Finite Difference Methods for Ordinary and Partial Differential Equations

Finite Difference Methods for Ordinary and Partial Differential Equations

Steady-State and Time-Dependent Problems

Randall J. LeVeque
University of Washington
Seattle, Washington

siam Society for Industrial and Applied Mathematics • Philadelphia

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

LeVeque, Randall J., 1955-
Finite difference methods for ordinary and partial differential equations : steady-state and time-dependent problems / Randall J. LeVeque.
p.cm.
Includes bibliographical references and index.
ISBN 978-0-898716-29-0 (alk. paper)
1. Finite differences. 2. Differential equations. I. Title.

QA431.L548 2007
515'.35—dc22 2007061732

Partial royalties from the sale of this book are placed in a fund to help students attend SIAM meetings and other SIAM-related activities. This fund is administered by SIAM, and qualified individuals are encouraged to write directly to SIAM for guidelines.

To my family,
Loyce, Ben, Bill, and Ann

Contents

Preface

This book evolved from lecture notes developed over the past 20+ years of teaching this material, mostly in Applied Mathematics 585–6 at the University of Washington. The course is taken by first-year graduate students in our department, along with graduate students from mathematics and a variety of science and engineering departments.

Exercises and student projects are an important aspect of any such course and many have been developed in conjunction with this book. Rather than lengthening the text, they are available on the book's Web page:

```
www.siam.org/books/OT98
```

Along with exercises that provide practice and further exploration of the topics in each chapter, some of the exercises introduce methods, techniques, or more advanced topics not found in the book.

The Web page also contains MATLAB® m-files that illustrate how to implement finite difference methods, and that may serve as a starting point for further study of the methods in exercises and projects. A number of the exercises require programming on the part of the student, or require changes to the MATLAB programs provided. Some of these exercises are fairly simple, designed to enable students to observe first hand the behavior of numerical methods described in the text. Others are more open-ended and could form the basis for a course project.

The exercises are available as PDF files. The LaTeX source is also provided, along with some hints on using LaTeX for the type of mathematics used in this field. Each exercise is in a separate file so that instuctors can easily construct customized homework assignments if desired. Students can also incorporate the source into their solutions if they use LaTeX to typeset their homework. Personally I encourage this when teaching the class, since this is a good opportunity for them to learn a valuable skill (and also makes grading homework considerably more pleasurable).

Organization of the Book

The book is organized into two main parts and a set of appendices. Part I deals with steady-state boundary value problems, starting with two-point boundary value problems in one dimension and then elliptic equations in two and three dimensions. Part I concludes with a chapter on iterative methods for large sparse linear systems, with an emphasis on systems arising from finite difference approximations.

Part II concerns time-dependent problems, starting with the initial value problem for ODEs and moving on to initial-boundary value problems for parabolic and hyperbolic PDEs. This part concludes with a chapter on mixed equations combining features of ordinary differential equations (ODEs) and parabolic and hyperbolic equations.

Part III consists of a set of appendices covering background material that is needed at various points in the main text. This material is collected at the end to avoid interrupting the flow of the main text and because many concepts are repeatedly used in different contexts in Parts I and II.

The organization of this book is somewhat different from the way courses are structured at many universities, where a course on ODEs (including both two-point boundary value problems and the initial value problem) is followed by a course on partial differential equations (PDEs) (including both elliptic boundary value problems and time-dependent hyperbolic and parabolic equations). Existing textbooks are well suited to this latter approach, since many books cover numerical methods for ODEs or for PDEs, but often not both. However, I have found over the years that the reorganization into boundary value problems followed by initial value problems works very well. The mathematical techniques are often similar for ODEs and PDEs and depend more on the steady-state versus time-dependent nature of the problem than on the number of dimensions involved. Concepts developed for each type of ODE are naturally extended to PDEs and the interplay between these theories is more clearly elucidated when they are covered together.

At the University of Washington, Parts I and II of this book are used for the second and third quarters of a year-long graduate course. Lectures are supplemented by material from the appendices as needed. The first quarter of the sequence covers direct methods for linear systems, eigenvalue problems, singular values, and so on. This course is currently taught out of Trefethen and Bau [91], which also serves as a useful reference text for the material in this book on linear algebra and iterative methods.

It should also be possible to use this book for a more traditional set of courses, teaching Chapters 1, 5, 6, 7, and 8 in an ODE course followed by Chapters 2, 3, 9, 10, and 11 in a PDE-oriented course.

Emphasis of the Book

The emphasis is on building an understanding of the essential ideas that underlie the development, analysis, and practical use of finite difference methods. Stability theory necessarily plays a large role, and I have attempted to explain several key concepts, their relation to one another, and their practical implications. I include some proofs of convergence in order to motivate the various definitions of "stability" and to show how they relate to error estimates, but have not attempted to rigorously prove all results in complete generality. I have also tried to give an indication of some of the more practical aspects of the algorithms without getting too far into implementation details. My goal is to form a foundation from which students can approach the vast literature on more advanced topics and further explore the theory and/or use of finite difference methods according to their interests and needs.

I am indebted to several generations of students who have worked through earlier versions of this book, found errors and omissions, and forced me to constantly rethink my understanding of this material and the way I present it. I am also grateful to many

colleagues who have taught out of my notes and given me valuable feedback, both at the University of Washington and at more than a dozen other universities where earlier versions have been used in courses. I take full responsibility for the remaining errors.

I have also been influenced by other books covering these same topics, and many excellent ones exist at all levels. Advanced books go into more detail on countless subjects only briefly discussed here, and I give pointers to some of these in the text. There are also a number of general introductory books that may be useful as complements to the presentation found here, including, for example, [27], [40], [49], [72], [84], and [93].

As already mentioned, this book has evolved over the past 20 years. This is true in part for the mundane reason that I have reworked (and perhaps improved) parts of it each time I teach the course. But it is also true for a more exciting reason—the field itself continues to evolve in significant ways. While some of the theory and methods in this book were very well known when I was a student, many of the topics and methods that should now appear in an introductory course had yet to be invented or were in their infancy. I give at least a flavor of some of these, though many other developments have not been mentioned. I hope that students will be inspired to further pursue the study of numerical methods, and perhaps invent even better methods in the future.

Randall J. LeVeque

Part I

Boundary Value Problems and Iterative Methods

Chapter 1
Finite Difference Approximations

Our goal is to approximate solutions to differential equations, i.e., to find a function (or some discrete approximation to this function) that satisfies a given relationship between various of its derivatives on some given region of space and/or time, along with some boundary conditions along the edges of this domain. In general this is a difficult problem, and only rarely can an analytic formula be found for the solution. A finite difference method proceeds by replacing the derivatives in the differential equations with finite difference approximations. This gives a large but finite algebraic system of equations to be solved in place of the differential equation, something that can be done on a computer.

Before tackling this problem, we first consider the more basic question of how we can approximate the derivatives of a known function by finite difference formulas based only on values of the function itself at discrete points. Besides providing a basis for the later development of finite difference methods for solving differential equations, this allows us to investigate several key concepts such as the *order of accuracy* of an approximation in the simplest possible setting.

Let $u(x)$ represent a function of one variable that, unless otherwise stated, will always be assumed to be smooth, meaning that we can differentiate the function several times and each derivative is a well-defined bounded function over an interval containing a particular point of interest $\bar{x}$.

Suppose we want to approximate $u'(\bar{x})$ by a finite difference approximation based only on values of u at a finite number of points near $\bar{x}$. One obvious choice would be to use

$$D_+u(\bar{x}) \equiv \frac{u(\bar{x}+h) - u(\bar{x})}{h} \tag{1.1}$$

for some small value of h. This is motivated by the standard definition of the derivative as the limiting value of this expression as $h \to 0$. Note that $D_+u(\bar{x})$ is the slope of the line interpolating u at the points $\bar{x}$ and $\bar{x}+h$ (see Figure 1.1).

The expression (1.1) is a *one-sided* approximation to u' since u is evaluated only at values of $x \geq \bar{x}$. Another one-sided approximation would be

$$D_-u(\bar{x}) \equiv \frac{u(\bar{x}) - u(\bar{x}-h)}{h}. \tag{1.2}$$

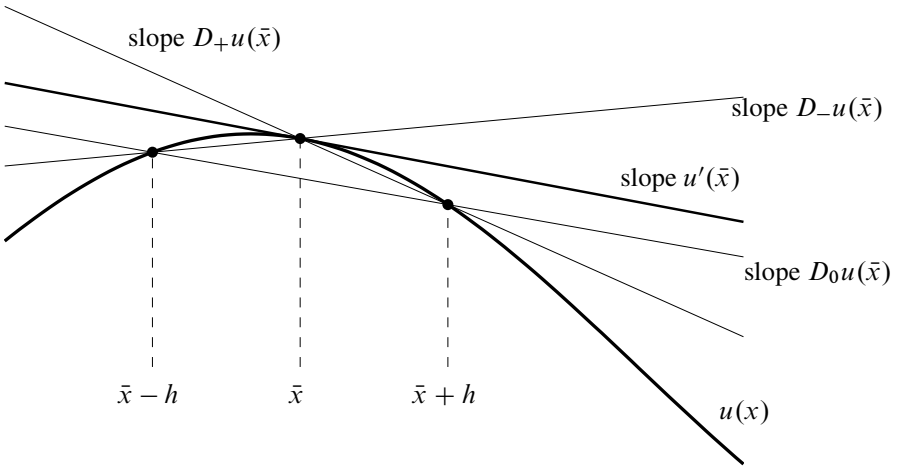


Figure 1.1. *Various approximations to $u'(\bar{x})$ interpreted as the slope of secant lines.*

Each of these formulas gives a *first order accurate* approximation to $u'(\bar{x})$, meaning that the size of the error is roughly proportional to h itself.

Another possibility is to use the *centered approximation*

$$D_0u(\bar{x}) \equiv \frac{u(\bar{x}+h) - u(\bar{x}-h)}{2h} = \frac{1}{2}(D_+u(\bar{x}) + D_-u(\bar{x})). \tag{1.3}$$

This is the slope of the line interpolating u at $\bar{x}-h$ and $\bar{x}+h$ and is simply the average of the two one-sided approximations defined above. From Figure 1.1 it should be clear that we would expect $D_0u(\bar{x})$ to give a better approximation than either of the one-sided approximations. In fact this gives a *second order accurate* approximation—the error is proportional to h^2 and hence is much smaller than the error in a first order approximation when h is small.

Other approximations are also possible, for example,

$$D_3u(\bar{x}) \equiv \frac{1}{6h}[2u(\bar{x}+h) + 3u(\bar{x}) - 6u(\bar{x}-h) + u(\bar{x}-2h)]. \tag{1.4}$$

It may not be clear where this came from or why it should approximate u' at all, but in fact it turns out to be a third order accurate approximation—the error is proportional to h^3 when h is small.

Our first goal is to develop systematic ways to derive such formulas and to analyze their accuracy and relative worth. First we will look at a typical example of how the errors in these formulas compare.

Example 1.1. Let $u(x) = \sin(x)$ and $\bar{x} = 1$; thus we are trying to approximate $u'(1) = \cos(1) = 0.5403023$. Table 1.1 shows the error $Du(\bar{x}) - u'(\bar{x})$ for various values of h for each of the formulas above.

We see that D_+u and D_-u behave similarly although one exhibits an error that is roughly the negative of the other. This is reasonable from Figure 1.1 and explains why D_0u, the average of the two, has an error that is much smaller than both.

Table 1.1. *Errors in various finite difference approximations to $u'(\bar{x})$.*

h	$D_+u(\bar{x})$	$D_-u(\bar{x})$	$D_0u(\bar{x})$	$D_3u(\bar{x})$
1.0e−01	−4.2939e−02	4.1138e−02	−9.0005e−04	6.8207e−05
5.0e−02	−2.1257e−02	2.0807e−02	−2.2510e−04	8.6491e−06
1.0e−02	−4.2163e−03	4.1983e−03	−9.0050e−06	6.9941e−08
5.0e−03	−2.1059e−03	2.1014e−03	−2.2513e−06	8.7540e−09
1.0e−03	−4.2083e−04	4.2065e−04	−9.0050e−08	6.9979e−11

We see that

$$\begin{aligned} D_+u(\bar{x}) - u'(\bar{x}) &\approx -0.42h, \\ D_0u(\bar{x}) - u'(\bar{x}) &\approx -0.09h^2, \\ D_3u(\bar{x}) - u'(\bar{x}) &\approx 0.007h^3, \end{aligned}$$

confirming that these methods are first order, second order, and third order accurate, respectively.

Figure 1.2 shows these errors plotted against h on a log-log scale. This is a good way to plot errors when we expect them to behave like some power of h, since if the error $E(h)$ behaves like

$$E(h) \approx Ch^p,$$

then

$$\log|E(h)| \approx \log|C| + p\log h.$$

So on a log-log scale the error behaves linearly with a slope that is equal to p, the order of accuracy.

1.1 Truncation errors

The standard approach to analyzing the error in a finite difference approximation is to expand each of the function values of u in a *Taylor series* about the point $\bar{x}$, e.g.,

$$u(\bar{x}+h) = u(\bar{x}) + hu'(\bar{x}) + \frac{1}{2}h^2u''(\bar{x}) + \frac{1}{6}h^3u'''(\bar{x}) + O(h^4), \tag{1.5a}$$

$$u(\bar{x}-h) = u(\bar{x}) - hu'(\bar{x}) + \frac{1}{2}h^2u''(\bar{x}) - \frac{1}{6}h^3u'''(\bar{x}) + O(h^4). \tag{1.5b}$$

These expansions are valid provided that u is sufficiently smooth. Readers unfamiliar with the "big-oh" notation $O(h^4)$ are advised to read Section A.2 of Appendix A at this point since this notation will be heavily used and a proper understanding of its use is critical.

Using (1.5a) allows us to compute that

$$D_+u(\bar{x}) = \frac{u(\bar{x}+h) - u(\bar{x})}{h} = u'(\bar{x}) + \frac{1}{2}hu''(\bar{x}) + \frac{1}{6}h^2u'''(\bar{x}) + O(h^3).$$

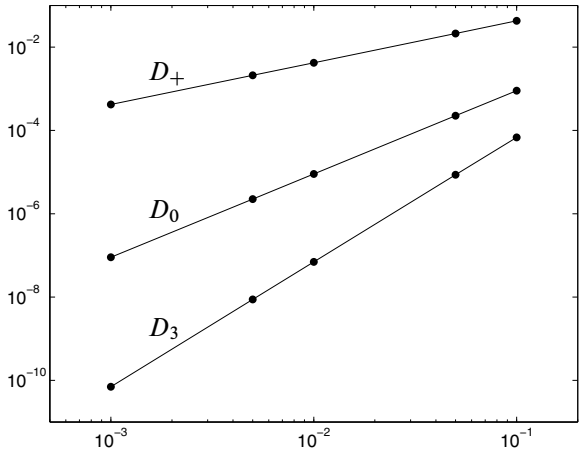


Figure 1.2. *The errors in* $Du(\bar{x})$ *from Table* 1.1 *plotted against* h *on a log-log scale.*

Recall that $\bar{x}$ is a fixed point so that $u''(\bar{x})$, $u'''(\bar{x})$, etc., are fixed constants independent of h. They depend on u of course, but the function is also fixed as we vary h.

For h sufficiently small, the error will be dominated by the first term $\frac{1}{2}hu''(\bar{x})$ and all the other terms will be negligible compared to this term, so we expect the error to behave roughly like a constant times h, where the constant has the value $\frac{1}{2}u''(\bar{x})$.

Note that in Example 1.1, where $u(x) = \sin x$, we have $\frac{1}{2}u''(1) = -0.4207355$, which agrees with the behavior seen in Table 1.1.

Similarly, from (1.5b) we can compute that the error in $D_-u(\bar{x})$ is

$$D_-u(\bar{x}) - u'(\bar{x}) = -\frac{1}{2}hu''(\bar{x}) + \frac{1}{6}h^2u'''(\bar{x}) + O(h^3),$$

which also agrees with our expectations.

Combining (1.5a) and (1.5b) shows that

$$u(\bar{x}+h) - u(\bar{x}-h) = 2hu'(\bar{x}) + \frac{1}{3}h^3u'''(\bar{x}) + O(h^5)$$

so that

$$D_0u(\bar{x}) - u'(\bar{x}) = \frac{1}{6}h^2u'''(\bar{x}) + O(h^4). \tag{1.6}$$

This confirms the second order accuracy of this approximation and again agrees with what is seen in Table 1.1, since in the context of Example 1.1 we have

$$\frac{1}{6}u'''(\bar{x}) = -\frac{1}{6}\cos(1) = -0.09005038.$$

Note that all the odd order terms drop out of the Taylor series expansion (1.6) for $D_0u(\bar{x})$. This is typical with *centered* approximations and typically leads to a higher order approximation.

To analyze $D_3 u$ we need to also expand $u(\bar{x} - 2h)$ as

$$u(\bar{x} - 2h) = u(\bar{x}) - 2hu'(\bar{x}) + \frac{1}{2}(2h)^2 u''(\bar{x}) - \frac{1}{6}(2h)^3 u'''(\bar{x}) + O(h^4). \tag{1.7}$$

Combining this with (1.5a) and (1.5b) shows that

$$D_3 u(\bar{x}) = u'(\bar{x}) + \frac{1}{12}h^3 u^{(4)}(\bar{x}) + O(h^4), \tag{1.8}$$

where $u^{(4)}$ is the fourth derivative of u.

1.2 Deriving finite difference approximations

Suppose we want to derive a finite difference approximation to $u'(\bar{x})$ based on some given set of points. We can use Taylor series to derive an appropriate formula, using the *method of undetermined coefficients*.

Example 1.2. Suppose we want a one-sided approximation to $u'(\bar{x})$ based on $u(\bar{x})$, $u(\bar{x} - h)$, and $u(\bar{x} - 2h)$ of the form

$$D_2 u(\bar{x}) = au(\bar{x}) + bu(\bar{x} - h) + cu(\bar{x} - 2h). \tag{1.9}$$

We can determine the coefficients a, b, and c to give the best possible accuracy by expanding in Taylor series and collecting terms. Using (1.5b) and (1.7) in (1.9) gives

$$\begin{aligned} D_2 u(\bar{x}) = (a + b + c)u(\bar{x}) - (b + 2c)hu'(\bar{x}) + \frac{1}{2}(b + 4c)h^2 u''(\bar{x}) \\ - \frac{1}{6}(b + 8c)h^3 u'''(\bar{x}) + \cdots . \end{aligned}$$

If this is going to agree with $u'(\bar{x})$ to high order, then we need

$$\begin{aligned} a + b + c &= 0, \\ b + 2c &= -1/h, \\ b + 4c &= 0. \end{aligned} \tag{1.10}$$

We might like to require that higher order coefficients be zero as well, but since there are only three unknowns a, b, and c, we cannot in general hope to satisfy more than three such conditions. Solving the linear system (1.10) gives

$$a = 3/2h, \qquad b = -2/h, \qquad c = 1/2h$$

so that the formula is

$$D_2 u(\bar{x}) = \frac{1}{2h}[3u(\bar{x}) - 4u(\bar{x} - h) + u(\bar{x} - 2h)]. \tag{1.11}$$

This approximation is used, for example, in the system of equations (2.57) for a 2-point boundary value problem with a Neumann boundary condition at the left boundary.

The error in this approximation is

$$\begin{aligned} D_2u(\bar{x}) - u'(\bar{x}) &= -\frac{1}{6}(b+8c)h^3u'''(\bar{x}) + \cdots \\ &= \frac{1}{12}h^2u'''(\bar{x}) + O(h^3). \end{aligned} \tag{1.12}$$

There are other ways to derive the same finite difference approximations. One way is to approximate the function $u(x)$ by some polynomial $p(x)$ and then use $p'(\bar{x})$ as an approximation to $u'(\bar{x})$. If we determine the polynomial by interpolating u at an appropriate set of points, then we obtain the same finite difference methods as above.

Example 1.3. To derive the method of Example 1.2 in this way, let $p(x)$ be the quadratic polynomial that interpolates u at $\bar{x}$, $\bar{x} - h$ and $\bar{x} - 2h$, and then compute $p'(\bar{x})$. The result is exactly (1.11).

1.3 Second order derivatives

Approximations to the second derivative $u''(x)$ can be obtained in an analogous manner. The standard second order centered approximation is given by

$$\begin{aligned} D^2u(\bar{x}) &= \frac{1}{h^2}[u(\bar{x}-h) - 2u(\bar{x}) + u(\bar{x}+h)] \\ &= u''(\bar{x}) + \frac{1}{12}h^2u''''(\bar{x}) + O(h^4). \end{aligned} \tag{1.13}$$

Again, since this is a symmetric centered approximation, all the odd order terms drop out. This approximation can also be obtained by the method of undetermined coefficients, or alternatively by computing the second derivative of the quadratic polynomial interpolating $u(x)$ at $\bar{x} - h$, $\bar{x}$, and $\bar{x} + h$, as is done in Example 1.4 below for the more general case of unequally spaced points.

Another way to derive approximations to higher order derivatives is by repeatedly applying first order differences. Just as the second derivative is the derivative of u', we can view $D^2u(\bar{x})$ as being a difference of first differences. In fact,

$$D^2u(\bar{x}) = D_+D_-u(\bar{x})$$

since

$$\begin{aligned} D_+(D_-u(\bar{x})) &= \frac{1}{h}[D_-u(\bar{x}+h) - D_-u(\bar{x})] \\ &= \frac{1}{h}\left[\left(\frac{u(\bar{x}+h)-u(\bar{x})}{h}\right) - \left(\frac{u(\bar{x})-u(\bar{x}-h)}{h}\right)\right] \\ &= D^2u(\bar{x}). \end{aligned}$$

Alternatively, $D^2(\bar{x}) = D_-D_+u(\bar{x})$, or we can also view it as a centered difference of centered differences, if we use a step size $h/2$ in each centered approximation to the first derivative. If we define

$$\hat{D}_0u(x) = \frac{1}{h}\left(u\left(x+\frac{h}{2}\right) - u\left(x-\frac{h}{2}\right)\right),$$

then we find that

$$\hat{D}_0(\hat{D}_0 u(\bar{x})) = \frac{1}{h}\left(\left(\frac{u(\bar{x}+h)-u(\bar{x})}{h}\right) - \left(\frac{u(\bar{x})-u(\bar{x}-h)}{h}\right)\right) = D^2 u(\bar{x}).$$

Example 1.4. Suppose we want to approximate $u''(x_2)$ based on data values U_1, U_2, and U_3, at three unequally spaced points x_1, x_2, and x_3. This approximation will be used in Section 2.18. Let $h_1 = x_2 - x_1$ and $h_2 = x_3 - x_2$. The approximation can be found by interpolating by a quadratic function and differentiating twice. Using the Newton form of the interpolating polynomial (see Section B.2.3),

$$p(x) = U[x_1] + U[x_1, x_2](x - x_1) + U[x_1, x_2, x_3](x - x_1)(x - x_2),$$

we see that the second derivative is constant and equal to twice the second order divided difference,

$$\begin{aligned} p''(x_2) &= 2U[x_1, x_2, x_3] \\ &= 2\left(\frac{U_3 - U_2}{h_2} - \frac{U_2 - U_1}{h_1}\right) \Big/ (h_1 + h_2) \\ &= c_1 U_1 + c_2 U_2 + c_3 U_3, \end{aligned} \tag{1.14}$$

where

$$c_1 = \frac{2}{h_1(h_1 + h_2)}, \qquad c_2 = -\frac{2}{h_1 h_2}, \qquad c_3 = \frac{2}{h_2(h_1 + h_2)}. \tag{1.15}$$

This would be our approximation to $u''(x_2)$. The same result can be found by the method of undetermined coefficients.

To compute the error in this approximation, we can expand $u(x_1)$ and $u(x_3)$ in Taylor series about x_2 and find that

$$\begin{aligned} &c_1 u(x_1) + c_2 u(x_2) + c_3 u(x_3) - u''(x_2) \\ &\quad = \frac{1}{3}(h_2 - h_1)u^{(3)}(x_2) + \frac{1}{12}\left(\frac{h_1^3 + h_2^3}{h_1 + h_2}\right)u^{(4)}(x_2) + \cdots. \end{aligned} \tag{1.16}$$

In general, if $h_1 \neq h_2$, the error is proportional to $\max(h_1, h_2)$ and this approximation is "first order" accurate.

In the special case $h_1 = h_2$ (equally spaced points), the approximation (1.14) reduces to the standard centered approximate $D^2 u(x_2)$ from (1.13) with the second order error shown there.

1.4 Higher order derivatives

Finite difference approximations to higher order derivatives can also be obtained using any of the approaches outlined above. Repeatedly differencing approximations to lower order derivatives is a particularly simple approach.

Example 1.5. As an example, here are two different approximations to $u'''(\bar{x})$. The first is uncentered and first order accurate:

$$D_+ D^2 u(\bar{x}) = \frac{1}{h^3}(u(\bar{x}+2h) - 3u(\bar{x}+h) + 3u(\bar{x}) - u(\bar{x}-h))$$
$$= u'''(\bar{x}) + \frac{1}{2}hu''''(\bar{x}) + O(h^2).$$

The next approximation is centered and second order accurate:

$$D_0 D_+ D_- u(\bar{x}) = \frac{1}{2h^3}(u(\bar{x}+2h) - 2u(\bar{x}+h) + 2u(\bar{x}-h) - u(\bar{x}-2h))$$
$$= u'''(\bar{x}) + \frac{1}{4}h^2 u'''''(\bar{x}) + O(h^4).$$

Another way to derive finite difference approximations to higher order derivatives is by interpolating with a sufficiently high order polynomial based on function values at the desired stencil points and then computing the appropriate derivative of this polynomial. This is generally a cumbersome way to do it. A simpler approach that lends itself well to automation is to use the method of undetermined coefficients, as illustrated in Section 1.2 for an approximation to the first order derivative and explained more generally in the next section.

1.5 A general approach to deriving the coefficients

The method illustrated in Section 1.2 can be extended to compute the finite difference coefficients for computing an approximation to $u^{(k)}(\bar{x})$, the kth derivative of $u(x)$ evaluated at $\bar{x}$, based on an arbitrary stencil of $n \geq k+1$ points $x_1, \ \ldots, \ x_n$. Usually $\bar{x}$ is one of the stencil points, but not necessarily.

We assume $u(x)$ is sufficiently smooth, namely, at least $n+1$ times continuously differentiable in the interval containing $\bar{x}$ and all the stencil points, so that the Taylor series expansions below are valid. Taylor series expansions of u at each point x_i in the stencil about $u(\bar{x})$ yield

$$u(x_i) = u(\bar{x}) + (x_i - \bar{x})u'(\bar{x}) + \cdots + \frac{1}{k!}(x_i - \bar{x})^k u^{(k)}(\bar{x}) + \cdots \tag{1.17}$$

for $i = 1, \ \ldots, \ n$. We want to find a linear combination of these values that agrees with $u^{(k)}(\bar{x})$ as well as possible. So we want

$$c_1 u(x_1) + c_2 u(x_2) + \cdots + c_n u(x_n) = u^{(k)}(\bar{x}) + O(h^p), \tag{1.18}$$

where p is as large as possible. (Here h is some measure of the width of the stencil. If we are deriving approximations on stencils with equally spaced points, then h is the mesh width, but more generally it is some "average mesh width," so that $\max_{1\leq i\leq n} |x_i - \bar{x}| \leq Ch$ for some small constant C.)

Following the approach of Section 1.2, we choose the coefficients c_j so that

$$\frac{1}{(i-1)!}\sum_{j=1}^{n} c_j (x_j - \bar{x})^{(i-1)} = \begin{cases} 1 & \text{if } i-1 = k, \\ 0 & \text{otherwise} \end{cases} \tag{1.19}$$

for $i = 1, \ \ldots, \ n$. Provided the points x_j are distinct, this $n \times n$ Vandermonde system is nonsingular and has a unique solution. If $n \leq k$ (too few points in the stencil), then the

right-hand side and solution are both the zero vector, but for $n > k$ the coefficients give a suitable finite difference approximation.

How accurate is the method? The right-hand side vector has a 1 in the $i = k + 1$ row, which ensures that this linear combination approximates the kth derivative. The 0 in the other component of the right-hand side ensures that the terms

$$\left(\sum_{j=1}^{n} c_j (x_j - \bar{x})^{(i-1)}\right) u^{(i-1)}(\bar{x})$$

drop out in the linear combination of Taylor series for $i - 1 \neq k$. For $i - 1 < k$ this is necessary to get even first order accuracy of the finite difference approximation. For $i - 1 > k$ (which is possible only if $n > k + 1$), this gives cancellation of higher order terms in the expansion and greater than first order accuracy. In general we expect the order of accuracy of the finite difference approximation to be at least $p \geq n - k$. It may be even higher if higher order terms happen to cancel out as well (as often happens with centered approximations, for example).

In MATLAB it is very easy to set up and solve this Vandermonde system. If `xbar` is the point $\bar{x}$ and `x(1:n)` are the desired stencil points, then the following function can be used to compute the coefficients:

```
function c = fdcoeffV(k,xbar,x)
A = ones(n,n);
xrow = (x(:)-xbar)';  % displacements as a row vector.
for i=2:n
   A(i,:) = (xrow .^ (i-1)) ./ factorial(i-1);
   end
b = zeros(n,1);     % b is right hand side,
b(k+1) = 1;         % so k'th derivative term remains
c = A\b;            % solve system for coefficients
c = c';             % row vector
```

If `u` is a column vector of n values $u(x_i)$, then in MATLAB the resulting approximation to $u^{(k)}(\bar{x})$ can be computed by `c*u`.

This function is implemented in the MATLAB function `fdcoeffV.m` available on the Web page for this book, which contains more documentation and data checking but is essentially the same as the above code. A row vector is returned since in applications we will often use the output of this routine as the row of a matrix approximating a differential operator (see Section 2.18, for example).

Unfortunately, for a large number of points this Vandermonde procedure is numerically unstable because the resulting linear system can be very poorly conditioned. A more stable procedure for calculating the weights is given by Fornberg [30], who also gives a FORTRAN implementation. This modified procedure is implemented in the MATLAB function `fdcoeffF.m` on the Web page.

Finite difference approximations of the sort derived in this chapter form the basis for finite difference algorithms for solving differential equations. In the next chapter we begin the study of this topic.

Chapter 2

Steady States and Boundary Value Problems

We will first consider ordinary differential equations (ODEs) that are posed on some interval $a < x < b$, together with some boundary conditions at each end of the interval. In the next chapter we will extend this to more than one space dimension and will study *elliptic partial differential equations* (ODEs) that are posed in some region of the plane or three-dimensional space and are solved subject to some boundary conditions specifying the solution and/or its derivatives around the boundary of the region. The problems considered in these two chapters are generally *steady-state* problems in which the solution varies only with the spatial coordinates but not with time. (But see Section 2.16 for a case where $[a, b]$ is a time interval rather than an interval in space.)

Steady-state problems are often associated with some time-dependent problem that describes the dynamic behavior, and the 2-point boundary value problem (BVP) or elliptic equation results from considering the special case where the solution is steady in time, and hence the time-derivative terms are equal to zero, simplifying the equations.

2.1 The heat equation

As a specific example, consider the flow of heat in a rod made out of some heat-conducting material, subject to some external heat source along its length and some boundary conditions at each end. If we assume that the material properties, the initial temperature distribution, and the source vary only with x, the distance along the length, and not across any cross section, then we expect the temperature distribution at any time to vary only with x and we can model this with a differential equation in one space dimension. Since the solution might vary with time, we let $u(x, t)$ denote the temperature at point x at time t, where $a < x < b$ along some finite length of the rod. The solution is then governed by the *heat equation*

$$u_t(x, t) = (\kappa(x)u_x(x, t))_x + \psi(x, t), \tag{2.1}$$

where $\kappa(x)$ is the coefficient of heat conduction, which may vary with x, and $\psi(x, t)$ is the heat source (or sink, if $\psi < 0$). See Appendix E for more discussion and a derivation. Equation (2.1) is often called the *diffusion equation* since it models diffusion processes more generally, and the diffusion of heat is just one example. It is assumed that the basic

theory of this equation is familiar to the reader. See standard PDE books such as [55] for a derivation and more introduction. In general it is extremely valuable to understand where the equation one is attempting to solve comes from, since a good understanding of the physics (or biology, etc.) is generally essential in understanding the development and behavior of numerical methods for solving the equation.

2.2 Boundary conditions

If the material is homogeneous, then $\kappa(x) \equiv \kappa$ is independent of x and the heat equation (2.1) reduces to

$$u_t(x,t) = \kappa u_{xx}(x,t) + \psi(x,t). \tag{2.2}$$

Along with the equation, we need initial conditions,

$$u(x,0) = u^0(x),$$

and boundary conditions, for example, the temperature might be specified at each end,

$$u(a,t) = \alpha(t), \qquad u(b,t) = \beta(t). \tag{2.3}$$

Such boundary conditions, where the value of the solution itself is specified, are called *Dirichlet boundary conditions*. Alternatively one end, or both ends, might be insulated, in which case there is zero heat flux at that end, and so $u_x = 0$ at that point. This boundary condition, which is a condition on the derivative of u rather than on u itself, is called a *Neumann boundary condition*. To begin, we will consider the Dirichlet problem for (2.2) with boundary conditions (2.3).

2.3 The steady-state problem

In general we expect the temperature distribution to change with time. However, if $\psi(x,t)$, $\alpha(t)$, and $\beta(t)$ are all time independent, then we might expect the solution to eventually reach a *steady-state* solution $u(x)$, which then remains essentially unchanged at later times. Typically there will be an initial *transient* time, as the initial data $u^0(x)$ approach $u(x)$ (unless $u^0(x) \equiv u(x)$), but if we are interested only in computing the steady-state solution itself, then we can set $u_t = 0$ in (2.2) and obtain an ODE in x to solve for $u(x)$:

$$u''(x) = f(x), \tag{2.4}$$

where we introduce $f(x) = -\psi(x)/\kappa$ to avoid minus signs below. This is a second order ODE, and from basic theory we expect to need two boundary conditions to specify a unique solution. In our case we have the boundary conditions

$$u(a) = \alpha, \qquad u(b) = \beta. \tag{2.5}$$

Remark: Having two boundary conditions does not necessarily guarantee that there exists a unique solution for a general second order equation—see Section 2.13.

The problem (2.4), (2.5) is called a *2-point* (BVP), since one condition is specified at each of the two endpoints of the interval where the solution is desired. If instead two data

values were specified at the same point, say, $u(a) = \alpha, u'(a) = \sigma$, and we want to find the solution for $t \geq a$, then we would have an *initial value problem* (IVP) instead. These problems are discussed in Chapter 5.

One approach to computing a numerical solution to a steady-state problem is to choose some initial data and march forward in time using a numerical method for the time-dependent PDE (2.2), as discussed in Chapter 9 on the solution of parabolic equations. However, this is typically not an efficient way to compute the steady state solution if this is all we want. Instead we can discretize and solve the 2-point BVP given by (2.4) and (2.5) directly. This is the first BVP that we will study in detail, starting in the next section. Later in this chapter we will consider some other BVPs, including more challenging nonlinear equations.

2.4 A simple finite difference method

As a first example of a finite difference method for solving a differential equation, consider the second order ODE discussed above,

$$u''(x) = f(x) \quad \text{for } 0 < x < 1, \tag{2.6}$$

with some given boundary conditions

$$u(0) = \alpha, \qquad u(1) = \beta. \tag{2.7}$$

The function $f(x)$ is specified and we wish to determine $u(x)$ in the interval $0 < x < 1$. This problem is called a *2-point* BVP since boundary conditions are given at two distinct points. This problem is so simple that we can solve it explicitly (integrate $f(x)$ twice and choose the two constants of integration so that the boundary conditions are satisfied), but studying finite difference methods for this simple equation will reveal some of the essential features of all such analysis, particularly the relation of the global error to the local truncation error and the use of stability in making this connection.

We will attempt to compute a grid function consisting of values $U_0, U_1, \ldots, U_m, U_{m+1}$, where U_j is our approximation to the solution $u(x_j)$. Here $x_j = jh$ and $h = 1/(m+1)$ is the *mesh width*, the distance between grid points. From the boundary conditions we know that $U_0 = \alpha$ and $U_{m+1} = \beta$, and so we have m unknown values $U_1, \ldots, U_m$ to compute. If we replace $u''(x)$ in (2.6) by the centered difference approximation

$$D^2 U_j = \frac{1}{h^2}(U_{j-1} - 2U_j + U_{j+1}),$$

then we obtain a set of algebraic equations

$$\frac{1}{h^2}(U_{j-1} - 2U_j + U_{j+1}) = f(x_j) \quad \text{for } j = 1, 2, \ldots, m. \tag{2.8}$$

Note that the first equation ($j = 1$) involves the value $U_0 = \alpha$ and the last equation ($j = m$) involves the value $U_{m+1} = \beta$. We have a linear system of m equations for the m unknowns, which can be written in the form

$$AU = F, \tag{2.9}$$

where U is the vector of unknowns $U = [U_1, U_2, \ldots, U_m]^T$ and

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix}, \qquad F = \begin{bmatrix} f(x_1) - \alpha/h^2 \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) - \beta/h^2 \end{bmatrix}. \quad (2.10)$$

This tridiagonal linear system is nonsingular and can be easily solved for U from any right-hand side F.

