cdot\|_h$. Because the error is not available in most problems, a more practical measure, the discrete L^2 norm of the residual $\mathbf{r}^h$, is also computed.

Table 4.1 shows the residual and error norms after each V-cycle. For each V-cycle, the two columns labeled *ratio* show the ratios of $\|\mathbf{r}^h\|_h$ and $\|\mathbf{e}\|_h$ between successive V-cycles. There are several points of interest. First consider the column

	$n = 16$				$n = 32$			
V-cycle	$\|\mathbf{r}^h\|_h$	ratio	$\|\mathbf{e}\|_h$	ratio	$\|\mathbf{r}^h\|_h$	ratio	$\|\mathbf{e}\|_h$	ratio
0	6.75e+02		5.45e−01		2.60e+03		5.61e−01	
1	4.01e+00	0.01	1.05e−02	0.02	1.97e+01	0.01	1.38e−02	0.02
2	1.11e−01	0.03	4.10e−04	0.04	5.32e−01	0.03	6.32e−04	0.05
3	3.96e−03	0.04	1.05e−04	0.26	2.06e−02	0.04	4.41e−05	0.07
4	1.63e−04	0.04	1.03e−04	0.98*	9.79e−04	0.05	2.59e−05	0.59
5	7.45e−06	0.05	1.03e−04	1.00*	5.20e−05	0.05	2.58e−05	1.00*
6	3.75e−07	0.05	1.03e−04	1.00*	2.96e−06	0.06	2.58e−05	1.00*
7	2.08e−08	0.06	1.03e−04	1.00*	1.77e−07	0.06	2.58e−05	1.00*
8	1.24e−09	0.06	1.03e−04	1.00*	1.10e−08	0.06	2.58e−05	1.00*
9	7.74e−11	0.06	1.03e−04	1.00*	7.16e−10	0.06	2.58e−05	1.00*
10	4.99e−12	0.06	1.03e−04	1.00*	4.79e−11	0.07	2.58e−05	1.00*
11	3.27e−13	0.07	1.03e−04	1.00*	3.29e−12	0.07	2.58e−05	1.00*
12	2.18e−14	0.07	1.03e−04	1.00*	2.31e−13	0.07	2.58e−05	1.00*
13	2.33e−15	0.11	1.03e−04	1.00*	1.80e−14	0.08	2.58e−05	1.00*
14	1.04e−15	0.45	1.03e−04	1.00*	6.47e−15	0.36	2.58e−05	1.00*
15	6.61e−16	0.63	1.03e−04	1.00*	5.11e−15	0.79	2.58e−05	1.00*

	$n = 64$				$n = 128$			
V-cycle	$\|\mathbf{r}^h\|_h$	ratio	$\|\mathbf{e}\|_h$	ratio	$\|\mathbf{r}^h\|_h$	ratio	$\|\mathbf{e}\|_h$	ratio
0	1.06e+04		5.72e−01		4.16e+04		5.74e−01	
1	7.56e+01	0.01	1.39e−02	0.02	2.97e+02	0.01	1.39e−02	0.02
2	2.07e+00	0.03	6.87e−04	0.05	8.25e+00	0.03	6.92e−04	0.05
3	8.30e−02	0.04	4.21e−05	0.06	3.37e−01	0.04	4.22e−05	0.06
4	4.10e−03	0.05	7.05e−06	0.17	1.65e−02	0.05	3.28e−06	0.08
5	2.29e−04	0.06	6.45e−06	0.91*	8.99e−04	0.05	1.63e−06	0.50
6	1.39e−05	0.06	6.44e−06	1.00*	5.29e−05	0.06	1.61e−06	0.99*
7	8.92e−07	0.06	6.44e−06	1.00*	3.29e−06	0.06	1.61e−06	1.00*
8	5.97e−08	0.07	6.44e−06	1.00*	2.14e−07	0.06	1.61e−06	1.00*
9	4.10e−09	0.07	6.44e−06	1.00*	1.43e−08	0.07	1.61e−06	1.00*
10	2.87e−10	0.07	6.44e−06	1.00*	9.82e−10	0.07	1.61e−06	1.00*
11	2.04e−11	0.07	6.44e−06	1.00*	6.84e−11	0.07	1.61e−06	1.00*
12	1.46e−12	0.07	6.44e−06	1.00*	4.83e−12	0.07	1.61e−06	1.00*
13	1.08e−13	0.07	6.44e−06	1.00*	3.64e−13	0.08	1.61e−06	1.00*
14	2.60e−14	0.24	6.44e−06	1.00*	1.03e−13	0.28	1.61e−06	1.00*
15	2.30e−14	0.88	6.44e−06	1.00*	9.19e−14	0.89	1.61e−06	1.00*

