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

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

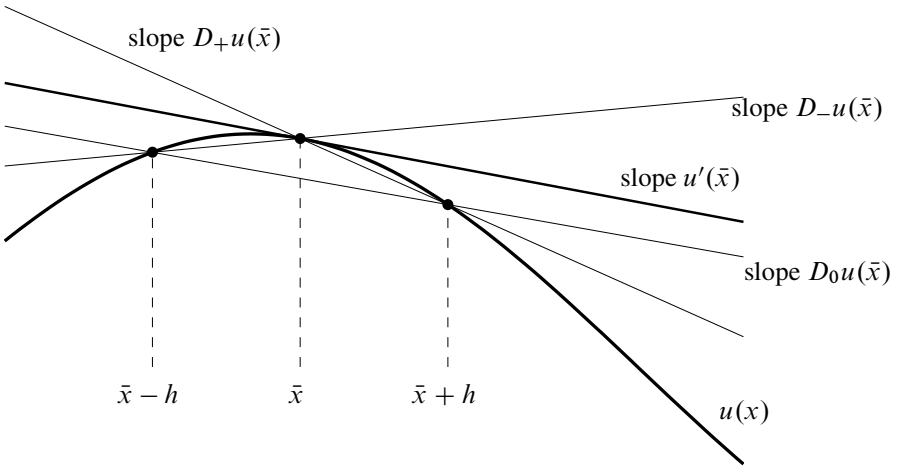

Figure 1.1. *Various approximations to $u'(\bar{x})$ interpreted as the slope of secant lines.*

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

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

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

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

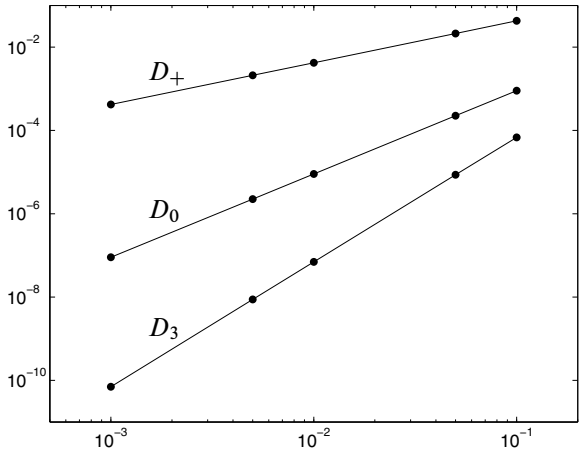

Figure 1.2. *The errors in* $Du(\bar{x})$ *from Table* 1.1 *plotted against* h *on a log-log scale.*

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^p_j for $j = 1, 2, \ldots, m$ given by

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

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

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

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

$$u^p_0 = u^p_{m+1} = 0,$$

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

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

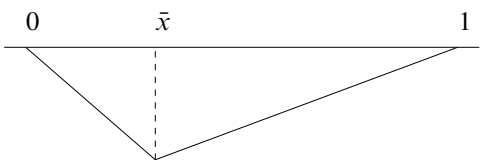

Figure 2.1. *The Green's function* $G(x; \bar{x})$ *from* (2.31).

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 $U_0,\ U_1,\ \ldots, U_m,\ U_{m+1}$:

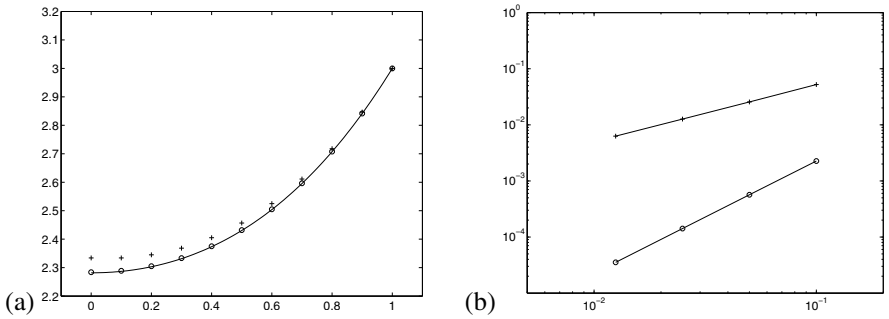

Figure 2.2. (a) *Sample solution to the steady-state heat equation with a Neumann boundary condition at the left boundary and Dirichlet at the right. The solid line is the true solution. The plus sign shows a solution on a grid with* 20 *points using* (2.53)*. The circle shows the solution on the same 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

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

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

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

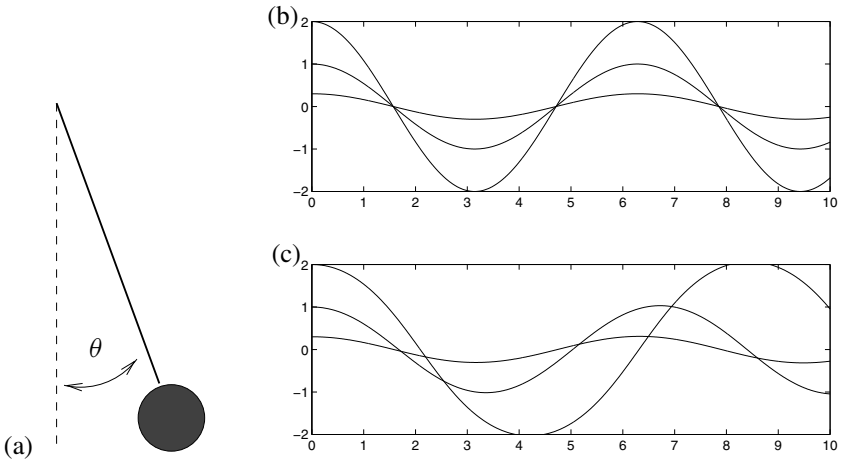

Figure 2.3. (a) *Pendulum.* (b) *Solutions to the linear equation* (2.76) *for various initial* θ *and zero initial velocity.* (c) *Solutions to the nonlinear equation* (2.75) *for various initial* θ *and zero initial velocity.*