How well does U approximate the function $u(x)$? We know that the centered difference approximation D^2, when applied to a known smooth function $u(x)$, gives a second order accurate approximation to $u''(x)$. But here we are doing something more complicated—we know the values of u'' at each point and are computing a whole set of discrete values $U_1, \ldots, U_m$ with the property that applying D^2 to these discrete values gives the desired values $f(x_j)$. While we might hope that this process also gives errors that are $O(h^2)$ (and indeed it does), this is certainly not obvious.

First we must clarify what we mean by the error in the discrete values $U_1, \ldots, U_m$ relative to the true solution $u(x)$, which is a function. Since U_j is supposed to approximate $u(x_j)$, it is natural to use the pointwise errors $U_j - u(x_j)$. If we let $\hat{U}$ be the vector of true values

$$\hat{U} = \begin{bmatrix} u(x_1) \\ u(x_2) \\ \vdots \\ u(x_m) \end{bmatrix}, \quad (2.11)$$

then the error vector E defined by

$$E = U - \hat{U}$$

contains the errors at each grid point.

Our goal is now to obtain a bound on the magnitude of this vector, showing that it is $O(h^2)$ as $h \to 0$. To measure the magnitude of this vector we must use some *norm*, for example, the max-norm

$$\|E\|_\infty = \max_{1 \le j \le m} |E_j| = \max_{1 \le j \le m} |U_j - u(x_j)|.$$

This is just the largest error over the interval. If we can show that $\|E\|_\infty = O(h^2)$, then it follows that each pointwise error must be $O(h^2)$ as well.

Other norms are often used to measure grid functions, either because they are more appropriate for a given problem or simply because they are easier to bound since some mathematical techniques work only with a particular norm. Other norms that are frequently used include the 1-norm

$$\|E\|_1 = h \sum_{j=1}^{m} |E_j|$$

and the 2-norm

$$\|E\|_2 = \left(h \sum_{j=1}^{m} |E_j|^2 \right)^{1/2}.$$

Note the factor of h that appears in these definitions. See Appendix A for a more thorough discussion of grid function norms and how they relate to standard vector norms.

Now let's return to the problem of estimating the error in our finite difference solution to BVP obtained by solving the system (2.9). The technique we will use is absolutely basic to the analysis of finite difference methods in general. It involves two key steps. We first compute the *local truncation error* (LTE) of the method and then use some form of *stability* to show that the *global error* can be bounded in terms of the LTE.

The global error simply refers to the error $U - \hat{U}$ that we are attempting to bound. The LTE refers to the error in our finite difference approximation of derivatives and hence is something that can be easily estimated using Taylor series expansions, as we have seen in Chapter 1. Stability is the magic ingredient that allows us to go from these easily computed bounds on the local error to the estimates we really want for the global error. Let's look at each of these in turn.

2.5 Local truncation error

The LTE is defined by replacing U_j with the true solution $u(x_j)$ in the finite difference formula (2.8). In general the true solution $u(x_j)$ won't satisfy this equation exactly and the discrepancy is the LTE, which we denote by τ_j:

$$\tau_j = \frac{1}{h^2}(u(x_{j-1}) - 2u(x_j) + u(x_{j+1})) - f(x_j) \tag{2.12}$$

for $j = 1, 2, \ldots, m$. Of course in practice we don't know what the true solution $u(x)$ is, but if we assume it is smooth, then by the Taylor series expansions (1.5a) we know that

$$\tau_j = \left[u''(x_j) + \frac{1}{12} h^2 u''''(x_j) + O(h^4) \right] - f(x_j). \tag{2.13}$$

Using our original differential equation (2.6) this becomes

$$\tau_j = \frac{1}{12} h^2 u''''(x_j) + O(h^4).$$

Although u'''' is in general unknown, it is some fixed function independent of h, and so $\tau_j = O(h^2)$ as $h \to 0$.

If we define τ to be the vector with components τ_j, then

$$\tau = A\hat{U} - F,$$

where $\hat{U}$ is the vector of true solution values (2.11), and so

$$A\hat{U} = F + \tau. \tag{2.14}$$

2.6 Global error

To obtain a relation between the local error τ and the global error $E = U - \hat{U}$, we subtract (2.14) from (2.9) that defines U, obtaining

$$AE = -\tau. \tag{2.15}$$

This is simply the matrix form of the system of equations

$$\frac{1}{h^2}(E_{j-1} - 2E_j + E_{j+1}) = -\tau(x_j) \quad \text{for } j = 1, 2, \ldots, m$$

with the boundary conditions

$$E_0 = E_{m+1} = 0$$

since we are using the exact boundary data $U_0 = \alpha$ and $U_{m+1} = \beta$. We see that the global error satisfies a set of finite difference equations that has exactly the same form as our original difference equations for U except that the right-hand side is given by $-\tau$ rather than F.

From this it should be clear why we expect the global error to be roughly the same magnitude as the local error τ. We can interpret the system (2.15) as a discretization of the ODE

$$e''(x) = -\tau(x) \quad \text{for } 0 < x < 1 \tag{2.16}$$

with boundary conditions

$$e(0) = 0, \quad e(1) = 0.$$

Since $\tau(x) \approx \frac{1}{12}h^2 u''''(x)$, integrating twice shows that the global error should be roughly

$$e(x) \approx -\frac{1}{12}h^2 u''(x) + \frac{1}{12}h^2 \left(u''(0) + x(u''(1) - u''(0))\right)$$

and hence the error should be $O(h^2)$.

2.7 Stability

The above argument is not completely convincing because we are relying on the assumption that solving the difference equations gives a decent approximation to the solution of the underlying differential equations (actually the converse now, that the solution to the differential equation (2.16) gives a good indication of the solution to the difference equations (2.15)). Since it is exactly this assumption we are trying to prove, the reasoning is rather circular.

Instead, let's look directly at the discrete system (2.15), which we will rewrite in the form

$$A^h E^h = -\tau^h, \tag{2.17}$$

where the superscript h indicates that we are on a grid with mesh spacing h. This serves as a reminder that these quantities change as we refine the grid. In particular, the matrix A^h is an $m \times m$ matrix with $h = 1/(m+1)$ so that its dimension is growing as $h \to 0$.

Let $(A^h)^{-1}$ be the inverse of this matrix. Then solving the system (2.17) gives

$$E^h = -(A^h)^{-1}\tau^h$$

and taking norms gives

$$\begin{aligned}\|E^h\| &= \|(A^h)^{-1}\tau^h\| \\ &\le \|(A^h)^{-1}\|\,\|\tau^h\|.\end{aligned}$$

We know that $\|\tau^h\| = O(h^2)$ and we are hoping the same will be true of $\|E^h\|$. It is clear what we need for this to be true: we need $\|(A^h)^{-1}\|$ to be bounded by some constant independent of h as $h \to 0$:

$$\|(A^h)^{-1}\| \le C \quad \text{for all } h \text{ sufficiently small.}$$

Then we will have

$$\|E^h\| \le C\|\tau^h\| \tag{2.18}$$

and so $\|E^h\|$ goes to zero at least as fast as $\|\tau^h\|$. This motivates the following definition of *stability* for linear BVPs.

Definition 2.1. *Suppose a finite difference method for a linear BVP gives a sequence of matrix equations of the form $A^hU^h = F^h$, where h is the mesh width. We say that the method is* stable *if $(A^h)^{-1}$ exists for all h sufficiently small (for $h < h_0$, say) and if there is a constant C, independent of h, such that*

$$\|(A^h)^{-1}\| \le C \quad \textit{for all } h < h_0. \tag{2.19}$$

2.8 Consistency

We say that a method is *consistent* with the differential equation and boundary conditions if

$$\|\tau^h\| \to 0 \quad \text{as } h \to 0. \tag{2.20}$$

This simply says that we have a sensible discretization of the problem. Typically $\|\tau^h\| = O(h^p)$ for some integer $p > 0$, and then the method is certainly consistent.

2.9 Convergence

A method is said to be *convergent* if $\|E^h\| \to 0$ as $h \to 0$. Combining the ideas introduced above we arrive at the conclusion that

$$\text{consistency} \;+\; \text{stability} \quad \Longrightarrow \quad \text{convergence}. \tag{2.21}$$

This is easily proved by using (2.19) and (2.20) to obtain the bound

$$\|E^h\| \le \|(A^h)^{-1}\|\,\|\tau^h\| \le C\|\tau^h\| \to 0 \quad \text{as } h \to 0.$$

Although this has been demonstrated only for the linear BVP, in fact most analyses of finite difference methods for differential equations follow this same two-tier approach, and the statement (2.21) is sometimes called the *fundamental theorem of finite difference methods*. In fact, as our above analysis indicates, this can generally be strengthened to say that

$$O(h^p) \text{ local truncation error} \quad + \quad \text{stability} \quad \Longrightarrow \quad O(h^p) \text{ global error.} \tag{2.22}$$

Consistency (and the order of accuracy) is usually the easy part to check. Verifying stability is the hard part. Even for the linear BVP just discussed it is not at all clear how to check the condition (2.19) since these matrices become larger as $h \to 0$. For other problems it may not even be clear how to define stability in an appropriate way. As we will see, there are many definitions of "stability" for different types of problems. The challenge in analyzing finite difference methods for new classes of problems often is to find an appropriate definition of "stability" that allows one to prove convergence using (2.21) while at the same time being sufficiently manageable that we can verify it holds for specific finite difference methods. For nonlinear PDEs this frequently must be tuned to each particular class of problems and relies on existing mathematical theory and techniques of analysis for this class of problems.

Whether or not one has a formal proof of convergence for a given method, it is always good practice to check that the computer program is giving convergent behavior, at the rate expected. Appendix A contains a discussion of how the error in computed results can be estimated.

2.10 Stability in the 2-norm

Returning to the BVP at the start of the chapter, let's see how we can verify stability and hence second order accuracy. The technique used depends on what norm we wish to consider. Here we will consider the 2-norm and see that we can show stability by explicitly computing the eigenvectors and eigenvalues of the matrix A. In Section 2.11 we show stability in the max-norm by different techniques.

Since the matrix A from (2.10) is symmetric, the 2-norm of A is equal to its spectral radius (see Section A.3.2 and Section C.9):

$$\|A\|_2 = \rho(A) = \max_{1 \le p \le m} |\lambda_p|.$$

(Note that λ_p refers to the pth eigenvalue of the matrix. Superscripts are used to index the eigenvalues and eigenvectors, while subscripts on the eigenvector below refer to components of the vector.)

The matrix A^{-1} is also symmetric, and the eigenvalues of A^{-1} are simply the inverses of the eigenvalues of A, so

$$\|A^{-1}\|_2 = \rho(A^{-1}) = \max_{1 \le p \le m} |(\lambda_p)^{-1}| = \left(\min_{1 \le p \le m} |\lambda_p| \right)^{-1}.$$

So all we need to do is compute the eigenvalues of A and show that they are bounded away from zero as $h \to 0$. Of course we have an infinite set of matrices A^h to consider,

as h varies, but since the structure of these matrices is so simple, we can obtain a general expression for the eigenvalues of each A^h. For more complicated problems we might not be able to do this, but it is worth going through in detail for this problem because one often considers model problems for which such an analysis is possible. We will also need to know these eigenvalues for other purposes when we discuss parabolic equations in Chapter 9. (See also Section C.7 for more general expressions for the eigenvalues of related matrices.)

We will now focus on one particular value of $h = 1/(m+1)$ and drop the superscript h to simplify the notation. Then the m eigenvalues of A are given by

$$\lambda_p = \frac{2}{h^2}(\cos(p\pi h) - 1) \quad \text{for } p = 1, 2, \ldots, m. \tag{2.23}$$

The eigenvector u^p corresponding to λ_p has components u_j^p for $j = 1, 2, \ldots, m$ given by

$$u_j^p = \sin(p\pi j h). \tag{2.24}$$

This can be verified by checking that $Au^p = \lambda_p u^p$. The jth component of the vector Au^p is

$$\begin{aligned}
(Au^p)_j &= \frac{1}{h^2}\left(u_{j-1}^p - 2u_j^p + u_{j+1}^p\right) \\
&= \frac{1}{h^2}(\sin(p\pi(j-1)h) - 2\sin(p\pi j h) + \sin(p\pi(j+1)h)) \\
&= \frac{1}{h^2}(\sin(p\pi j h)\cos(p\pi h) - 2\sin(p\pi j h) + \sin(p\pi j h)\cos(p\pi h)) \\
&= \lambda_p u_j^p.
\end{aligned}$$

Note that for $j = 1$ and $j = m$ the jth component of Au^p looks slightly different (the u_{j-1}^p or u_{j+1}^p term is missing) but that the above form and trigonometric manipulations are still valid provided that we define

$$u_0^p = u_{m+1}^p = 0,$$

as is consistent with (2.24). From (2.23) we see that the smallest eigenvalue of A (in magnitude) is

$$\begin{aligned}
\lambda_1 &= \frac{2}{h^2}(\cos(\pi h) - 1) \\
&= \frac{2}{h^2}\left(-\frac{1}{2}\pi^2 h^2 + \frac{1}{24}\pi^4 h^4 + O(h^6)\right) \\
&= -\pi^2 + O(h^2).
\end{aligned}$$

This is clearly bounded away from zero as $h \to 0$, and so we see that the method is stable in the 2-norm. Moreover we get an error bound from this:

$$\|E^h\|_2 \le \|(A^h)^{-1}\|_2 \|\tau^h\|_2 \approx \frac{1}{\pi^2}\|\tau^h\|_2.$$

Since $\tau_j^h \approx \frac{1}{12}h^2 u''''(x_j)$, we expect $\|\tau^h\|_2 \approx \frac{1}{12}h^2\|u''''\|_2 = \frac{1}{12}h^2\|f''\|_2$. The 2-norm of the function f'' here means the grid-function norm of this function evaluated at the discrete points x_j, although this is approximately equal to the function space norm of f'' defined using (A.14).

Note that the eigenvector (2.24) is closely related to the eigenfunction of the corresponding differential operator $\frac{\partial^2}{\partial x^2}$. The functions

$$u^p(x) = \sin(p\pi x), \qquad p = 1,\ 2,\ 3,\ \ldots,$$

satisfy the relation

$$\frac{\partial^2}{\partial x^2}u^p(x) = \mu_p u^p(x)$$

with eigenvalue $\mu_p = -p^2\pi^2$. These functions also satisfy $u^p(0) = u^p(1) = 0$, and hence they are eigenfunctions of $\frac{\partial^2}{\partial x^2}$ on $[0, 1]$ with homogeneous boundary conditions. The discrete approximation to this operator given by the matrix A has only m eigenvalues instead of an infinite number, and the corresponding eigenvectors (2.24) are simply the first m eigenfunctions of $\frac{\partial^2}{\partial x^2}$ evaluated at the grid points. The eigenvalue λ_p is not exactly the same as μ_p, but at least for small values of p it is very nearly the same, since Taylor series expansion of the cosine in (2.23) gives

$$\begin{aligned}\lambda_p &= \frac{2}{h^2}\left(-\frac{1}{2}p^2\pi^2h^2 + \frac{1}{24}p^4\pi^4h^4 + \cdots\right)\\ &= -p^2\pi^2 + O(h^2) \quad \text{as } h \to 0 \text{ for } p \text{ fixed.}\end{aligned}$$

This relationship will be illustrated further when we study numerical methods for the heat equation (2.1).

2.11 Green's functions and max-norm stability

In Section 2.10 we demonstrated that A from (2.10) is stable in the 2-norm, and hence that $\|E\|_2 = O(h^2)$. Suppose, however, that we want a bound on the maximum error over the interval, i.e., a bound on $\|E\|_\infty = \max|E_j|$. We can obtain one such bound directly from the bound we have for the 2-norm. From (A.19) we know that

$$\|E\|_\infty \le \frac{1}{\sqrt{h}}\|E\|_2 = O(h^{3/2}) \quad \text{as } h \to 0.$$

However, this does not show the second order accuracy that we hope to have. To show that $\|E\|_\infty = O(h^2)$ we will explicitly calculate the inverse of A and then show that $\|A^{-1}\|_\infty = O(1)$, and hence

$$\|E\|_\infty \le \|A^{-1}\|_\infty\|\tau\|_\infty = O(h^2)$$

since $\|\tau\|_\infty = O(h^2)$. As in the computation of the eigenvalues in the last section, we can do this only because our model problem (2.6) is so simple. In general it would be impossible to obtain closed form expressions for the inverse of the matrices A^h as h varies.

But again it is worth working out the details for this simple case because it gives a great deal of insight into the nature of the inverse matrix and what it represents more generally.

Each column of the inverse matrix can be interpreted as the solution of a particular BVP. The columns are discrete approximations to the *Green's functions* that are commonly introduced in the study of the differential equation. An understanding of this is valuable in developing an intuition for what happens if we introduce relatively large errors at a few points within the interval. Such difficulties arise frequently in practice, typically at the boundary or at an internal interface where there are discontinuities in the data or solution.

We begin by reviewing the Green's function solution to the BVP

$$u''(x) = f(x) \qquad \text{for } 0 < x < 1 \tag{2.25}$$

with Dirichlet boundary conditions

$$u(0) = \alpha, \qquad u(1) = \beta. \tag{2.26}$$

To keep the expressions simple below we assume we are on the unit interval, but everything can be shifted to an arbitrary interval $[a, b]$.

For any fixed point $\bar{x} \in [0, 1]$, the Green's function $G(x; \bar{x})$ is the function of x that solves the particular BVP of the above form with $f(x) = \delta(x - \bar{x})$ and $\alpha = \beta = 0$. Here $\delta(x - \bar{x})$ is the "delta function" centered at $\bar{x}$. The delta function, $\delta(x)$, is not an ordinary function but rather the mathematical idealization of a sharply peaked function that is nonzero only on an interval $(-\epsilon, \epsilon)$ near the origin and has the property that

$$\int_{-\infty}^{\infty} \phi_\epsilon(x)\, dx = \int_{-\epsilon}^{\epsilon} \phi_\epsilon(x)\, dx = 1. \tag{2.27}$$

For example, we might take

$$\phi_\epsilon(x) = \begin{cases} (\epsilon + x)/\epsilon & \text{if } -\epsilon \le x \le 0, \\ (\epsilon - x)/\epsilon & \text{if } 0 \le x \le \epsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{2.28}$$

This piecewise linear function is the "hat function" with width ϵ and height $1/\epsilon$. The exact shape of ϕ_ϵ is not important, but note that it must attain a height that is $O(1/\epsilon)$ in order for the integral to have the value 1. We can think of the delta function as being a sort of limiting case of such functions as $\epsilon \to 0$. Delta functions naturally arise when we differentiate functions that are discontinuous. For example, consider the *Heaviside function* (or step function) $H(x)$ that is defined by

$$H(x) = \begin{cases} 0 & x < 0, \\ 1 & x \ge 0. \end{cases} \tag{2.29}$$

What is the derivative of this function? For $x \neq 0$ the function is constant and so $H'(x) = 0$. At $x = 0$ the derivative is not defined in the classical sense. But if we smooth out the function a little bit, making it continuous and differentiable by changing $H(x)$ only on the interval $(-\epsilon, \epsilon)$, then the new function $H_\epsilon(x)$ is differentiable everywhere and has a

derivative $H'_\epsilon(x)$ that looks something like $\phi_\epsilon(x)$. The exact shape of $H'_\epsilon(x)$ depends on how we choose $H_\epsilon(x)$, but note that regardless of its shape, its integral must be 1, since

$$\begin{aligned}\int_{-\infty}^{\infty} H'_\epsilon(x)\,dx &= \int_{-\epsilon}^{\epsilon} H'_\epsilon(x)\,dx \\ &= H_\epsilon(\epsilon) - H_\epsilon(-\epsilon) \\ &= 1 - 0 = 1.\end{aligned}$$

This explains the normalization (2.27). By letting $\epsilon \to 0$, we are led to define

$$H'(x) = \delta(x).$$

This expression makes no sense in terms of the classical definition of derivatives, but it can be made rigorous mathematically through the use of "distribution theory"; see, for example, [31]. For our purposes it suffices to think of the delta function as being a very sharply peaked function that is nonzero only on a very narrow interval but with total integral 1.

If we interpret the problem (2.25) as a steady-state heat conduction problem with source $\psi(x) = -f(x)$, then setting $f(x) = \delta(x - \bar{x})$ in the BVP is the mathematical idealization of a heat sink that has unit magnitude but that is concentrated near a single point. It might be easier to first consider the case $f(x) = -\delta(x-\bar{x})$, which corresponds to a heat source localized at $\bar{x}$, the idealization of a blow torch pumping heat into the rod at a single point. With the boundary conditions $u(0) = u(1) = 0$, holding the temperature fixed at each end, we would expect the temperature to be highest at the point $\bar{x}$ and to fall linearly to zero to each side (linearly because $u''(x) = 0$ away from $\bar{x}$). With $f(x) = \delta(x - \bar{x})$, a heat sink at $\bar{x}$, we instead have the minimum temperature at $\bar{x}$, rising linearly to each side, as shown in Figure 2.1. This figure shows a typical Green's function $G(x; \bar{x})$ for one particular choice of $\bar{x}$. To complete the definition of this function we need to know the value $G(\bar{x}; \bar{x})$ that it takes at the minimum. This value is determined by the fact that the jump in slope at this point must be 1, since

$$\begin{aligned}u'(\bar{x} + \epsilon) - u'(\bar{x} - \epsilon) &= \int_{\bar{x}-\epsilon}^{\bar{x}+\epsilon} u''(x)\,dx \\ &= \int_{\bar{x}-\epsilon}^{\bar{x}+\epsilon} \delta(x - \bar{x})\,dx \\ &= 1.\end{aligned} \tag{2.30}$$

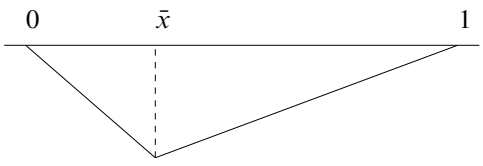


Figure 2.1. *The Green's function* $G(x; \bar{x})$ *from* (2.31).

A little algebra shows that the piecewise linear function $G(x;\bar{x})$ is given by

$$G(x;\bar{x}) = \begin{cases} (\bar{x}-1)x & \text{for } 0 \le x \le \bar{x}, \\ \bar{x}(x-1) & \text{for } \bar{x} \le x \le 1. \end{cases} \tag{2.31}$$

Note that by linearity, if we replaced $f(x)$ with $c\delta(x-\bar{x})$ for any constant c, the solution to the BVP would be $cG(x;\bar{x})$. Moreover, any linear combination of Green's functions at different points $\bar{x}$ is a solution to the BVP with the corresponding linear combination of delta functions on the right-hand side. So if we want to solve

$$u''(x) = 3\delta(x-0.3) - 5\delta(x-0.7), \tag{2.32}$$

for example (with $u(0) = u(1) = 0$), the solution is simply

$$u(x) = 3G(x;0.3) - 5G(x;0.7). \tag{2.33}$$

This is a piecewise linear function with jumps in slope of magnitude 3 at $x = 0.3$ and -5 at $x = 0.7$. More generally, if the right-hand side is a sum of weighted delta functions at any number of points,

$$f(x) = \sum_{k=1}^{n} c_k \delta(x - x_k), \tag{2.34}$$

then the solution to the BVP is

$$u(x) = \sum_{k=1}^{n} c_k G(x;x_k). \tag{2.35}$$

Now consider a general source $f(x)$ that is not a discrete sum of delta functions. We can view this as a continuous distribution of point sources, with $f(\bar{x})$ being a density function for the weight assigned to the delta function at $\bar{x}$, i.e.,

$$f(x) = \int_0^1 f(\bar{x})\delta(x-\bar{x})\,d\bar{x}. \tag{2.36}$$

(Note that if we smear out δ to ϕ_ϵ, then the right-hand side becomes a weighted average of values of f very close to x.) This suggests that the solution to $u''(x) = f(x)$ (still with $u(0) = u(1) = 0$) is

$$u(x) = \int_0^1 f(\bar{x})G(x;\bar{x})\,d\bar{x}, \tag{2.37}$$

and indeed it is.

Now let's consider more general boundary conditions. Since each Green's function $G(x;\bar{x})$ satisfies the homogeneous boundary conditions $u(0) = u(1) = 0$, any linear combination does as well. To incorporate the effect of nonzero boundary conditions, we introduce two new functions $G_0(x)$ and $G_1(x)$ defined by the BVPs

$$G_0''(x) = 0, \quad G_0(0) = 1, \quad G_0(1) = 0 \tag{2.38}$$

and

$$G_1''(x) = 0, \quad G_1(0) = 0, \quad G_1(1) = 1. \tag{2.39}$$

The solutions are

$$\begin{aligned} G_0(x) &= 1 - x, \\ G_1(x) &= x. \end{aligned} \tag{2.40}$$

These functions give the temperature distribution for the heat conduction problem with the temperature held at 1 at one boundary and 0 at the other with no internal heat source. Adding a scalar multiple of $G_0(x)$ to the solution $u(x)$ of (2.37) will change the value of $u(0)$ without affecting $u''(x)$ or $u(1)$, so adding $\alpha G_0(x)$ will allow us to satisfy the boundary condition at $x = 0$, and similarly adding $\beta G_1(x)$ will give the desired boundary value at $x = 1$. The full solution to (2.25) with boundary conditions (2.26) is thus

$$u(x) = \alpha G_0(x) + \beta G_1(x) + \int_0^1 f(\bar{x}) G(x; \bar{x})\, d\bar{x}. \tag{2.41}$$

Note that using the formula (2.31), we can rewrite this as

$$u(x) = \left(\alpha - \int_0^x \bar{x} f(\bar{x})\, d\bar{x}\right)(1 - x) + \left(\beta + \int_x^1 (\bar{x} - 1) f(\bar{x})\, d\bar{x}\right) x. \tag{2.42}$$

Of course this simple BVP can also be solved simply by integrating the function f twice, and the solution (2.42) can be put in this same form using integration by parts. But for our current purposes it is the form (2.41) that is of interest, since it shows clearly how the effect of each boundary condition and the local source at each point feeds into the global solution. The values α, β, and $f(x)$ are the data for this linear differential equation and (2.41) writes the solution as a linear operator applied to this data, analogous to writing the solution to the linear system $AU = F$ as $U = A^{-1}F$.

We are finally ready to return to the study of the max-norm stability of the finite difference method, which will be based on explicitly determining the inverse matrix for the matrix arising in this discretization. We will work with a slightly different formulation of the linear algebra problem in which we view U_0 and U_{m+1} as additional "unknowns" in the problem and introduce two new equations in the system that simply state that $U_0 = \alpha$ and $u_{m+1} = \beta$. The modified system has the form $AU = F$, where now

$$A = \frac{1}{h^2}\begin{bmatrix} h^2 & 0 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & 1 & -2 & 1 & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 0 & h^2 \end{bmatrix}, \quad U = \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ \vdots \\ U_{m-1} \\ U_m \\ U_{m+1} \end{bmatrix}, \quad F = \begin{bmatrix} \alpha \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) \\ \beta \end{bmatrix}. \tag{2.43}$$

While we could work directly with the matrix A from (2.10), this reformulation has two advantages:

1. It separates the algebraic equations corresponding to the boundary conditions from the algebraic equations corresponding to the ODE $u''(x) = f(x)$. In the system (2.10), the first and last equations contain a mixture of ODE and boundary conditions. Separating these terms will make it clearer how the inverse of A relates to the Green's function representation of the true solution found above.

2. In the next section we will consider Neumann boundary conditions $u'(0) = \sigma$ in place of $u(0) = \alpha$. In this case the value U_0 really is unknown and our new formulation is easily extended to this case by replacing the first row of A with a discretization of this boundary condition.

Let B denote the $(m + 2) \times (m + 2)$ inverse of A from (2.43), $B = A^{-1}$. We will index the elements of B by B_{00} through $B_{m+1,m+1}$ in the obvious manner. Let B_j denote the jth column of B for $j = 0, 1, \ldots, m + 1$. Then

$$AB_j = e_j,$$

where e_j is the jth column of the identity matrix. We can view this as a linear system to be solved for B_j. Note that this linear system is simply the discretization of the BVP for a special choice of right-hand side F in which only one element of this vector is nonzero. This is exactly analogous to the manner in which the Green's function for the ODE is defined. The column B_0 corresponds to the problem with $\alpha = 1$, $f(x) = 0$, and $\beta = 0$, and so we expect B_0 to be a discrete approximation of the function $G_0(x)$. In fact, the first (i.e., $j = 0$) column of B has elements obtained by simply evaluating G_0 at the grid points,

$$B_{i0} = G_0(x_i) = 1 - x_i. \tag{2.44}$$

Since this is a linear function, the second difference operator applied at any point yields zero. Similarly, the last ($j = m + 1$) column of B has elements

$$B_{i,m+1} = G_1(x_i) = x_i. \tag{2.45}$$

The interior columns ($1 \le j \le m$) correspond to the Green's function for zero boundary conditions and the source concentrated at a single point, since $F_j = 1$ and $F_i = 0$ for $i \ne j$. Note that this is a discrete version of $h\delta(x - x_j)$ since as a grid function F is nonzero over an interval of length h but has value 1 there, and hence total mass h. Thus we expect that the column B_j will be a discrete approximation to the function $hG(x; x_j)$. In fact, it is easy to check that

$$B_{ij} = hG(x_i; x_j) = \begin{cases} h(x_j - 1)x_i, & i = 1, 2, \ldots, j, \\ h(x_i - 1)x_j, & i = j, j + 1, \ldots, m. \end{cases} \tag{2.46}$$

An arbitrary right-hand side F for the linear system can be written as

$$F = \alpha e_0 + \beta e_{m+1} + \sum_{j=1}^{m} f_j e_j, \tag{2.47}$$

and the solution $U = BF$ is

$$U = \alpha B_0 + \beta B_{m+1} + \sum_{j=1}^{m} f_j B_j \tag{2.48}$$

with elements

$$U_i = \alpha(1 - x_i) + \beta x_i + h \sum_{j=1}^{m} f_j G(x_i; x_j). \tag{2.49}$$

This is the discrete analogue of (2.41).

In fact, something more is true: suppose we define a function $v(x)$ by

$$v(x) = \alpha(1 - x) + \beta x + h \sum_{j=1}^{m} f_j G(x; x_j). \tag{2.50}$$

Then $U_i = v(x_i)$ and $v(x)$ is the piecewise linear function that interpolates the numerical solution. This function $v(x)$ is the exact solution to the BVP

$$v''(x) = h \sum_{j=1}^{m} f(x_j)\delta(x - x_j), \quad v(0) = \alpha, \quad v(1) = \beta. \tag{2.51}$$

Thus we can interpret the discrete solution as the exact solution to a modified problem in which the right-hand side $f(x)$ has been replaced by a finite sum of delta functions at the grid points x_j, with weights $hf(x_j) \approx \int_{x_{j-1/2}}^{x_{j+1/2}} f(x)\,dx$.

To verify max-norm stability of the numerical method, we must show that $\|B\|_\infty$ is uniformly bounded as $h \to 0$. The infinity norm of the matrix is given by

$$\|B\|_\infty = \max_{0 \le j \le m+1} \sum_{j=0}^{m+1} |B_{ij}|,$$

the maximum row sum of elements in the matrix. Note that the first row of B has $B_{00} = 1$ and $B_{0j} = 0$ for $j > 0$, and hence row sum 1. Similarly the last row contains all zeros except for $B_{m+1,m+1} = 1$. The intermediate rows are dense and the first and last elements (from columns B_0 and B_{m+1}) are bounded by 1. The other m elements of each of these rows are all bounded by h from (2.46), and hence

$$\sum_{j=0}^{m+1} |B_{ij}| \le 1 + 1 + mh < 3$$

since $h = 1/(m + 1)$. Every row sum is bounded by 3 at most, and so $\|A^{-1}\|_\infty < 3$ for all h, and stability is proved.

While it may seem like we've gone to a lot of trouble to prove stability, the explicit representation of the inverse matrix in terms of the Green's functions is a useful thing to have, and if it gives additional insight into the solution process. Note, however, that it would *not* be a good idea to use the explicit expressions for the elements of $B = A^{-1}$ to solve the linear system by computing $U = BF$. Since B is a dense matrix, doing this matrix-vector multiplication requires $O(m^2)$ operations. We are much better off solving the original system $AU = F$ by Gaussian elimination. Since A is tridiagonal, this requires only $O(m)$ operations.

The Green's function representation also clearly shows the effect that each local truncation error has on the global error. Recall that the global error E is related to the local truncation error by $AE = -\tau$. This continues to hold for our reformulation of the problem, where we now define τ_0 and τ_{m+1} as the errors in the imposed boundary conditions, which are typically zero for the Dirichlet problem. Solving this system gives $E = -B\tau$. If we did make an error in one of the boundary conditions, setting F_0 to $\alpha + \tau_0$, the effect on the global error would be $\tau_0 B_0$. The effect of this error is thus nonzero across the entire interval, decreasing linearly from the boundary where the error is made at the other end. Each truncation error τ_i for $1 \le i \le m$ in the difference approximation to $u''(x_i) = f(x_i)$ likewise has an effect on the global error everywhere, although the effect is largest at the grid point x_i, where it is $hG(x_i; x_i)\tau_i$, and decays linearly toward each end. Note that since $\tau_i = O(h^2)$, the contribution of this error to the global error at each point is only $O(h^3)$. However, since all m local errors contribute to the global error at each point, the total effect is $O(mh^3) = O(h^2)$.

As a final note on this topic, observe that we have also worked out the inverse of the original matrix A defined in (2.10). Because the first row of B consists of zeros beyond the first element, and the last row consists of zeros, except for the last element, it is easy to check that the inverse of the $m \times m$ matrix from (2.10) is the $m \times m$ central block of B consisting of B_{11} through B_{mm}. The max-norm of this matrix is bounded by 1 for all h, so our original formulation is stable as well.

2.12 Neumann boundary conditions

Now suppose that we have one or more Neumann boundary conditions instead of Dirichlet boundary conditions, meaning that a boundary condition on the derivative u' is given rather than a condition on the value of u itself. For example, in our heat conduction example we might have one end of the rod insulated so that there is no heat flux through this end, and hence $u' = 0$ there. More generally we might have heat flux at a specified rate giving $u' = \sigma$ at this boundary.

We will see in the next section that imposing Neumann boundary conditions at both ends gives an ill-posed problem that has either no solution or infinitely many solutions. In this section we consider (2.25) with one Neumann condition, say,

$$u'(0) = \sigma, \qquad u(1) = \beta. \tag{2.52}$$

Figure 2.2 shows the solution to this problem with $f(x) = e^x$, $\sigma = 0$, and $\beta = 0$ as one example.

To solve this problem numerically, we need to determine U_0 as one of the unknowns. If we use the formulation of (2.43), then the first row of the matrix A must be modified to model the boundary condition (2.52).