Table 4.1: *The* V(2,1) *scheme with red-black Gauss–Seidel applied to a two-dimensional problem on fine grids with* $n = 16$, 32, 64, *and* 128 *points. The discrete* L^2 *norms of the residual and error are shown after each* V*-cycle. The* ratio *columns give the ratios of residual and error norms of successive* V*-cycles. The* * *in the error ratio column indicates that the level of discretization error has been reached.*

of error norms. For each of the four grid sizes, the error norms decrease rapidly and then level off abruptly as the scheme reaches the level of discretization error. We confirm this by comparing the final error norms, $\|\mathbf{e}\|_h$, on the four grids ($1.03\text{e}-04$, $2.58\text{e}-05$, $6.44\text{e}-06$, and $1.61\text{e}-06$). These norms decrease by a factor of four as the resolution doubles, which is consistent with the second-order discretization we have used. The residual norms also decrease rapidly for 12 to 14 V-cycles, with the value in the corresponding *ratio* column reaching a nearly constant value, until the last few cycles. This constant value is a good estimate of the asymptotic convergence factor of the scheme (approximately 0.07) for this problem. The sharp increase in the residual norm ratio during the last two cycles reflects the fact that the algebraic approximation is already accurate to near machine precision.

In the course of our development, we described several different schemes for relaxation, restriction, and interpolation. Specifically, we worked with weighted Jacobi, Gauss–Seidel, and red-black Gauss–Seidel relaxation schemes; injection and full weighting restriction operators; and linear interpolation. We now investigate how various combinations of these schemes perform when used in V-schemes applied to model problem (4.8).

For completeness, we introduce two more schemes, *half-injection* and *cubic interpolation.* Half-injection, as the name implies, is simply defined in one dimension by $v_j^{2h} = 0.5v_{2j}^h$, with a similar extension to two dimensions. Half-injection is designed for use on the model problem with red-black relaxation and may be understood most easily by considering the one-dimensional case. The idea is that because one sweep of the red-black relaxation produces zero residuals at every other point, full-weighting and half-injection are equivalent. We will see shortly that the scheme indeed works well for this case.

Cubic interpolation is one of the many interpolation schemes that could be applied and is probably the most commonly used in multigrid after linear interpolation. As its name implies, the method interpolates cubic (or lower degree) polynomials exactly. It represents a good compromise between the desire for greater interpolation accuracy and the increase in computational cost required to achieve the desired accuracy. In one dimension, the basic cubic interpolation operator is given by

$$\begin{aligned} v_{2j}^h &= v_j^{2h}, \\ v_{2j+1}^h &= \frac{1}{16}\left(-v_{j-1}^{2h} + 9v_j^{2h} + 9v_{j+1}^{2h} - v_{j+2}^{2h}\right). \end{aligned}$$

Table 4.2 gives comparative results for many experiments. For each of the Jacobi, Gauss–Seidel, and red-black Gauss–Seidel relaxation schemes, we performed six V-cycles, using all possible combinations of the half-injection, injection, and full weighting restriction operators with linear or cubic interpolations. In each case, the experiment was performed using (1,0), (1,1), and (2,1) V-cycles, where (ν_1, ν_2) indicates ν_1 pre-correction relaxation sweeps and ν_2 post-correction relaxation sweeps. The entries in the table give the average convergence factor for the last five V-cycles of each experiment. A dash indicates that a particular scheme diverged. The entries in the *cost* line reflect the cost of the method, in terms of operation count, shown as a multiple of the cost of the (1,0) scheme using linear interpolation and injection. Notice that the cost is independent of the different relaxation schemes, as they all perform the same number of operations.