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

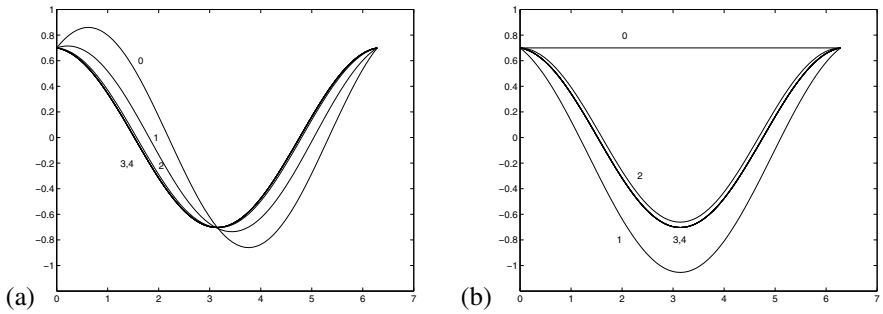

Figure 2.4. *Convergence of Newton iterates toward a solution of the pendulum problem. The iterates $\theta^{[k]}$ for $k = 1, 2, \ldots$ are denoted by the number k in the plots.* (a) *Starting from* $\theta_i^{[0]} = 0.7\cos(t_i) + 0.5\sin(t_i)$. (b) *Starting from* $\theta_i^{[0]} = 0.7$.

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 $T > 0$.

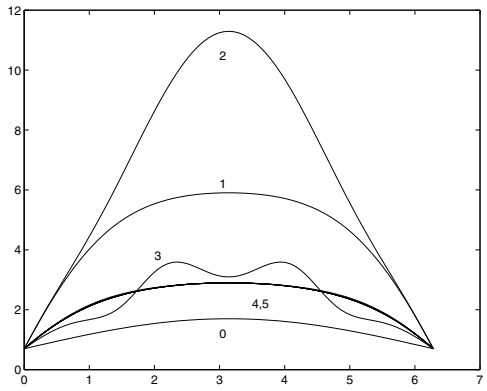

Figure 2.5. *Convergence of Newton iterates toward a different solution of the pendulum problem starting with initial guess* $\theta_i^{[0]} = 0.7 + \sin(t_i/2)$. *The iterates* k *for* $k = 1, 2, \ldots$ *are denoted by the number* k *in the plots.*

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.

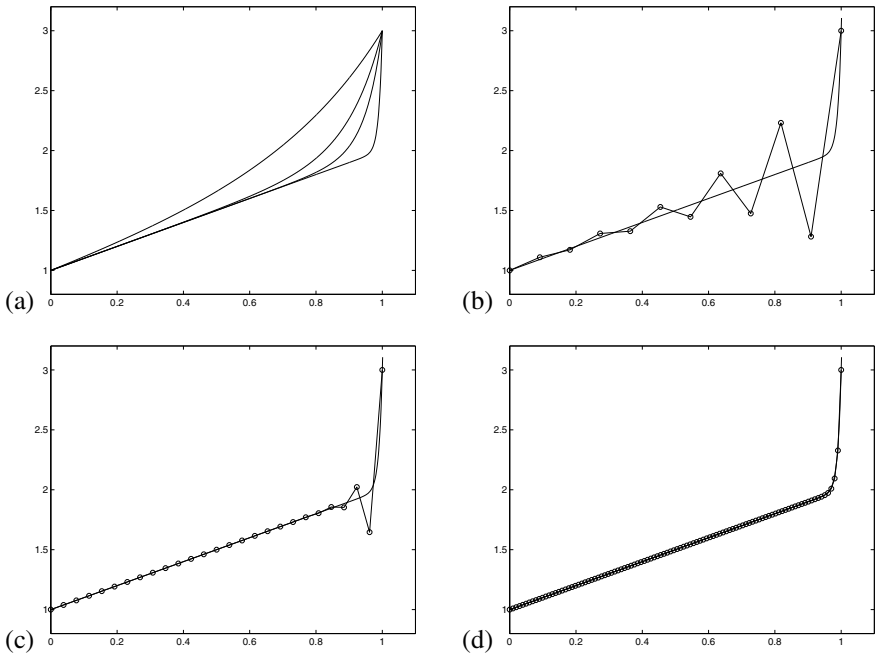

Figure 2.6. (a) *Solutions to the steady state advection-diffusion equation* (2.88) *for different values of* ϵ. *The four lines correspond to* $\epsilon = 0.3,\ 0.1,\ 0.05$, *and* 0.01 *from top to bottom.* (b) *Numerical solution with* $\epsilon = 0.01$ *and* $h = 1/10$. (c) $h = 1/25$. (d) $h = 1/100$.