First approach. As a first try, we might use a one-sided expression for $u'(0)$, such as

$$\frac{U_1 - U_0}{h} = \sigma. \tag{2.53}$$

If we use this equation in place of the first line of the system (2.43), we obtain the following system of equations for the unknowns $U_0,\ U_1,\ \ldots, U_m,\ U_{m+1}$:

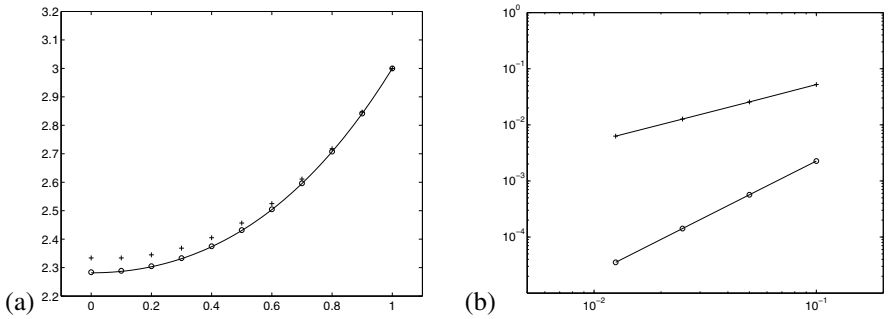


Figure 2.2. (a) *Sample solution to the steady-state heat equation with a Neumann boundary condition at the left boundary and Dirichlet at the right. The solid line is the true solution. The plus sign shows a solution on a grid with* 20 *points using* (2.53)*. The circle shows the solution on the same grid using* (2.55)*.* (b) *A log-log plot of the max-norm error as the grid is refined is also shown for each case.*

$$\frac{1}{h^2}\begin{bmatrix} -h & h & & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & 1 & -2 & 1 & & & & & \\ & & 1 & -2 & 1 & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & 1 & -2 & 1 & \\ & & & & & & 0 & h^2 & \end{bmatrix}\begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{m-1} \\ U_m \\ U_{m+1} \end{bmatrix} = \begin{bmatrix} \sigma \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) \\ \beta \end{bmatrix}. \tag{2.54}$$

Solving this system of equations does give an approximation to the true solution (see Figure 2.2), but checking the errors shows that this is only first order accurate. Figure 2.2 also shows a log-log plot of the max-norm errors as we refine the grid. The problem is that the local truncation error of the approximation (2.53) is $O(h)$, since

$$\begin{aligned} \tau_0 &= \frac{1}{h^2}(hu(x_1) - hu(x_0)) - \sigma \\ &= u'(x_0) + \frac{1}{2}hu''(x_0) + O(h^2) - \sigma \\ &= \frac{1}{2}hu''(x_0) + O(h^2). \end{aligned}$$

This translates into a global error that is only $O(h)$ as well.

Remark: It is sometimes possible to achieve second order accuracy even if the local truncation error is $O(h)$ at a single point, as long as it is $O(h^2)$ everywhere else. This is true here if we made an $O(h)$ truncation error at a single interior point, since the effect on the global error would be this $\tau_j B_j$, where B_j is the jth column of the appropriate inverse matrix. As in the Dirichlet case, this column is given by the corresponding Green's function scaled by h, and so the $O(h)$ local error would make an $O(h^2)$ contribution to the global error at each point. However, introducing an $O(h)$ error in τ_0 gives a contribution of $\tau_0 B_0$

to the global error, and as in the Dirichlet case this first column of B contains elements that are $O(1)$, resulting in an $O(h)$ contribution to the global error at every point.

Second approach. To obtain a second order accurate method, we can use a centered approximation to $u'(0) = \sigma$ instead of the one-sided approximation (2.53). We might introduce another unknown U_{-1} and, instead of the single equation (2.53), use the following two equations:

$$\begin{aligned} \frac{1}{h^2}(U_{-1} - 2U_0 + U_1) &= f(x_0), \\ \frac{1}{2h}(U_1 - U_{-1}) &= \sigma. \end{aligned} \tag{2.55}$$

This results in a system of $m + 3$ equations.

Introducing the unknown U_{-1} outside the interval $[0, 1]$ where the original problem is posed may seem unsatisfactory. We can avoid this by eliminating the unknown U_{-1} from the two equations (2.55), resulting in a single equation that can be written as

$$\frac{1}{h}(-U_0 + U_1) = \sigma + \frac{h}{2}f(x_0). \tag{2.56}$$

We have now reduced the system to one with only $m + 2$ equations for the unknowns $U_0, U_1, \ldots, U_{m+1}$. The matrix is exactly the same as the matrix in (2.54), which came from the one-sided approximation. The only difference in the linear system is that the first element in the right-hand side of (2.54) is now changed from σ to $\sigma + \frac{h}{2}f(x_0)$. We can interpret this as using the one-sided approximation to $u'(0)$, but with a modified value for this Neumann boundary condition that adjusts for the fact that the approximation has an $O(h)$ error by introducing the same error in the data σ.

Alternatively, we can view the left-hand side of (2.56) as a centered approximation to $u'(x_0 + h/2)$ and the right-hand side as the first two terms in the Taylor series expansion of this value,

$$u'\left(x_0 + \frac{h}{2}\right) = u'(x_0) + \frac{h}{2}u''(x_0) + \cdots = \sigma + \frac{h}{2}f(x_0) + \cdots.$$

Third approach. Rather than using a second order accurate centered approximation to the Neumann boundary condition, we could instead use a second order accurate one-sided approximation based on the three unknowns U_0, U_1, and U_2. An approximation of this form was derived in Example 1.2, and using this as the boundary condition gives the equation

$$\frac{1}{h}\left(\frac{3}{2}U_0 - 2U_1 + \frac{1}{2}U_2\right) = \sigma.$$

This results in the linear system

$$\frac{1}{h^2}\begin{bmatrix} 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & 1 & -2 & 1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & 1 \\ & & & & & & 0 & h^2 \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{m-1} \\ U_m \\ U_{m+1} \end{bmatrix} = \begin{bmatrix} \sigma \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) \\ \beta \end{bmatrix}. \tag{2.57}$$

This boundary condition is second order accurate from the error expression (1.12). The use of this equation slightly disturbs the tridiagonal structure but adds little to the cost of solving the system of equations and produces a second order accurate result. This approach is often the easiest to generalize to other situations, such as higher order accurate methods, nonuniform grids, or more complicated boundary conditions.

2.13 Existence and uniqueness

In trying to solve a mathematical problem by a numerical method, it is always a good idea to check that the original problem has a solution and in fact that it is *well posed* in the sense developed originally by Hadamard. This means that the problem should have a unique solution that depends continuously on the data used to define the problem. In this section we will show that even seemingly simple BVPs may fail to be well posed.

Consider the problem of Section 2.12 but now suppose we have Neumann boundary conditions at both ends, i.e., we have (2.6) with

$$u'(0) = \sigma_0, \qquad u'(1) = \sigma_1.$$

In this case the techniques of Section 2.12 would naturally lead us to the discrete system

$$\frac{1}{h^2}\begin{bmatrix} -h & h & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & 1 & -2 & 1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 1 & \\ & & & & & h & -h & \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_m \\ U_{m+1} \end{bmatrix} = \begin{bmatrix} \sigma_0 + \frac{h}{2} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_m) \\ -\sigma_1 + \frac{h}{2} f(x_{m+1}) \end{bmatrix}. \tag{2.58}$$

If we try to solve this system, however, we will soon discover that the matrix is singular, and in general the system has no solution. (Or, if the right-hand side happens to lie in the range of the matrix, it has infinitely many solutions.) It is easy to verify that the matrix is singular by noting that the constant vector $e = [1,\ 1,\ \ldots,\ 1]^T$ is a null vector.

This is not a failure in our numerical model. In fact it reflects that the problem we are attempting to solve is not well posed, and the differential equation will also have either no solution or infinitely many solutions. This can be easily understood physically by again considering the underlying heat equation discussed in Section 2.1. First consider the case

where $\sigma_0 = \sigma_1 = 0$ and $f(x) \equiv 0$ so that both ends of the rod are insulated, there is no heat flux through the ends, and there is no heat source within the rod. Recall that the BVP is a simplified equation for finding the steady-state solution of the heat equation (2.2) with some initial data $u^0(x)$. How does $u(x,t)$ behave with time? In the case now being considered the total heat energy in the rod must be conserved with time, so $\int_0^1 u(x,t)\,dx \equiv \int_0^1 u^0(x)\,dx$ for all time. Diffusion of the heat tends to redistribute it until it is uniformly distributed throughout the rod, so we expect the steady state solution $u(x)$ to be constant in x,

$$u(x) = c, \tag{2.59}$$

where the constant c depends on the initial data $u^0(x)$. In fact, by conservation of energy, $c = \int_0^1 u^0(x)\,dx$ for our rod of unit length. But notice now that *any* constant function of the form (2.59) is a solution of the steady-state BVP, since it satisfies all the conditions $u''(x) \equiv 0$, $u'(0) = u'(1) = 0$. The ODE has infinitely many solutions in this case. The physical problem has only one solution, but in attempting to simplify it by solving for the steady state alone, we have thrown away a crucial piece of data, which is the heat content of the initial data for the heat equation. If at least one boundary condition is a Dirichlet condition, then it can be shown that the steady-state solution is *independent* of the initial data and we can solve the BVP uniquely, but not in the present case.

Now suppose that we have a source term $f(x)$ that is not identically zero, say, $f(x) < 0$ everywhere. Then we are constantly adding heat to the rod (recall that $f = -\psi$ in (2.4)). Since no heat can escape through the insulated ends, we expect the temperature to keep rising without bound. In this case we never reach a steady state, and the BVP has no solution. On the other hand, if f is positive over part of the interval and negative elsewhere, and the net effect of the heat sources and sinks exactly cancels out, then we expect that a steady state might exist. In fact, solving the BVP exactly by integrating twice and trying to determine the constants of integration from the boundary conditions shows that a solution exists (in the case of insulated boundaries) only if $\int_0^1 f(x)\,dx = 0$, in which case there are infinitely many solutions. If σ_0 and/or σ_1 are nonzero, then there is heat flow at the boundaries and the net heat source must cancel the boundary fluxes. Since

$$u'(1) = u'(0) + \int_0^1 u''(x)\,dx = \int_0^1 f(x)\,dx, \tag{2.60}$$

this requires

$$\int_0^1 f(x)\,dx = \sigma_1 - \sigma_0. \tag{2.61}$$

Similarly, the singular linear system (2.58) has a solution (in fact infinitely many solutions) only if the right-hand side F is orthogonal to the null space of A^T. This gives the condition

$$\frac{h}{2} f(x_0) + h \sum_{i=1}^{m} f(x_i) + \frac{h}{2} f(x_{m+1}) = \sigma_1 - \sigma_0, \tag{2.62}$$

which is the trapezoidal rule approximation to the condition (2.61).

2.14 Ordering the unknowns and equations

Note that in general we are always free to change the order of the equations in a linear system without changing the solution. Modifying the order corresponds to permuting the rows of the matrix and right-hand side. We are also free to change the ordering of the unknowns in the vector of unknowns, which corresponds to permuting the columns of the matrix. As an example, consider the difference equations given by (2.9). Suppose we reordered the unknowns by listing first the unknowns at odd numbered grid points and then the unknowns at even numbered grid points, so that $\tilde{U} = [U_1,\ U_3,\ U_5,\ \ldots, U_2,\ U_4,\ \ldots]^T$. If we also reorder the equations in the same way, i.e., we write down first the difference equation centered at U_1, then at U_3, U_5, etc., then we would obtain the following system:

$$\frac{1}{h^2}\left[\begin{array}{ccccccc|ccccccc}
-2 & & & & & & & 1 & & & & & & \\
 & -2 & & & & & & 1 & 1 & & & & & \\
 & & -2 & & & & & & 1 & 1 & & & & \\
 & & & \ddots & & & & & & \ddots & \ddots & & & \\
 & & & & & -2 & & & & & & 1 & 1 & \\
\hline
1 & 1 & & & & & & -2 & & & & & & \\
 & 1 & 1 & & & & & & -2 & & & & & \\
 & & 1 & 1 & & & & & & -2 & & & & \\
 & & & \ddots & \ddots & & & & & & \ddots & & & \\
 & & & & 1 & & & & & & & -2 & &
\end{array}\right]\left[\begin{array}{c} U_1 \\ U_3 \\ U_5 \\ \vdots \\ U_{m-1} \\ \hline U_2 \\ U_4 \\ U_6 \\ \vdots \\ U_m \end{array}\right] \tag{2.63}$$

$$= \left[\begin{array}{c} f(x_1) - \alpha/h^2 \\ f(x_3) \\ f(x_5) \\ \vdots \\ f(x_{m-1}) \\ \hline f(x_2) \\ f(x_4) \\ f(x_6) \\ \vdots \\ f(x_m) - \beta/h^2 \end{array}\right].$$

This linear system has the same solution as (2.9) modulo the reordering of unknowns, but it looks very different. For this one-dimensional problem there is no point in reordering things this way, and the natural ordering $[U_1,\ U_2,\ U_3,\ \ldots]^T$ clearly gives the optimal matrix structure for the purpose of applying Gaussian elimination. By ordering the unknowns so that those which occur in the same equation are close to one another in the vector, we keep the nonzeros in the matrix clustered near the diagonal. In two or three space dimensions there are more interesting consequences of choosing different orderings, a topic we return to in Section 3.3.

2.15 A general linear second order equation

We now consider the more general linear equation

$$a(x)u''(x) + b(x)u'(x) + c(x)u(x) = f(x), \tag{2.64}$$

together with two boundary conditions, say, the Dirichlet conditions

$$u(a) = \alpha, \qquad u(b) = \beta. \tag{2.65}$$

This equation can be discretized to second order by

$$a_i \left(\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} \right) + b_i \left(\frac{U_{i+1} - U_{i-1}}{2h} \right) + c_i U_i = f_i, \tag{2.66}$$

where, for example, $a_i = a(x_i)$. This gives the linear system $AU = F$, where A is the tridiagonal matrix

$$A = \frac{1}{h^2} \begin{bmatrix} (h^2c_1 - 2a_1) & (a_1 + hb_1/2) & & & \\ (a_2 - hb_2/2) & (h^2c_2 - 2a_2) & (a_2 + hb_2/2) & & \\ \ddots & \ddots & \ddots & & \\ & (a_{m-1} - hb_{m-1}/2) & (h^2c_{m-1} - 2a_{m-1}) & (a_{m-1} + hb_{m-1}/2) & \\ & & (a_m - hb_m/2) & (h^2c_m - 2a_m) \end{bmatrix} \tag{2.67}$$

and

$$U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_{m-1} \\ U_m \end{bmatrix}, \qquad F = \begin{bmatrix} f_1 - (a_1/h^2 - b_1/2h)\alpha \\ f_2 \\ \vdots \\ f_{m-1} \\ f_m - (a_m/h^2 + b_m/2h)\beta \end{bmatrix}. \tag{2.68}$$

This linear system can be solved with standard techniques, assuming the matrix is nonsingular. A singular matrix would be a sign that the discrete system does not have a unique solution, which may occur if the original problem, or a nearby problem, is not well posed (see Section 2.13).

The discretization used above, while second order accurate, may not be the best discretization to use for certain problems of this type. Often the physical problem has certain properties that we would like to preserve with our discretization, and it is important to understand the underlying problem and be aware of its mathematical properties before blindly applying a numerical method. The next example illustrates this.

Example 2.1. Consider heat conduction in a rod with varying heat conduction properties, where the parameter $\kappa(x)$ varies with x and is always positive. The steady-state heat-conduction problem is then

$$(\kappa(x)u'(x))' = f(x) \tag{2.69}$$

together with some boundary conditions, say, the Dirichlet conditions (2.65). To discretize this equation we might be tempted to apply the chain rule to rewrite (2.69) as

$$\kappa(x)u''(x) + \kappa'(x)u'(x) = f(x) \tag{2.70}$$

and then apply the discretization (2.67), yielding the matrix

$$A = \frac{1}{h^2}\begin{bmatrix} -2\kappa_1 & (\kappa_1 + h\kappa_1'/2) & & & \\ (\kappa_2 - h\kappa_2'/2) & -2\kappa_2 & (\kappa_2 + h\kappa_2'/2) & & \\ \ddots & \ddots & \ddots & & \\ & (\kappa_{m-1} - h\kappa_{m-1}'/2) & -2\kappa_{m-1} & (\kappa_{m-1} + h\kappa_{m-1}'/2) & \\ & & (\kappa_m - h\kappa_m'/2) & -2\kappa_m \end{bmatrix}. \tag{2.71}$$

However, this is not the best approach. It is better to discretize the physical problem (2.69) directly. This can be done by first approximating $\kappa(x)u'(x)$ at points halfway between the grid points, using a centered approximation

$$\kappa(x_{i+1/2})u'(x_{i+1/2}) = \kappa_{i+1/2}\left(\frac{U_{i+1} - U_i}{h}\right)$$

and the analogous approximation at $x_{i-1/2}$. Differencing these then gives a centered approximation to $(\kappa u')'$ at the grid point x_i:

$$\begin{aligned}(\kappa u')'(x_i) &\approx \frac{1}{h}\left[\kappa_{i+1/2}\left(\frac{U_{i+1} - U_i}{h}\right) - \kappa_{i-1/2}\left(\frac{U_i - U_{i-1}}{h}\right)\right] \\ &= \frac{1}{h^2}[\kappa_{i-1/2}U_{i-1} - (\kappa_{i-1/2} + \kappa_{i+1/2})U_i + \kappa_{i+1/2}U_{i+1}].\end{aligned} \tag{2.72}$$

This leads to the matrix

$$A = \frac{1}{h^2}\begin{bmatrix} -(\kappa_{1/2} + \kappa_{3/2}) & \kappa_{3/2} & & & \\ \kappa_{3/2} & -(\kappa_{3/2} + \kappa_{5/2}) & \kappa_{5/2} & & \\ \ddots & \ddots & \ddots & & \\ & \kappa_{m-3/2} & -(\kappa_{m-3/2} + \kappa_{m-1/2}) & \kappa_{m-1/2} & \\ & & \kappa_{m-1/2} & -(\kappa_{m-1/2} + \kappa_{m+1/2}) \end{bmatrix}. \tag{2.73}$$

Comparing (2.71) to (2.73), we see that they agree to $O(h^2)$, noting, for example, that

$$\kappa(x_{i+1/2}) = \kappa(x_i) + \frac{1}{2}h\kappa'(x_i) + O(h^2) = \kappa(x_{i+1}) - \frac{1}{2}h\kappa'(x_{i+1}) + O(h^2).$$

However, the matrix (2.73) has the advantage of being *symmetric*, as we would hope, since the original differential equation is *self-adjoint*. Moreover, since $\kappa > 0$, the matrix can be shown to be nonsingular and *negative definite*. This means that all the eigenvalues are negative, a property also shared by the differential operator $\frac{\partial}{\partial x}\kappa(x)\frac{\partial}{\partial x}$ (see Section C.8). It is generally desirable to have important properties such as these modeled by the discrete approximation to the differential equation. One can then show, for example, that the solution to the difference equations satisfies a *maximum principle* of the same type as the solution to the differential equation: for the homogeneous equation with $f(x) \equiv 0$, the values of $u(x)$ lie between the values of the boundary values α and β everywhere, so the maximum and minimum values of u arise on the boundaries. For the heat conduction problem this is physically obvious: the steady-state temperature in the rod won't exceed what's imposed at the boundaries if there is no heat source.

When solving the resulting linear system by iterative methods (see Chapters 3 and 4) it is also often desirable that the matrix have properties such as negative definiteness, since some iterative methods (e.g., the conjugate-gradient (CG) method in Section 4.3) depend on such properties.

2.16 Nonlinear equations

We next consider a nonlinear BVP to illustrate the new complications that arise in this case. We will consider a specific example that has a simple physical interpretation which makes it easy to understand and interpret solutions. This example also illustrates that not all 2-point BVPs are steady-state problems.

Consider the motion of a pendulum with mass m at the end of a rigid (but massless) bar of length L, and let $\theta(t)$ be the angle of the pendulum from vertical at time t, as illustrated in Figure 2.3. Ignoring the mass of the bar and forces of friction and air resistance, we see that the differential equation for the pendulum motion can be well approximated by

$$\theta''(t) = -(g/L)\sin(\theta(t)), \tag{2.74}$$

where g is the gravitational constant. Taking $g/L = 1$ for simplicity we have

$$\theta''(t) = -\sin(\theta(t)) \tag{2.75}$$

as our model problem.

For small amplitudes of the angle θ it is possible to approximate $\sin(\theta) \approx \theta$ and obtain the approximate *linear* differential equation

$$\theta''(t) = -\theta(t) \tag{2.76}$$

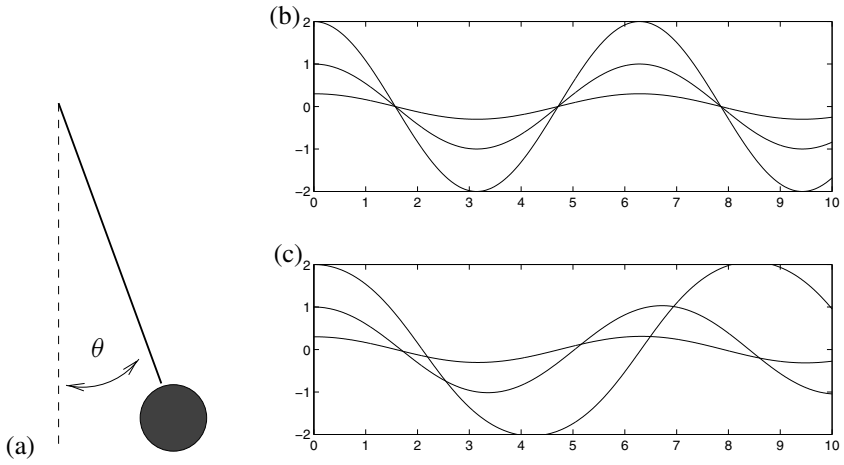


Figure 2.3. (a) *Pendulum.* (b) *Solutions to the linear equation* (2.76) *for various initial* θ *and zero initial velocity.* (c) *Solutions to the nonlinear equation* (2.75) *for various initial* θ *and zero initial velocity.*

with general solutions of the form $A\cos(t) + B\sin(t)$. The motion of a pendulum that is oscillating only a small amount about the equilibrium at $\theta = 0$ can be well approximated by this sinusoidal motion, which has period 2π independent of the amplitude. For larger-amplitude motions, however, solving (2.76) does not give good approximations to the true behavior. Figures 2.3(b) and (c) show some sample solutions to the two equations.

To fully describe the problem we also need to specify two auxiliary conditions in addition to the second order differential equation (2.75). For the pendulum problem the IVP is most natural—we set the pendulum swinging from some initial position $\theta(0)$ with some initial angular velocity $\theta'(0)$, which gives two initial conditions that are enough to determine a unique solution at all later times.

To obtain instead a BVP, consider the situation in which we wish to set the pendulum swinging from some initial given location $\theta(0) = \alpha$ with some unknown angular velocity $\theta'(0)$ in such a way that the pendulum will be at the desired location $\theta(T) = \beta$ at some specified later time T. Then we have a 2-point BVP

$$\begin{aligned} &\theta''(t) = -\sin(\theta(t)) \quad \text{for } 0 < t < T, \\ &\theta(0) = \alpha, \qquad \theta(T) = \beta. \end{aligned} \tag{2.77}$$

Similar BVPs do arise in more practical situations, for example, trying to shoot a missile in such a way that it hits a desired target. In fact, this latter example gives rise to the name *shooting method* for another approach to solving 2-point BVPs that is discussed in [4] and [54], for example.

2.16.1 Discretization of the nonlinear boundary value problem

We can discretize the nonlinear problem (2.75) in the obvious manner, following our approach for linear problems, to obtain the system of equations

$$\frac{1}{h^2}(\theta_{i-1} - 2\theta_i + \theta_{i+1}) + \sin(\theta_i) = 0 \tag{2.78}$$

for $i = 1, 2, \ldots, m$, where $h = T/(m+1)$ and we set $\theta_0 = \alpha$ and $\theta_{m+1} = \beta$. As in the linear case, we have a system of m equations for m unknowns. However, this is now a *nonlinear system* of equations of the form

$$G(\theta) = 0, \tag{2.79}$$

where $G : \mathbb{R}^m \to \mathbb{R}^m$. This cannot be solved as easily as the tridiagonal linear systems encountered so far. Instead of a direct method we must generally use some *iterative method*, such as Newton's method. If $\theta^{[k]}$ is our approximation to θ in step k, then *Newton's method* is derived via the Taylor series expansion

$$G(\theta^{[k+1]}) = G(\theta^{[k]}) + G'(\theta^{[k]})(\theta^{[k+1]} - \theta^{[k]}) + \cdots .$$

Setting $G(\theta^{[k+1]}) = 0$ as desired, and dropping the higher order terms, results in

$$0 = G(\theta^{[k]}) + G'(\theta^{[k]})(\theta^{[k+1]} - \theta^{[k]}).$$

This gives the Newton update

$$\theta^{[k+1]} = \theta^{[k]} + \delta^{[k]}, \tag{2.80}$$

where $\delta^{[k]}$ solves the linear system

$$J(\theta^{[k]})\delta^{[k]} = -G(\theta^{[k]}). \tag{2.81}$$

Here $J(\theta) \equiv G'(\theta) \in \mathbb{R}^{m\times m}$ is the *Jacobian matrix* with elements

$$J_{ij}(\theta) = \frac{\partial}{\partial\theta_j} G_i(\theta),$$

where $G_i(\theta)$ is the ith component of the vector-valued function G. In our case $G_i(\theta)$ is exactly the left-hand side of (2.78), and hence

$$J_{ij}(\theta) = \begin{cases} 1/h^2 & \text{if } j = i-1 \text{ or } j = i+1, \\ -2/h^2 + \cos(\theta_i) & \text{if } j = i, \\ 0 & \text{otherwise,} \end{cases}$$

so that

$$J(\theta) = \frac{1}{h^2}\begin{bmatrix} (-2+h^2\cos(\theta_1)) & 1 & & & \\ 1 & (-2+h^2\cos(\theta_2)) & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & (-2+h^2\cos(\theta_m)) \end{bmatrix}. \tag{2.82}$$

In each iteration of Newton's method we must solve a tridiagonal linear system similar to the single tridiagonal system that must be solved in the linear case.

Consider the nonlinear problem with $T = 2\pi$, $\alpha = \beta = 0.7$. Note that the linear problem (2.76) has infinitely many solutions in this particular case since the linearized pendulum has period 2π independent of the amplitude of motion; see Figure 2.3. This is not true of the nonlinear equation, however, and so we might expect a unique solution to the full nonlinear problem. With Newton's method we need an initial guess for the solution, and in Figure 2.4(a) we take a particular solution to the linearized problem, the one with initial angular velocity 0.5, as a first approximation, i.e., $\theta_i^{[0]} = 0.7\cos(t_i) + 0.5\sin(t_i)$. Figure 2.4(a) shows the different $\theta^{[k]}$ for $k = 0,\ 1,\ 2,\ \ldots$ that are obtained as we iterate with Newton's method. They rapidly converge to a solution to the nonlinear system (2.78). (Note that the solution looks similar to the solution to the linearized equation with $\theta'(0) = 0$, as we should have expected, and taking this as the initial guess, $\theta^{[0]} = 0.7\cos(t)$, would have given even more rapid convergence.)

Table 2.1 shows $\|\delta^{[k]}\|_\infty$ in each iteration, which measures the change in the solution. As expected, Newton's method appears to be converging quadratically.

If we start with a different initial guess $\theta^{[0]}$ (but still close enough to this solution), we would find that the method still converges to this same solution. For example, Figure 2.4(b) shows the iterates $\theta^{[k]}$ for $k = 0,\ 1,\ 2,\ \ldots$ with a different choice of $\theta^{[0]} \equiv 0.7$.

Newton's method can be shown to converge if we start with an initial guess that is sufficiently close to a solution. How close depends on the nature of the problem. For the

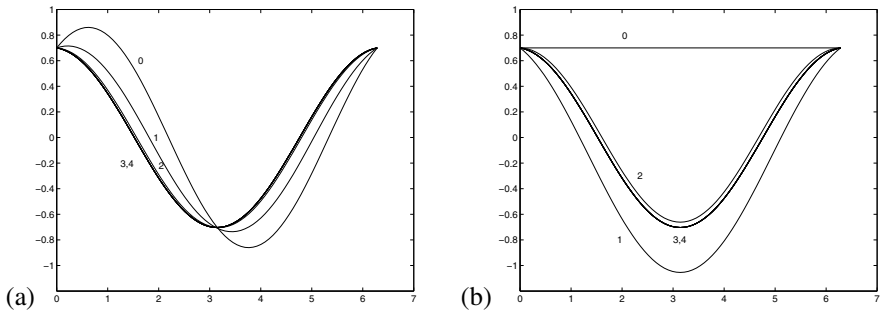


Figure 2.4. *Convergence of Newton iterates toward a solution of the pendulum problem. The iterates $\theta^{[k]}$ for $k = 1, 2, \ldots$ are denoted by the number k in the plots.* (a) *Starting from* $\theta_i^{[0]} = 0.7\cos(t_i) + 0.5\sin(t_i)$. (b) *Starting from* $\theta_i^{[0]} = 0.7$.

Table 2.1. *Change* $\|\delta^{[k]}\|_\infty$ *in solution in each iteration of Newton's method.*

k	Figure 2.4(a)	Figure 2.5
0	3.2841e−01	4.2047e+00
1	1.7518e−01	5.3899e+00
2	3.1045e−02	8.1993e+00
3	2.3739e−04	7.7111e−01
4	1.5287e−08	3.8154e−02
5	5.8197e−15	2.2490e−04
6	1.5856e−15	9.1667e−09
7		1.3395e−15

problem considered above one need not start very close to the solution to converge, as seen in the examples, but for more sensitive problems one might have to start extremely close. In such cases it may be necessary to use a technique such as *continuation* to find suitable initial data; see Section 2.19.

2.16.2 Nonuniqueness

The nonlinear problem does not have an infinite family of solutions the way the linear equation does on the interval $[0, 2\pi]$, and the solution found above is an *isolated solution* in the sense that there are no other solutions very nearby (it is also said to be *locally unique*). However, it does not follow that this is the unique solution to the BVP (2.77). In fact physically we should expect other solutions. The solution we found corresponds to releasing the pendulum with nearly zero initial velocity. It swings through nearly one complete cycle and returns to the initial position at time T.

Another possibility would be to propel the pendulum upward so that it rises toward the top (an unstable equilibrium) at $\theta = \pi$, before falling back down. By specifying the correct velocity we should be able to arrange it so that the pendulum falls back to $\theta = 0.7$ again at $T = 2\pi$. In fact it is possible to find such a solution for any $T > 0$.

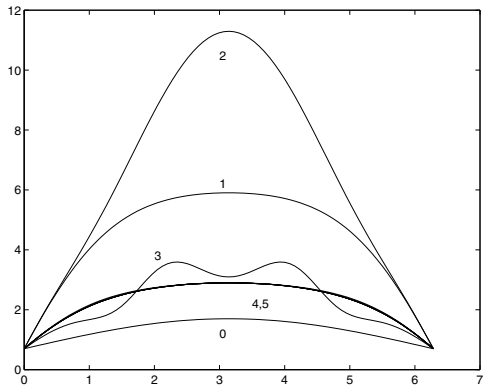


Figure 2.5. *Convergence of Newton iterates toward a different solution of the pendulum problem starting with initial guess* $\theta_i^{[0]} = 0.7 + \sin(t_i/2)$. *The iterates* k *for* $k = 1, 2, \ldots$ *are denoted by the number* k *in the plots.*

Physically it seems clear that there is a second solution to the BVP. To find it numerically we can use the same iteration as before, but with a different initial guess $\theta^{[0]}$ that is sufficiently close to this solution. Since we are now looking for a solution where θ initially increases and then falls again, let's try a function with this general shape. In Figure 2.5 we see the iterates $\theta^{[k]}$ generated with data $\theta_i^{[0]} = 0.7 + \sin(t_i/2)$. We have gotten lucky here on our first attempt, and we get convergence to a solution of the desired form. (See Table 2.1.) Different guesses with the same general shape might not work. Note that some of the iterates $\theta^{[k]}$ obtained along the way in Figure 2.5 do not make physical sense (since θ goes above π and then back down—what does this mean?), but the method still converges.

2.16.3 Accuracy on nonlinear equations

The solutions plotted above are not exact solutions to the BVP (2.77). They are only solutions to the discrete system of (2.78) with $h = 1/80$. How well do they approximate true solutions of the differential equation? Since we have used a second order accurate centered approximation to the second derivative in (2.8), we again hope to obtain second order accuracy as the grid is refined. In this section we will investigate this.

Note that it is very important to keep clear the distinction between the convergence of Newton's method to a solution of the finite difference equations and the convergence of this finite difference approximation to the solution of the differential equation. Table 2.1 indicates that we have obtained a solution to machine accuracy (roughly 10^{-15}) of the non linear system of equations by using Newton's method. This does *not* mean that our solution agrees with the true solution of the differential equation to the same degree. This depends on the size of h, the size of the truncation error in our finite difference approximation, and the relation between the local truncation error and the resulting global error.

Let's start by computing the local truncation error of the finite difference formula. Just as in the linear case, we define this by inserting the true solution of the differential

equation into the finite difference equations. This will not satisfy the equations exactly, and the residual is what we call the *local truncation error* (LTE):

$$\begin{aligned}\tau_i &= \frac{1}{h^2}(\theta(t_{i-1}) - 2\theta(t_i) + \theta(t_{i+1})) + \sin(\theta(t_i)) \\ &= (\theta''(t_i) + \sin(\theta(t_i))) + \frac{1}{12}h^2\theta''''(t_i) + O(h^4) \\ &= \frac{1}{12}h^2\theta''''(t_i) + O(h^4).\end{aligned} \tag{2.83}$$

Note that we have used the differential equation to set $\theta''(t_i) + \sin(\theta(t_i)) = 0$, which holds exactly since $\theta(t)$ is the exact solution. The LTE is $O(h^2)$ and has exactly the same form as in the linear case. (For a more complicated nonlinear problem it might not work out so simply, but similar expressions result.) The vector τ with components τ_i is simply $G(\hat{\theta})$, where $\hat{\theta}$ is the vector made up of the true solution at each grid point. We now want to obtain an estimate on the global error E based on this local error. We can attempt to follow the path used in Section 2.6 for linear problems. We have

$$\begin{aligned}G(\theta) &= 0, \\ G(\hat{\theta}) &= \tau,\end{aligned}$$

and subtracting gives

$$G(\theta) - G(\hat{\theta}) = -\tau. \tag{2.84}$$

We would like to derive from this a relation for the global error $E = \theta - \hat{\theta}$. If G were linear (say, $G(\theta) = A\theta - F$), we would have $G(\theta) - G(\hat{\theta}) = A\theta - A\hat{\theta} = A(\theta - \hat{\theta}) = AE$, giving an expression in terms of the global error $E = \theta - \hat{\theta}$. This is what we used in Section 2.7.

In the nonlinear case we cannot express $G(\theta) - G(\hat{\theta})$ directly in terms of $\theta - \hat{\theta}$. However, we can use Taylor series expansions to write

$$G(\theta) = G(\hat{\theta}) + J(\hat{\theta})E + O(\|E\|^2),$$

where $J(\hat{\theta})$ is again the Jacobian matrix of the difference formulas, evaluated now at the exact solution. Combining this with (2.84) gives

$$J(\hat{\theta})E = -\tau + O(\|E\|^2).$$

If we ignore the higher order terms, then we again have a linear relation between the local and global errors.

This motivates the following definition of stability. Here we let $\hat{J}^h$ denote the Jacobian matrix of the difference formulas evaluated at the true solution on a grid with grid spacing h.

Definition 2.2. *The nonlinear difference method $G(\theta) = 0$ is stable in some norm $\|\cdot\|$ if the matrices $(\hat{J}^h)^{-1}$ are uniformly bounded in this norm as $h \to 0$, i.e., there exist constants C and h_0 such that*

$$\|(\hat{J}^h)^{-1}\| \le C \quad \text{for all } h < h_0. \tag{2.85}$$

It can be shown that if the method is stable in this sense, and consistent in this norm ($\|\tau^h\| \to 0$), then the method converges and $\|E^h\| \to 0$ as $h \to 0$. This is not obvious in the nonlinear case: we obtain a linear system for E only by dropping the $O(\|E\|^2)$ nonlinear terms. Since we are trying to show that E is small, we can't necessarily assume that these terms are negligible in the course of the proof, at least not without some care. See [54] for a proof.

It makes sense that it is uniform boundedness of the inverse Jacobian at the exact solution that is required for stability. After all, it is essentially this Jacobian matrix that is used in solving linear systems in the course of Newton's method, once we get very close to the solution.

Warning: We state a final reminder that there is a difference between convergence of the difference method as $h \to 0$ and convergence of Newton's method, or some other iterative method, to the solution of the difference equations for some particular h. Stability of the difference method does not imply that Newton's method will converge from a poor initial guess. It can be shown, however, that with a stable method, Newton's method will converge from a sufficiently good initial guess; see [54]. Also, the fact that Newton's method has converged to a solution of the nonlinear system of difference equations, with an error of 10^{-15}, say, does not mean that we have a good solution to the original differential equation. The global error of the difference equations determines this.

2.17 Singular perturbations and boundary layers

In this section we consider some singular perturbation problems to illustrate the difficulties that can arise when numerically solving problems with boundary layers or other regions where the solution varies rapidly. See [55], [56] for more detailed discussions of singular perturbation problems. In particular, the example used here is very similar to one that can be found in [55], where solution by matched asymptotic expansions is discussed.

As a simple example we consider a steady-state advection-diffusion equation. The time-dependent equation has the form

$$u_t + au_x = \kappa u_{xx} + \psi \tag{2.86}$$

in the simplest case. This models the temperature $u(x,t)$ of a fluid flowing through a pipe with constant velocity a, where the fluid has constant heat diffusion coefficient κ and ψ is a source term from heating through the walls of the tube.