	Relaxation	Injection		Full Weighting		Half-Injection	
(ν_1,ν_2)	Scheme	Linear	Cubic	Linear	Cubic	Linear	Cubic
(1,0)	Jacobi	–	–	0.49	0.49	0.55	0.62
	GS	0.89	0.66	0.33	0.34	0.38	0.37
	RBGS	–	–	0.21	0.23	0.45	0.42
	Cost	1.00	1.25	1.13	1.39	1.01	1.26
(1,1)	Jacobi	0.94	0.56	0.35	0.34	0.54	0.52
	GS	0.16	0.16	0.14	0.14	0.45	0.43
	RBGS	–	–	0.06	0.05	0.12	0.16
	Cost	1.49	1.75	1.63	1.88	1.51	1.76
(2,1)	Jacobi	0.46	0.31	0.24	0.24	0.46	0.45
	GS	0.07	0.07	0.08	0.07	0.40	0.39
	RBGS	–	–	0.04	0.03	0.03	0.07
	Cost	1.99	2.24	2.12	3.37	1.51	1.76

Table 4.2: *Average convergence factors over five* V*-cycles on model problem* (4.8) *for various combinations of relaxation, restriction, and interpolation operators. The dashes indicate divergent schemes. The* cost *line gives the computational cost of a* V(ν_1,ν_2)*-cycle scheme using the specified choice of restriction and interpolation, as a multiple of the cost of a* (1,0) V*-cycle scheme using injection and linear interpolation.*

A few observations are in order. At least for this problem, cubic interpolation is only noticeably more effective than linear interpolation when injection is used as the restriction operator. It is also apparent that half-injection is useful only with red-black Gauss–Seidel, as expected; even then, several smoothing sweeps are required. Finally, and not surprisingly, the table indicates that you get what you pay for: combinations that produce the best convergence factors are also those with the higher costs. Parameter selection is largely the art of finding a compromise between performance and cost.

The final set of experiments concerns the effectiveness of FMG schemes and, in particular, whether FMG schemes are more or less effective than V-cycles. Table 4.3 describes the performance of three FMG schemes. Square grids with up to $n = 2048$ points in each direction are used. The FMG(ν_1,ν_2) scheme uses ν_1 relaxation sweeps on the descent phase and ν_2 relaxation sweeps on the ascent phase of each V-cycle. Red-black Gauss–Seidel relaxation is used with full weighting and linear interpolation. In each case, only one complete FMG cycle is done.

The table shows the discrete L^2 norm of the error for each scheme. Evidence that the FMG code solves the problem to the level of discretization error on each grid is that the ratio of the error norms between successive levels is roughly 0.25 (for this two-dimensional problem). If the ratio is noticeably greater than 0.25, the solution is probably not accurate to the level of discretization. Based on this observation, we suspect that the FMG(1,0) scheme does not solve the problem to the level of discretization error on any level. This is confirmed when we observe the FMG(1,1) and FMG(2,1) schemes, which *do* solve to the level of the discretization error on all levels. The FMG(2,1) scheme requires more work than the FMG(1,1) with little gain in accuracy; so it appears that FMG(1,1) is the best choice for this problem.

	FMG(1,0)		FMG(1,1)		FMG(2,1)		FMG(1,1)	V(2,1)	V(2,1)
N	$\|\mathbf{e}\|_h$	ratio	$\|\mathbf{e}\|_h$	ratio	$\|\mathbf{e}\|_h$	ratio	WU	cycles	WU
2	5.86e−03		5.86e−03		5.86e−03				
4	5.37e−03	0.917	2.49e−03	0.424	2.03e−03	0.347	7/2	3	12
8	2.78e−03	0.518	9.12e−04	0.367	6.68e−04	0.328	7/2	4	16
16	1.19e−03	0.427	2.52e−04	0.277	1.72e−04	0.257	7/2	4	16
32	4.70e−04	0.395	6.00e−05	0.238	4.00e−05	0.233	7/2	5	20
64	1.77e−04	0.377	1.36e−05	0.227	9.36e−06	0.234	7/2	5	20
128	6.49e−05	0.366	3.12e−06	0.229	2.26e−06	0.241	7/2	6	24
256	2.33e−05	0.359	7.35e−07	0.235	5.56e−07	0.246	7/2	7	28
512	8.26e−06	0.354	1.77e−07	0.241	1.38e−07	0.248	7/2	7	28
1024	2.90e−06	0.352	4.35e−08	0.245	3.44e−08	0.249	7/2	8	32
2048	1.02e−06	0.351	1.08e−08	0.247	8.59e−09	0.250	7/2	9	36