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

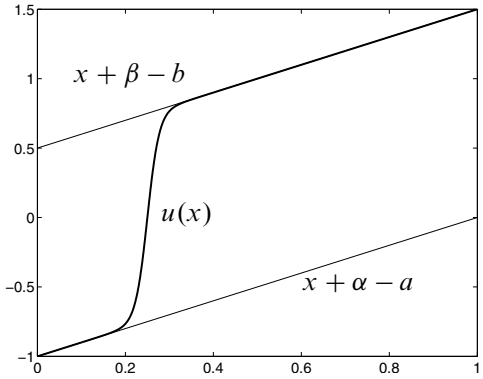

Figure 2.7. *Outer solutions and full solution to the singular perturbation problem with* $a = 0$, $b = 1$, $\alpha = -1$, *and* $\beta = 1.5$. *The solution has an interior layer centered about* $\bar{x} = 0.25$.

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.

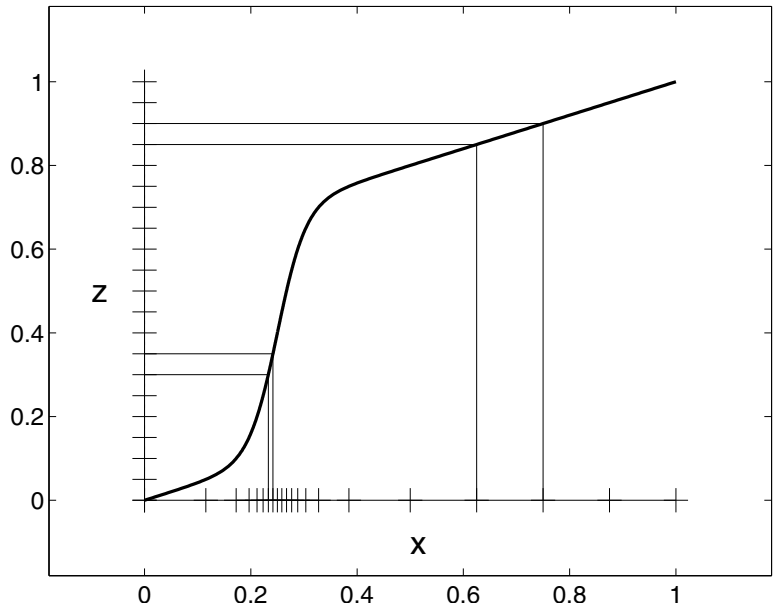

Figure 2.8. *Grid mapping from a uniform grid in* $0 \le z \le 1$ *(vertical axis) to the nonuniform grid in physical* x*-space shown on the horizontal axis. This particular mapping may be useful for solving the singular perturbation problem illustrated in Fig. 2.7.*

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

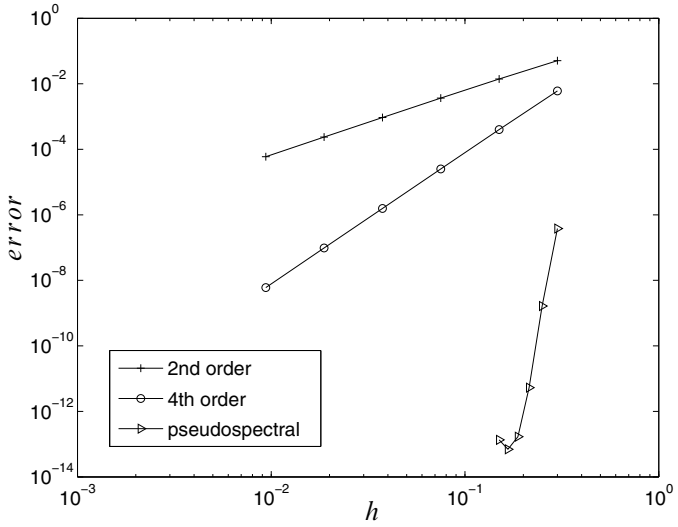

Figure 2.9. *Error as a function of h for two finite difference methods and the Chebyshev pseudospectral method on* (2.121).

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.

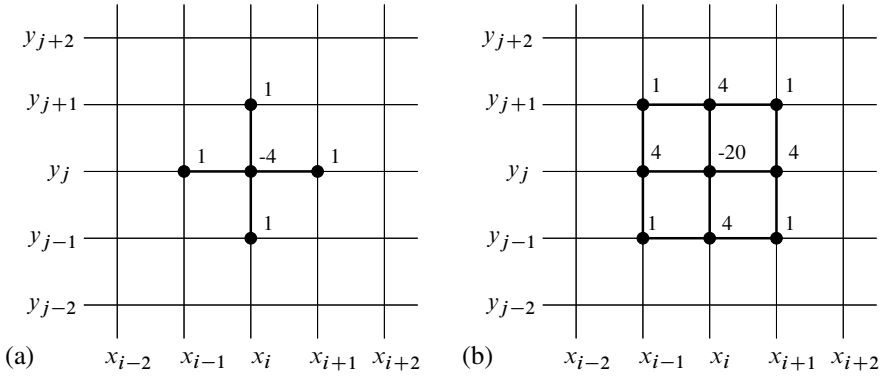

Figure 3.1. *Portion of the computational grid for a two-dimensional elliptic equation. (a) The 5-point stencil for the Laplacian about the point* (i, j) *is also indicated. (b) The 9-point stencil is indicated, which is discussed in Section 3.5.*

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.