If $a > 0$, then we naturally have a boundary condition at the left boundary (say, $x = 0$),

$$u(0,t) = \alpha(t),$$

specifying the temperature of the incoming fluid. At the right boundary (say, $x = 1$) the fluid is flowing out and so it may seem that the temperature is determined only by what is happening in the pipe, and no boundary condition is needed here. This is correct if $\kappa = 0$ since the first order advection equation needs only one boundary condition and we are allowed to specify u only at the left boundary. However, if $\kappa > 0$, then heat can diffuse upstream, and we need to also specify $u(1,t) = \beta(t)$ to determine a unique solution.

If α, β, and ψ are all independent of t, then we expect a steady-state solution, which we hope to find by solving the linear 2-point boundary value problem

$$\begin{aligned} au'(x) &= \kappa u''(x) + \psi(x), \\ u(0) &= \alpha, \qquad u(1) = \beta. \end{aligned} \tag{2.87}$$

This can be discretized using the approach of Section 2.4. If a is small relative to κ, then this problem is easy to solve. In fact for $a = 0$ this is just the steady-state heat equation discussed in Section 2.15, and for small a the solution appears nearly identical.

But now suppose a is large relative to κ (i.e., we crank up the velocity, or we decrease the ability of heat to diffuse with the velocity $a > 0$ fixed). More properly we should work in terms of the nondimensional *Péclet number*, which measures the ratio of advection velocity to transport speed due to diffusion. Here we introduce a parameter ϵ which is like the inverse of the Péclet number, $\epsilon = \kappa/a$, and rewrite (2.87) in the form

$$\epsilon u''(x) - u'(x) = f(x). \tag{2.88}$$

Then taking a large relative to κ (large Péclet number) corresponds to the case $\epsilon \ll 1$.

We should expect difficulties physically in this case where advection overwhelms diffusion. It would be very difficult to maintain a fixed temperature at the outflow end of the tube in this situation. If we had a thermal device that was capable of doing so by instantaneously heating the fluid to the desired temperature as it passes the right boundary, independent of the temperature of the fluid flowing toward this point, then we would expect the temperature distribution to be essentially discontinuous at this boundary.

Mathematically we expect trouble as $\epsilon \to 0$ because in the limit $\epsilon = 0$ the equation (2.88) reduces to a *first order* equation (the steady advection equation)

$$-u'(x) = f(x), \tag{2.89}$$

which allows only one boundary condition, rather than two. For $\epsilon > 0$, no matter how small, we have a second order equation that needs two conditions, but we expect to perhaps see strange behavior at the outflow boundary as $\epsilon \to 0$, since in the limit we are over specifying the problem.

Figure 2.6(a) shows how solutions to (2.88) appear for various values of ϵ in the case $\alpha = 1$, $\beta = 3$, and $f(x) = -1$. In this case the exact solution is

$$u(x) = \alpha + x + (\beta - \alpha - 1)\left(\frac{e^{x/\epsilon} - 1}{e^{1/\epsilon} - 1}\right). \tag{2.90}$$

Note that as $\epsilon \to 0$ the solution tends toward a discontinuous function that jumps to the value β at the last possible moment. This region of rapid transition is called the *boundary layer* and it can be shown that for this problem the width of this layer is $O(\epsilon)$ as $\epsilon \to 0$.

The equation (2.87) with $0 < \epsilon \ll 1$ is called a *singularly perturbed equation*. It is a small perturbation of (2.89), but this small perturbation completely changes the character of the equation (from a first order to a second order equation). Typically any differential equation having a small parameter multiplying the highest order derivative will give a singular perturbation problem.

By contrast, going from the pure diffusion equation $\kappa u_{xx} = f$ to an advection diffusion equation $\kappa u_{xx} - a u_x = f$ for very small a is a *regular perturbation*. Both of these equations are second order differential equations requiring the same number of

boundary conditions. The solution of the perturbed equation looks nearly identical to the solution of the unperturbed equation for small a, and the difference in solutions is $O(a)$ as $a \to 0$.

Singular perturbation problems cause numerical difficulties because the solution changes rapidly over a very small interval in space. In this region derivatives of $u(x)$ are large, giving rise to large errors in our finite difference approximations. Recall that the error in our approximation to $u''(x)$ is proportional to $h^2 u''''(x)$, for example. If h is not small enough, then the local truncation error will be very large in the boundary layer. Moreover, even if the truncation error is large only in the boundary layer, the resulting global error may be large everywhere. (Recall that the global error E is obtained from the truncation error τ by solving a linear system $AE = -\tau$, which means that each element of E depends on *all* elements of τ since A^{-1} is a dense matrix.) This is clearly seen in Figure 2.6(b), where the numerical solution with $h = 1/10$ is plotted. Errors are large even in regions where the exact solution is nearly linear and $u'''' \approx 0$.

On finer grids the solution looks better (see Figure 2.6(c) and (d)), and as $h \to 0$ the method does exhibit second order accurate convergence. But it is necessary to have a sufficiently fine grid before reasonable results are obtained; we need enough grid points to enable the boundary layer to be well resolved.

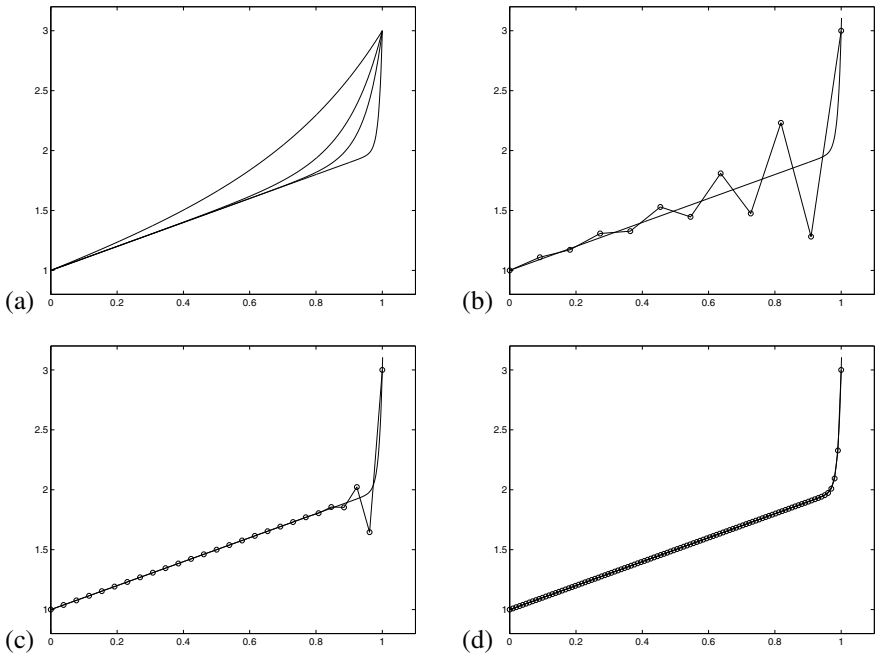


Figure 2.6. (a) *Solutions to the steady state advection-diffusion equation* (2.88) *for different values of* ϵ. *The four lines correspond to* $\epsilon = 0.3,\ 0.1,\ 0.05$, *and* 0.01 *from top to bottom.* (b) *Numerical solution with* $\epsilon = 0.01$ *and* $h = 1/10$. (c) $h = 1/25$. (d) $h = 1/100$.

2.17.1 Interior layers

The above example has a boundary layer, a region of rapid transition at one boundary. Other problems may have *interior layers* instead. In this case the solution is smooth except for some thin region interior to the interval where a rapid transition occurs. Such problems can be even more difficult to solve since we often don't know a priori where the interior layer will be. Perturbation theory can often be used to analyze singular perturbation problems and predict where the layers will occur, how wide they will be (as a function of the small parameter ϵ), and how the solution behaves. The use of perturbation theory to obtain good approximations to problems of this type is a central theme of classical applied mathematics.

These analytic techniques can often be used to good advantage along with numerical methods, for example, to obtain a good initial guess for Newton's method, or to choose an appropriate nonuniform grid as discussed in the next section. In some cases it is possible to develop special numerical methods that have the correct singular behavior built into the approximation in such a way that far better accuracy is achieved than with a naive numerical method.

Example 2.2. Consider the nonlinear boundary value problem

$$\begin{aligned} &\epsilon u'' + u(u' - 1) = 0 \qquad \text{for } a \le x \le b, \\ &u(a) = \alpha, \quad u(b) = \beta. \end{aligned} \tag{2.91}$$

For small ϵ this is a singular perturbation problem since ϵ multiplies the highest order derivative. Setting $\epsilon = 0$ gives a reduced equation

$$u(u' - 1) = 0 \tag{2.92}$$

for which we generally can enforce only one boundary condition. Solutions to (2.92) are $u(x) \equiv 0$ or $u(x) = x + C$ for some constant C. If the boundary condition imposed at $x = a$ or $x = b$ is nonzero, then the solution has the latter form and is either

$$u(x) = x + \alpha - a \quad \text{if } u(a) = \alpha \text{ is imposed} \tag{2.93}$$

or

$$u(x) = x + \beta - b \quad \text{if } u(b) = \beta \text{ is imposed.} \tag{2.94}$$

These two solutions are shown in Figure 2.7.

For $0 < \epsilon \ll 1$, the full equation (2.91) has a solution that satisfies both boundary conditions, and Figure 2.7 also shows such a solution. Over most of the domain the solution is smooth and u'' is small, in which case $\epsilon u''$ is negligible and the solution must nearly satisfy (2.92). Thus over most of the domain the solution follows one of the linear solutions to the reduced equation. Both boundary conditions can be satisfied by following one solution (2.93) near $x = a$ and the other solution (2.94) near $x = b$. Connecting these two smooth portions of the solution is a narrow zone (the interior solution) where $u(x)$ is rapidly varying. In this layer u'' is very large and the $\epsilon u''$ term of (2.91) is not negligible, and hence $u(u' - 1)$ may be far from zero in this region.

To determine the location and width of the interior layer, and the approximate form of the solution in this layer, we can use perturbation theory. Focusing attention on this

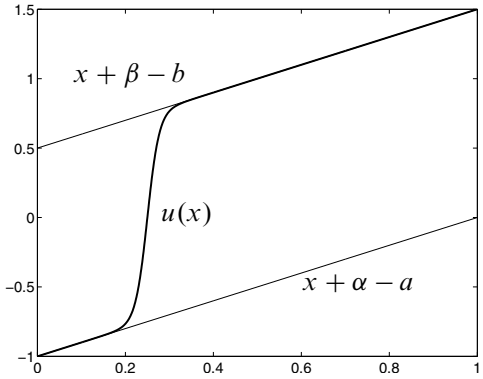


Figure 2.7. *Outer solutions and full solution to the singular perturbation problem with* $a = 0$, $b = 1$, $\alpha = -1$, *and* $\beta = 1.5$. *The solution has an interior layer centered about* $\bar{x} = 0.25$.

layer, which we now assume is centered at some location $x = \bar{x}$, we can zoom in on the solution by assuming that $u(x)$ has the approximate form

$$u(x) = W((x - \bar{x})/\epsilon^k) \tag{2.95}$$

for some power k to be determined. We are zooming in on a layer of width $O(\epsilon^k)$ asymptotically, so determining k will tell us how wide the layer is. From (2.95) we compute

$$\begin{aligned} u'(x) &= \epsilon^{-k} W'((x - \bar{x})/\epsilon^k), \\ u''(x) &= \epsilon^{-2k} W'((x - \bar{x})/\epsilon^k). \end{aligned} \tag{2.96}$$

Inserting these expressions in (2.91) gives

$$\epsilon \cdot \epsilon^{-2k} W''(\xi) + W(\xi)(\epsilon^{-k} W'(\xi) - 1) = 0,$$

where $\xi = (x - \bar{x})/\epsilon^k$. Multiply by ϵ^{2k-1} to obtain

$$W''(\xi) + W(\xi)(\epsilon^{k-1} W'(\xi) - \epsilon^{2k-1}) = 0. \tag{2.97}$$

By rescaling the independent variable by a factor ϵ^k, we have converted the singular perturbation problem (2.91) into a problem where the highest order derivative W'' has coefficient 1 and the small parameter appears only in the lower order term. However, the lower order term behaves well in the limit $\epsilon \to 0$ only if we take $k \geq 1$. For smaller values of k (zooming in on too large a region asymptotically), the lower order term blows up as $\epsilon \to 0$, or dividing by ϵ^{k-1} shows that we still have a singular perturbation problem. This gives us some information on k.

If we fix x at any value away from $\bar{x}$, then $\xi \to \pm\infty$ as $\epsilon \to 0$. So the boundary value problem (2.97) for $W(\xi)$ has boundary conditions at $\pm\infty$,

$$\begin{aligned} W(\xi) &\to \bar{x} + \alpha - a \quad \text{as } \xi \to -\infty, \\ W(\xi) &\to \bar{x} + \beta - b \quad \text{as } \xi \to +\infty. \end{aligned} \tag{2.98}$$

The "inner solution" $W(\xi)$ will then match up with the "outer solutions" given by (2.93) and (2.94) at the edges of the layer. We also require

$$W'(\xi) \to 0 \quad \text{as } \xi \to \pm\infty \tag{2.99}$$

since outside the layer the linear functions (2.93) and (2.94) have the desired slope.

For (2.97) to give a reasonable 2-point boundary value problem with these three boundary conditions (2.98) and (2.99), we must take $k = 1$. We already saw that we need $k \geq 1$, but we also cannot take $k > 1$ since in this case the lower order term in (2.97) vanishes as $\epsilon \to 0$ and the equation reduces to $W''(\xi) = 0$. In this case we are zooming in too far on the solution near $x = \bar{x}$ and the solution simply appears linear, as does any sufficiently smooth function if we zoom in on its behavior at a fixed point. While this does reveal the behavior extremely close to $\bar{x}$, it does not allow us to capture the full behavior in the interior layer. We cannot satisfy all four boundary conditions on W with a solution to $W''(x) = 0$.

Taking $k = 1$ gives the proper interior problem, and (2.97) becomes

$$W''(\xi) + W(\xi)(W'(\xi) - \epsilon) = 0. \tag{2.100}$$

Now letting $\epsilon \to 0$ we obtain

$$W''(\xi) + W(\xi)W'(\xi) = 0. \tag{2.101}$$

This equation has solutions of the form

$$W(\xi) = w_0 \tanh(w_0\xi/2) \tag{2.102}$$

for arbitrary constants w_0. The boundary conditions (2.98) lead to

$$w_0 = \frac{1}{2}(a - b + \beta - \alpha) \tag{2.103}$$

and

$$\bar{x} = \frac{1}{2}(a + b - \alpha - \beta). \tag{2.104}$$

To match this solution to the outer solutions, we require $a < \bar{x} < b$. If the value of $\bar{x}$ determined by (2.104) doesn't satisfy this condition, then the original problem has a boundary layer at $x = a$ (if $\bar{x} \leq a$) or at $x = b$ (if $\bar{x} \geq b$) instead of an interior layer. For the remainder of this discussion we assume $a < \bar{x} < b$.

We can combine the inner and outer solutions to obtain an approximate solution of the form

$$u(x) \approx \tilde{u}(x) \equiv x - \bar{x} + w_0 \tanh(w_0(x - \bar{x})/2\epsilon). \tag{2.105}$$

Singular perturbation analysis has given us a great deal of information about the solution to the problem (2.91). We know that the solution has an interior layer of width $O(\epsilon)$ at $x = \bar{x}$ with roughly linear solution (2.93), (2.94) outside the layer. This type of information may be all we need to know about the solution for some applications. If we want to determine a more detailed numerical approximation to the full solution, this analytical

information can be helpful in devising an accurate and efficient numerical method, as we now consider.

The problem (2.91) can be solved numerically on a uniform grid using the finite difference equations

$$G_i(U) \equiv \epsilon \left(\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} \right) + U_i \left(\frac{U_{i+1} - U_{i-1}}{2h} - 1 \right) = 0 \tag{2.106}$$

for $i = 1, 2, \ldots, m$ with $U_0 = \alpha$ and $U_{m+1} = \beta$ (where, as usual, $h = (b-a)/(m+1)$). This gives a nonlinear system of equations $G(U) = 0$ that can be solved using Newton's method as described in Section 2.16.1. One way to use the singular perturbation approximation is to generate a good initial guess for Newton's method, e.g.,

$$U_i = \tilde{u}(x_i), \tag{2.107}$$

where $\tilde{u}(x)$ is the approximate solution from (2.105). We then have an initial guess that is already very accurate at nearly all grid points. Newton's method converges rapidly from such a guess. If the grid is fine enough that the interior layer is well resolved, then a good approximation to the full solution is easily obtained. By contrast, starting with a more naive initial guess such as $U_i = \alpha + (x - a)(\beta - \alpha)/(b - a)$ leads to nonconvergence when ϵ is small.

When ϵ is very small, highly accurate numerical results can be obtained with less computation by using a nonuniform grid, with grid points clustered in the layer. To construct such a grid we can use the singular perturbation analysis to tell us where the points should be clustered (near $\bar{x}$) and how wide to make the clustering zone. The width of the layer is $O(\epsilon)$ and, moreover, from (2.102) we expect that most of the transition occurs for, say, $|\frac{1}{2} w_0 \xi| < 2$. This translates into

$$|x - \bar{x}| < 4\epsilon / w_0, \tag{2.108}$$

where w_0 is given by (2.103). The construction and use of nonuniform grids is pursued further in the next section.

2.18 Nonuniform grids

From Figure 2.6 it is clear that we need to choose our grid to be fine enough so that several points are within the boundary layer and we can obtain a reasonable solution. If we wanted high accuracy within the boundary layer we would have to choose a much finer grid than shown in this figure. With a uniform grid this means using a very large number of grid points, the vast majority of which are in the region where the solution is very smooth and could be represented well with far fewer points. This waste of effort may be tolerable for simple one-dimensional problems but can easily be intolerable for more complicated problems, particularly in more than one dimension.

Instead it is preferable to use a nonuniform grid for such calculations, with grid points clustered in regions where they are most needed. This requires using formulas that are sufficiently accurate on nonuniform grids. For example, a four-point stencil can be used to obtain second order accuracy for the second derivative operator. Using this for a linear

problem would give a banded matrix with four nonzero diagonals. A little extra care is needed at the boundaries.

One way to specify nonuniform grid points is to start with a uniform grid in some "computational coordinate" z, which we will denote by $z_i = ih$ for $i = 0, 1, \ldots, m+1$, where $h = 1/(m+1)$, and then use some appropriate *grid mapping* function $X(z)$ to define the "physical grid points" $x_i = X(z_i)$. This is illustrated in Figure 2.8, where z is plotted on the vertical axis and x is on the horizontal axis. The curve plotted represents a function $X(z)$, although with this choice of axes it is more properly the graph of the inverse function $z = X^{-1}(x)$. The horizontal and vertical lines indicate how the uniform grid points on the z axis are mapped to nonuniform points in x. If the problem is posed on the interval $[a, b]$, then the function $X(z)$ should be monotonically increasing and satisfy $X(0) = a$ and $X(1) = b$.

Note that grid points are clustered in regions where the curve is steepest, which means that $X(z)$ varies slowly with z, and spread apart in regions where $X(z)$ varies rapidly with z. Singular perturbation analysis of the sort done in the previous section may provide guidelines for where the grid points should be clustered.

Once a set of grid points x_i is chosen, it is necessary to set up and solve an appropriate system of difference equations on this grid. In general a different set of finite difference coefficients will be required at each grid point, depending on the spacing of the grid points nearby.

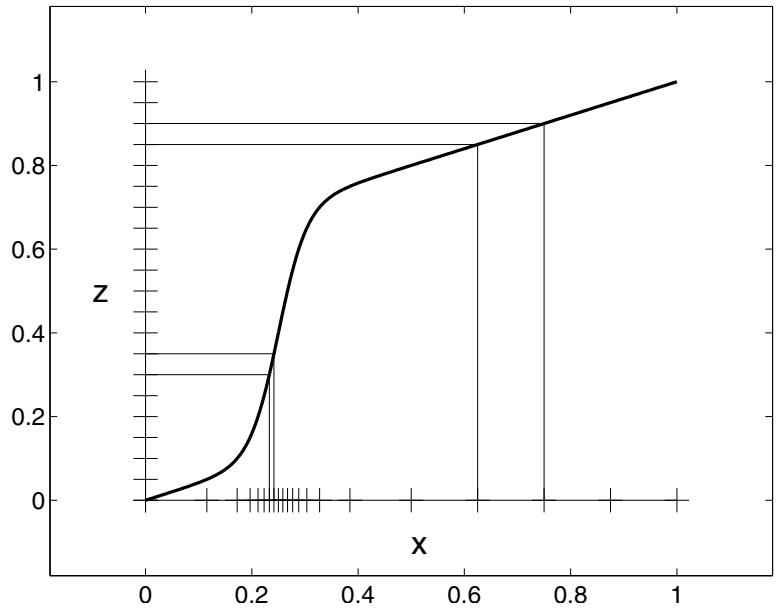


Figure 2.8. *Grid mapping from a uniform grid in* $0 \le z \le 1$ *(vertical axis) to the nonuniform grid in physical* x*-space shown on the horizontal axis. This particular mapping may be useful for solving the singular perturbation problem illustrated in Fig. 2.7.*

Example 2.3. As an example, again consider the simple problem $u''(x) = f(x)$ with the boundary conditions (2.52), $u'(0) = \sigma$, and $u(1) = \beta$. We would like to generalize the matrix system (2.57) to the situation where the x_i are nonuniformly distributed in the interval $[0, 1]$. In MATLAB this is easily accomplished using the `fdcoeffV` function discussed in Section 1.5, and the code fragment below shows how this matrix can be computed. Note that in MATLAB the vector `x` must be indexed from 1 to `m+2` rather than from 0 to `m+1`.

```
A = speye(m+2);   % initialize using sparse storage
% first row for Neumann BC, approximates u'(x(1))
A(1,1:3) = fdcoeffV(1, x(1), x(1:3));
% interior rows approximate u''(x(i))
for i=2:m+1
   A(i,i-1:i+1) = fdcoeffV(2, x(i), x((i-1):(i+1)));
   end
% last row for Dirichlet BC, approximates u(x(m+2))
A(m+2,m:m+2) = fdcoeffV(0,x(m+2),x(m:m+2));
```

A complete program that uses this and tests the order of accuracy of this method is available on the Web page.

Note that in this case the finite difference coefficients for the 3-point approximations to $u''(x_i)$ also can be explicitly calculated from the formula (1.14), but it is simpler to use `fdcoeffV`, and the above code also can be easily generalized to higher order methods by using more points in the stencils.

What accuracy do we expect from this method? In general if x_{i-1}, x_i and x_{i+1} are not equally spaced, then we expect an approximation to the second derivative $u''(x_i)$ based on these three points to be only first order accurate ($n = 3$ and $k = 2$ in the terminology of Section 1.5, so we expect $p = n - k = 1$). This is confirmed by the error expression (1.16), and this is generally what we will observe if we take randomly spaced grid points x_i.

However, in practice we normally use grids that are smoothly varying, for example, $x_i = X(z_i)$, where $X(z)$ is some smooth function, as discussed in Section 2.18. In this case it turns out that we should expect to achieve second order accuracy with the method just presented, and that is what is observed in practice. This can be seen from the error expressions (1.16): the "first order" portion of the error is proportional to

$$h_2 - h_1 = (x_{i+1} - x_i) - (x_i - x_{i-1}) = X(z_{i+1}) - 2X(z_i) + X(z_{i-1}) \approx h^2 X''(z_i),$$

where $h = \Delta z$ is the grid spacing in z. So we see that for a smoothly varying grid the difference $h_2 - h_1$ is actually $O(h^2)$. Hence the local truncation error is $O(h^2)$ at each grid point and we expect second order accuracy globally.

2.18.1 Adaptive mesh selection

Ideally a numerical method would work robustly on problems with interior or boundary layers without requiring that the user know beforehand how the solution is behaving. This can often be achieved by using methods that incorporate some form of *adaptive mesh selection*, which means that the method selects the mesh based on the behavior of the solution

and automatically clusters grid points in regions where they are needed. A discussion of this topic is beyond the scope of this book. See, for example, [4].

Readers who wish to use methods on nonuniform grids are encouraged to investigate software that will automatically choose an appropriate grid for a given problem (perhaps with some initial guidance) and take care of all the details of discretizing on this grid. In MATLAB, the routine `bvp4c` can be used and links to other software may be found on the book's Web page.

2.19 Continuation methods

For a difficult problem (e.g., a boundary layer or interior layer problem with $\epsilon \ll 1$), an adaptive mesh refinement program may not work well unless a reasonable initial grid is provided that already has some clustering in the appropriate layer location. Moreover, Newton's method may not converge unless we have a good initial guess for the solution. We have seen how information about the layer location and width and the approximate form of the solution can sometimes be obtained by using singular perturbation analysis.

There is another approach that is often easier in practice, known as *continuation* or the *homotopy method*. As an example, consider again the interior layer problem considered in Example 2.2 and suppose we want to solve this problem for a very small value of ϵ, say, $\epsilon = 10^{-6}$. Rather than immediately tackling this problem, we could first solve the problem with a much larger value of ϵ, say, $\epsilon = 0.1$, for which the solution is quite smooth and convergence is easily obtained on a uniform grid with few points. This solution can then be used as an initial guess for the problem with a smaller value of ϵ, say, $\epsilon = 10^{-2}$. We can repeat this process as many times as necessary to get to the desired value of ϵ, perhaps also adapting the grid as we go along to cluster points near the location of the interior layer (which is independent of ϵ and becomes clearly defined as we reduce ϵ).

More generally, the idea of following the solution to a differential equation as some parameter in the equation varies arises in other contexts as well. Difficulties sometimes arise at particular parameter values, such as *bifurcation points*, where two paths of solutions intersect.

2.20 Higher order methods

So far we have considered only second order methods for solving BVPs. Several approaches can be used to obtain higher order accurate methods. In this section we will look at various approaches to achieving higher polynomial order, such as fourth order or sixth order approximations. In Section 2.21 we briefly introduce spectral methods that can achieve convergence at exponential rates under some conditions.

2.20.1 Fourth order differencing

The obvious approach is to use a better approximation to the second derivative operator in place of the second order difference used in (2.8). For example, the finite difference approximation

$$\frac{1}{12h^2}[-U_{j-2} + 16U_{j-1} - 30U_j + 16U_{j+1} - U_{j+2}] \tag{2.109}$$

gives a fourth order accurate approximation to $u''(x_j)$. Note that this formula can be easily found in MATLAB by `fdcoeffV(2,0,-2:2)`.

For the BVP $u''(x) = f(x)$ on a grid with m interior points, this approximation can be used at grid points $j = 2, 3, \ldots, m-1$ but not for $j = 1$ or $j = m$. At these points we must use methods with only one point in the stencil to the left or right, respectively. Suitable formulas can again be found using `fdcoeffV`; for example,

$$\frac{1}{12h^2}[11U_0 - 20U_1 + 6U_2 + 4U_3 - U_4] \tag{2.110}$$

is a third order accurate formula for $u''(x_1)$ and

$$\frac{1}{12h^2}[10U_0 - 15U_1 - 4U_2 + 14U_3 - 6U_4 + U_5] \tag{2.111}$$

is fourth order accurate. As in the case of the second order methods discussed above, we can typically get away with one less order at one point near the boundary, but somewhat better accuracy is expected if (2.111) is used.

These methods are easily extended to nonuniform grids using the same approach as in Section 2.18. The matrix is essentially pentadiagonal except in the first and last two rows, and using sparse matrix storage ensures that the system is solved in $O(m)$ operations. Fourth order accuracy is observed as long as the grid is smoothly varying.

2.20.2 Extrapolation methods

Another approach to obtaining fourth order accuracy is to use the second order accurate method on two different grids, with spacing h (the coarse grid) and $h/2$ (the fine grid), and then to extrapolate in h to obtain a better approximation on the coarse grid that turns out to have $O(h^4)$ errors for this problem.

Denote the coarse grid solution by

$$U_j \approx u(jh), \qquad i = 1, 2, \ldots, m,$$

and the fine grid solution by

$$V_i \approx u(ih/2), \qquad i = 1, 2, \ldots, 2m+1,$$

and note that both U_j and V_{2j} approximate $u(jh)$. Because the method is a centered second order accurate method, it can be shown that the error has the form of an even-order expansion in powers of h,

$$U_j - u(jh) = C_2h^2 + C_4h^4 + C_6h^6 + \cdots, \tag{2.112}$$

provided $u(x)$ is sufficiently smooth. The coefficients $C_2, C_4, \ldots$ depend on high order derivatives of u but are independent of h at each fixed point jh. (This follows from the fact that the local truncation error has an expansion of this form and the fact that the inverse matrix has columns that are an exact discretization of the Green's function, as shown in Section 2.11, but we omit the details of justifying this.)

On the fine grid we therefore have an error of the form

$$\begin{aligned} V_{2j} - u(jh) &= C_2\left(\frac{h}{2}\right)^2 + C_4\left(\frac{h}{2}\right)^4 + C_6\left(\frac{h}{2}\right)^6 + \cdots \\ &= \frac{1}{4}C_2h^2 + \frac{1}{16}C_4h^4 + \frac{1}{64}C_6h^6 + \cdots . \end{aligned} \tag{2.113}$$

The extrapolated value is given by

$$\bar{U}_j = \frac{1}{3}(4V_{2j} - U_j), \tag{2.114}$$

which is chosen so that the h^2 term of the errors cancels out and we obtain

$$\bar{U}_j - u(jh) = \frac{1}{3}\left(\frac{1}{4} - 1\right)C_4h^4 + O(h^6). \tag{2.115}$$

The result has fourth order accuracy as h is reduced and a much smaller error than either U_j or V_{2j} (provided C_4h^2 is not larger than C_2, and usually it is much smaller).

Implementing extrapolation requires solving the problem twice, once on the coarse grid and once on the fine grid, but to obtain similar accuracy with the second order method alone would require a far finer grid than either of these and therefore much more work.

The extrapolation method is more complicated to implement than the fourth order method described in Section 2.20.1, and for this simple one-dimensional boundary value problem it is probably easier to use the fourth order method directly. For more complicated problems, particularly in more than one dimension, developing a higher order method may be more difficult and extrapolation is often a powerful tool.

It is also possible to extrapolate further to obtain higher order accurate approximations. If we also solve the problem on a grid with spacing $h/4$, then this solution can be combined with V to obtain a fourth order accurate approximation on the $(h/2)$-grid. This can be combined with $\bar{U}$ determined above to eliminate the $O(h^4)$ error and obtain a sixth order accurate approximation on the original grid.

2.20.3 Deferred corrections

Another way to combine two different numerical solutions to obtain a higher order accurate approximation, called deferred corrections, has the advantage that it solves both of the problems on the same grid rather than refining the grid as in the extrapolation method. We first solve the system $AU = F$ of Section 2.4 to obtain the second order accurate approximation U. Recall that the global error $E = U - \hat{U}$ satisfies the difference equation (2.15),

$$AE = -\tau, \tag{2.116}$$

where τ is the local truncation error. Suppose we knew the vector τ. Then we could solve the system (2.116) to obtain the global error E and hence obtain the exact solution $\hat{U}$ as $\hat{U} = U - E$. We cannot do this exactly because the local truncation error has the form

$$\tau_j = \frac{1}{12}h^2u''''(x_j) + O(h^4)$$

and depends on the exact solution, which we do not know. However, from the approximate solution U we can estimate τ by approximating the fourth derivative of U.

For the simple problem $u''(x) = f(x)$ that we are now considering we have $u''''(x) = f''(x)$, and so the local truncation error can be estimated directly from the given function $f(x)$. In fact for this simple problem we can avoid solving the problem twice by simply modifying the right-hand side of the original problem $AU = F$ by setting

$$F_j = f(x_j) + \frac{1}{12}h^2 f''(x_j) \tag{2.117}$$

with boundary terms added at $j = 1$ and $j = m$. Solving $AU = F$ then gives a fourth order accurate solution directly. An analogue of this for the two-dimensional Poisson problem is discussed in Section 3.5.

For other problems, we would typically have to use the computed solution U to estimate τ_j and then solve a second problem to estimate E. This general approach is called the method of deferred corrections. In summary, the procedure is to use the approximate solution to estimate the local truncation error and then solve an auxiliary problem of the form (2.116) to estimate the global error. The global error estimate can then be used to improve the approximate solution. For more details see, e.g., [54], [4].

2.21 Spectral methods

The term *spectral method* generally refers to a numerical method that is capable (under suitable smoothness conditions) of converging at a rate that is faster than polynomial in the mesh width h. Originally the term was more precisely defined. In the classical spectral method the solution to the differential equation is approximated by a function $U(x)$ that is a linear combination of a finite set of orthogonal basis functions, say,

$$U(x) = \sum_{j=1}^{N} c_j \phi_j(x), \tag{2.118}$$

and the coefficients chosen to minimize an appropriate norm of the residual function ($= U''(x) - f(x)$ for the simple BVP (2.4)). This is sometimes called a *Galerkin approach*. The method we discuss in this section takes a different approach and can be viewed as expressing $U(x)$ as in (2.118) but then requiring $U''(x_i) = f(x_i)$ at $N - 2$ grid points, along with the two boundary conditions. The differential equation will be exactly satisfied at the grid points by the function $U(x)$, although in between the grid points the ODE generally will not be satisfied. This is called *collocation* and the method presented below is sometimes called a *spectral collocation* or *pseudospectral* method.

In Section 2.20.1 we observed that the second order accurate method could be extended to obtain fourth order accuracy by using more points in the stencil at every grid point, yielding better approximations to the second derivative. We can increase the order further by using even wider stencils.

Suppose we take this idea to its logical conclusion and use the data at *all* the grid points in the domain in order to approximate the derivative at each point. This is easy to try in MATLAB using a simple extension of the approach discussed in Example 2.3. For

the test problem considered with a Neumann boundary condition at the left boundary and a Dirichlet condition at the right, the code from Example 2.3 can be rewritten to use all the grid values in every stencil as

```
A = zeros(m+2);   % A is dense
% first row for Neumann BC, approximates u'(x(1))
A(1,:) = fdcoeffF(1, x(1), x);
% interior rows approximate u''(x(i))
for i=2:m+1
   A(i,:) = fdcoeffF(2, x(i), x);
   end
% last row for Dirichlet BC, approximates u(x(m+2))
A(m+2,:) = fdcoeffF(0,x(m+2),x);
```

We have also switched from using `fdcoeffV` to the more stable `fdcoeffF`, as discussed in Section 1.5.

Note that the matrix will now be dense, since each finite difference stencil involves all the grid points. Recall that `x` is a vector of length $m + 2$ containing all the grid points, so each vector returned by a call to `fdcoeffF` is a full row of the matrix A.

If we apply the resulting A to a vector $U = [U_1\ U_2\ \cdots U_{m+2}]^T$, the values in $W = AU$ will simply be

$$\begin{aligned} W_1 &= p'(x_1), \\ W_i &= p''(x_i) \quad \text{for } i = 2,\ \ldots,\ m+1, \\ W_{m+2} &= p(x_{m+2}), \end{aligned} \tag{2.119}$$

where we're now using the MATLAB indexing convention as in the code and $p(x)$ is the unique polynomial of degree $m + 1$ that interpolates the $m + 2$ data points in U. The same high degree polynomial is used to approximate the derivatives at every grid point.

What sort of accuracy might we hope for? Interpolation through n points generally gives $O(h^{(n-2)})$ accuracy for the second derivative, or one higher order if the stencil is symmetric. We are now interpolating through $m + 2$ points, where $m = O(1/h)$, as we refine the grid, so we might hope that the approximation is $O(h^{1/h})$ accurate. Note that $h^{1/h}$ approaches zero faster than any fixed power of h as $h \to 0$. So we might expect very rapid convergence and small errors.

However, it is not at all clear that we will really achieve the accuracy suggested by the argument above, since increasing the number of interpolation points spread over a fixed interval as $h \to 0$ is qualitatively different than interpolating at a fixed number of points that are all approaching a single point as $h \to 0$. In particular, if we take the points x_i to be equally spaced, then we generally expect to obtain disastrous results. High order polynomial interpolation at equally spaced points on a fixed interval typically leads to a highly oscillatory polynomial that does not approximate the underlying smooth function well at all away from the interpolation points (the Runge phenomenon), and it becomes exponentially *worse* as the grid is refined and the degree increases. Approximating second derivatives by twice differentiating such a function would not be wise and would lead to an unstable method.