Table 4.3: *Three different FMG schemes applied to the two-dimensional problem on square grids with up to* $n = 2048$ *points in each direction. The FMG(*ν_1*,*ν_2*) scheme uses* ν_1 *red-black Gauss–Seidel relaxation sweeps on the descent phase and* ν_2 *relaxation sweeps on the ascent phase of each* V*-cycle. The discrete* L^2 *norm of the error and the ratio of errors at each grid level are shown. Solution to the level of discretization error is indicated when the* ratio *column shows a reduction of at least* 0.25 *in the error norm. For comparison, the* V-cycles *column shows the number of* V(2,1)*-cycles needed to converge to the level of discretization error, while the* V-cycle WU *column shows the number of work units needed to converge to the level of discretization error.*

A useful question is whether an FMG(1,1) scheme is more efficient than, say, the V(2,1) scheme in achieving a solution accurate to the level of discretization error. We answer this question by performing the V(2,1) method (as in Table 4.1) for all grid sizes from $n = 4$ through $n = 2048$ (over 4 million fine-grid points!) and recording the number of V-cycles required to converge to the level of discretization error. These results are presented in the second-to-last column of Table 4.3. It is apparent that the number of cycles required to solve to the level of discretization error increases with the problem size.

We can now make some comparisons. Recall our discussion of computational costs earlier in the chapter. We determined that a (1,1) V-cycle in $d = 2$ dimensions costs about $\frac{8}{3}$ WUs (Exercise 3); therefore, a (2,1) V-cycle costs half again as much, or 4 WU. The last column of Table 4.3 shows the costs in WUs of solving to the level of discretization error with the V(2,1) scheme on various grids. We also saw (Exercise 4) that the FMG(1,0) scheme, which did not converge to the level of discretization error in this case, requires just under 2 WUs, while the FMG(1,1) and FMG(2,1) schemes, which did achieve the desired accuracy, require approximately $\frac{7}{2}$ and $\frac{16}{3}$ WUs, respectively; these costs are the same for all grid sizes. Thus, on all of the grids shown in Table 4.3, the FMG(1,1) scheme is significantly less expensive in WUs than the V(2,1) scheme. This confirms the observation that for converging to the level of discretization error, full multigrid methods are generally preferable to simple V-cycles. ◇◇

Exercises

Data Structures and Complexity

1. **Data structures.** Work out the details of the data structures given in Fig. 4.1. Assume that for a one-dimensional problem, the finest grid has $n - 1 =$

$2^L - 1$ interior points. Let $h = \frac{1}{n}$ be the grid spacing on Ω^h. Let level l have grid spacing $2^{l-1}h$. As suggested in the text, store the approximations $\mathbf{v}^h, \mathbf{v}^{2h}, \ldots$ contiguously in a single array $\mathbf{v}$, with the level L values stored in v_1, v_2, v_3; the level $L-1$ values in $v_4, \ldots, v_8$; etc. Use a similar arrangement for the right-side values $\mathbf{f}^h$, $\mathbf{f}^{2h}, \ldots$. How many values are stored on level l, where $1 \le l \le L$? What is the starting index in the $\mathbf{v}$ array for the level l values, where $1 \le l \le L$?

2. **Data structure for two dimensions.** Now consider the two-dimensional model problem. The one-dimensional data structure may be retained in the main program. However, the initial index for each grid will now be different. Compute these indices, assuming that on the finest grid Ω^h there are $(n-1)^2$ interior points, where $n - 1 = 2^L - 1$.

3. **Storage requirements.** Verify the statement in the text that for a one-dimensional problem ($d = 1$), the storage requirement is less than twice that of the fine-grid problem alone. Show that for problems in two or more dimensions, the requirement drops to less than $\frac{4}{3}$ of the fine-grid problem alone.

4. **V-cycle computation cost.** Verify the statement in the text that a single V-cycle costs about 4 WUs for a one-dimensional ($d = 1$) problem, about $\frac{8}{3}$ WUs for $d = 2$, and $\frac{16}{7}$ WUs for $d = 3$.

5. **FMG computation cost.** Verify the statement in the text that an FMG cycle costs 8 WUs for a one-dimensional problem; the cost is about $\frac{7}{2}$ WUs for $d = 2$ and $\frac{5}{2}$ WUs for $d = 3$.

Local Mode Analysis

6. **One-dimensional weighted Jacobi.**

   (a) Verify the Jacobi updating step (4.2).

   (b) Show that the error $e_j = u(x_j) - v_j$ satisfies (4.3).