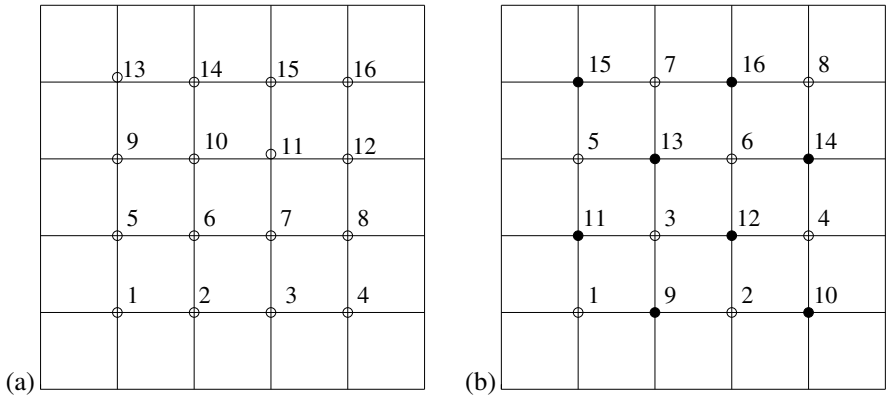

Figure 3.2. (a) *The natural rowwise order of unknowns and equations on a* 4×4 *grid.* (b) *The red-black ordering.*

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 $u'' = f$:

$$\begin{aligned} u_i^{\mathrm{GS}} &= \frac{1}{2}\left(u_{i-1}^{[k+1]} + u_{i+1}^{[k]} - h^2 f_i\right), \\ u_i^{[k+1]} &= u_i^{[k]} + \omega\left(u_i^{\mathrm{GS}} - u_i^{[k]}\right), \end{aligned} \tag{4.22}$$

where ω is some scalar parameter. If $\omega = 1$, then $u_i^{[k+1]} = u_i^{\mathrm{GS}}$ 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

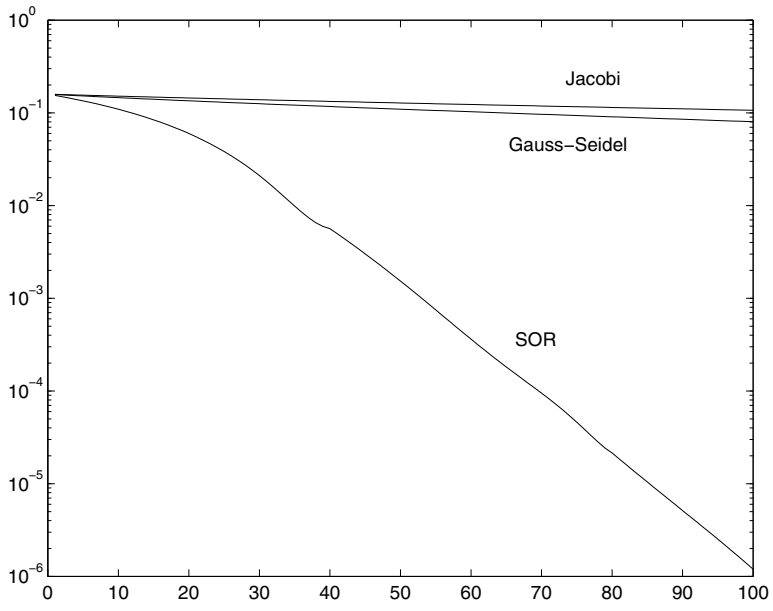

Figure 4.1. *Errors versus k for three methods.*

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

$$\phi(u) = \phi(u_1, u_2) = \frac{1}{2}(a_{11}u_1^2 + 2a_{12}u_1u_2 + a_{22}u_2^2) - u_1 f_1 - u_2 f_2.$$

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

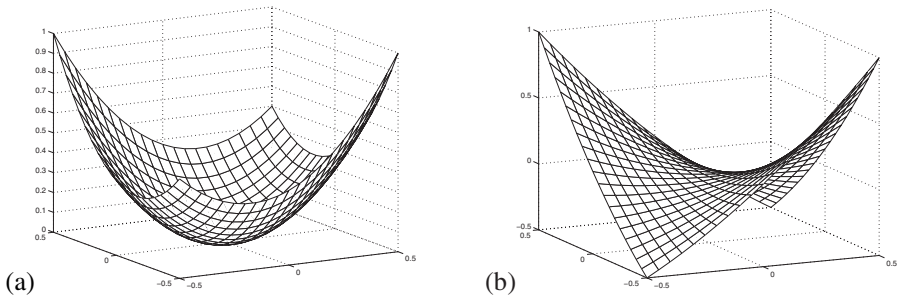

Figure 4.2. (a) *The function $\phi(u)$ for $m = 2$ in a case where A is symmetric and positive definite.* (b) *The function $\phi(u)$ for $m = 2$ in a case where A is symmetric but indefinite.*

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 $k = 1, 2, \ldots$
    $r_{k-1} = f - Au_{k-1}$
    if $\|r_{k-1}\|$ is less than some tolerance then stop
    $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(r_{k-1}^T Ar_{k-1})$
    $u_k = u_{k-1} + \alpha_{k-1} r_{k-1}$
    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
$r_0 = f - Au_0$
for $k = 1, 2, \ldots$
    $w_{k-1} = Ar_{k-1}$
    $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(r_{k-1}^T w_{k-1})$
    $u_k = u_{k-1} + \alpha_{k-1} r_{k-1}$
    $r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
    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