This idea can be saved, however, by choosing the grid points to be clustered near the ends of the interval in a particular manner. A very popular choice, which can be shown to be optimal in a certain sense, is to use the extreme points of the Chebyshev polynomial of degree $m + 1$, shifted to the interval $[a, b]$. The expression (B.25) in Appendix B gives the extreme points of $T_m(x)$ on the interval $[-1, 1]$. Shifting to the desired interval, changing m to $m + 1$, and reordering them properly gives the *Chebyshev grid points*

$$x_i = a + \frac{1}{2}(b - a)(1 + \cos(\pi(1 - z_i))) \quad \text{for } i = 0,\ 1, \ldots,\ m + 1, \tag{2.120}$$

where the z_i are again $m + 2$ equally spaced points in the unit interval, $z_i = i/(m + 1)$ for $i = 0,\ 1,\ \ldots,\ m + 1$.

The resulting method is called a *Chebyshev spectral method* (or pseudospectral/ spectral collocation method). For many problems these methods give remarkably good accuracy with relatively few grid points. This is certainly true for the simple boundary value problem $u''(x) = f(x)$, as the following example illustrates.

Example 2.4. Figure 2.9 shows the error as a function of h for three methods we have discussed on the simplest BVP of the form

$$\begin{aligned} u''(x) &= e^x \quad \text{for } 0 \le x \le 3, \\ u(0) &= -5, \quad u(3) = 3. \end{aligned} \tag{2.121}$$

The error behaves in a textbook fashion: the errors for the second order method of Section 2.4 lie on a line with slope 2 (in this log-log plot), and those obtained with the fourth order method of Section 2.20.1 lie on a line with slope 4. The Chebyshev pseudospectral method behaves extremely well for this problem; an error less than 10^{-6} is already

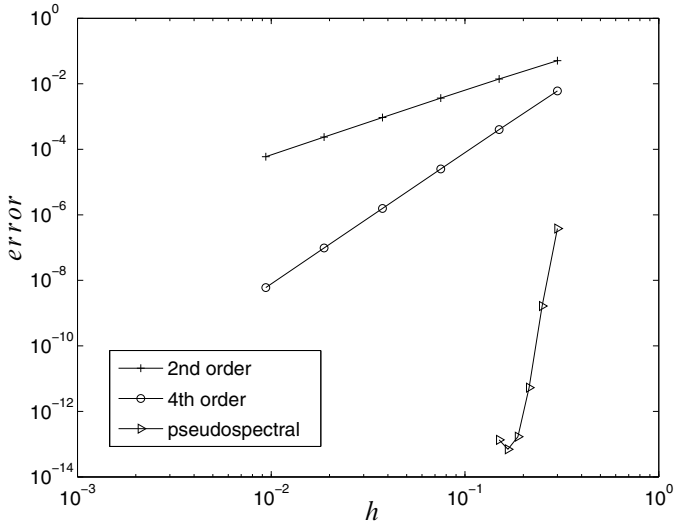


Figure 2.9. *Error as a function of h for two finite difference methods and the Chebyshev pseudospectral method on* (2.121).

observed on the coarsest grid (with $m = 10$) and rounding errors become a problem by $m = 20$. The finest grids used for the finite difference methods in this figure had $m = 320$ points.

For many problems, spectral or pseudospectral methods are well suited and should be seriously considered, although they can have difficulties of their own on realistic nonlinear problems in complicated geometries, or for problems where the solution is not sufficiently smooth. In fact the solution is required to be analytic in some region of the complex plane surrounding the interval over which the solution is being computed in order to get full "spectral accuracy."

Note that the polynomial $p(x)$ in (2.119) is exactly the function $U(x)$ from (2.118), although in the way we have presented the method we do not explicitly compute the coefficients of this polynomial in terms of polynomial basis functions. One could compute this interpolating polynomial if desired once the grid values U_j are known. This may be useful if one needs to approximate the solution $u(x)$ at many more points in the interval than were used in solving the BVP.

For some problems it is natural to use Fourier series representations for the function $U(x)$ in (2.118) rather than polynomials, in particular for problems with periodic boundary conditions. In this case the dense matrix systems that arise can generally be solved using fast Fourier transform (FFT) algorithms. The FFT also can be used in solving problems with Chebyshev polynomials because of the close relation between these polynomials and trigonometric functions, as discussed briefly in Section B.3.2. In many applications, however, a spectral method uses sufficiently few grid points that using direct solvers (Gaussian elimination) is a reasonable approach.

The analysis and proper application of pseudospectral methods goes well beyond the scope of this book. See, for example, [10], [14], [29], [38], or [90] for more thorough introductions to spectral methods.

Note also that spectral approximation of derivatives often is applied only in spatial dimensions. For time-dependent problems, a time-stepping procedure is often based on finite difference methods, of the sort developed in Part II of this book. For PDEs this time stepping may be coupled with a spectral approximation of the spatial derivatives, a topic we briefly touch on in Sections 9.9 and 10.13. One time-stepping procedure that has the flavor of a spectral procedure in time is the recent spectral deferred correction method presented in [28].

Chapter 3
Elliptic Equations

In more than one space dimension, the steady-state equations discussed in Chapter 2 generalize naturally to *elliptic* partial differential equations, as discussed in Section E.1.2. In two space dimensions a constant-coefficient elliptic equation has the form

$$a_1 u_{xx} + a_2 u_{xy} + a_3 u_{yy} + a_4 u_x + a_5 u_y + a_6 u = f, \tag{3.1}$$

where the coefficients a_1, a_2, a_3 satisfy

$$a_2^2 - 4a_1 a_3 < 0. \tag{3.2}$$

This equation must be satisfied for all (x, y) in some region of the plane Ω, together with some boundary conditions on $\partial\Omega$, the boundary of Ω. For example, we may have Dirichlet boundary conditions in which case $u(x, y)$ is given at all points $(x, y) \in \partial\Omega$. If the ellipticity condition (3.2) is satisfied, then this gives a well-posed problem. If the coefficients vary with x and y, then the ellipticity condition must be satisfied at each point in Ω.

3.1 Steady-state heat conduction

Equations of elliptic character often arise as steady-state equations in some region of space, associated with some time-dependent physical problem. For example, the diffusion or heat conduction equation in two space dimensions takes the form

$$u_t = (\kappa u_x)_x + (\kappa u_y)_y + \psi, \tag{3.3}$$

where $\kappa(x, y) > 0$ is a diffusion or heat conduction coefficient that may vary with x and y, and $\psi(x, y, t)$ is a source term. The solution $u(x, y, t)$ generally will vary with time as well as space. We also need initial conditions $u(x, y, 0)$ in Ω and boundary conditions at each point in time at every point on the boundary of Ω. If the boundary conditions and source terms are independent of time, then we expect a steady state to exist, which we can find by solving the elliptic equation

$$(\kappa u_x)_x + (\kappa u_y)_y = f, \tag{3.4}$$

where again we set $f(x, y) = -\psi(x, y)$, together with the boundary conditions. Note that (3.2) is satisfied at each point, provided $\kappa > 0$ everywhere.

We first consider the simplest case where $\kappa \equiv 1$. We then have the *Poisson problem*

$$u_{xx} + u_{yy} = f. \tag{3.5}$$

In the special case $f \equiv 0$, this reduces to *Laplace's equation*,

$$u_{xx} + u_{yy} = 0. \tag{3.6}$$

We also need to specify boundary conditions all around the boundary of the region Ω. These could be Dirichlet conditions, where the temperature $u(x, y)$ is specified at each point on the boundary, or Neumann conditions, where the normal derivative (the heat flux) is specified. We may have Dirichlet conditions specified at some points on the boundary and Neumann conditions at other points.

In one space dimension the corresponding Laplace's equation $u''(x) = 0$ is trivial: the solution is a linear function connecting the two boundary values. In two dimensions even this simple equation in nontrivial to solve, since boundary values can now be specified at every point along the curve defining the boundary. Solutions to Laplace's equation are called *harmonic functions*. You may recall from complex analysis that if $g(z)$ is any complex analytic function of $z = x + iy$, then the real and imaginary parts of this function are harmonic. For example, $g(z) = z^2 = (x^2 - y^2) + 2ixy$ is analytic and the functions $x^2 - y^2$ and $2xy$ are both harmonic.

The operator ∇^2 defined by

$$\nabla^2 u = u_{xx} + u_{yy}$$

is called the *Laplacian*. The notation ∇^2 comes from the fact that, more generally,

$$(\kappa u_x)_x + (\kappa u_y)_y = \nabla \cdot (\kappa \nabla u),$$

where ∇u is the gradient of u,

$$\nabla u = \begin{bmatrix} u_x \\ u_y \end{bmatrix}, \tag{3.7}$$

and $\nabla\cdot$ is the divergence operator,

$$\nabla \cdot \begin{bmatrix} u \\ v \end{bmatrix} = u_x + v_y. \tag{3.8}$$

The symbol Δ is also often used for the Laplacian but would lead to confusion in numerical work where Δx and Δy are often used for grid spacing.

3.2 The 5-point stencil for the Laplacian

To discuss discretizations, first consider the Poisson problem (3.5) on the unit square $0 \le x \le 1$, $0 \le y \le 1$ and suppose we have Dirichlet boundary conditions. We will use a uniform Cartesian grid consisting of grid points (x_i, y_j), where $x_i = i\Delta x$ and $y_j = j\Delta y$. A section of such a grid is shown in Figure 3.1.

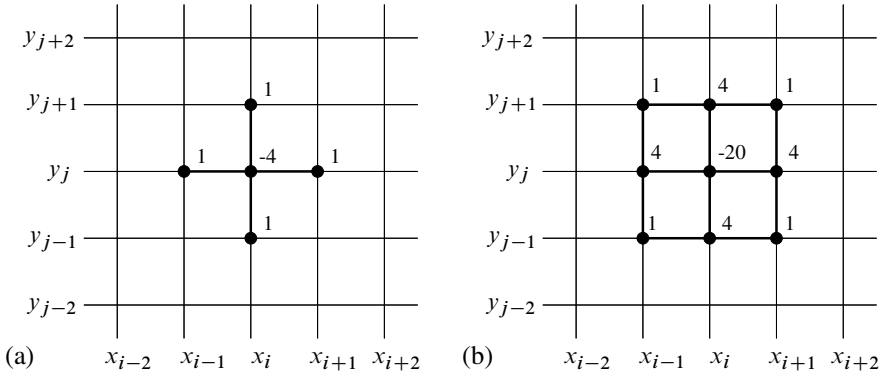


Figure 3.1. *Portion of the computational grid for a two-dimensional elliptic equation. (a) The 5-point stencil for the Laplacian about the point* (i, j) *is also indicated. (b) The 9-point stencil is indicated, which is discussed in Section 3.5.*

Let u_{ij} represent an approximation to $u(x_i, y_j)$. To discretize (3.5) we replace the x- and y-derivatives with centered finite differences, which gives

$$\frac{1}{(\Delta x)^2}(u_{i-1,j} - 2u_{ij} + u_{i+1,j}) + \frac{1}{(\Delta y)^2}(u_{i,j-1} - 2u_{ij} + u_{i,j+1}) = f_{ij}. \tag{3.9}$$

For simplicity of notation we will consider the special case where $\Delta x = \Delta y \equiv h$, although it is easy to handle the general case. We can then rewrite (3.9) as

$$\frac{1}{h^2}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}) = f_{ij}. \tag{3.10}$$

This finite difference scheme can be represented by the *5-point stencil* shown in Figure 3.1. We have both an unknown u_{ij} and an equation of the form (3.10) at each of m^2 grid points for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, m$, where $h = 1/(m + 1)$ as in one dimension. We thus have a linear system of m^2 unknowns. The difference equations at points near the boundary will of course involve the known boundary values, just as in the one-dimensional case, which can be moved to the right-hand side.

3.3 Ordering the unknowns and equations

If we collect all these equations together into a matrix equation, we will have an $m^2 \times m^2$ matrix that is very *sparse*, i.e., most of the elements are zero. Since each equation involves at most five unknowns (fewer near the boundary), each row of the matrix has at most five nonzeros and at least $m^2 - 5$ elements that are zero. This is analogous to the tridiagonal matrix (2.9) seen in the one-dimensional case, in which each row has at most three nonzeros.

Recall from Section 2.14 that the structure of the matrix depends on the order we choose to enumerate the unknowns. Unfortunately, in two space dimensions the structure of the matrix is not as compact as in one dimension, no matter how we order the

unknowns, and the nonzeros cannot be as nicely clustered near the main diagonal. One obvious choice is the *natural rowwise ordering*, where we take the unknowns along the bottom row, $u_{11}, u_{21}, u_{31}, \ldots, u_{m1}$, followed by the unknowns in the second row, $u_{12}, u_{22}, \ldots, u_{m2}$, and so on, as illustrated in Figure 3.2(a). The vector of unknowns is partitioned as

$$u = \begin{bmatrix} u^{[1]} \\ u^{[2]} \\ \vdots \\ u^{[m]} \end{bmatrix}, \quad \text{where } u^{[j]} = \begin{bmatrix} u_{1j} \\ u_{2j} \\ \vdots \\ u_{mj} \end{bmatrix}. \tag{3.11}$$

This gives a matrix equation where A has the form

$$A = \frac{1}{h^2} \begin{bmatrix} T & I & & & & \\ I & T & I & & & \\ & I & T & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & I & T \end{bmatrix}, \tag{3.12}$$

which is an $m \times m$ *block tridiagonal matrix* in which each block T or I is itself an $m \times m$ matrix,

$$T = \begin{bmatrix} -4 & 1 & & & & \\ 1 & -4 & 1 & & & \\ & 1 & -4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -4 \end{bmatrix},$$

and I is the $m \times m$ identity matrix. While this has a nice structure, the 1 values in the I matrices are separated from the diagonal by $m-1$ zeros, since these coefficients correspond to grid points lying above or below the central point in the stencil and hence are in the next or previous row of unknowns.

Another possibility, which has some advantages in the context of certain iterative methods, is to use the *red-black ordering* (or checkerboard ordering) shown in Figure 3.2. This is the two-dimensional analogue of the odd-even ordering that leads to the matrix (2.63) in one dimension. This ordering is significant because all four neighbors of a red grid point are black points, and vice versa, and it leads to a matrix equation with the structure

$$\begin{bmatrix} D & H \\ H^T & D \end{bmatrix} \begin{bmatrix} u_{\text{red}} \\ u_{\text{black}} \end{bmatrix} = \begin{bmatrix} f_{\text{red}} \\ -f_{\text{black}} \end{bmatrix}, \tag{3.13}$$

where $D = -\frac{4}{h^2} I$ is a diagonal matrix of dimension $m^2/2$ and H is a banded matrix of the same dimension with four nonzero diagonals.

When direct methods such as Gaussian elimination are used to solve the system, one typically wants to order the equations and unknowns so as to reduce the amount of fill-in during the elimination procedure as much as possible. This is done automatically if the backslash operator in MATLAB is used to solve the system, provided it is set up using sparse storage; see Section 3.7.1.

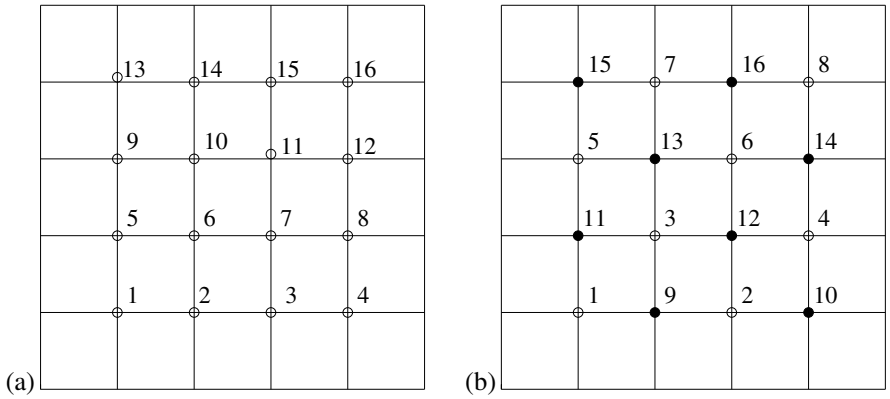


Figure 3.2. (a) *The natural rowwise order of unknowns and equations on a* 4×4 *grid.* (b) *The red-black ordering.*

3.4 Accuracy and stability

The discretization of the two-dimensional Poisson problem can be analyzed using exactly the same approach as we used for the one-dimensional boundary value problem. The local truncation error τ_{ij} at the (i, j) grid point is defined in the obvious way,

$$\tau_{ij} = \frac{1}{h^2}(u(x_{i-1}, y_j)+u(x_{i+1}, y_j)+u(x_i, y_{j-1})+u(x_i, y_{j+1})-4u(x_i, y_j))-f(x_i, y_j),$$

and by splitting this into the second order difference in the x- and y-directions it is clear from previous results that

$$\tau_{ij} = \frac{1}{12}h^2(u_{xxxx} + u_{yyyy}) + O(h^4).$$

For this linear system of equations the global error $E_{ij} = u_{ij} - u(x_i, y_j)$ then solves the linear system

$$A^h E^h = -\tau^h$$

just as in one dimension, where A^h is now the discretization matrix with mesh spacing h, e.g., the matrix (3.12) if the rowwise ordering is used. The method will be globally second order accurate in some norm provided that it is stable, i.e., that $\|(A^h)^{-1}\|$ is uniformly bounded as $h \to 0$.

In the 2-norm this is again easy to check for this simple problem, since we can explicitly compute the spectral radius of the matrix, as we did in one dimension in Section 2.10. The eigenvalues and eigenvectors of A can now be indexed by two parameters p and k corresponding to wave numbers in the x- and y-directions for $p,\ k = 1,\ 2,\ \ldots,\ m$. The (p, q) eigenvector $u^{p,q}$ has the m^2 elements

$$u_{ij}^{p,q} = \sin(p\pi ih)\sin(q\pi jh). \tag{3.14}$$

The corresponding eigenvalue is

$$\lambda_{p,q} = \frac{2}{h^2}\left((\cos(p\pi h) - 1) + (\cos(q\pi h) - 1)\right). \tag{3.15}$$

The eigenvalues are strictly negative (A is negative definite) and the one closest to the origin is

$$\lambda_{1,1} = -2\pi^2 + O(h^2).$$

The spectral radius of $(A^h)^{-1}$, which is also the 2-norm, is thus

$$\rho((A^h)^{-1}) = 1/\lambda_{1,1} \approx -1/2\pi^2.$$

Hence the method is stable in the 2-norm.

While we're at it, let's also compute the condition number of the matrix A^h, since it turns out that this is a critical quantity in determining how rapidly certain iterative methods converge. Recall that the 2-norm condition number is defined by

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2.$$

We've just seen that $\|(A^h)^{-1}\|_2 \approx -1/2\pi^2$ for small h, and the norm of A is given by its spectral radius. The largest eigenvalue of A (in magnitude) is

$$\lambda_{m,m} \approx -\frac{8}{h^2}$$

and so

$$\kappa_2(A) \approx \frac{4}{\pi^2 h^2} = O\left(\frac{1}{h^2}\right) \quad \text{as } h \to 0. \tag{3.16}$$

The fact that the matrix becomes very ill-conditioned as we refine the grid is responsible for the slow-down of iterative methods, as discussed in Chapter 4.

3.5 The 9-point Laplacian

Above we used the 5-point Laplacian, which we will denote by $\nabla_5^2 u_{ij}$, where this denotes the left-hand side of equation (3.10). Another possible approximation is the 9-point Laplacian

$$\begin{aligned}\nabla_9^2 u_{ij} = \frac{1}{6h^2}[&4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} \\ &+ u_{i-1,j-1} + u_{i-1,j+1} + u_{i+1,j-1} + u_{i+1,j+1} - 20u_{ij}]\end{aligned} \tag{3.17}$$

as indicated in Figure 3.1. If we apply this to the true solution and expand in Taylor series, we find that

$$\nabla_9^2 u(x_i, y_j) = \nabla^2 u + \frac{1}{12}h^2(u_{xxxx} + 2u_{xxyy} + u_{yyyy}) + O(h^4).$$

At first glance this discretization looks no better than the 5-point discretization since the error is still $O(h^2)$. However, the additional terms lead to a very nice form for the dominant error term, since

$$u_{xxxx} + 2u_{xxyy} + u_{yyyy} = \nabla^2(\nabla^2 u) \equiv \nabla^4 u.$$

This is the Laplacian of the Laplacian of u and ∇^4 is called the *biharmonic operator*. If we are solving $\nabla^2 u = f$, then we have

$$u_{xxxx} + 2u_{xxyy} + u_{yyyy} = \nabla^2 f.$$

Hence we can compute the dominant term in the truncation error easily from the known function f without knowing the true solution u to the problem.

In particular, if we are solving Laplace's equation, where $f = 0$, or more generally if f is a harmonic function, then this term in the local truncation error vanishes and the 9-point Laplacian would give a fourth order accurate discretization of the differential equation.

More generally, we can obtain a fourth order accurate method of the form

$$\nabla_9^2 u_{ij} = f_{ij} \tag{3.18}$$

for arbitrary smooth functions $f(x, y)$ by defining

$$f_{ij} = f(x_i, y_j) + \frac{h^2}{12}\nabla^2 f(x_i, y_j). \tag{3.19}$$

We can view this as deliberately introducing an $O(h^2)$ error into the right-hand side of the equation that is chosen to cancel the $O(h^2)$ part of the local truncation error. Taylor series expansion easily shows that the local truncation error of the method (3.18) is now $O(h^4)$. This is the two-dimensional analogue of the modification (2.117) that gives fourth order accuracy for the boundary value problem $u''(x) = f(x)$.

If we have only data $f(x_i, y_j)$ at the grid points (but we know that the underlying function is sufficiently smooth), then we can still achieve fourth order accuracy by using

$$f_{ij} = f(x_i, y_j) + \frac{h^2}{12}\nabla_5^2 f(x_i, y_j)$$

instead of (3.19).

This is a trick that often can be used in developing numerical methods—introducing an "error" into the equations that is carefully chosen to cancel some other error.

Note that the same trick wouldn't work with the 5-point Laplacian, or at least not as directly. The form of the truncation error in this method depends on $u_{xxxx} + u_{yyyy}$. There is no way to compute this directly from the original equation without knowing u. The extra points in the 9-point stencil convert this into the Laplacian of f, which can be computed if f is sufficiently smooth.

On the other hand, a two-pass approach could be used with the 5-point stencil, in which we first estimate u by solving with the standard 5-point scheme to get a second order accurate estimate of u. We then use this estimate of u to approximate $u_{xxxx} + u_{yyyy}$ and then solve a second time with a right-hand side that is modified to eliminate the dominant term of the local truncation error. This would be more complicated for this particular problem, but this idea can be used much more generally than the above trick, which depends on the special form of the Laplacian. This is the method of *deferred corrections*, already discussed for one dimension in Section 2.20.3.

3.6 Other elliptic equations

In Chapter 2 we started with the simplest boundary value problem for the constant coefficient problem $u''(x) = f(x)$ but then introduced various, more interesting problems, such as variable coefficients, nonlinear problems, singular perturbation problems, and boundary or interior layers.

In the multidimensional case we have discussed only the simplest Poisson problem, which in one dimension reduces to $u''(x) = f(x)$. All the further complications seen in one dimension can also arise in multidimensional problems. For example, heat conduction in a heterogeneous two-dimensional domain gives rise to the equation

$$(\kappa(x, y)u_x(x, y))_x + (\kappa(x, y)u_y(x, y))_y = f(x, y), \tag{3.20}$$

where $\kappa(x, y)$ is the varying heat conduction coefficient. In any number of space dimensions this equation can be written as

$$\nabla \cdot (\kappa \nabla u) = f. \tag{3.21}$$

These problems can be solved by generalizations of the one-dimensional methods. The terms $(\kappa(x, y)u_x(x, y))_x$ and $(\kappa(x, y)u_y(x, y))_y$ can each be discretized as in the one-dimensional case, again resulting in a 5-point stencil in two dimensions.

Nonlinear elliptic equations also arise in multidimensions, in which case a system of nonlinear algebraic equations will result from the discretization. A Newton method can be used as in one dimension, but now in each Newton iteration a large sparse linear system will have to be solved. Typically the Jacobian matrix has a sparsity pattern similar to those seen above for linear elliptic equations. See Section 4.5 for a brief discussion of Newton–Krylov iterative methods for such problems.

In multidimensional problems there is an additional potential complication that is not seen in one dimension: the domain Ω where the boundary value problem is posed may not be a simple rectangle as we have supposed in our discussion so far. When the solution exhibits boundary or interior layers, then we would also like to cluster grid points or adaptively refine the grid in these regions. This often presents a significant challenge that we will not tackle in this book.

3.7 Solving the linear system

Two fundamentally different approaches could be used for solving the large linear systems that arise from discretizing elliptic equations. A *direct method* such as Gaussian elimination produces an exact solution (or at least would in exact arithmetic) in a finite number of operations. An *iterative method* starts with an initial guess for the solution and attempts to improve it through some iterative procedure, halting after a sufficiently good approximation has been obtained.

For problems with large sparse matrices, iterative methods are often the method of choice, and Chapter 4 is devoted to a study of several iterative methods. Here we briefly consider the operation counts for Gaussian elimination to see the potential pitfalls of this approach.

It should be noted, however, that on current computers direct methods can be successfully used for quite large problems, provided appropriate sparse storage and efficient

elimination procedures are used. See Section 3.7.1 for some comments on setting up sparse matrices such as (3.12) in MATLAB.

It is well known (see, e.g., [35], [82], [91]) that for a general $N \times N$ *dense* matrix (one with few elements equal to zero), performing Gaussian elimination requires $O(N^3)$ operations. (There are $N(N-1)/2 = O(N^2)$ elements below the diagonal to eliminate, and eliminating each one requires $O(N)$ operations to take a linear combination of the rows.)

Applying a general Gaussian elimination program blindly to the matrices we are now dealing with would be disastrous, or at best extremely wasteful of computer resources. Suppose we are solving the three-dimensional Poisson problem on a $100 \times 100 \times 100$ grid—a modest problem these days. Then $N = m^3 = 10^6$ and $N^3 = 10^{18}$. On a reasonably fast desktop that can do on the order of 10^{10} floating point operations per second (10 gigaflops), this would take on the order of 10^8 seconds, which is more than 3 years. More sophisticated methods can solve this problem in seconds.

Moreover, even if speed were not an issue, memory would be. Storing the full matrix A in order to modify the elements and produce L and U would require N^2 memory locations. In 8-byte arithmetic this requires $8\,N^2$ bytes. For the problem mentioned above, this would be 8×10^{12} bytes, or eight terabytes. One advantage of iterative methods is that they do not store the matrix at all and at most need to store the nonzero elements.

Of course with Gaussian elimination it would be foolish to store all the elements of a sparse matrix, since the vast majority are zero, or to apply the procedure blindly without taking advantage of the fact that so many elements are already zero and hence do not need to be eliminated.

As an extreme example, consider the one-dimensional case where we have a tridiagonal matrix as in (2.9). Applying Gaussian elimination requires eliminating only the nonzeros along the subdiagonal, only $N-1$ values instead of $N(N-1)/2$. Moreover, when we take linear combinations of rows in the course of eliminating these values, in most columns we will be taking linear combinations of zeros, producing zero again. If we do not do pivoting, then only the diagonal elements are modified. Even with partial pivoting, at most we will introduce one extra superdiagonal of nonzeros in the upper triangular U that were not present in A. As a result, it is easy to see that applying Gaussian elimination to an $m \times m$ tridiagonal system requires only $O(m)$ operations, not $O(m^3)$, and that the storage required is $O(m)$ rather than $O(m^2)$.

Note that this is the best we could hope for in one dimension, at least in terms of the order of magnitude. There are m unknowns and even if we had exact formulas for these values, it would require $O(m)$ work to evaluate them and $O(m)$ storage to save them.

In two space dimensions we can also take advantage of the sparsity and structure of the matrix to greatly reduce the storage and work required with Gaussian elimination, although not to the minimum that one might hope to attain. On an $m \times m$ grid there are $N = m^2$ unknowns, so the best one could hope for is an algorithm that computes the solution in $O(N) = O(m^2)$ work using $O(m^2)$ storage. Unfortunately, this cannot be achieved with a direct method.

One approach that is better than working with the full matrix is to observe that the A is a banded matrix with bandwidth m both above and below the diagonal. Since a general $N \times N$ banded matrix with a nonzero bands above the diagonal and b below the diagonal

can be factored in $O(Nab)$ operations, this results in an operation count of $O(m^4)$ for the two-dimensional Poisson problem.

A more sophisticated approach that takes more advantage of the special structure (and the fact that there are already many zeros within the bandwidth) is the *nested dissection* algorithm [34]. This algorithm requires $O(m^3)$ operations in two dimensions. It turns out this is the best that can be achieved with a direct method based on Gaussian elimination. George proved (see [34]) that any elimination method for solving this problem requires at least $O(m^3)$ operations.

For certain special problems, very fast direct methods can be used, which are much better than standard Gaussian elimination. In particular, for the Poisson problem on a rectangular domain there are *fast Poisson solvers* based on the fast Fourier transform that can solve on an $m \times m$ grid in two dimensions in $O(m^2 \log m)$ operations, which is nearly optimal. See [87] for a review of this approach.

3.7.1 Sparse storage in MATLAB

If you are going to work in MATLAB with sparse matrices arising from finite difference methods, it is important to understand and use the sparse matrix commands that set up matrices using sparse storage, so that only the nonzeros are stored. Type `help sparse` to get started.

As one example, the matrix of (3.12) can be formed in MATLAB by the commands

```
I = eye(m);
e = ones(m,1);
T = spdiags([e -4*e e],[-1 0 1],m,m);
S = spdiags([e e],[-1 1],m,m);
A = (kron(I,T) + kron(S,I))/h^2;
```

The `spy(A)` command is also useful for looking at the nonzero structure of a matrix.

The backslash command in MATLAB can be used to solve systems using sparse storage, and it implements highly efficient direct methods using sophisticated algorithms for dynamically ordering the equations to minimize fill-in, as described by Davis [24].

Chapter 4
Iterative Methods for Sparse Linear Systems

This chapter contains an overview of several iterative methods for solving the large sparse linear systems that arise from discretizing elliptic equations. Large sparse linear systems arise from many other practical problems, too, of course, and the methods discussed here are useful in other contexts as well. Except when the matrix has very special structure and fast direct methods of the type discussed in Section 3.7 apply, iterative methods are usually the method of choice for large sparse linear systems.

The classical Jacobi, Gauss–Seidel, and successive overrelaxation (SOR) methods are introduced and briefly discussed. The bulk of the chapter, however, concerns more modern methods for solving linear systems that are typically much more effective for large-scale problems: preconditioned conjugate-gradient (CG) methods, Krylov space methods such as generalized minimum residual (GMRES), and multigrid methods.

4.1 Jacobi and Gauss–Seidel

In this section two classical iterative methods, Jacobi and Gauss–Seidel, are introduced to illustrate the main issues. It should be stressed at the beginning that these are poor methods in general which converge very slowly when used as standalone methods, but they have the virtue of being simple to explain. Moreover, these methods are sometimes used as building blocks in more sophisticated methods, e.g., Jacobi may be used as a smoother for the multigrid method, as discussed in Section 4.6.

We again consider the Poisson problem where we have the system of equations (3.10). We can rewrite this equation as

$$u_{ij} = \frac{1}{4}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) - \frac{h^2}{4} f_{ij}. \tag{4.1}$$

In particular, note that for Laplace's equation (where $f_{ij} \equiv 0$), this simply states that the value of u at each grid point should be the average of its four neighbors. This is the discrete analogue of the well-known fact that a harmonic function has the following property: the value at any point (x, y) is equal to the average value around a closed curve containing the point, in the limit as the curve shrinks to the point. Physically this also makes sense if we

think of the heat equation. Unless the temperature at this point is equal to the average of the temperature at neighboring points, there will be a net flow of heat toward or away from this point.

The equation (4.1) suggests the following iterative method to produce a new estimate $u^{[k+1]}$ from a current guess $u^{[k]}$:

$$u_{ij}^{[k+1]} = \frac{1}{4}\left(u_{i-1,j}^{[k]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k]} + u_{i,j+1}^{[k]}\right) - \frac{h^2}{4} f_{ij}. \tag{4.2}$$

This is the *Jacobi* iteration for the Poisson problem, and it can be shown that for this particular problem it converges from any initial guess $u^{[0]}$ (although very slowly).

Here is a short section of MATLAB code that implements the main part of this iteration:

```
for iter=0:maxiter
   for j=2:(m+1)
      for i=2:(m+1)
         unew(i,j) = 0.25*(u(i-1,j) + u(i+1,j) + ...
                     u(i,j-1) + u(i,j+1) - h^2 * f(i,j));
      end
   end
   u = unew;
   end
```

Here it is assumed that `u` initially contains the guess $u^{[0]}$ and that boundary data are stored in `u(1,:), u(m+2,:), u(:,1), and u(:,m+2)`. The indexing is off by 1 from what might be expected since MATLAB begins arrays with index 1, not 0.

Note that one might be tempted to dispense with the variable `unew` and replace the above code with

```
for iter=0:maxiter
   for j=2:(m+1)
      for i=2:(m+1)
         u(i,j) = 0.25*(u(i-1,j) + u(i+1,j) + ...
                     u(i,j-1) + u(i,j+1) - h^2 * f(i,j));
      end
   end
   end
```

This would not give the same results, however. In the correct code for Jacobi we compute new values of `u` based entirely on old data from the previous iteration, as required from (4.2). In the second code we have already updated `u(i-1,j)` and `u(i,j-1)` before updating `u(i,j)`, and these new values will be used instead of the old ones. The latter code thus corresponds to the method

$$u_{ij}^{[k+1]} = \frac{1}{4}\left(u_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k+1]} + u_{i,j+1}^{[k]}\right) - \frac{h^2}{4} f_{ij}. \tag{4.3}$$

This is what is known as the *Gauss–Seidel* method, and it would be a lucky coding error since this method generally converges about twice as fast as Jacobi does. The Jacobi

method is sometimes called the *method of simultaneous displacements*, while Gauss–Seidel is known as the *method of successive displacements*. Later we'll see that Gauss–Seidel can be improved by using SOR.

Note that if one actually wants to implement Jacobi in MATLAB, looping over i and j is quite slow and it is much better to write the code in vectorized form, e.g.,

```
I = 2:(m+1);
J = 2:(m+1);
for iter=0:maxiter
   u(I,J) = 0.25*(u(I-1,J) + u(I+1,J) + u(I,J-1) ...
                          + u(I,J+1) - h^2 * f(I,J));
   end
```

It is somewhat harder to implement Gauss–Seidel in vectorized form.

Convergence of these methods will be discussed in Section 4.2. First we note some important features of these iterative methods:

- The matrix A is never stored. In fact, for this simple constant coefficient problem, we don't even store all the $5m^2$ nonzeros which all have the value $1/h^2$ or $-4/h^2$. The values 0.25 and h^2 in the code are the only values that are "stored." (For a variable coefficient problem where the coefficients are different at each point, we would in general have to store all the nonzeros.)

- Hence the storage is optimal—essentially only the m^2 solution values are stored in the Gauss–Seidel method. The above code for Jacobi uses $2m^2$ since `unew` is stored as well as `u`, but one could eliminate most of this with more careful coding.

- Each iteration requires $O(m^2)$ work. The total work required will depend on how many iterations are required to reach the desired level of accuracy. We will see that with these particular methods we require $O(m^2 \log m)$ iterations to reach a level of accuracy consistent with the expected global error in the solution (as $h \to 0$ we should require more accuracy in the solution to the linear system). Combining this with the work per iteration gives a total operation count of $O(m^4 \log m)$. This looks worse than Gaussian elimination with a banded solver, although since $\log m$ grows so slowly with m it is not clear which is really more expensive for a realistic-size matrix. (And the iterative method definitely saves on storage.)

Other iterative methods also typically require $O(m^2)$ work per iteration but may converge much faster and hence result in less overall work. The ideal would be to converge in a number of iterations that is independent of h so that the total work is simply $O(m^2)$. Multigrid methods (see Section 4.6) can achieve this, not only for Poisson's problem but also for many other elliptic equations.

4.2 Analysis of matrix splitting methods

In this section we study the convergence of the Jacobi and Gauss–Seidel methods. As a simple example we will consider the one-dimensional analogue of the Poisson problem, $u''(x) = f(x)$ as discussed in Chapter 2. Then we have a tridiagonal system of equations

(2.9) to solve. In practice we would never use an iterative method for this system, since it can be solved directly by Gaussian elimination in $O(m)$ operations, but it is easier to illustrate the iterative methods in the one-dimensional case, and all the analysis done here carries over almost unchanged to the two-dimensional and three-dimensional cases.