   (c) Verify that the amplification factor for the method is given by

$$G(\theta) = 1 - 2\omega \sin^2\left(\frac{\theta}{2}\right).$$

7. **One-dimensional Gauss–Seidel.** Verify the error updating step (4.4). Then show that the amplification factor for the method is given by

$$G(\theta) = \frac{e^{i\theta}}{2 - e^{-i\theta}}.$$

8. **Two-dimensional weighted Jacobi.**

   (a) Verify the error updating step (4.6).

   (b) Show that the amplification factor for the method is given by

$$G(\theta_1, \theta_2) = 1 - \omega\left(\sin^2\left(\frac{\theta_1}{2}\right) + \sin^2\left(\frac{\theta_2}{2}\right)\right).$$

(c) Show that the optimal smoothing factor is obtained with $\omega = \frac{4}{5}$ and that its value is $\mu = |G(\pm\pi, \pm\pi)| = 0.6$. Hint: Note that $\mu(\omega) = \max |G(\theta_1, \theta_2)|$ is a function of ω. The optimal value of ω is that which minimizes μ, viewed as a function of ω. The substitutions $\xi = \sin^2(\frac{\theta_1}{2})$, $\eta = \sin^2(\frac{\theta_2}{2})$ may be helpful.

9. **Two-dimensional Gauss–Seidel.**

(a) Verify the error updating step (4.7).

(b) Show that the amplification factor for the method is given by

$$G(\theta_1, \theta_2) = \frac{e^{\iota\theta_1} + e^{\iota\theta_2}}{4 - e^{-\iota\theta_1} - e^{-\iota\theta_2}}.$$

(c) Show that the smoothing factor is given by

$$\mu = G\left(\frac{\pi}{2}, \cos^{-1}\left(\frac{4}{5}\right)\right) = \frac{1}{2}.$$

10. **Nine-point stencil.** Consider the nine-point stencil for the operator $-u_{xx} - u_{yy}$ given by

$$\frac{1}{3h^2}\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}.$$

Find the amplification factors for the weighted Jacobi method and Gauss–Seidel relaxation applied to this system.

11. **Anisotropic operator.** Consider the five-point stencil for the operator $-\epsilon u_{xx} - u_{yy}$ given by

$$\frac{1}{h^2}\begin{pmatrix} 0 & -1 & 0 \\ -\epsilon & 2(1+\epsilon) & -\epsilon \\ 0 & -1 & 0 \end{pmatrix}.$$

Find the amplification factors for weighted Jacobi and Gauss–Seidel applied to this system. Discuss the effect of the parameter ϵ in the case that $\epsilon << 1$.

12. **Eigenvalue calculation in two dimensions.** Consider the weighted Jacobi method applied to the model Poisson equation in two dimensions on the unit square. Assume a uniform grid of $h = \frac{1}{n}$ in each direction.

(a) Let v_{ij} be the approximation to the solution at the grid point (x_i, y_j). Write the (i,j)th equation of the corresponding discrete problem, where $1 \le i, j \le n-1$.

(b) Letting A be the matrix of coefficients for the discrete system, write the (i,j)th equation for the eigenvalue problem $A\mathbf{v} = \lambda\mathbf{v}$.

(c) Assume an eigenvector solution of the form

$$v_{ij} = \sin\left(\frac{ik\pi}{n}\right)\sin\left(\frac{j\ell\pi}{n}\right), \quad 1 \le k, \ell \le n-1.$$

Using sine addition rules, simplify this eigenvalue equation, cancel common terms, and show that the eigenvalues are

$$\lambda_{k\ell} = 4\left[\sin^2\left(\frac{k\pi}{2n}\right) + \sin^2\left(\frac{\ell\pi}{2n}\right)\right], \quad 1 \le k, \ell \le n-1.$$

(d) As in the one-dimensional case, note that the iteration matrix of the weighted Jacobi method is given by $P_\omega = I - \omega D^{-1}A$, where D corresponds to the diagonal terms of A. Find the eigenvalues of P_ω.

(e) Using a graphing utility, find a suitable way to present the two-dimensional set of eigenvalues (either a surface plot or multiple curves). Plot the eigenvalues for $\omega = \frac{2}{3}, \frac{4}{5}, 1$, and $n = 16$.

(f) In each case, discuss the effect of the weighted Jacobi method on low- and high-frequency modes. Be sure to note that modes can have a high frequencies in one direction and low frequencies in the other direction.

(g) What do you conclude about the optimal value of ω for the two-dimensional problem?

Implementation

13. **V-Cycle program.** Develop a V-cycle program for the one-dimensional model problem. Write a subroutine for each individual component of the algorithm as follows.