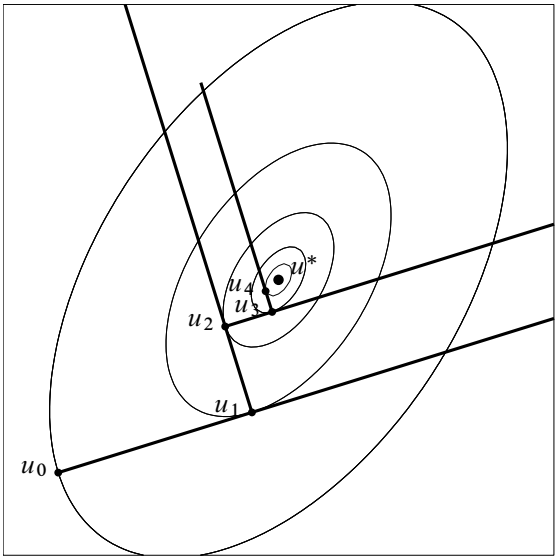

Figure 4.3. *Several iterates of the method of steepest descent in the case $m = 2$. The concentric ellipses are level sets of $\phi(u)$.*

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

$$A(v_j - u^*) = \lambda_j(v_j - u^*) \tag{4.35}$$

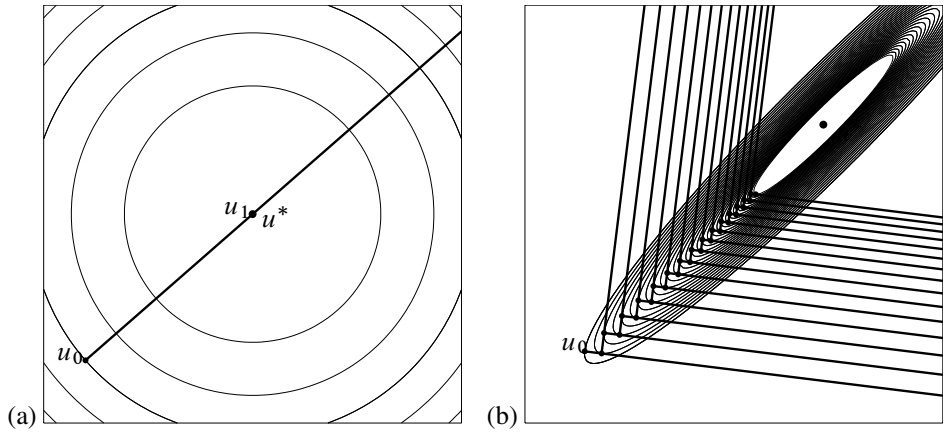

Figure 4.4. (a) *If A is a scalar multiple of the identity, then the level sets of $\phi(u)$ are circular and steepest descent converges in one iteration from any initial guess u_0.* (b) *If the level sets of $\phi(u)$ are far from circular, then steepest descent may converge slowly.*

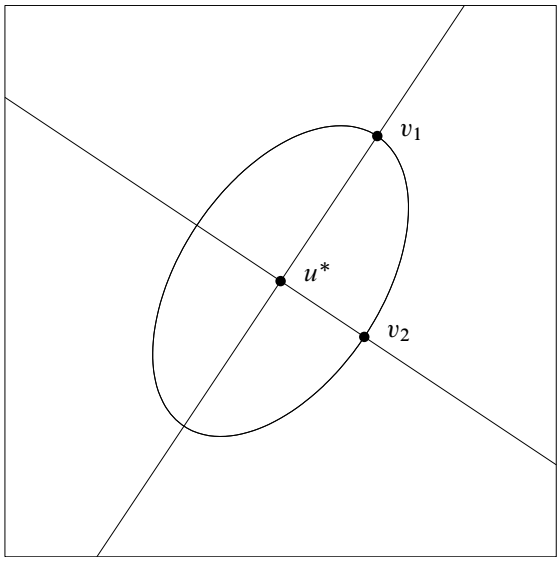

Figure 4.5. *The major and minor axes of the elliptical level set of $\phi(u)$ point in the directions of the eigenvectors of A.*

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

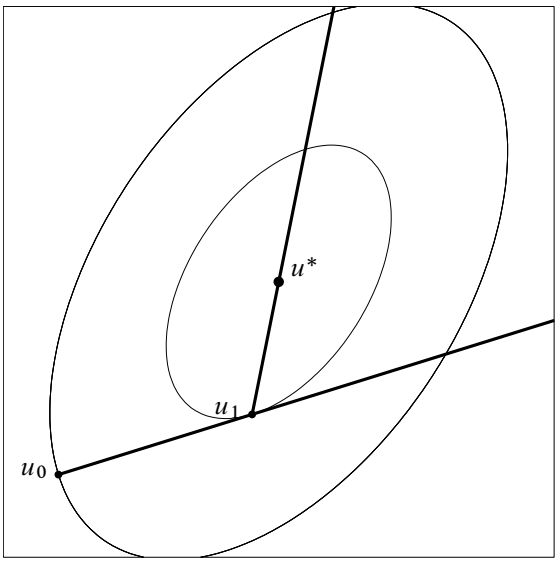

Figure 4.6. *The CG algorithm converges in two iterations from any initial guess u_0 in the case $m = 2$. The two search directions used are A-conjugate.*