The Jacobi and Gauss–Seidel methods for this problem take the form

$$\text{Jacobi} \qquad u_i^{[k+1]} = \frac{1}{2}\left(u_{i-1}^{[k]} + u_{i+1}^{[k]} - h^2 f_i\right), \tag{4.4}$$

$$\text{Gauss–Seidel} \qquad u_i^{[k+1]} = \frac{1}{2}\left(u_{i-1}^{[k+1]} + u_{i+1}^{[k]} - h^2 f_i\right). \tag{4.5}$$

Both methods can be analyzed by viewing them as based on a splitting of the matrix A into

$$A = M - N, \tag{4.6}$$

where M and N are two $m \times m$ matrices. Then the system $Au = f$ can be written as

$$Mu - Nu = f \quad \Longrightarrow \quad Mu = Nu + f,$$

which suggests the iterative method

$$Mu^{[k+1]} = Nu^{[k]} + f. \tag{4.7}$$

In each iteration we assume $u^{[k]}$ is known and we obtain $u^{[k+1]}$ by solving a linear system with the matrix M. The basic idea is to define the splitting so that M contains as much of A as possible (in some sense) while keeping its structure sufficiently simple that the system (4.7) is much easier to solve than the original system with the full A. Since systems involving diagonal, lower, or upper triangular matrices are relatively simple to solve, there are some obvious choices for the matrix M. To discuss these in a unified framework, write

$$A = D - L - U \tag{4.8}$$

in general, where D is the diagonal of A, $-L$ is the strictly lower triangular part, and $-U$ is the strictly upper triangular part. For example, the tridiagonal matrix (2.10) would give

$$D = \frac{1}{h^2}\begin{bmatrix} -2 & 0 & & & & & \\ 0 & -2 & 0 & & & & \\ & 0 & -2 & 0 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 0 & -2 & 0 & \\ & & & & 0 & -2 & \end{bmatrix}, \quad L = -\frac{1}{h^2}\begin{bmatrix} 0 & 0 & & & & & \\ 1 & 0 & 0 & & & & \\ & 1 & 0 & 0 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 0 & 0 & \\ & & & & 1 & 0 & \end{bmatrix}$$

with $-U = -L^T$ being the remainder of A.

In the Jacobi method, we simply take M to be the diagonal part of A, $M = D$, so that

$$M = -\frac{2}{h^2}I, \qquad N = L + U = D - A = -\frac{1}{h^2}\begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix}.$$

The system (4.7) is then diagonal and extremely easy to solve:

$$u^{[k+1]} = \frac{1}{2}\begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix} u^{[k]} - \frac{h^2}{2} f, \tag{4.9}$$

which agrees with (4.4).

In Gauss–Seidel, we take M to be the full lower triangular portion of A, so $M = D - L$ and $N = U$. The system (4.7) is then solved using forward substitution, which results in (4.5).

To analyze these methods, we derive from (4.7) the update formula

$$\begin{aligned} u^{[k+1]} &= M^{-1}Nu^{[k]} + M^{-1}f \\ &\equiv Gu^{[k]} + c, \end{aligned} \tag{4.10}$$

where $G = M^{-1}N$ is the *iteration matrix* and $c = M^{-1}f$.

Let u^* represent the true solution to the system $Au = f$. Then

$$u^* = Gu^* + c. \tag{4.11}$$

This shows that the true solution is a fixed point, or equilibrium, of the iteration (4.10), i.e., if $u^{[k]} = u^*$, then $u^{[k+1]} = u^*$ as well. However, it is not clear that this is a *stable equilibrium*, i.e., that we would converge toward u^* if we start from some incorrect initial guess.

If $e^{[k]} = u^{[k]} - u^*$ represents the error, then subtracting (4.11) from (4.10) gives

$$e^{[k+1]} = Ge^{[k]},$$

and so after k steps we have

$$e^{[k]} = G^k e^{[0]}. \tag{4.12}$$

From this we can see that the method will converge from any initial guess $u^{[0]}$, provided $G^k \to 0$ (an $m \times m$ matrix of zeros) as $k \to \infty$. When is this true?

For simplicity, assume that G is a diagonalizable matrix, so that we can write

$$G = R\Gamma R^{-1},$$

where R is the matrix of right eigenvectors of G and Γ is a diagonal matrix of eigenvalues $\gamma_1, \gamma_2, \ldots, \gamma_m$. Then

$$G^k = R\Gamma^k R^{-1}, \tag{4.13}$$

where

$$\Gamma^k = \begin{bmatrix} \gamma_1^k & & & \\ & \gamma_2^k & & \\ & & \ddots & \\ & & & \gamma_m^k \end{bmatrix}.$$

Clearly the method converges if $|\gamma_p| < 1$ for all $p = 1, 2, \ldots, m$, i.e., if $\rho(G) < 1$, where ρ is the spectral radius. See Appendix D for a more general discussion of the asymptotic properties of matrix powers.

4.2.1 Rate of convergence

From (4.12) we can also determine how rapidly the method can be expected to converge in cases where it is convergent. Using (4.13) in (4.12) and using the 2-norm, we obtain

$$\|e^{[k]}\|_2 \le \|\Gamma^k\|_2 \|R\|_2 \|R^{-1}\|_2 \|e^{[0]}\|_2 = \rho^k \kappa_2(R) \|e^{[0]}\|_2, \tag{4.14}$$

where $\rho \equiv \rho(G)$, and $\kappa_2(R) = \|R\|_2 \|R^{-1}\|_2$ is the condition number of the eigenvector matrix.

If the matrix G is a normal matrix (see Section C.4), then the eigenvectors are orthogonal and $\kappa_2(R) = 1$. In this case we have

$$\|e^{[k]}\|_2 \le \rho^k \|e^{[0]}\|_2. \tag{4.15}$$

If G is nonnormal, then the spectral radius of G gives information about the asymptotic rate of convergence as $k \to \infty$ but may not give a good indication of the behavior of the error for small k. See Section D.4 for more discussion of powers of nonnormal matrices and see Chapters 24–27 of [92] for some discussion of iterative methods on highly nonnormal problems.

Note: These methods are *linearly* convergent, in the sense that $\|e^{[k+1]}\| \le \rho \|e^{[k]}\|$ and it is the first power of $\|e^{[k]}\|$ that appears on the right. Recall that Newton's method is typically quadratically convergent, and it is the square of the previous error that appears on the right-hand side. But Newton's method is for a nonlinear problem and requires solving a linear system in each iteration. Here we are looking at solving such a linear system.

Example 4.1. For the Jacobi method we have

$$G = D^{-1}(D - A) = I - D^{-1}A.$$

If we apply this method to the boundary value problem $u'' = f$, then

$$G = I + \frac{h^2}{2}A.$$

The eigenvectors of this matrix are the same as the eigenvectors of A, and the eigenvalues are hence

$$\gamma_p = 1 + \frac{h^2}{2}\lambda_p,$$

where λ_p is given by (2.23). So

$$\gamma_p = \cos(p\pi h), \quad p = 1, 2, \ldots, m,$$

where $h = 1/(m + 1)$. The spectral radius is

$$\rho(G) = |\gamma_1| = \cos(\pi h) \approx 1 - \frac{1}{2}\pi^2 h^2 + O(h^4). \tag{4.16}$$

The spectral radius is less than 1 for any $h > 0$ and the Jacobi method converges. Moreover, the G matrix for Jacobi is symmetric as seen in (4.9), and so (4.15) holds and the error is monotonically decreasing at a rate given precisely by the spectral radius. Unfortunately, though, for small h this value is very close to 1, resulting in very slow convergence.

How many iterations are required to obtain a good solution? Suppose we want to reduce the error to $\|e^{[k]}\| \approx \epsilon \|e^{[0]}\|$ (where typically $\|e^{[0]}\|$ is on the order of 1).[1] Then we want $\rho^k \approx \epsilon$ and so

$$k \approx \log(\epsilon)/\log(\rho). \tag{4.17}$$

How small should we choose ϵ? To get full machine precision we might choose ϵ to be close to the machine round-off level. However, this typically would be very wasteful. For one thing, we rarely need this many correct digits. More important, however, we should keep in mind that even the *exact* solution u^* of the linear system $Au = f$ is only an *approximate* solution of the differential equation we are actually solving. If we are using a second order accurate method, as in this example, then u_i^* differs from $u(x_i)$ by something on the order of h^2 and so we cannot achieve better accuracy than this no matter how well we solve the linear system. In practice we should thus take ϵ to be something related to the expected global error in the solution, e.g., $\epsilon = Ch^2$ for some fixed C.

To estimate the order of work required asymptotically as $h \to 0$, we see that the above choice gives

$$k = (\log(C) + 2\log(h))/\log(\rho). \tag{4.18}$$

For Jacobi on the boundary value problem we have $\rho \approx 1 - \frac{1}{2}\pi^2 h^2$ and hence $\log(\rho) \approx -\frac{1}{2}\pi^2 h^2$. Since $h = 1/(m+1)$, using this in (4.18) gives

$$k = O(m^2 \log m) \quad \text{as } m \to \infty. \tag{4.19}$$

Since each iteration requires $O(m)$ work in this one-dimensional problem, the total work required to solve the problem is

$$\text{total work} = O(m^3 \log m).$$

Of course this tridiagonal problem can be solved exactly in $O(m)$ work, so we would be foolish to use an iterative method at all here!

For a Poisson problem in two or three dimensions it can be verified that (4.19) still holds, although now the work required per iteration is $O(m^2)$ or $O(m^3)$, respectively, if there are m grid points in each direction. In two dimensions we would thus find that

$$\text{total work} = O(m^4 \log m). \tag{4.20}$$

Recall from Section 3.7 that Gaussian elimination on the banded matrix requires $O(m^4)$ operations, while other direct methods can do much better, so Jacobi is still not competitive. Luckily there are much better iterative methods.

[1] Assuming we are using some grid function norm, as discussed in Appendix A. Note that for the 2-norm in one dimension this requires introducing a factor of $\sqrt{h}$ in the definitions of both $\|e^{[k]}\|$ and $\|e^{[0]}\|$, but these factors cancel out in choosing an appropriate ϵ.

For the Gauss–Seidel method applied to the Poisson problem in any number of space dimensions, it can be shown that

$$\rho(G) = 1 - \pi^2 h^2 + O(h^4) \quad \text{as } h \to 0. \tag{4.21}$$

This still approaches 1 as $h \to 0$, but it is better than (4.16) by a factor of 2, and the number of iterations required to reach a given tolerance typically will be half the number required with Jacobi. The order of magnitude figure (4.20) still holds, however, and this method also is not widely used.

4.2.2 Successive overrelaxation

If we look at how iterates $u^{[k]}$ behave when Gauss–Seidel is applied to a typical problem, we will often see that $u_i^{[k+1]}$ is closer to u_i^* than $u_i^{[k]}$ was, but only by a little bit. The Gauss–Seidel update moves u_i in the right direction but is far too conservative in the amount it allows u_i to move. This suggests that we use the following two-stage update, illustrated again for the problem $u'' = f$:

$$\begin{aligned} u_i^{\text{GS}} &= \frac{1}{2}\left(u_{i-1}^{[k+1]} + u_{i+1}^{[k]} - h^2 f_i\right), \\ u_i^{[k+1]} &= u_i^{[k]} + \omega\left(u_i^{\text{GS}} - u_i^{[k]}\right), \end{aligned} \tag{4.22}$$

where ω is some scalar parameter. If $\omega = 1$, then $u_i^{[k+1]} = u_i^{\text{GS}}$ is the Gauss–Seidel update. If $\omega > 1$, then we move farther than Gauss–Seidel suggests. In this case the method is known as *successive overrelaxation* (SOR).

If $\omega < 1$, then we would be underrelaxing, rather than overrelaxing. This would be even less effective than Gauss–Seidel as a standalone iterative method for most problems, although underrelaxation is sometimes used in connection with multigrid methods (see Section 4.6).

The formulas in (4.22) can be combined to yield

$$u_i^{[k+1]} = \frac{\omega}{2}\left(u_{i-1}^{[k+1]} + u_{i+1}^{[k]} - h^2 f_i\right) + (1-\omega)u_i^{[k]}. \tag{4.23}$$

For a general system $Au = f$ with $A = D - L - U$ it can be shown that SOR with forward sweeps corresponds to a matrix splitting method of the form (4.7) with

$$M = \frac{1}{\omega}(D - \omega L), \qquad N = \frac{1}{\omega}((1-\omega)D + \omega U). \tag{4.24}$$

Analyzing this method is considerably trickier than with the Jacobi or Gauss–Seidel methods because of the form of these matrices. A theorem of Ostrowski states that if A is symmetric positive definite (SPD) and $D - \omega L$ is nonsingular, then the SOR method converges for all $0 < \omega < 2$. Young [105] showed how to find the optimal ω to obtain the most rapid convergence for a wide class of problems (including the Poisson problem). This elegant theory can be found in many introductory texts. (For example, see [37], [42], [96], [106]. See also [67] for a different introductory treatment based on Fourier series

and modified equations in the sense of Section 10.9, and see [3] for applications of this approach to the 9-point Laplacian.)

For the Poisson problem in any number of space dimensions it can be shown that the SOR method converges most rapidly if ω is chosen as

$$\omega_{\text{opt}} = \frac{2}{1 + \sin(\pi h)} \approx 2 - 2\pi h.$$

This is nearly equal to 2 for small h. One might be tempted to simply set $\omega = 2$ in general, but this would be a poor choice since SOR does not then converge! In fact the convergence rate is quite sensitive to the value of ω chosen. With the optimal ω it can be shown that the spectral radius of the corresponding G matrix is

$$\rho_{\text{opt}} = \omega_{\text{opt}} - 1 \approx 1 - 2\pi h,$$

but if ω is changed slightly this can deteriorate substantially.

Even with the optimal ω we see that $\rho_{\text{opt}} \to 1$ as $h \to 0$, but only linearly in h rather than quadratically as with Jacobi or Gauss–Seidel. This makes a substantial difference in practice. The expected number of iterations to converge to the required $O(h^2)$ level, the analogue of (4.19), is now

$$k_{opt} = O(m \log m).$$

Figure 4.1 shows some computational results for the methods described above on the two-point boundary value problem $u'' = f$. The SOR method with optimal ω is

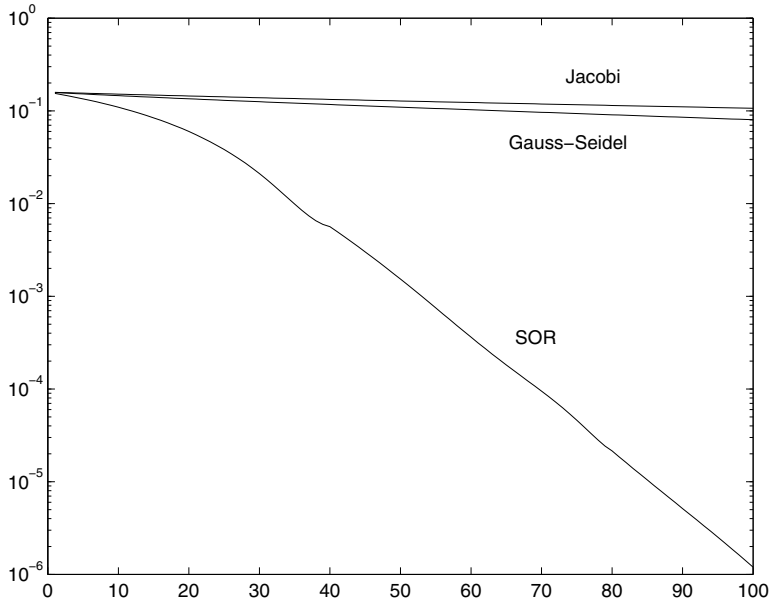


Figure 4.1. *Errors versus k for three methods.*

far superior to Gauss–Seidel or Jacobi, at least for this simple problem with a symmetric coefficient matrix. For more complicated problems it can be difficult to estimate the optimal ω, however, and other approaches are usually preferred.

4.3 Descent methods and conjugate gradients

The CG method is a powerful technique for solving linear systems $Au = f$ when the matrix A is SPD, or negative definite since negating the system then gives an SPD matrix. This may seem like a severe restriction, but SPD methods arise naturally in many applications, such as the discretization of elliptic equations. There are several ways to introduce the CG method and the reader may wish to consult texts such as [39], [79], [91] for other approaches and more analysis. Here the method is first motivated as a *descent method* for solving a *minimization problem*.

Consider the function $\phi : \mathbb{R}^m \to \mathbb{R}$ defined by

$$\phi(u) = \frac{1}{2}u^T Au - u^T f. \tag{4.25}$$

This is a quadratic function of the variables $u_1, \ldots, u_m$. For example, if $m = 2$, then

$$\phi(u) = \phi(u_1, u_2) = \frac{1}{2}(a_{11}u_1^2 + 2a_{12}u_1u_2 + a_{22}u_2^2) - u_1f_1 - u_2f_2.$$

Note that since A is symmetric, $a_{21} = a_{12}$. If A is positive definite, then plotting $\phi(u)$ as a function of u_1 and u_2 gives a parabolic bowl as shown in Figure 4.2(a). There is a unique value u^* that minimizes $\phi(u)$ over all choices of u. At the minimum, the partial derivative of ϕ with respect to each component of u is zero, which gives the equations

$$\begin{aligned} \frac{\partial \phi}{\partial u_1} &= a_{11}u_1 + a_{12}u_2 - f_1 = 0, \\ \frac{\partial \phi}{\partial u_2} &= a_{21}u_1 + a_{22}u_2 - f_2 = 0. \end{aligned} \tag{4.26}$$

This is exactly the linear system $Au = f$ that we wish to solve. So finding u^* that solves this system can equivalently be approached as finding u^* to minimize $\phi(u)$. This is true more generally when $u \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times m}$ is SPD. The function $\phi(u)$ in (4.25) has a unique minimum at the point u^*, where $\nabla\phi(u^*) = 0$, and

$$\nabla\phi(u) = Au - f, \tag{4.27}$$

so the minimizer solves the linear system $Au = f$.

If A is negative definite, then $\phi(u)$ instead has a unique maximum at u^*, which again solves the linear system. If A is *indefinite* (neither positive nor negative definite), i.e., if the eigenvalues of A are not all of the same sign, then the function $\phi(u)$ still has a stationary point with $\nabla\phi(u^*) = 0$ at the solution to $Au = f$, but this is a saddle point rather than a minimum or maximum, as illustrated in Figure 4.2(b). It is much harder to find a saddle point than a minimum. An iterative method can find a minimum by always heading downhill, but if we are looking for a saddle point, it is hard to tell if we need to

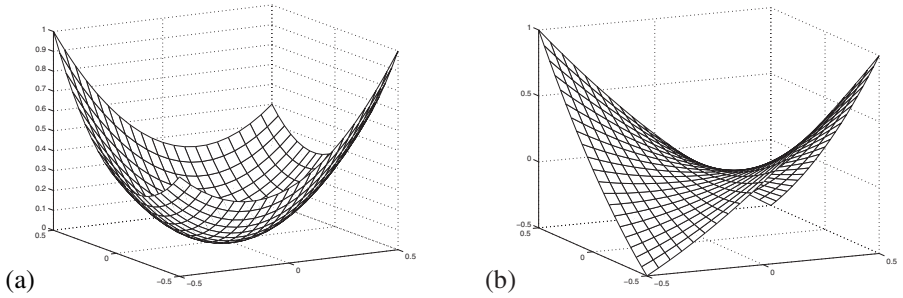


Figure 4.2. (a) *The function $\phi(u)$ for $m = 2$ in a case where A is symmetric and positive definite.* (b) *The function $\phi(u)$ for $m = 2$ in a case where A is symmetric but indefinite.*

head uphill or downhill from the current approximation. Since the CG method is based on minimization, it is necessary for the matrix to be SPD. By viewing CG in a different way it is possible to generalize it and obtain methods that also work on indefinite problems, such as the GMRES algorithm described in Section 4.4.

4.3.1 The method of steepest descent

As a prelude to studying CG, we first review the method of steepest descent for minimizing $\phi(u)$. As in all iterative methods we start with an initial guess u_0 and iterate to obtain $u_1, u_2, \ldots$. For notational convenience we now use subscripts to denote the iteration number: u_k instead of $u^{[k]}$. This is potentially confusing since normally we use subscripts to denote components of the vector, but the formulas below get too messy otherwise and we will not need to refer to the components of the vector in the rest of this chapter.

From one estimate u_{k-1} to u^* we wish to obtain a better estimate u_k by moving downhill, based on values of $\phi(u)$. It seems sensible to move in the direction in which ϕ is decreasing most rapidly, and go in this direction for as far as we can before $\phi(u)$ starts to increase again. This is easy to implement, since the gradient vector $\nabla\phi(u)$ always points in the direction of most rapid increase of ϕ. So we want to set

$$u_k = u_{k-1} - \alpha_{k-1}\nabla\phi(u_{k-1}) \tag{4.28}$$

for some scalar α_{k-1}, chosen to solve the minimization problem

$$\min_{\alpha\in\mathbb{R}} \phi\left(u_{k-1} - \alpha\nabla\phi(u_{k-1})\right). \tag{4.29}$$

We expect $\alpha_{k-1} \geq 0$ and $\alpha_{k-1} = 0$ only if we are already at the minimum of ϕ, i.e., only if $u_{k-1} = u^*$.

For the function $\phi(u)$ in (4.25), the gradient is given by (4.27) and so

$$\nabla\phi(u_{k-1}) = Au_{k-1} - f \equiv -r_{k-1}, \tag{4.30}$$

where $r_{k-1} = f - Au_{k-1}$ is the *residual vector* based on the current approximation u_{k-1}. To solve the minimization problem (4.29), we compute the derivative with respect to α and set this to zero. Note that

$$\phi(u+\alpha r) = \left(\frac{1}{2}u^T Au - u^T f\right) + \alpha(r^T Au - r^T f) + \frac{1}{2}\alpha^2 r^T Ar \tag{4.31}$$

and so

$$\frac{d\phi(u+\alpha r)}{d\alpha} = r^T Au - r^T f + \alpha r^T Ar.$$

Setting this to zero and solving for α gives

$$\alpha = \frac{r^T r}{r^T Ar}. \tag{4.32}$$

The steepest descent algorithm thus takes the form

```
choose a guess u_0
for k = 1, 2, ...
    r_{k-1} = f - A u_{k-1}
    if ||r_{k-1}|| is less than some tolerance then stop
    α_{k-1} = (r_{k-1}^T r_{k-1}) / (r_{k-1}^T A r_{k-1})
    u_k = u_{k-1} + α_{k-1} r_{k-1}
    end
```

Note that implementing this algorithm requires only that we be able to multiply a vector by A, as with the other iterative methods discussed earlier. We do not need to store the matrix A, and if A is very sparse, then this multiplication can be done quickly.

It appears that in each iteration we must do two matrix-vector multiplies, Au_{k-1} to compute r_{k-1} and then Ar_{k-1} to compute α_{k-1}. However, note that

$$\begin{aligned} r_k &= f - Au_k \\ &= f - A(u_{k-1} + \alpha_{k-1} r_{k-1}) \\ &= r_{k-1} - \alpha_{k-1} Ar_{k-1}. \end{aligned} \tag{4.33}$$

So once we have computed Ar_{k-1} as needed for α_{k-1}, we can also use this result to compute r_k. A better way to organize the computation is thus:

```
choose a guess u_0
r_0 = f - A u_0
for k = 1, 2, ...
    w_{k-1} = A r_{k-1}
    α_{k-1} = (r_{k-1}^T r_{k-1}) / (r_{k-1}^T w_{k-1})
    u_k = u_{k-1} + α_{k-1} r_{k-1}
    r_k = r_{k-1} - α_{k-1} w_{k-1}
    if ||r_k|| is less than some tolerance then stop
    end
```

Figure 4.3 shows how this iteration proceeds for a typical case with $m = 2$. This figure shows a contour plot of the function $\phi(u)$ in the u_1-u_2 plane (where u_1 and u_2 mean the components of u here), along with several iterates u_n of the steepest descent algorithm. Note that the gradient vector is always orthogonal to the contour lines. We move along the direction of the gradient (the "search direction" for this algorithm) to the

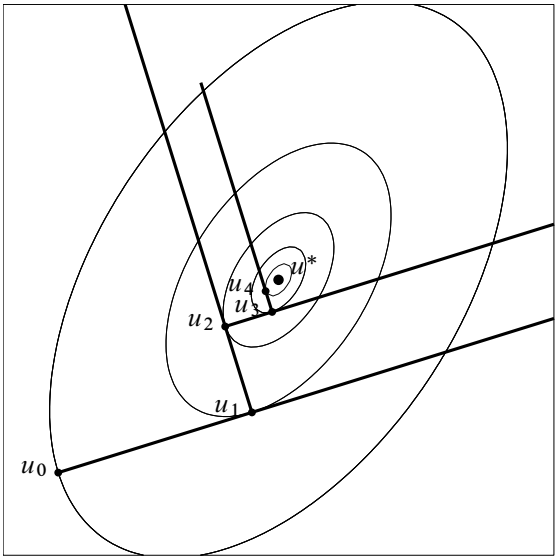


Figure 4.3. *Several iterates of the method of steepest descent in the case $m = 2$. The concentric ellipses are level sets of $\phi(u)$.*

point where $\phi(u)$ is minimized along this line. This will occur at the point where this line is tangent to a contour line. Consequently, the next search direction will be orthogonal to the current search direction, and in two dimensions we simply alternate between only two search directions. (Which particular directions depend on the location of u_0.)

If A is SPD, then the contour lines (level sets of ϕ) are always ellipses. How rapidly this algorithm converges depends on the geometry of these ellipses and on the particular starting vector u_0 chosen. Figure 4.4(a) shows the best possible case, where the ellipses are circles. In this case the iterates converge in one step from any starting guess, since the first search direction r_0 generates a line that always passes through the minimum u^* from any point.

Figure 4.4(b) shows a bad case, where the ellipses are long and skinny and the iteration slowly traverses back and forth in this shallow valley searching for the minimum. In general steepest descent is a slow algorithm, particularly when m is large, and should not be used in practice. Shortly we will see a way to improve this algorithm dramatically.

The geometry of the level sets of $\phi(u)$ is closely related to the eigenstructure of the matrix A. In the case $m = 2$ as shown in Figures 4.3 and 4.4, each ellipse can be characterized by a major and minor axis, as shown in Figure 4.5 for a typical level set. The points v_1 and v_2 have the property that the gradient $\nabla\phi(v_j)$ lies in the direction that connects v_j to the center u^*, i.e.,

$$Av_j - f = \lambda_j(v_j - u^*) \tag{4.34}$$

for some scalar λ_j. Since $f = Au^*$, this gives

$$A(v_j - u^*) = \lambda_j(v_j - u^*) \tag{4.35}$$

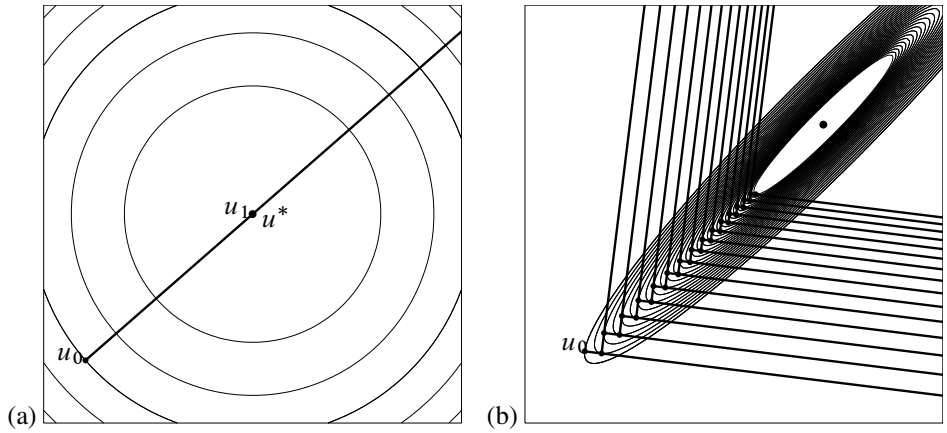


Figure 4.4. (a) *If A is a scalar multiple of the identity, then the level sets of $\phi(u)$ are circular and steepest descent converges in one iteration from any initial guess u_0.* (b) *If the level sets of $\phi(u)$ are far from circular, then steepest descent may converge slowly.*

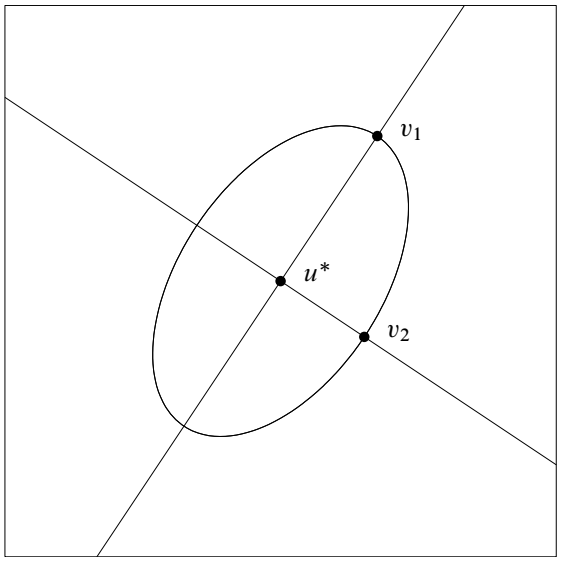


Figure 4.5. *The major and minor axes of the elliptical level set of $\phi(u)$ point in the directions of the eigenvectors of A.*

and hence each direction $v_j - u^*$ is an eigenvector of the matrix A, and the scalar λ_j is an eigenvalue.

If the eigenvalues of A are distinct, then the ellipse is noncircular and there are two unique directions for which the relation (4.34) holds, since there are two one-dimensional eigenspaces. Note that these two directions are always orthogonal since a symmetric matrix

A has orthogonal eigenvectors. If the eigenvalues of A are equal, $\lambda_1 = \lambda_2$, then every vector is an eigenvector and the level curves of $\phi(u)$ are circular. For $m = 2$ this happens only if A is a multiple of the identity matrix, as in Figure 4.4(a).

The length of the major and minor axes is related to the magnitude of λ_1 and λ_2. Suppose that v_1 and v_2 lie on the level set along which $\phi(u) = 1$, for example. (Note that $\phi(u^*) = -\frac{1}{2}u^{*T}Au^* \leq 0$, so this is reasonable.) Then

$$\frac{1}{2}v_j^T Av_j - v_j^T Au^* = 1. \tag{4.36}$$

Taking the inner product of (4.35) with $(v_j - u^*)$ and combining with (4.36) yields

$$\|v_j - u^*\|_2^2 = \frac{2 + u^{*T}Au^*}{\lambda_j}. \tag{4.37}$$

Hence the ratio of the length of the major axis to the length of the minor axis is

$$\frac{\|v_1 - u^*\|_2}{\|v_2 - u^*\|_2} = \sqrt{\frac{\lambda_2}{\lambda_1}} = \sqrt{\kappa_2(A)}, \tag{4.38}$$

where $\lambda_1 \leq \lambda_2$ and $\kappa_2(A)$ is the 2-norm condition number of A. (Recall that in general $\kappa_2(A) = \max_j |\lambda_j| / \min_j |\lambda_j|$ when A is symmetric.)

A multiple of the identity is perfectly conditioned, $\kappa_2 = 1$, and has circular level sets. Steepest descent converges in one iteration. An ill-conditioned matrix ($\kappa_2 \gg 1$) has long skinny level sets, and steepest descent may converge very slowly. The example shown in Figure 4.4(b) has $\kappa_2 = 50$, which is not particularly ill-conditioned compared to the matrices that often arise in solving differential equations.

When $m > 2$ the level sets of $\phi(u)$ are ellipsoids in m-dimensional space. Again the eigenvectors of A determine the directions of the principal axes and the spread in the size of the eigenvalues determines how stretched the ellipse is in each direction.

4.3.2 The A-conjugate search direction

The steepest descent direction can be generalized by choosing a search direction p_{k-1} in the kth iteration that might be different from the gradient direction r_{k-1}. Then we set

$$u_k = u_{k-1} + \alpha_{k-1} p_{k-1}, \tag{4.39}$$

where α_{k-1} is chosen to minimize $\phi(u_{k-1} + \alpha p_{k-1})$ over all scalars α. In other words, we perform a *line search* along the line through u_{k-1} in the direction p_{k-1} and find the minimum of ϕ on this line. The solution is at the point where the line is tangent to a contour line of ϕ, and

$$\alpha_{k-1} = \frac{p_{k-1}^T r_{k-1}}{p_{k-1}^T Ap_{k-1}}. \tag{4.40}$$

A *bad* choice of search direction p_{k-1} would be a direction orthogonal to r_{k-1}, since then p_{k-1} would be tangent to the level set of ϕ at u_{k-1}, $\phi(u)$ could only increase along

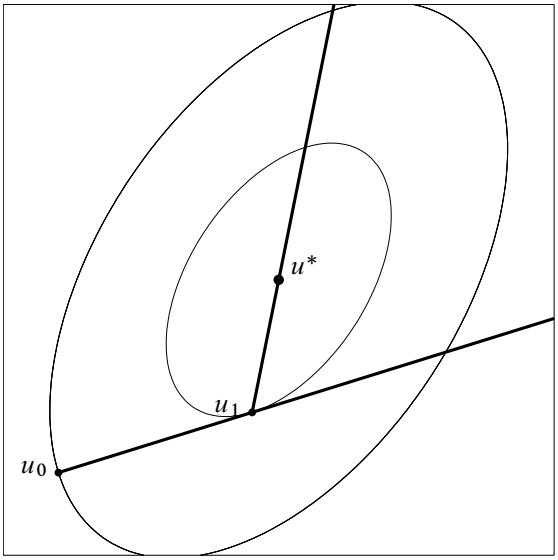


Figure 4.6. *The CG algorithm converges in two iterations from any initial guess u_0 in the case $m = 2$. The two search directions used are A-conjugate.*

this line, and so $u_k = u_{k-1}$. But as long as $p_{k-1}^T r_{k-1} \neq 0$, the new point u_k will be different from u_{k-1} and will satisfy $\phi(u_k) < \phi(u_{k-1})$.

Intuitively we might suppose that the best choice for p_{k-1} would be the direction of steepest descent r_{k-1}, but Figure 4.4(b) illustrates that this does not always give rapid convergence. A much better choice, if we could arrange it, would be to choose the direction p_{k-1} to point directly toward the solution u^*, as shown in Figure 4.6. Then minimizing ϕ along this line would give $u_k = u^*$, in which case we would have converged.

Since we don't know u^*, it seems there is little hope of determining this direction in general. But in two dimensions ($m = 2$) it turns out that we can take an arbitrary initial guess u_0 and initial search direction p_0 and then from the next iterate u_1 determine the direction p_1 that leads directly to the solution, as illustrated in Figure 4.6. Once we obtain u_1 by the formulas (4.39) and (4.40), we choose the next search direction p_1 to be a vector satisfying

$$p_1^T A p_0 = 0. \tag{4.41}$$

Below we will show that this is the optimal search direction, leading directly to $u_2 = u^*$. When $m > 2$ we generally cannot converge in two iterations, but we will see below that it is possible to define an algorithm that converges in at most m iterations to the exact solution (in exact arithmetic, at least).

Two vectors p_0 and p_1 that satisfy (4.41) are said to be *A-conjugate*. For any SPD matrix A, the vectors u and v are A-conjugate if the inner product of u with Av is zero, $u^T Av = 0$. If $A = I$, this just means the vectors are orthogonal, and A-conjugacy is a natural generalization of the notion of orthogonality. This concept is easily explained in terms of the ellipses that are level sets of the function $\phi(u)$ defined by (4.25). Consider

an arbitrary point on an ellipse. The direction tangent to the ellipse at this point and the direction that points toward the center of the ellipse are always A-conjugate. This is the fact that allows us to determine the direction toward the center once we know a tangent direction, which has been achieved by the line search in the first iteration. If $A = I$ then the ellipses are circles and the direction toward the center is simply the radial direction, which is orthogonal to the tangent direction.

To prove that the two directions shown in Figure 4.6 are A-conjugate, note that the direction p_0 is tangent to the level set of ϕ at u_1 and so p_0 is orthogonal to the residual $r_1 = f - Au_1 = A(u^* - u_1)$, which yields

$$p_0^T A(u^* - u_1) = 0. \tag{4.42}$$

On the other hand, $u^* - u_1 = \alpha p_1$ for some scalar $\alpha \neq 0$ and using this in (4.42) gives (4.41).