(a) Given an approximation array $\mathbf{v}$, a right-side array $\mathbf{f}$, and a level number $1 \le l \le L$, write a subroutine that will carry out ν weighted Jacobi sweeps on level l.

(b) Given an array $\mathbf{f}$ and a level number $1 \le l \le L-1$, write a subroutine that will carry out full weighting between level l and level $l+1$.

(c) Given an array $\mathbf{v}$ and a level number $2 \le l \le L$, write a subroutine that will carry out linear interpolation between level l and level $l-1$.

(d) Write a driver program that initializes the data arrays and carries out a V-cycle by calling the three preceding subroutines. The program should be tested on simple problems for which the exact solution is known. For example, for fixed k, take $f(x) = C\sin(k\pi x)$ on the interval $0 \le x \le 1$, where C is a constant. Then

$$u(x) = \frac{C}{\pi^2 k^2 + \sigma}\sin(k\pi x)$$

is an exact solution to model problem (1.1). Another subroutine that computes norms of errors and residuals will be useful.

14. **Modification of V-cycle code.** It is now easy to modify this program and make comparisons.

(a) Vary ν and vary the number of V-cycles.

(b) Replace the weighted Jacobi subroutine, first by a Gauss–Seidel subroutine, then by a red-black Gauss–Seidel subroutine.

(c) Replace the full weighting subroutine by an injection subroutine. Observe that using red-black Gauss–Seidel and injection impairs convergence. Try to remedy this by using half-injection. In all cases, determine experimentally how these changes affect convergence rates and computation time.

(d) Explain the result of using black-red (rather than red-black) Gauss–Seidel with injection.

15. **Two-dimensional program.** For the two-dimensional problem, proceed again in a modular way.

    (a) Write a subroutine that performs weighted Jacobi on a two-dimensional grid. Within the subroutine, it is easiest to refer to $\mathbf{v}$ and $\mathbf{f}$ as two-dimensional arrays.

    (b) Make the appropriate modifications to the one-dimensional code to implement bilinear interpolation and full weighting on a two-dimensional grid.

    (c) Make the (minor) changes required in the main program to create a two-dimensional V-cycle program. Test this program on problems with known exact solutions. For example, for fixed k and ℓ, take $f(x,y) = C\sin(k\pi x)\sin(\ell\pi y)$ on the unit square ($0 \le x, y \le 1$), where C is a constant. Then

    $$u(x,y) = \frac{C}{\pi^2 k^2 + \pi^2 \ell^2 + \sigma}\sin(k\pi x)\sin(\ell\pi y)$$

    is an exact soultion to model problem (1.4).

16. **FMG programs.** Modify the one- and two-dimensional V-cycle programs to carry out the FMG scheme.

17. **A convection-diffusion problem.** Consider the following convection-diffusion problem on the unit square $\Omega = \{(x,y) : 0 < x < 1, 0 < y < 1\}$:

    $$\begin{aligned} -\epsilon(u_{xx} + u_{yy}) + au_x &= A\sin(\ell\pi y)(C_2x^2 + C_1x + C_0) \quad \text{on } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

    where $\epsilon > 0$, $A \in \mathbf{R}$, $a \in \mathbf{R}$, ℓ is an integer, $C_2 = -\epsilon\ell^2\pi^2$, $C_1 = \epsilon\ell^2\pi^2 - 2a$, and $C_0 = a + 2\epsilon$. It has the exact solution $u(x,y) = Ax(1-x)\sin(\ell\pi y)$. Apply the multigrid algorithms discussed in this chapter to this problem. Compare the algorithms and explore how their performance changes for $\epsilon = 0.01, 0.1, 1$; $a = 0.1, 1, 10$; $n = 32, 64, 128$; $l = 1, 3, 16$.

18. **Discrete L^2 norm**. Let $u(x) = x^{m/2}$, where $m > -1$ is an integer, on $\Omega = [0,1]$, with grid spacing $h = \frac{1}{n}$. Let $u_i^h = x_i^{m/2} = (ih)^{m/2}$. Show that the continuous L^2 norm is

    $$\|u\|_2 = \left(\int_0^1 x^{m/2}\right)^{1/2} = \frac{1}{\sqrt{m+1}},$$

    while the corresponding discrete L^2 norm satisfies

    $$\|\mathbf{u}^h\|_h \overset{h\to 0}{=} \frac{1}{\sqrt{m+1}}.$$