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

$$\begin{aligned}
\max_{1\le j\le m} |\tilde{P}_1(\lambda_j)| = \tilde{P}_1(\lambda_1) &= 1 - \frac{2\lambda_1}{\lambda_m + \lambda_1} = \frac{\lambda_m/\lambda_1 - 1}{\lambda_m/\lambda_1 + 1} \\
&= \frac{\kappa - 1}{\kappa + 1},
\end{aligned} \tag{4.54}$$

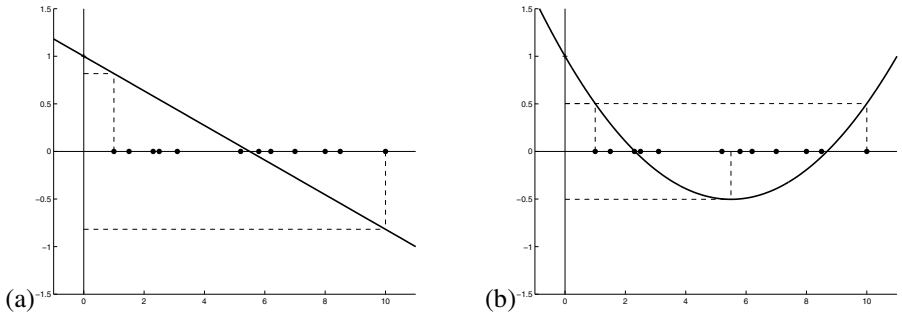

Figure 4.7. (a) *The polynomial* $\tilde{P}_1(x)$ *based on a sample set of eigenvalues marked by dots on the* x*-axis.* (b) *The polynomial* $\tilde{P}_2(x)$ *for the same set of eigenvalues.*

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 $k = 1, 2, \ldots$
    $v = Aq_k$
    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.

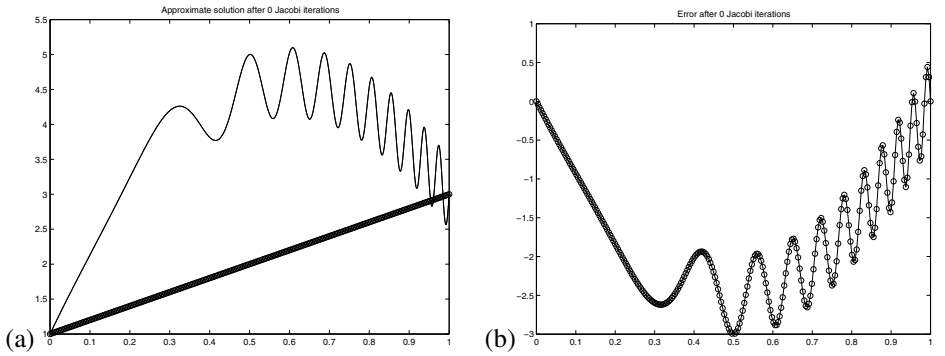

Figure 4.8. (a) *The solution $u(x)$ (solid line) and initial guess u_0 (circles).* (b) *The error e_0 in the initial guess.*