Now consider the case $m = 3$, from which the essential features of the general algorithm will be more apparent. In this case the level sets of the function $\phi(u)$ are concentric ellipsoids, two-dimensional surfaces in $\mathbb{R}^3$ for which the cross section in any two-dimensional plane is an ellipse. We start at an arbitrary point u_0 and choose a search direction p_0 (typically $p_0 = r_0$, the residual at u_0). We minimize $\phi(u)$ along the one-dimensional line $u_0 + \alpha p_0$, which results in the choice (4.40) for α_0, and we set $u_1 = u_0 + \alpha_0 p_0$. We now choose the search direction p_1 to be A-conjugate to p_0. In the previous example with $m = 2$ this determined a unique direction, which pointed straight to u^*. With $m = 3$ there is a two-dimensional space of vectors p_1 that are A-conjugate to p_0 (the plane orthogonal to the vector Ap_0). In the next section we will discuss the full CG algorithm, where a specific choice is made that is computationally convenient, but for the moment suppose p_1 is any vector that is both A-conjugate to p_0 and also linearly independent from p_0. We again use (4.40) to determine α_1 so that $u_2 = u_1 + \alpha_1 p_1$ minimizes $\phi(u)$ along the line $u_1 + \alpha p_1$.

We now make an observation that is crucial to understanding the CG algorithm for general m. The two vectors p_0 and p_1 are linearly independent and so they span a plane that cuts through the ellipsoidal level sets of $\phi(u)$, giving a set of concentric ellipses that are the contour lines of $\phi(u)$ within this plane. The fact that p_0 and p_1 are A-conjugate means that the point u_2 lies at the *center* of these ellipses. In other words, when restricted to this plane the algorithm so far looks exactly like the $m = 2$ case illustrated in Figure 4.6.

This means that u_2 not only minimizes $\phi(u)$ over the one-dimensional line $u_1 + \alpha p_1$ but in fact minimizes $\phi(u)$ over the entire two-dimensional plane $u_0 + \alpha p_0 + \beta p_1$ for all choices of α and β (with the minimum occurring at $\alpha = \alpha_0$ and $\beta = \alpha_1$).

The next step of the algorithm is to choose a new search direction p_2 that is A-conjugate to *both* p_0 and p_1. It is important that it be A-conjugate to both the previous directions, not just the most recent direction. This defines a unique direction (the line orthogonal to the plane spanned by Ap_0 and Ap_1). We now minimize $\phi(u)$ over the line $u_2 + \alpha p_2$ to obtain $u_3 = u_2 + \alpha_2 p_2$ (with α_2 given by (4.40)). It turns out that this always gives $u_3 = u^*$, the center of the ellipsoids and the solution to our original problem $Au = f$.

In other words, the direction p_2 always points from u_2 directly through the center of the concentric ellipsoids. This follows from the three-dimensional version of the result we showed above in two dimensions, that the direction tangent to an ellipse and the direction

toward the center are always A-conjugate. In the three-dimensional case we have a plane spanned by p_0 and p_1 and the point u_2 that minimized $\phi(u)$ over this plane. This plane must be the tangent plane to the level set of $\phi(u)$ through u_2. This tangent plane is always A-conjugate to the line connecting u_2 to u^*.

Another way to interpret this process is the following. After one step, u_1 minimizes $\phi(u)$ over the one-dimensional line $u_0 + \alpha p_0$. After two steps, u_2 minimizes $\phi(u)$ over the two-dimensional plane $u_0 + \alpha p_0 + \beta p_1$. After three steps, u_3 minimizes $\phi(u)$ over the three-dimensional space $u_0 + \alpha p_0 + \beta p_1 + \gamma p_2$. But this is all of $\mathbb{R}^3$ (provided p_0, p_1, and p_2 are linearly independent) and so $u_3 = u_0 + \alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2$ must be the global minimizer u^*.

For $m = 3$ this procedure always converges in *at most* three iterations (in exact arithmetic, at least). It may converge to u^* in fewer iterations. For example, if we happen to choose an initial guess u_0 that lies along one of the axes of the ellipsoids, then r_0 will already point directly toward u^*, and so $u_1 = u^*$ (although this is rather unlikely).

However, there are certain matrices A for which it will always take fewer iterations no matter what initial guess we choose. For example, if A is a multiple of the identity matrix, then the level sets of $\phi(u)$ are concentric *circles*. In this case r_0 points toward u^* from any initial guess u_0 and we always obtain convergence in one iteration. Note that in this case all three eigenvalues of A are equal, $\lambda_1 = \lambda_2 = \lambda_3$.

In the "generic" case (i.e., a random SPD matrix A), all the eigenvalues of A are distinct and three iterations are typically required. An intermediate case is if there are only two distinct eigenvalues, *e.g.*, $\lambda_1 = \lambda_2 \neq \lambda_3$. In this case the level sets of ϕ appear circular when cut by certain planes but appear elliptical when cut at other angles. As we might suspect, it can be shown that the CG algorithm always converges in at most *two* iterations in this case, from any initial u_0.

This generalizes to the following result for the analogous algorithm in m dimensions: in exact arithmetic, an algorithm based on A-conjugate search directions as discussed above converges in at most n iterations, where n is the number of distinct eigenvalues of the matrix $A \in \mathbb{R}^{m \times m}$ $(n \leq m)$.

4.3.3 The conjugate-gradient algorithm

In the above description of algorithms based on A-conjugate search directions we required that each search direction p_k be A-conjugate to all previous search directions, but we did not make a specific choice for this vector. In this section the full "conjugate gradient algorithm" is presented, in which a specific recipe for each p_k is given that has very nice properties both mathematically and computationally. The CG method was first proposed in 1952 by Hestenes and Stiefel [46], but it took some time for this and related methods to be fully understood and widely used. See Golub and O'Leary [36] for some history of the early developments.

This method has the feature mentioned at the end of the previous section: it always converges to the exact solution of $Au = f$ in a finite number of iterations $n \leq m$ (in exact arithmetic). In this sense it is not really an iterative method mathematically. We can view it as a "direct method" like Gaussian elimination, in which a finite set of operations produces the exact solution. If we programmed it to always take m iterations, then in principle we would always obtain the solution, and with the same asymptotic work estimate

as for Gaussian elimination (since each iteration takes at most $O(m^2)$ operations for matrix-vector multiplies, giving $O(m^3)$ total work). However, there are two good reasons why CG is better viewed as an iterative method than a direct method:

- In theory it produces the exact solution in n iterations (where n is the number of distinct eigenvalues) but in finite precision arithmetic u_n will not be the exact solution, and may not be substantially better than u_{n-1}. Hence it is not clear that the algorithm converges at all in finite precision arithmetic, and the full analysis of this turns out to be quite subtle [39].
- On the other hand, in practice CG frequently "converges" to a sufficiently accurate approximation to u^* in *far less* than n iterations. For example, consider solving a Poisson problem using the 5-point Laplacian on a 100×100 grid, which gives a linear system of dimension $m = 10{,}000$ and a matrix A that has $n \approx 5000$ distinct eigenvalues. An approximation to u^* consistent with the truncation error of the difference formula is obtained after approximately 150 iterations, however (after preconditioning the matrix appropriately).

That effective convergence often is obtained in far fewer iterations is crucial to the success and popularity of CG, since the operation count of Gaussian elimination is far too large for most sparse problems and we wish to use an iterative method that is much quicker. To obtain this rapid convergence it is often necessary to *precondition* the matrix, which effectively moves the eigenvalues around so that they are distributed more conducively for rapid convergence. This is discussed in Section 4.3.5, but first we present the basic CG algorithm and explore its convergence properties more fully.

The CG algorithm takes the following form:

Choose initial guess u_0 (possibly the zero vector)
$r_0 = f - Au_0$
$p_0 = r_0$
for $k = 1, 2, \ldots$
  $w_{k-1} = Ap_{k-1}$
  $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(p_{k-1}^T w_{k-1})$
  $u_k = u_{k-1} + \alpha_{k-1} p_{k-1}$
  $r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
  if $\|r_k\|$ is less than some tolerance then stop
  $\beta_{k-1} = (r_k^T r_k)/(r_{k-1}^T r_{k-1})$
  $p_k = r_k + \beta_{k-1} p_{k-1}$
  end

As with steepest descent, only one matrix-vector multiply is required at each iteration in computing w_{k-1}. In addition, two inner products must be computed each iteration. (By more careful coding than above, the inner product of each residual with itself can be computed once and reused twice.) To arrange this, we have used the fact that

$$p_{k-1}^T r_{k-1} = r_{k-1}^T r_{k-1}$$

to rewrite the expression (4.40).

Compare this algorithm to the steepest descent algorithm presented on page 80. Up through the convergence check it is essentially the same except that the A-conjugate search direction p_{k-1} is used in place of the steepest descent search direction r_{k-1} in several places.

The final two lines in the loop determine the next search direction p_k. This simple choice gives a direction p_k with the required property that p_k is A-conjugate to all the previous search directions p_j for $j = 0, 1, , \ldots, k-1$. This is part of the following theorem, which is similar to Theorem 38.1 of Trefethen and Bau [91], although there it is assumed that $u_0 = 0$. See also Theorem 2.3.2 in Greenbaum [39].

Theorem 4.1. *The vectors generated in the CG algorithm have the following properties, provided $r_k \neq 0$ (if $r_k = 0$, then we have converged):*

1. *p_k is A-conjugate to all the previous search directions,* i.e., *$p_k^T A p_j = 0$ for $j = 0, 1, , \ldots, k-1$.*
2. *The residual r_k is orthogonal to all previous residuals, $r_k^T r_j = 0$ for $j = 0, 1, , \ldots, k-1$.*
3. *The following three subspaces of $\mathbb{R}^m$ are identical:*

$$\begin{aligned} &\text{span}(p_0,\ p_1,\ p_2,\ \ldots,\ p_{k-1}),\\ &\text{span}(r_0,\ Ar_0,\ A^2r_0,\ \ldots,\ A^{k-1}r_0),\\ &\text{span}(Ae_0,\ A^2e_0,\ A^3e_0,\ \ldots,\ A^ke_0). \end{aligned} \tag{4.43}$$

The subspace $\mathcal{K}_k = \text{span}(r_0,\ Ar_0,\ A^2r_0,\ \ldots,\ A^{k-1}r_0)$ spanned by the vector r_0 and the first $k-1$ powers of A applied to this vector is called a *Krylov space* of dimension k associated with this vector.

The iterate u_k is formed by adding multiples of the search directions p_j to the initial guess u_0 and hence must lie in the affine spaces $u_0 + \mathcal{K}_k$ (i.e., the vector $u_k - u_0$ is in the linear space $\mathcal{K}_k$).

We have seen that the CG algorithm can be interpreted as minimizing the function $\phi(u)$ over the space $u_0 + \text{span}(p_0, p_1, \ldots, p_{k-1})$ in the kth iteration, and by the theorem above this is equivalent to minimizing $\phi(u)$ over the $u_0 + \mathcal{K}_k$. Many other iterative methods are also based on the idea of solving problems on an expanding sequence of Krylov spaces; see Section 4.4.

4.3.4 Convergence of conjugate gradient

The convergence theory for CG is related to the fact that u_k minimizes $\phi(u)$ over the affine space $u_0 + \mathcal{K}_k$ defined in the previous section. We now show that a certain norm of the error is also minimized over this space, which is useful in deriving estimates about the size of the error and rate of convergence.

Since A is assumed to be SPD, the A-norm defined by

$$\|e\|_A = \sqrt{e^T A e} \tag{4.44}$$

satisfies the requirements of a vector norm in Section A.3, as discussed further in Section C.10. This is a natural norm to use because

$$\begin{aligned}\|e\|_A^2 &= (u-u^*)^T A(u-u^*) \\ &= u^T Au - 2u^T Au^* + u^{*T} Au^* \\ &= 2\phi(u) + u^{*T} Au^*.\end{aligned} \tag{4.45}$$

Since $u^{*T} Au^*$ is a fixed number, we see that minimizing $\|e\|_A$ is equivalent to minimizing $\phi(u)$.

Since

$$u_k = u_0 + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_{k-1} p_{k-1},$$

we find by subtracting u^* that

$$e_k = e_0 + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_{k-1} p_{k-1}.$$

Hence $e_k - e_0$ is in $\mathcal{K}_k$ and by Theorem 4.1 lies in span$(Ae_0, A^2e_0, \ldots, A^ke_0)$. So $e_k = e_0 + c_1Ae_0 + c_2A^2e_0 + \cdots + c_kA^ke_0$ for some coefficients $c_1, \ \ldots, \ c_k$. In other words,

$$e_k = P_k(A)e_0, \tag{4.46}$$

where

$$P_k(A) = I + c_1A + c_2A^2 + \cdots + c_kA^k \tag{4.47}$$

is a polynomial in A. For a scalar value x we have

$$P_k(x) = 1 + c_1x + c_2x^2 + \cdots + c_kx^k \tag{4.48}$$

and $P_k \in \mathcal{P}_k$, where

$$\mathcal{P}_k = \{\text{polynomials } P(x) \text{ of degree at most } k \text{ satisfying } P(0) = 1\}. \tag{4.49}$$

The polynomial P_k constructed implicitly by the CG algorithm solves the minimization problem

$$\min_{P \in \mathcal{P}_k} \|P(A)e_0\|_A. \tag{4.50}$$

To understand how a polynomial function of a diagonalizable matrix behaves, recall that

$$A = V\Lambda V^{-1} \quad \Longrightarrow \quad A^j = V\Lambda^j V^{-1},$$

where V is the matrix of right eigenvectors, and so

$$P_k(A) = VP_k(\Lambda)V^{-1},$$

where

$$P_k(\Lambda) = \begin{bmatrix} P_k(\lambda_1) & & & \\ & P_k(\lambda_2) & & \\ & & \ddots & \\ & & & P_k(\lambda_m) \end{bmatrix}.$$

Note, in particular, that if $P_k(x)$ has a root at each eigenvalue $\lambda_1, \ \ldots, \ \lambda_m$, then $P_k(\Lambda)$ is the zero matrix and so $e_k = P_k(A)e_0 = 0$. If A has only $n \le m$ distinct eigenvalues $\lambda_1, \ \ldots, \ \lambda_n$, then there is a polynomial $P_n \in \mathcal{P}_n$ that has these roots, and hence the CG algorithm converges in at most n iterations, as was previously claimed.

To get an idea of how small $\|e_0\|_A$ will be at some earlier point in the iteration, we will show that for any polynomial $P(x)$ we have

$$\frac{\|P(A)e_0\|_A}{\|e_0\|_A} \le \max_{1 \le j \le m} |P(\lambda_j)| \tag{4.51}$$

and then exhibit one polynomial $\tilde{P}_k \in \mathcal{P}_k$ for which we can use this to obtain a useful upper bound on $\|e_k\|_A / \|e_0\|_k$.

Since A is SPD, the eigenvectors are orthogonal and we can choose the matrix V so that $V^{-1} = V^T$ and $A = V\Lambda V^{-1}$. In this case we obtain

$$\begin{aligned} \|P(A)e_0\|_A^2 &= e_0^T P(A)^T A P(A) e_0 \\ &= e_0^T V P(\Lambda) V^T A V P(\Lambda) V^T e_0 \\ &= e_0^T V \operatorname{diag}(\lambda_j P(\lambda_j)^2) V^T e_0 \\ &\le \max_{1 \le j \le m} P(\lambda_j)^2 \left(e_0^T V \Lambda V^T e_0 \right). \end{aligned} \tag{4.52}$$

Taking square roots and rearranging results in (4.51).

We will now show that for a particular choice of polynomials $\tilde{P}_k \in \mathcal{P}_k$ we can evaluate the right-hand side of (4.51) and obtain a bound that decreases with increasing k. Since the polynomial P_k constructed by CG solves the problem (4.50), we know that

$$\|P_k(A)e_0\|_A \le \|\tilde{P}_k(A)e_0\|_A,$$

and so this will give a bound for the convergence rate of the CG algorithm.

Consider the case $k = 1$, after one step of CG. We choose the linear function

$$\tilde{P}_1(x) = 1 - \frac{2x}{\lambda_m + \lambda_1}, \tag{4.53}$$

where we assume the eigenvalues are ordered $0 < \lambda_1 \le \lambda_2 \le \cdots \le \lambda_m$. A typical case is shown in Figure 4.7(a). The linear function $\tilde{P}_1(x) = 1 + c_1 x$ must pass through $P_1(0) = 1$ and the slope c_1 has been chosen so that

$$\tilde{P}_1(\lambda_1) = -\tilde{P}_1(\lambda_m),$$

which gives

$$1 + c_1\lambda_1 = -1 - c_1\lambda_m \quad \Longrightarrow \quad c_1 = -\frac{2}{\lambda_m + \lambda_1}.$$

If the slope were made any larger or smaller, then the value of $|\tilde{P}_1(\lambda)|$ would increase at either λ_m or λ_1, respectively; see Figure 4.7(a). For this polynomial we have

$$\begin{aligned} \max_{1 \le j \le m} |\tilde{P}_1(\lambda_j)| = \tilde{P}_1(\lambda_1) &= 1 - \frac{2\lambda_1}{\lambda_m + \lambda_1} = \frac{\lambda_m/\lambda_1 - 1}{\lambda_m/\lambda_1 + 1} \\ &= \frac{\kappa - 1}{\kappa + 1}, \end{aligned} \tag{4.54}$$

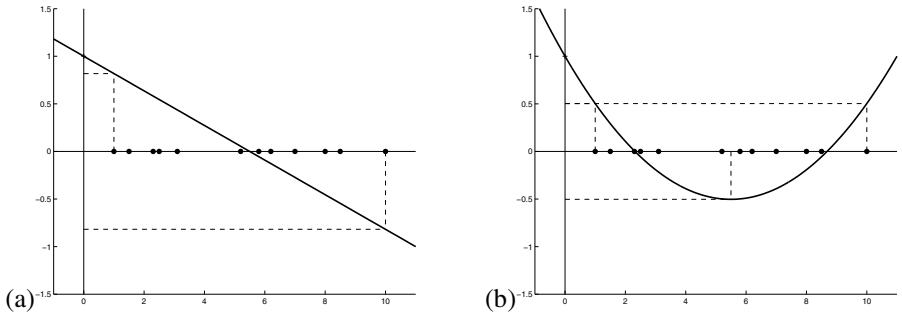


Figure 4.7. (a) *The polynomial* $\tilde{P}_1(x)$ *based on a sample set of eigenvalues marked by dots on the* x*-axis.* (b) *The polynomial* $\tilde{P}_2(x)$ *for the same set of eigenvalues.*

where $\kappa = \kappa_2(A)$ is the condition number of A. This gives an upper bound on the reduction of the error in the first step of the CG algorithm and is the best estimate we can obtain by knowing only the distribution of eigenvalues of A. The CG algorithm constructs the actual $P_1(x)$ based on e_0 as well as A and may do better than this for certain initial data. For example, if $e_0 = a_j v_j$ has only a single eigencomponent, then $P_1(x) = 1 - x/\lambda_j$ reduces the error to zero in one step. This is the case where the initial guess lies on an axis of the ellipsoid and the residual points directly to the solution $u^* = A^{-1} f$. But the above bound is the best we can obtain that holds for any e_0.

Now consider the case $k = 2$, after two iterations of CG. Figure 4.7(b) shows the quadratic function $\tilde{P}_2(x)$ that has been chosen so that

$$\tilde{P}_2(\lambda_1) = -\tilde{P}_1\left((\lambda_m + \lambda_1)/2\right) = \tilde{P}_2(\lambda_m).$$

This function equioscillates at three points in the interval $[\lambda_1, \lambda_m]$, where the maximum amplitude is taken. This is the polynomial from $\mathcal{P}_2$ that has the smallest maximum value on this interval, i.e., it minimizes

$$\max_{\lambda_1 \le x \le \lambda_m} |P(x)|.$$

This polynomial does not necessarily solve the problem of minimizing

$$\max_{1 \le j \le m} |P(\lambda_j)|$$

unless $(\lambda_1 + \lambda_m)/2$ happens to be an eigenvalue, since we could possibly reduce this quantity by choosing a quadratic with a slightly larger magnitude near the midpoint of the interval but a smaller magnitude at each eigenvalue. However, it has the great virtue of being easy to compute based only on λ_1 and λ_m. Moreover, we can compute the analogous polynomial $\tilde{P}_k(x)$ for arbitrary degree k, the polynomial from $\tilde{P}_k$ with the property of minimizing the maximum amplitude over the entire interval $[\lambda_1, \lambda_m]$. The resulting maximum amplitude also can be computed in terms of λ_1 and λ_m and in fact depends only on the ratio of these and hence depends only on the condition number of A. This gives an upper bound for the convergence rate of CG in terms of the condition number of A that often is quite realistic.

The polynomials we want are simply shifted and scaled versions of the Chebyshev polynomials discussed in Section B.3.2. Recall that $T_k(x)$ equioscillates on the interval $[-1, 1]$ with the extreme values ± 1 being taken at $k + 1$ points, including the endpoints. We shift this to the interval $[\lambda_1, \lambda_m]$, scale it so that the value at $x = 0$ is 1, and obtain

$$\tilde{P}_k(x) = \frac{T_k\left(\frac{\lambda_m+\lambda_1-2x}{\lambda_m-\lambda_1}\right)}{T_k\left(\frac{\lambda_m+\lambda_1}{\lambda_m-\lambda_1}\right)}. \tag{4.55}$$

For $k = 1$ this gives (4.53) since $T_1(x) = x$. We now need only compute

$$\max_{1\le j\le m} |\tilde{P}_k(\lambda_j)| = \tilde{P}_k(\lambda_1)$$

to obtain the desired bound on $\|e_k\|_A$. We have

$$\tilde{P}_k(\lambda_1) = \frac{T_k(1)}{T_k\left(\frac{\lambda_m+\lambda_1}{\lambda_m-\lambda_1}\right)} = \frac{1}{T_k\left(\frac{\lambda_m+\lambda_1}{\lambda_m-\lambda_1}\right)}. \tag{4.56}$$

Note that

$$\frac{\lambda_m + \lambda_1}{\lambda_m - \lambda_1} = \frac{\lambda_m/\lambda_1 + 1}{\lambda_m/\lambda_1 - 1} = \frac{\kappa + 1}{\kappa - 1} > 1$$

so we need to evaluate the Chebyshev polynomial at a point outside the interval $[-1, 1]$, which according to (B.27) is

$$T_k(x) = \cosh(k \cosh^{-1} x).$$

We have

$$\cosh(z) = \frac{e^z + e^{-z}}{2} = \frac{1}{2}(y + y^{-1})$$

where $y = e^z$, so if we make the change of variables $x = \frac{1}{2}(y + y^{-1})$, then $\cosh^{-1} x = z$ and

$$T_k(x) = \cosh(kz) = \frac{e^{kz} + e^{-kz}}{2} = \frac{1}{2}(y^k + y^{-k}).$$

We can find y from any given x by solving the quadratic equation $y^2 - 2xy + 1 = 0$, yielding

$$y = x \pm \sqrt{x^2 - 1}.$$

To evaluate (4.56) we need to evaluate T_k at $x = (\kappa + 1)/(\kappa - 1)$, where we obtain

$$\begin{aligned} y &= \frac{\kappa + 1}{\kappa - 1} \pm \sqrt{\left(\frac{\kappa + 1}{\kappa - 1}\right)^2 - 1} \\ &= \frac{\kappa + 1 \pm \sqrt{4\kappa}}{\kappa - 1} \\ &= \frac{(\sqrt{\kappa} \pm 1)^2}{(\sqrt{\kappa} + 1)(\sqrt{\kappa} - 1)} \\ &= \frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1} \quad \text{or} \quad \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}. \end{aligned} \tag{4.57}$$

Either choice of y gives the same value for

$$T_k\left(\frac{\kappa+1}{\kappa-1}\right) = \frac{1}{2}\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right]. \tag{4.58}$$

Using this in (4.56) and combining with (4.51) gives

$$\frac{\|P(A)e_0\|_A}{\|e_0\|_A} \le 2\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right]^{-1} \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k. \tag{4.59}$$

This gives an upper bound on the error when the CG algorithm is used. In practice the error may be smaller, either because the initial error e_0 happens to be deficient in some eigencoefficients or, more likely, because the optimal polynomial $P_k(x)$ is much smaller at all the eigenvalues λ_j than our choice $\tilde{P}_k(x)$ used to obtain the above bound. This typically happens if the eigenvalues of A are clustered near fewer than m points. Then the $P_k(x)$ constructed by CG will be smaller near these points and larger on other parts of the interval $[\lambda_1, \lambda_m]$ where no eigenvalues lie. As an iterative method it is really the number of clusters, not the number of mathematically distinct eigenvalues, that then determines how rapidly CG converges in practical terms.

The bound (4.59) is realistic for many matrices, however, and shows that in general the convergence rate depends on the size of the condition number κ. If κ is large, then

$$2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx 2\left(1-\frac{2}{\sqrt{\kappa}}\right)^k \approx 2e^{-2k/\sqrt{\kappa}}, \tag{4.60}$$

and we expect that the number of iterations required to reach a desired tolerance will be $k = O(\sqrt{\kappa})$.

For example, the standard second order discretization of the Poisson problem on a grid with m points in each direction gives a matrix with $\kappa = O(1/h^2)$, where $h = 1/(m+1)$. The bound (4.60) suggests that CG will require $O(m)$ iterations to converge, which is observed in practice. This is true in any number of space dimensions. In one dimension where there are only m unknowns this does not look very good (and of course it's best just to solve the tridiagonal system by elimination). In two dimensions there are m^2 unknowns and m^2 work per iteration is required to compute Ap_{k-1}, so CG requires $O(m^3)$ work to converge to a fixed tolerance, which is significantly better than Gauss elimination and comparable to SOR with the optimal ω. Of course for this problem a fast Poisson solver could be used, requiring only $O(m^2 \log m)$ work. But for other problems, such as variable coefficient elliptic equations with symmetric coefficient matrices, CG may still work very well while SOR works well only if the optimal ω is found, which may be impossible, and fast Fourier transform (FFT) methods are inapplicable. Similar comments apply in three dimensions.

4.3.5 Preconditioners

We saw in Section 4.3.4 that the convergence rate of CG generally depends on the condition number of the matrix A. Often *preconditioning* the system can reduce the condition

number of the matrix involved and speed up convergence. In fact preconditioning is absolutely essential for most practical problems, and there are many papers in the literature on the development of effective preconditioners for specific applications or general classes of problems.

If M is any nonsingular matrix, then

$$Au = f \quad \Longleftrightarrow \quad M^{-1}Au = M^{-1}f. \tag{4.61}$$

So we could solve the system on the right instead of the system on the left. If M is some approximation to A, then $M^{-1}A$ may have a much smaller condition number than A. If $M = A$, then $M^{-1}A$ is perfectly conditioned but we'd still be faced with the problem of computing $M^{-1}f = A^{-1}f$.

Of course in practice we don't actually form the matrix $M^{-1}A$. As we will see below, the preconditioned conjugate gradient (PCG) algorithm has the same basic form as CG, but a step is added in which a system of the form $Mz = r$ is solved, and it is here that the preconditioner is "applied." The idea is to choose an M for which $M^{-1}A$ is better conditioned than A but for which systems involving M are much easier to solve than systems involving A. Often this can be done by solving some approximation to the original physical problem (e.g., by solving on a coarser grid and then interpolating, by solving a nearby constant-coefficient problem).

A very simple preconditioner that is effective for some problems is simply to use $M = \text{diag}(A)$, a diagonal matrix for which solving linear systems is trivial. This doesn't help for the Poisson problem on a rectangle, where this is just a multiple of the identity matrix, and hence doesn't change the condition number at all, but for other problems such as variable coefficient elliptic equations with large variation in the coefficients, this can make a significant difference.

Another popular approach is to use an incomplete Cholesky factorization of the matrix A, as discussed briefly in Section 4.3.6. Other iterative methods are sometimes used as a preconditioner, for example, the multigrid algorithm of Section 4.6. Other preconditioners are discussed in many places; for example, there is a list of possible approaches in Trefethen and Bau [91].

A problem with the approach to preconditioning outlined above is that $M^{-1}A$ may not be symmetric, even if M^{-1} and A are, in which case CG could not be applied to the system on the right in (4.61). Instead we can consider solving a different system, again equivalent to the original:

$$(C^{-T}AC^{-1})(Cu) = C^{-T}f, \tag{4.62}$$

where C is a nonsingular matrix. Write this system as

$$\tilde{A}\tilde{u} = \tilde{f}. \tag{4.63}$$

Note that since $A^T = A$, the matrix $\tilde{A}$ is also symmetric even if C is not. Moreover $\tilde{A}$ is positive definite (provided A is) since

$$u^T\tilde{A}u = u^TC^{-T}AC^{-1}u = (C^{-1}u)^TA(C^{-1}u) > 0$$

for any vector $u \neq 0$.

Now the problem is that it may not be clear how to choose a reasonable matrix C in this formulation. The goal is to make the condition number of $\tilde{A}$ small, but C appears twice in the definition of $\tilde{A}$ so C should be chosen as some sort of "square root" of A. But note that the condition number of $\tilde{A}$ depends only on the eigenvalues of this matrix, and we can apply a similarity transformation to $\tilde{A}$ without changing its eigenvalues, e.g.,

$$C^{-1}\tilde{A}C = C^{-1}C^{-T}A = (C^TC)^{-1}A. \tag{4.64}$$

The matrix $\tilde{A}$ thus has the same condition number as $(C^TC)^{-1}A$. So if we have a sensible way to choose a preconditioner M in (4.61) that is SPD, we could in principle determine C by a Cholesky factorization of the matrix M.

In practice this is not necessary, however. There is a way to write the PCG algorithm in such a form that it only requires solving systems involving M (without ever computing C) but that still corresponds to applying CG to the SPD system (4.63).

To see this, suppose we apply CG to (4.63) and generate vectors $\tilde{u}_k$, $\tilde{p}_k$, $\tilde{w}_k$, and $\tilde{r}_k$. Now define

$$u_k = C^{-1}\tilde{u}_k, \quad p_k = C^{-1}\tilde{p}_k, \quad w_k = C^{-1}\tilde{w}_k, \text{ and } r_k = C^T\tilde{r}_k.$$

Note that $\tilde{r}_k$ is multiplied by C^T, not C^{-1}. Here $\tilde{r}_k$ is the residual when $\tilde{u}_k$ is used in the system (4.63). Note that if $\tilde{u}_k$ approximates the solution to (4.62), then u_k will approximate the solution to the original system $Au = f$. Moreover, we find that

$$r_k = C(\tilde{f} - \tilde{A}\tilde{u}_k) = f - Au_k$$

and so r_k is the residual for the original system. Rewriting this CG algorithm in terms of the variables u_k, p_k, w_k, and r_k, we find that it can be rewritten as the following PCG algorithm:

$r_0 = f - Au_0$
Solve $Mz_0 = r_0$ for z_0
$p_0 = z_0$
for $k = 1, 2, \ldots$
    $w_{k-1} = Ap_{k-1}$
    $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(p_{k-1}^T w_{k-1})$
    $u_k = u_{k-1} + \alpha_{k-1} p_{k-1}$
    $r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
    if $\|r_k\|$ is less than some tolerance then stop
    Solve $Mz_k = r_k$ for z_k
    $\beta_{k-1} = (z_k^T r_k)/(r_{k-1}^T r_{k-1})$
    $p_k = z_k + \beta_{k-1} p_{k-1}$
    end

Note that this is essentially the same as the CG algorithm on page 87, but we solve the system $Mz_k = r_k$ for $z_k = M^{-1}r_k$ in each iteration and then use this vector in place of r_k in two places in the last two lines.

4.3.6 Incomplete Cholesky and ILU preconditioners

There is one particular preconditioning strategy where the matrix C is in fact computed and used. Since A is SPD it has a Cholesky factorization of the form $A = R^T R$, where R is an upper triangular matrix (this is just a special case of the LU factorization). The problem with computing and using this factorization to solve the original system $Au = f$ is that the elimination process used to compute R generates a lot of nonzeros in the R matrix, so that it is typically much less sparse than A.

A popular preconditioner that is often very effective is to do an *incomplete Cholesky factorization* of the matrix A, in which nonzeros in the factors are allowed to appear only in positions where the corresponding element of A is nonzero, simply throwing away the other elements as we go along. This gives an approximate factorization of the form $A \approx C^T C$. This defines a preconditioner $M = C^T C$. To solve systems of the form $Mz = r$ required in the PCG algorithm we use the known Cholesky factorization of M and only need to do forward and back substitutions for these lower and upper triangular systems. This approach can be generalized by specifying a *drop tolerance* and dropping only those elements of R that are smaller than this tolerance. A smaller drop tolerance will give a better approximation to A but a denser matrix C.

Methods for nonsymmetric linear systems (e.g., the GMRES algorithm in the next section) also generally benefit greatly from preconditioners and this idea can be extended to *incomplete LU (ILU) factorizations* as a preconditioner for nonsymmetric systems.

4.4 The Arnoldi process and GMRES algorithm

For linear systems that are not SPD, many other iterative algorithms have been developed. We concentrate here on just one of these, the popular GMRES (generalized minimum residual) algorithm. In the course of describing this method we will also see the Arnoldi process, which is useful in other applications.

In the kth step of GMRES a least squares problem is solved to find the best approximation to the solution of $Au = f$ from the affine space $u_0 + \mathcal{K}_k$, where again $\mathcal{K}_k$ is the k-dimensional Krylov space $\mathcal{K}_k = \text{span}(r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0)$ based on the initial residual $r_0 = f - Au_0$. To do this we build up a matrix of the form

$$Q_k = [q_1\ q_2\ \cdots\ q_k] \in \mathbb{R}^{m \times k},$$

whose columns form an orthonormal basis for the space $\mathcal{K}_k$. In the kth iteration we determine the vector q_{k+1} by starting with some vector v_j that is not in $\mathcal{K}_k$ and orthogonalizing it to $q_1,\ q_2,\ \ldots,\ q_k$ using a Gram–Schmidt-type procedure. How should we choose v_k? One obvious choice might be $v_k = A^k r_0$. This is a bad choice, however. The vectors $r_0,\ Ar_0,\ A^2 r_0,\ \ldots$, although linearly independent and a natural choice from our definition of the Krylov space, tend to become more and more closely aligned (nearly linearly dependent) as k grows. (In fact they converge to the eigenvector direction of the dominant eigenvalue of A since this is just the power method.) In other words the Krylov matrix

$$K_{k+1} = [r_0\ Ar_0\ A^2 r_0\ \cdots\ A^k r_0]$$

has rank $k + 1$ but has some very small singular values. Applying the orthogonalization procedure using $v_k = A^k r_0$ would amount to doing a QR factorization of the matrix K_{k+1},

which is numerically unstable in this case. Moreover, it is not clear how we would use the resulting basis to find the least square approximation to $Au = f$ in the affine space $u_0 + \mathcal{K}_k$.

Instead we choose $v_k = Aq_k$ as the starting point in the kth step. Since q_k has already been orthogonalized to all the previous basis vectors, this does not tend to be aligned with an eigendirection. In addition, the resulting procedure can be viewed as building up a factorization of the matrix A itself that can be directly used to solve the desired least squares problem.

This procedure is called the *Arnoldi process*. This algorithm is important in other applications as well as in the solution of linear systems, as we will see below. Here is the basic algorithm, with an indication of where a least squares problem should be solved in each iteration to compute the GMRES approximations u_k to the solution of the linear system:

```
q_1 = r_0/‖r_0‖_2
for k = 1, 2, ...
    v = Aq_k
    for i = 1 : k
        h_ik = q_i^T v
        v = v − h_ik q_i        % orthogonalize to previous vectors
        end
    h_{k+1,k} = ‖v‖_2
    q_{k+1} = v/h_{k+1,k}       % normalize
    % For GMRES: Check residual of least squares problem (4.75).
    % If it's sufficiently small, halt and compute u_k
    end
```

Before discussing the least squares problem, we must investigate the form of the matrix factorization we are building up with this algorithm. After k iterations we have

$$Q_k = [q_1\ q_2\ \cdots\ q_k] \in \mathbb{R}^{m\times k}, \qquad Q_{k+1} = [Q_k\ q_{k+1}] \in \mathbb{R}^{m\times(k+1)},$$

which form orthonormal bases for $\mathcal{K}_k$ and $\mathcal{K}_{k+1}$, respectively. Let

$$H_k = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1,k-1} & h_{1k} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2,k-1} & h_{2k} \\ & h_{32} & h_{33} & \cdots & h_{3,k-1} & h_{3k} \\ & & \ddots & \ddots & & \vdots \\ & & & & h_{k,k-1} & h_{kk} \end{bmatrix} \in \mathbb{R}^{k\times k} \tag{4.65}$$

be the upper Hessenberg matrix consisting of the h values computed so far. We will also need the matrix $\tilde{H}_k \in \mathbb{R}^{(k+1)\times k}$ consisting of H_k with an additional row that is all zeros except for the $h_{k+1,k}$ entry, also computed in the kth step of Arnoldi.