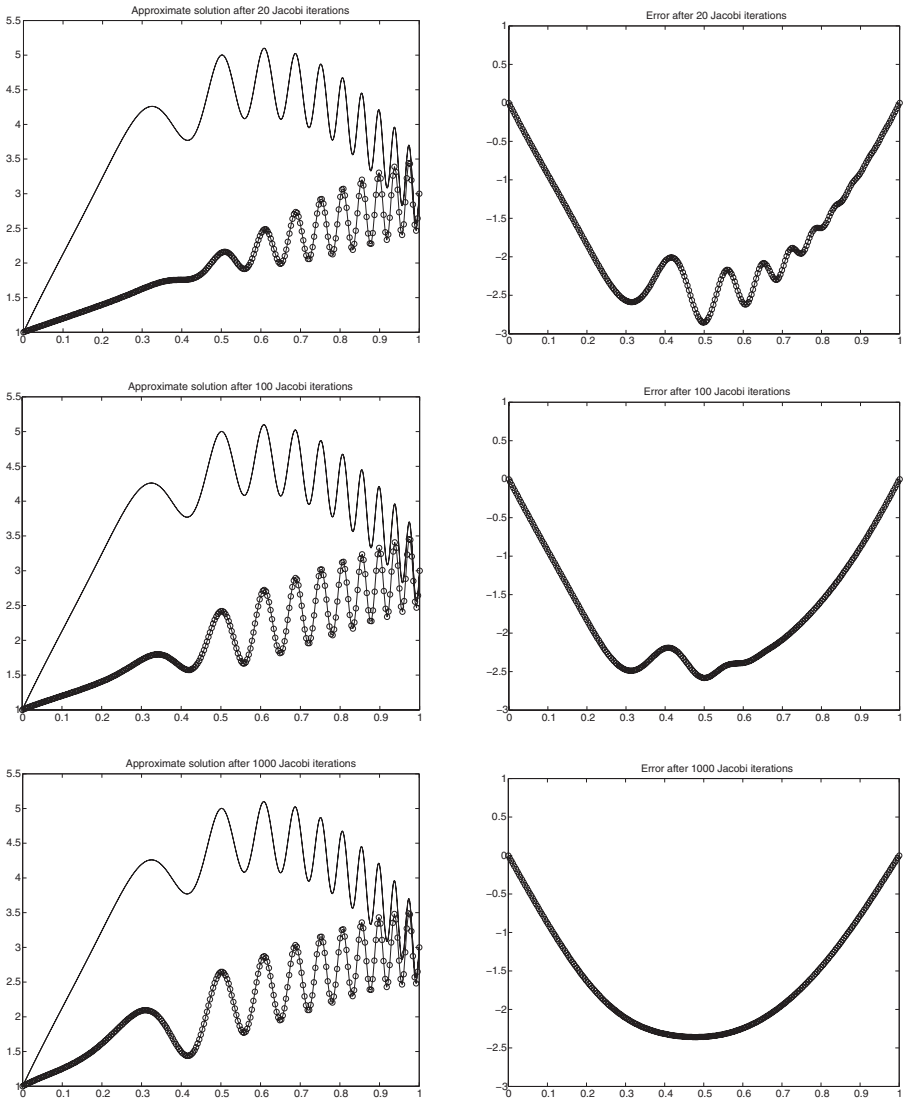

Figure 4.9. *On the left: The solution $u(x)$ (solid line) and Jacobi iterate u_k after k iterations. On the right: The error e_k, shown for $k = 20$ (top), $k = 100$ (middle), and $k = 1000$ (bottom).*

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

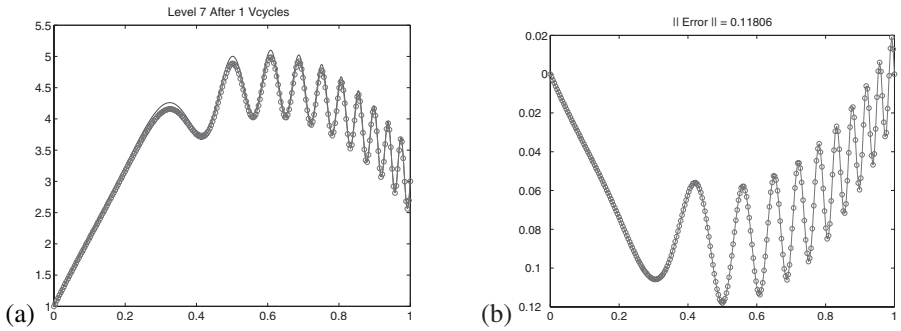

Figure 4.10. (a) *The solution $u(x)$ (solid line) and approximate solution (circles) obtained after one V-cycle of the multigrid algorithm with $\nu = 3$.* (b) *The error in this approximation. Note the change in scale from* Figure 4.9(b).

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

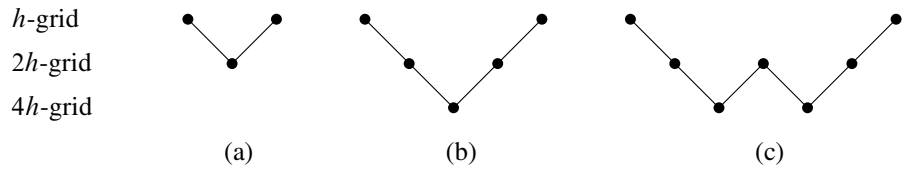

Figure 4.11. (a) *One V-cycle with two levels.* (b) *One V-cycle with three levels.* (c) *One W-cycle with three levels.*

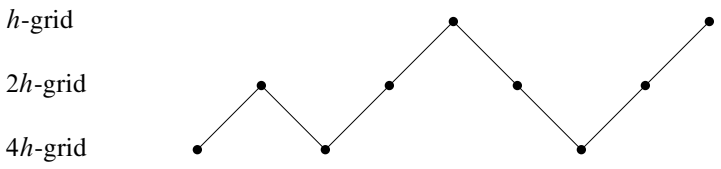

Figure 4.12. *FMG with one V-cycle on three levels.*

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].

Part II

Initial Value Problems

Chapter 5

The Initial Value Problem for Ordinary Differential Equations

In this chapter we begin a study of time-dependent differential equations, beginning with the initial value problem (IVP) for a time-dependent ordinary differential equation (ODE). Standard introductory texts are Ascher and Petzold [5], Lambert [59], [60], and Gear [33]. Henrici [45] gives the details on some theoretical issues, although stiff equations are not discussed. Butcher [12] and Hairer, Nørsett, and Wanner [43, 44] provide more recent surveys of the field.

The IVP takes the form

$$u'(t) = f(u(t), t) \quad \text{for } t > t_0 \tag{5.1}$$

with some initial data

$$u(t_0) = \eta. \tag{5.2}$$

We will often assume $t_0 = 0$ for simplicity.

In general, (5.1) may represent a system of ODEs, i.e., u may be a vector with s components $u_1, \ \ldots, \ u_s$, and then $f(u, t)$ also represents a vector with components $f_1(u, t), \ \ldots, \ f_s(u, t)$, each of which can be a nonlinear function of all the components of u.

We will consider only the first order equation (5.1), but in fact this is more general than it appears since we can reduce higher order equations to a system of first order equations.

Example 5.1. Consider the IVP for the ODE,

$$v'''(t) = v'(t)v(t) - 2t(v''(t))^2 \quad \text{for } t > 0.$$

This third order equation requires three initial conditions, typically specified as

$$\begin{aligned} v(0) &= \eta_1, \\ v'(0) &= \eta_2, \\ v''(0) &= \eta_3. \end{aligned} \tag{5.3}$$

We can rewrite this as a system of the form (5.1), (5.2) by introducing the variables

$$\begin{aligned} u_1(t) &= v(t), \\ u_2(t) &= v'(t), \\ u_3(t) &= v''(t). \end{aligned}$$

Then the equations take the form

$$\begin{aligned} u_1'(t) &= u_2(t), \\ u_2'(t) &= u_3(t), \\ u_3'(t) &= u_1(t)u_2(t) - 2tu_3^2(t), \end{aligned}$$

which defines the vector function $f(u,t)$. The initial condition is simply (5.2), where the three components of η come from (5.3). More generally, any single equation of order m can be reduced to m first order equations by defining $u_j(t) = v^{(j-1)}(t)$, and an mth order system of s equations can be reduced to a system of ms first order equations.