Now consider the matrix product

$$AQ_k = [Aq_1\ Aq_2\ \cdots\ Aq_k].$$

The jth column of this matrix has the form of the starting vector v used in the jth iteration of Arnoldi, and unraveling the computations done in the jth step shows that

$$h_{j+1,j}q_{j+1} = Aq_j - h_{1j}q_1 - h_{2j}q_2 - \cdots - h_{jj}q_j.$$

This can be rearranged to give

$$Aq_j = h_{1j}q_1 + h_{2j}q_2 + \cdots + h_{jj}q_j + h_{j+1,j}q_{j+1}. \tag{4.66}$$

The left-hand side is the jth column of AQ_k and the right-hand side, at least for $j < k$, is the jth column of the matrix $Q_k H_k$. We find that

$$AQ_k = Q_k H_k + h_{k+1,k}q_{k+1}e_k^T. \tag{4.67}$$

In the final term the vector $e_k^T = [0\ 0\ \cdots\ 0\ 1]$ is the vector of length k with a 1 in the last component and $h_{k+1,k}q_{k+1}e_k^T$ is the $m \times k$ matrix that is all zeros except the last column, which is $h_{k+1,k}q_{k+1}$. This term corresponds to the last term in the expression (4.66) for $j = k$. The expression (4.67) can also be rewritten as

$$AQ_k = Q_{k+1}\tilde{H}_k. \tag{4.68}$$

If we run the Arnoldi process to completion (i.e., up to $k = m$, the dimension of A), then we will find in the final step that $v = Aq_m$ lies in the Krylov space $\mathcal{K}_m$ (which is already all of $\mathbb{R}^m$), so orthogonalizing it to each of the q_i for $i = 1 : m$ will leave us with $v = 0$. So in this final step there is no $h_{m+1,m}$ value or q_{m+1} vector and setting $Q = Q_m$ and $H = H_m$ gives the result

$$AQ = QH,$$

which yields

$$Q^T AQ = H \qquad \text{or} \qquad A = QHQ^T. \tag{4.69}$$

We have reduced A to Hessenberg form by a similarity transformation.

Our aim at the moment is not to reduce A all the way by running the algorithm to $k = m$ but rather to approximate the solution to $Au = f$ well in a few iterations. After k iterations we have (4.67) holding. We wish to compute u_k, an approximation to $u = A^{-1}f$ from the affine space $u_0 + \mathcal{K}_k$, by minimizing the 2-norm of the residual $r_k = f - Au_k$ over this space. Since the columns of Q_k form a basis for $\mathcal{K}_k$, we must have

$$u_k = u_0 + Q_k y_k \tag{4.70}$$

for some vector $y_k \in \mathbb{R}^k$, and so the residual is

$$\begin{aligned} r_k &= f - A(u_0 + Q_k y_k) \\ &= r_0 - AQ_k y_k \\ &= r_0 - Q_{k+1}\tilde{H}_k y_k, \end{aligned} \tag{4.71}$$

where we have used (4.68). But recall that the first column of Q_{k+1} is just $q_1 = r_0/\|r_0\|_2$ so we have $r_0 = Q_{k+1}\eta$, where η is the vector

$$\eta = \begin{bmatrix} \|r_0\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{k+1}. \tag{4.72}$$

Hence

$$r_k = Q_{k+1}(\eta - \tilde{H}_k y_k). \tag{4.73}$$

Since $Q_{k+1}^T Q_{k+1} = I$, computing $r_k^T r_k$ shows that

$$\|r_k\|_2 = \|\eta - \tilde{H}_k y_k\|_2. \tag{4.74}$$

In the kth iteration of GMRES we choose y_k to solve the least squares problem

$$\min_{y \in \mathbb{R}^k} \|\eta - \tilde{H}_k y\|_2, \tag{4.75}$$

and the approximation u_k is then given by (4.70).

Note the following (see, e.g., Greenbaum [39] for details):

- $\tilde{H}_k \in \mathbb{R}^{(k+1)\times k}$ and $\eta \in \mathbb{R}^{k+1}$, so this is a small least squares problem when $k \ll m$.
- $\tilde{H}_k$ is already nearly upper triangular, so solving the least squares problem by computing the QR factorization of this matrix is relatively cheap.
- Moreover, in each iteration $\tilde{H}_k$ consists of $\tilde{H}_{k-1}$ with one additional row and column added. Since the QR factorization of $\tilde{H}_{k-1}$ has already been computed in the previous iteration, the QR factorization of $\tilde{H}_k$ is easily computed with little additional work.
- Once the QR factorization is known, it is possible to compute the residual in the least squares problem (4.75) without actually solving for y_k (which requires solving an upper triangular system of size k using the R matrix from QR). So in practice only the residual is checked each iteration and the final y_k and u_k are actually computed only after the convergence criterion is satisfied.

Notice, however, one drawback of GMRES, and the Arnoldi process more generally, for nonsymmetric matrices: in the kth iteration we must orthogonalize v to all k previous basis vectors, so we must keep all these vectors in storage. For practical problems arising from discretizing a multidimensional partial differential equation (PDE), each of these "vectors" is an approximation to the solution over the full grid, which may consist of millions of grid points. Taking more than a few iterations may consume a great deal of storage.

Often in GMRES the iteration is restarted periodically to save storage: the approximation u_k at some point is used as the initial guess for a new GMRES iteration. There's a large literature on this and other variations of GMRES.

4.4.1 Krylov methods based on three term recurrences

Note that if A is symmetric, then so is the Hessenberg matrix H, since

$$H^T = (Q^T A Q)^T = Q^T A^T Q = Q^T A Q = H,$$

and hence H must be tridiagonal. In this case the Arnoldi iteration simplifies in a very important way: $h_{ik} = 0$ for $i = 1, 2, \ldots, (k-2)$ and in the kth iteration of Arnoldi v

only needs to be orthogonalized to the previous two basis vectors. There is a three-term recurrence relation for each new basis vector in terms of the previous two. This means only the two previous vectors need to be stored at any time, rather than all the previous q_i vectors, which is a dramatic improvement for systems of large dimension.

The special case of Arnoldi on a symmetric matrix (or more generally a complex Hermitian matrix) is called the *Lanczos iteration* and plays an important role in many numerical algorithms, not just for linear systems but also for eigenvalue problems and other applications.

There are also several iterative methods for nonsymmetric systems of equations that are based on three-term recurrence relations using the idea of *biorthogonalization*—in addition to building up a Krylov space based on powers of the matrix A, a second Krylov space based on powers of the matrix A^H is simultaneously determined. Basis vectors v_i and w_i for the two spaces are found that are not orthogonal sets separately, but are instead "biorthogonal" in the sense that

$$v_i^H w_j = 0 \quad \text{if } i \neq j.$$

It turns out that there are three-term recurrence relations for these sets of basis vectors, eliminating the need for storing long sequences of vectors. The disadvantage is that two matrix-vector multiplies must be performed each iteration, one involving A and another involving A^H. One popular method of this form is Bi-CGSTAB (bi-conjugate gradient stabilized), introduced by Van der Vorst [95]. See, e.g., [39], [91] for more discussion of this method and other variants.

4.4.2 Other applications of Arnoldi

The Arnoldi process has other applications besides the approximate solution of linear systems. Note from (4.67) that

$$Q_k^T A Q_k = H_k \tag{4.76}$$

since $Q_k^T Q_k = I$ and $Q_k^T q_{k+1} = 0$. This looks much like (4.69), but here Q_k is a rectangular matrix (for $k < m$) and so this is not a similarity transformation and H_k does not have the same eigenvalues as A (or even the same number of eigenvalues, since it has only k). However, a very useful fact is that the eigenvalues of H_k are typically good approximations to the dominant eigenvalues of A (those with largest magnitude). In many eigenvalue applications where A is a large sparse matrix, the primary interest is in determining the dominant eigenvalues (e.g., in determining stability or asymptotic growth properties of matrix iterations or exponentials). In this case we can run the Arnoldi process (which requires only matrix-vector multiplies with A) and then calculate the eigenvalues of the small matrix H_k in the kth iteration as an approximation to the dominant eigenvalues of A. This approach is implemented in the ARPACK software [62], which is used, for example, by the `eigs` command in MATLAB.

Also note that from (4.76), by multiplying on the left by Q_k and on the right by Q_k^T we obtain

$$Q_k Q_k^T A Q_k Q_k^T = Q_k H_k Q_k^T. \tag{4.77}$$

If $k = m$, then $Q_k Q_k^T = I$ and this is simply (4.69). For $k < m$, $Q_k Q_k^T$ is the projection matrix that projects any vector z in $\mathbb{R}^m$ onto the k-dimensional Krylov space $\mathcal{K}_k$.

So the operator on the left of (4.77), when applied to any vector in $z \in \mathbb{R}^m$, has the following effect: the vector is first projected to $\mathcal{K}_k$, then A is applied, and then the result is again projected to $\mathcal{K}_k$. The operator on the right does the same thing in a different form: $Q_k^T z \in \mathbb{R}^k$ consists of the coefficients of the basis vectors of Q_k for the projected vector. Multiplying by H_k transforms these coefficients according to the effect of A, and $H_k Q_k^T z$ are then the modified coefficients used to form a linear combination of the basis vectors when this is multiplied by Q_k. Hence we can view H_k as the restriction of A to the k-dimensional Krylov space $\mathcal{K}_k$. Thus it is not so surprising, for example, that the eigenvalues of H_k approximate the dominant eigenvalues of A. As commented above, the basis vectors $f,\ Af,\ A^2 f,\ \ldots$ for $\mathcal{K}_k$ tend to align with the dominant eigenvectors of A, and if an eigenvector of A lies in $\mathcal{K}_k$, then it is also an eigenvector of the restriction of A to this space.

We will see another use of Krylov space methods in Section 11.6, where we consider exponential time differencing methods for time-dependent ordinary differential equations (ODEs). The matrix exponential applied to a vector, $e^{At}v$, arises in solving linear systems of ODEs. This often can be effectively approximated by $Q_k e^{H_k t} Q_k^T v$ for $k \ll m$. More generally, other functions $\phi(z)$ can be extended to matrix arguments (using the Cauchy integral formula (D.4), for example) and their action often approximated by $\phi(A)v \approx Q_k \phi(H_k t) Q_k^T v$.

4.5 Newton–Krylov methods for nonlinear problems

So far in this chapter we have considered only linear problems and a variety of iterative methods that can be used to solve sparse linear systems of the form $Au = f$. However, many differential equations are nonlinear and these naturally give rise to nonlinear systems of equations after discretization. In Section 2.16 we considered a nonlinear boundary value problem and discussed the use of Newton's method for its solution. Recall that Newton's method is an iterative method based on linearizing the problem about the current approximation to the solution and then solving a linear system of equations involving the Jacobian matrix to determine the next update to the approximation. If the nonlinear system is written as $G(u) = 0$, then the Newton update is

$$u^{[j+1]} = u^{[j]} - \delta^{[j]}, \tag{4.78}$$

where $\delta^{[j]}$ is the solution to the linear system

$$J^{[j]}\delta^{[j]} = G(u^{[j]}). \tag{4.79}$$

Here $J^{[j]} = G'(u^{[j]})$ is the Jacobian matrix evaluated at the current iterate. For the one-dimensional problem of Section 2.16 the Jacobian matrix is tridiagonal and the linear system is easily solved in each iteration by a direct method.

For a nonlinear problem in more space dimensions the Jacobian matrix typically will have the same nonzero structure as the matrices discussed in the context of linear elliptic equations in Chapter 3. (Of course for a linear problem $Au = f$ we have $G(u) = Au - f$ and the matrix A is the Jacobian matrix.) Hence when solving a nonlinear elliptic equation by a Newton method we must solve, in each Newton iteration, a sparse linear system of the type we are tackling in this chapter. For practical problems the Jacobian matrix is often

nonsymmetric and Krylov space methods such as GMRES are a popular choice. This gives an obvious way to combine Newton's method with Krylov space methods: in each iteration of Newton's method determine all the elements of the Jacobian matrix $J^{[j]}$ and then apply a Krylov space method to solve the system (4.79).

However, the term *Newton–Krylov method* often refers to something slightly different, in which the calculation of the full Jacobian matrix is avoided in performing the Krylov space iteration. These methods are also called *Jacobian–free Newton–Krylov methods* (JFNK), and a good survey of these methods and their history and applicability is given in the review paper of Knoll and Keyes [57].

To explain the basic idea, consider a single iteration of the Newton method and drop the superscript j for notational convenience. So we need to solve a linear system of the form

$$J(u)\delta = G(u), \tag{4.80}$$

where u a fixed vector (the current Newton iterate $u^{[j]}$).

When the GMRES algorithm (or any other iterative method requiring only matrix vector products) is applied to the linear system (4.80), we require only the product $J(u)q_k$ for certain vectors q_k (where k is the iteration index of the linear solver). The key to JFNK is to recognize that since $J(u)$ is a Jacobian matrix, the vector $J(u)q_k$ is simply the directional derivative of the nonlinear function G at this particular u in the direction q_k. The Jacobian matrix contains all the information needed to compute the directional derivative in any arbitrary direction, but there is no need to compute the full matrix if, in the course of the Krylov iteration, we are only going to need the directional derivative in relatively few directions. This is the case if we hope that the Krylov iteration will converge in very few iterations relative to the dimension of the system.

How do we compute the directional derivative $J(u)q_k$ without knowing $J(u)$? The standard approach is to use a simple finite difference approximation,

$$J(u)q_k \approx (G(u + \epsilon q_k) - G(u))/\epsilon, \tag{4.81}$$

where ϵ is some small real number. This approximation is first order accurate in ϵ but is sufficiently accurate for the needs of the Krylov space method if we take ϵ quite small. If ϵ is too small, however, then numerical cancellation can destroy the accuracy of the approximation in finite precision arithmetic. For scalar problems the optimal trade-off typically occurs at $\epsilon = \sqrt{\epsilon_{\text{mach}}}$, the square root of the machine precision (i.e., $\epsilon \approx 10^{-8}$ for 64-bit double precision calculations). See [57] for some comments on good choices.

JFNK is particularly advantageous for problems where the derivatives required in the Jacobian matrix cannot be easily computed analytically, for example, if the computation of $G(u)$ involves table look-ups or requires solving some other nonlinear problem. A subroutine evaluating $G(u)$ is already needed for a Krylov space method in order to evaluate the right-hand side of (4.80), and the JFNK method simply calls this in each iteration of the Krylov method to compute $G(u + \epsilon q_k)$.

Good preconditioners generally are required to obtain good convergence properties and limit the number of Krylov iterations (and hence nonlinear G evaluations) required. As with Newton's method in other contexts, a good initial guess is often required to achieve convergence of the Newton iteration, regardless of how the system (4.79) is solved in each iteration. See [57] for more comments on these and other issues.

4.6 Multigrid methods

We return to the solution of linear systems $Au = f$ and discuss a totally different approach to the solution of such systems. Multigrid methods also can be applied directly to nonlinear problems and there is a vast literature on variations of these methods and applications to a variety of problems. Here we concentrate on understanding the main idea of multigrid methods in the context of the one-dimensional model problem $u''(x) = f(x)$. For more discussion, see, for example, [11], [52], [41], [101].

4.6.1 Slow convergence of Jacobi

Let

$$f(x) = -20 + a\phi''(x)\cos(\phi(x)) - a(\phi'(x))^2 \sin(\phi(x)), \tag{4.82}$$

where $a = 0.5$, $\phi(x) = 20\pi x^3$, and consider the boundary value problem $u''(x) = f(x)$ with Dirichlet boundary conditions $u(0) = 1$ and $u(1) = 3$. The true solution is

$$u(x) = 1 + 12x - 10x^2 + a\sin(\phi(x)), \tag{4.83}$$

which is plotted in Figure 4.8(a). This function has been chosen because it clearly contains variations on many different spatial scales, i.e., large components of many different frequencies.

We discretize this problem with the standard tridiagonal systems (2.10) and apply the Jacobi iterative method of Section 4.1 to the linear initial guess u_0 with components $1 + 2x_i$, which is also shown in Figure 4.8(a). Figure 4.8(b) shows the error e_0 in this initial guess on a grid with $m = 255$ grid points.

The left column of Figure 4.9 shows the approximations obtained after $k = 20$, 100, and 1000 iterations of Jacobi. This method converges very slowly and it would take about 10^5 iterations to obtain a useful approximation to the solution. However, notice something very interesting in Figure 4.9. The more detailed features of the solution develop relatively quickly and it is the larger-scale features that are slow to appear. At first this may seem counterintuitive since we might expect the small-scale features to be harder to capture.

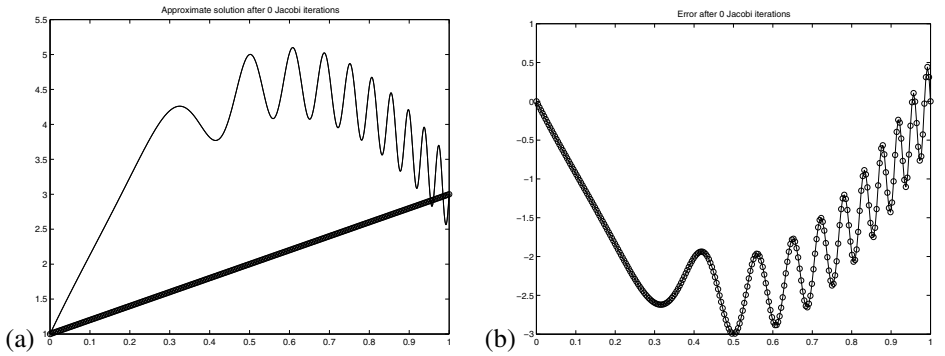


Figure 4.8. (a) *The solution $u(x)$ (solid line) and initial guess u_0 (circles).* (b) *The error e_0 in the initial guess.*

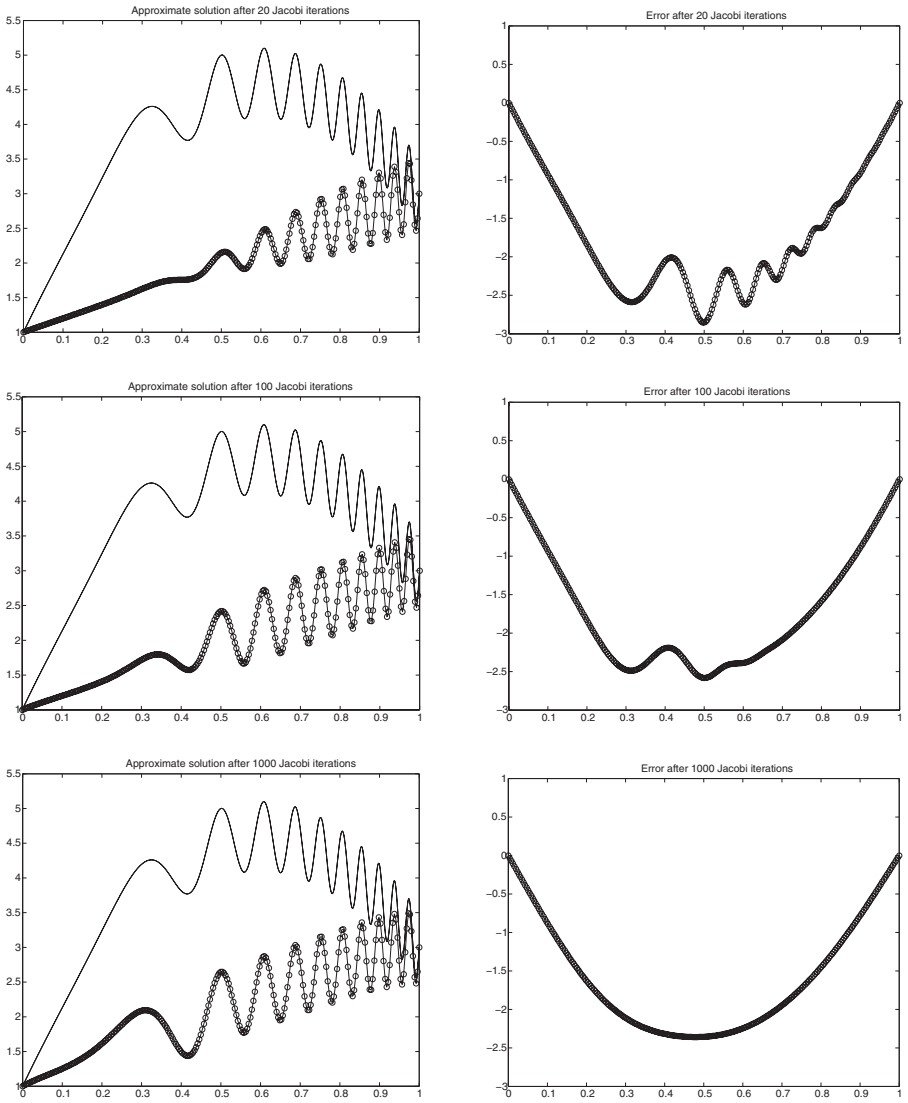


Figure 4.9. *On the left: The solution $u(x)$ (solid line) and Jacobi iterate u_k after k iterations. On the right: The error e_k, shown for $k = 20$ (top), $k = 100$ (middle), and $k = 1000$ (bottom).*

This is easier to understand if we look at the errors shown on the right. The initial error is highly oscillatory but these oscillations are rapidly damped by the Jacobi iteration, and after only 20 iterations the error is much smoother than the initial error. After 100 iterations it is considerably smoother and after 1000 iterations only the smoothest components of the error remain. This component takes nearly forever to be damped out, and it is this component that dominates the error and renders the approximate solution worthless.

To understand why higher frequency components of the error are damped most rapidly, recall from Section 4.2 that the error $e_k = u_k = u^*$ satisfies

$$e_k = Ge_{k-1},$$

where, for the tridiagonal matrix A,

$$G = I + \frac{h^2}{2}A = \begin{bmatrix} 0 & 1/2 & & & & & \\ 1/2 & 0 & 1/2 & & & & \\ & 1/2 & 0 & 1/2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1/2 & 0 & 1/2 & \\ & & & & 1/2 & 0 & \end{bmatrix}.$$

The ith element of e_k is simply obtained by averaging the $(i-1)$ and $(i+1)$ elements of e_{k-1} and this averaging damps out higher frequencies more rapidly than low frequencies. This can be quantified by recalling from Section 4.1 that the eigenvectors of G are the same as the eigenvectors of A. The eigenvector u^p has components

$$u_j^p = \sin(\pi p x_j) \qquad (x_j = jh, \;\; j = 1, \, 2, \, \ldots, \, m), \tag{4.84}$$

while the corresponding eigenvalue is

$$\gamma_p = \cos(p\pi h) \tag{4.85}$$

for $p = 1, \, 2, \, \ldots, \, m$. If we decompose the initial error e_0 into eigencomponents,

$$e_0 = c_1 u^1 + c_2 u^2 + \cdots + c_m u^m, \tag{4.86}$$

then we have

$$e_k = c_1 \gamma_1^k u^1 + c_2 \gamma_2^k u^2 + \cdots + c_m \gamma_m^k u^m. \tag{4.87}$$

Hence the pth eigencomponent decays at the rate γ_p^k as k increases. For large k the error is dominated by the components $c_1 \gamma_1^k u^1$ and $c_m \gamma_m^k u^m$, since these eigenvalues are closest to 1:

$$\gamma_1 = -\gamma_m \approx 1 - \frac{1}{2}\pi^2 h^2.$$

This determines the overall convergence rate, as discussed in Section 4.1.

Other components of the error, however, decay much more rapidly. In fact, for half the eigenvectors, those with $m/4 \le p \le 3m/4$, the eigenvalue γ_p satisfies

$$|\gamma_p| \le \frac{1}{\sqrt{2}} \approx 0.7$$

and $|\gamma_p|^{20} < 10^{-3}$, so that 20 iterations are sufficient to reduce these components of the error by a factor of 1000. Decomposing the error e_0 as in (4.86) gives a Fourier sine series representation of the error, since u^p in (4.84) is simply a discretized version of the sine

function with frequency p. Hence eigencomponents $c_p u^p$ for larger p represent higher-frequency components of the initial error e_0, and so we see that higher-frequency components decay more rapidly.

Actually it is the middle range of frequencies, those nearest $p \approx m/2$, that decay most rapidly. The highest frequencies $p \approx m$ decay just as slowly as the lowest frequencies $p \approx 1$. The error e_0 shown in Figure 4.9 has a negligible component of these highest frequencies, however, and we are observing the rapid decay of the intermediate frequencies in this figure.

For this reason Jacobi is not the best method to use in the context of multigrid. A better choice is *underrelaxed Jacobi*, where

$$u_{k+1} = (1-\omega)u_k + \omega G u_k \tag{4.88}$$

with $\omega = 2/3$. The iteration matrix for this method is

$$G_\omega = (1-\omega)I + \omega G \tag{4.89}$$

with eigenvalues

$$\gamma_p = (1-\omega) + \omega \cos(p\pi h). \tag{4.90}$$

The choice $\omega = 2/3$ minimizes $\max_{m/2<p\le m} |\gamma_p|$, giving optimal smoothing of high frequencies. With this choice of ω, all frequencies above the midpoint $p = m/2$ have $|\gamma_p| \le 1/3$.

As a standalone iterative method this would be even worse than Jacobi, since low-frequency components of the error decay even more slowly (γ_1 is now $\frac{1}{3} + \frac{2}{3}\cos(\pi h) \approx 1 - \frac{1}{3}\pi^2 h^2$), but in the context of multigrid this does not concern us. What is important is that the upper half of the range frequencies are all damped by a factor of at least $1/3$ per iteration, giving a reduction by a factor of $(1/3)^3 \approx 0.037$ after only three iterations, for example.

4.6.2 The multigrid approach

We are finally ready to introduce the multigrid algorithm. If we use underrelaxed Jacobi, then after only three iterations the high-frequency components of the error have already decayed significantly, but convergence starts to slow down because of the lower-frequency components. But because the error is now much smoother, we can represent the remaining part of the problem on a coarser grid. The key idea in multigrid is to switch now to a coarser grid to estimate the remaining error. This has two advantages. Iterating on a coarser grid takes less work than iterating further on the original grid. This is nice but is a relatively minor advantage. Much more important, the convergence rate for some components of the error is greatly improved by transferring the error to a coarser grid.

For example, consider the eigencomponent $p = m/4$ that is not damped so much by underrelaxed Jacobi, $\gamma_{m/4} \approx 0.8$, and after three iterations on this grid this component of the error is damped only by a factor $(0.8)^3 = 0.512$. The value $p = m/4$ is not in the upper half of frequencies that can be represented on a grid with m points—it is right in the middle of the lower half.

However, if we transfer this function to a grid with only half as many points, it is suddenly at the halfway point of the frequencies we can represent on the coarser grid

($p \approx m_c/2$ now, where $m_c = (m-1)/2$ is the number of grid points on the coarser grid). Hence this same component of the error is damped by a factor of $(1/3)^3 \approx 0.037$ after only three iterations on this coarser grid. This is the essential feature of multigrid.

But how do we transfer the remaining part of the problem to a coarser grid? We don't try to solve the original problem on a coarser grid. Instead we solve an equation for the error. Suppose we have taken ν iterations on the original grid and now want to estimate the error $e_\nu = u_\nu - u^*$. This is related to the residual vector $r_\nu = f - Au_\nu$ by the linear system

$$Ae_\nu = -r_\nu. \tag{4.91}$$

If we can solve this equation for e_ν, then we can subtract e_ν from u_ν to obtain the desired solution u^*. The system (4.91) is the one we approximate on a coarsened grid. After taking a few iterations of Jacobi on the original problem, we know that e_ν is smoother than the solution u to the original problem, and so it makes sense that we can approximate this problem well on a coarser grid and then interpolate back to the original grid to obtain the desired approximation to e_ν. As noted above, iterating on the coarsened version of this problem leads to much more rapid decay of some components of the error.

The basic multigrid algorithm can be informally described as follows:

1. Take a fixed number of iterations (e.g., $\nu = 3$) of a simple iterative method (e.g., underrelaxed Jacobi or another choice of "smoother") on the original $m \times m$ system $Au = f$. This gives an approximation $u_\nu \in \mathbb{R}^m$.

2. Compute the residual $r_\nu = f - Au_\nu \in \mathbb{R}^m$.

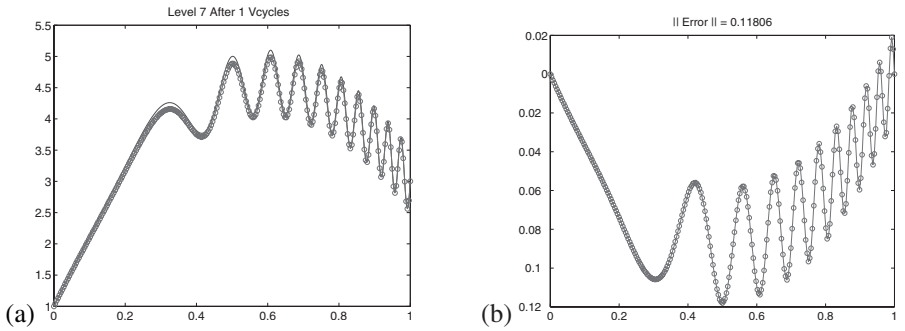
3. Coarsen the residual: approximate the grid function r_ν on a grid with $m_c = (m-1)/2$ points to obtain $\tilde{r} \in \mathbb{R}^{m_c}$.

4. Approximately solve the system $\tilde{A}\tilde{e} = -\tilde{r}$, where $\tilde{A}$ is the $m_c \times m_c$ version of A (the tridiagonal approximation to d^2/dx^2 on a grid with m_c points).

5. The vector $\tilde{e}$ approximates the error in u_ν but only at m_c points on the coarse grid. Interpolate this grid function back to the original grid with m points to obtain an approximation to e_ν. Subtract this from u_ν to get a better approximation to u^*.

6. Using this as a starting guess, take a few more iterations (e.g., $\nu = 3$) of a simple iterative method (e.g., underrelaxed Jacobi) on the original $m \times m$ system $Au = f$ to smooth out errors introduced by this interpolation procedure.

The real power of multigrid comes from recursively applying this idea. In step 4 of the algorithm above we must approximately solve the linear system $\tilde{A}\tilde{e} = -\tilde{r}$ of size m_c. As noted, some components of the error that decayed slowly when iterating on the original system will now decay quickly. However, if m_c is still quite large, then there will be other lower-frequency components of the error that still decay abysmally slowly on this coarsened grid. The key is to recurse. We only iterate a few times on this problem before resorting to a coarser grid with $(m_c - 1)/2$ grid points to speed up the solution to this problem. In other words, the entire algorithm given above is applied within step 4 to solve the linear system $\tilde{A}\tilde{e} = -\tilde{r}$. In a recursive programming language (such as MATLAB) this is not hard to implement and allows one to recurse back as far as possible. If $m + 1$ is a



(a) (b)

Figure 4.10. (a) *The solution $u(x)$ (solid line) and approximate solution (circles) obtained after one V-cycle of the multigrid algorithm with $\nu = 3$.* (b) *The error in this approximation. Note the change in scale from* Figure 4.9(b).

power of 2, then in principle one could recurse all the way back to a coarse grid with only a single grid point, but in practice the recursion is generally stopped once the problem is small enough that an iterative method converges very quickly or a direct method such as Gaussian elimination is easily applied.

Figure 4.10 shows the results obtained when the above algorithm is used starting with $m = 2^8 - 1 = 255$, using $\nu = 3$, and recursing down to a grid with three grid points, i.e., seven levels of grids. On each level we apply three iterations of underrelaxed Jacobi, do a coarse grid correction, and then apply three more iterations of under-relaxed Jacobi. Hence a total of six Jacobi iterations are used on each grid, and this is done on grids with $2^j - 1$ points for $j = 8,\ 7,\ 6,\ 5,\ 4,\ 3,\ 2$, since the coarse grid correction at each level requires doing this recursively at coarser levels. A total of 42 underrelaxed Jacobi iterations are performed, but most of these are on relatively coarse grids. The total number of grid values that must be updated in the course of these iterations is

$$6 \sum_{j=2}^{8} 2^j \approx 6 \cdot 2^9 = 3072,$$

roughly the same amount of work as 12 iterations on the original grid would require. But the improvement in accuracy is dramatic—compare Figure 4.10 to the results in Figure 4.9 obtained by simply iterating on the original grid with Jacobi.

More generally, suppose we start on a grid with $m + 1 = 2^J$ points and recurse all the way down, taking ν iterations of Jacobi both before and after the coarse grid correction on each level. Then the work is proportional to the total number of grid values updated, which is

$$2\nu \sum_{j=2}^{J} 2^j \approx 4\nu 2^J \approx 4\nu m = O(m). \tag{4.92}$$

Note that this is *linear* in the number of grid points m, although as m increases we are using an increasing number of coarser grids. The number of grids grows at the rate of $\log_2(m)$ but the work on each grid is half as much as the previous finer grid and, so the total work

is $O(m)$. This is the work required for one "V-cycle" of the multigrid algorithm, starting on the finest grid, recursing down to the coarsest grid and then back up as illustrated in Figure 4.11(a) and (b). Taking a single V-cycle often results in a significant reduction in the error, as illustrated in Figure 4.10, but more than one V-cycle might be required to obtain a sufficiently accurate solution. In fact, it can be shown that for this model problem $O(\log(m))$ V-cycles would be needed to reach a given level of error, so that the total work would grow like $O(m \log m)$.

We might also consider taking more that one iteration of the cycle on each of the coarser grids to solve the coarse grid problems within each cycle on the finest grid. Suppose, for example, that we take two cycles at each stage on each of the finer grids. This gives the W-cycle illustrated in Figure 4.11(c).

Even better results are typically obtained by using the "full multigrid" (FMG) algorithm, which consists of starting the process on the coarsest grid level instead of the finest grid. The original problem $u''(x) = f(x)$ is discretized and solved on the coarsest level first, using a direct solver or a few iterations of some iterative method. This approximation to $u(x)$ is then interpolated to the next finer grid to obtain a good initial guess for solving the problem on this grid. The two-level multigrid algorithm is used on this level to solve the problem. The result is then interpolated to the next-level grid to give good initial data there, and so on. By the time we get to the finest grid (our original grid, where we want the solution), we have a very good initial guess to start the multigrid process described above. This process is illustrated using the V-cycle in Figure 4.12.

This start-up phase of the computation adds relatively little work since it is mostly iterating on coarser grids. The total work for FMG with one V-cycle is only about 50% more than for the V-cycle alone. With this initialization process it often turns out that one V-cycle then suffices to obtain good accuracy, regardless of the number of grid points. In

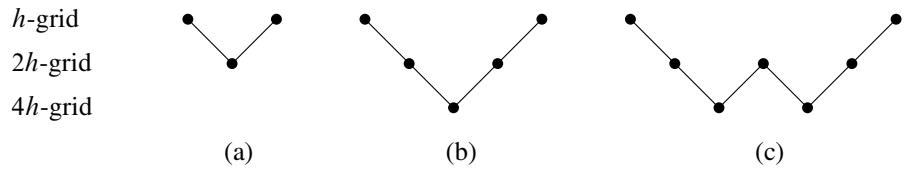


Figure 4.11. (a) *One V-cycle with two levels.* (b) *One V-cycle with three levels.* (c) *One W-cycle with three levels.*

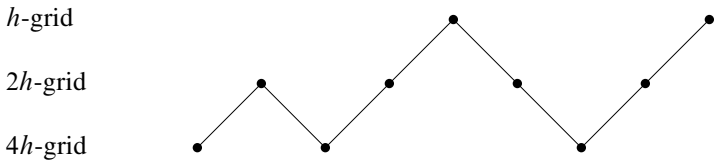


Figure 4.12. *FMG with one V-cycle on three levels.*

this case the total work is $O(m)$, which is optimal. For the example shown in Figure 4.10, switching to FMG gives an error of magnitude 6×10^{-3} after a single V-cycle.

Of course in one dimension simply solving the tridiagonal system requires only $O(m)$ work and is easier to implement, so this is not so impressive. But the same result carries over to more space dimensions. The FMG algorithm for the Poisson problem on an $m \times m$ grid in two dimensions requires $O(m^2)$ work, which is again optimal since there are this many unknowns to determine. Recall that fast Poisson solvers based on the FFT require $O(m^2 \log m)$ work, while the best possible direct method would require (m^3). Applying multigrid to more complicated problems can be more difficult, but optimal results of this sort have been achieved for a wide variety of problems.

The multigrid method described here is intimately linked to the finite difference grid being used and the natural manner in which a vector and matrix can be "coarsened" by discretizing the same differential operator on a coarser grid. However, the ideas of multigrid can also be applied to other sparse matrix problems arising from diverse applications where it may not be at all clear how to coarsen the problem. This more general approach is called *algebraic multigrid* (AMG); see, for example, [76], [86].