See Section D.3.1 for an example of how this procedure can be used to determine the general solution of an rth order linear differential equation.

It is also sometimes useful to note that any explicit dependence of f on t can be eliminated by introducing a new variable that is simply equal to t. In the above example we could define

$$u_4(t) = t$$

so that

$$u_4'(t) = 1 \quad \text{and} \quad u_4(t_0) = t_0.$$

The system then takes the form

$$u'(t) = f(u(t)) \tag{5.4}$$

with

$$f(u) = \begin{bmatrix} u_2 \\ u_3 \\ u_1u_2 - 2u_4u_3^2 \\ 1 \end{bmatrix} \quad \text{and} \quad u(t_0) = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \\ t_0 \end{bmatrix}.$$

The equation (5.4) is said to be *autonomous* since it does not depend explicitly on time. It is often convenient to assume f is of this form since it simplifies notation.

5.1 Linear ordinary differential equations

The system of ODEs (5.1) is *linear* if

$$f(u,t) = A(t)u + g(t), \tag{5.5}$$

where $A(t) \in \mathbb{R}^{s\times s}$ and $g(t) \in \mathbb{R}^s$. An important special case is the *constant coefficient linear system*

$$u'(t) = Au(t) + g(t), \tag{5.6}$$

where $A \in \mathbb{R}^{s\times s}$ is a constant matrix. If $g(t) \equiv 0$, then the equation is *homogeneous*. The solution to the homogeneous system $u' = Au$ with data (5.2) is

$$u(t) = e^{A(t-t_0)}\eta, \tag{5.7}$$

where the matrix exponential is defined as in Appendix D. In the scalar case we often use λ in place of A.

5.1.1 Duhamel's principle

If $g(t)$ is not identically zero, then the solution to the constant coefficient system (5.6) can be written as

$$u(t) = e^{A(t-t_0)}\eta + \int_{t_0}^{t} e^{A(t-\tau)}g(\tau)\,d\tau. \tag{5.8}$$

This is known as *Duhamel's principle*. The matrix $e^{A(t-\tau)}$ is the solution operator for the homogeneous problem; it maps data at time τ to the solution at time t when solving the homogeneous equation. Duhamel's principle states that the inhomogeneous term $g(\tau)$ at any instant τ has an effect on the solution at time t given by $e^{A(t-\tau)}g(\tau)$. Note that this is very similar to the idea of a Green's function for the boundary value problem (BVP).

As a special case, if $A = 0$, then the ODE is simply

$$u'(t) = g(t) \tag{5.9}$$

and of course the solution (5.8) reduces to the integral of g:

$$u(t) = \eta + \int_{t_0}^{t} g(\tau)\,d\tau. \tag{5.10}$$

As another special case, suppose A is constant and so is $g(t) \equiv g \in \mathbb{R}^s$. Then (5.8) reduces to

$$u(t) = e^{A(t-t_0)}\eta + \left(\int_{t_0}^{t} e^{A(t-\tau)}\,d\tau\right) g. \tag{5.11}$$

This integral can be computed, e.g., by expressing $e^{A(t-\tau)}$ as a Taylor series as in (D.31) and then integrating term by term. This gives

$$\int_{t_0}^{t} e^{A(t-\tau)}\,d\tau = A^{-1}\left(e^{A(t-t_0)} - I\right) \tag{5.12}$$

and so

$$u(t) = e^{A(t-t_0)}\eta + A^{-1}\left(e^{A(t-t_0)} - I\right) g. \tag{5.13}$$

This may be familiar in the scalar case and holds also for constant coefficient systems (provided A is nonsingular). This form of the solution is used explicitly in exponential time differencing methods; see Section 11.6.

5.2 Lipschitz continuity

In the last section we considered linear ODEs, for which there is always a unique solution. In most applications, however, we are concerned with nonlinear problems for which there is usually no explicit formula for the solution. The standard theory for the existence of a solution to the initial value problem

$$u'(t) = f(u,t), \quad u(0) = \eta \tag{5.14}$$

is discussed in many texts, e.g., [15]. To guarantee that there is a unique solution it is necessary to require a certain amount of smoothness in the function $f(u,t)$ of (5.14). We say that the function $f(u,t)$ is *Lipschitz continuous* in u over some domain

$$\mathcal{D} = \{(u,t) : |u-\eta| \le a \ t_0 \le t \le t_1\}$$

if there exists some constant $L \ge 0$ so that

$$|f(u,t) - f(u^*,t)| \le L|u-u^*| \tag{5.15}$$

for all (u,t) and (u^*,t) in $\mathcal{D}$. This is slightly stronger than mere continuity, which only requires that $|f(u,t) - f(u^*,t)| \to 0$ as $u \to u^*$. Lipschitz continuity requires that $|f(u,t) - f(u^*,t)| = O(|u-u^*|)$ as $u \to u^*$.

If $f(u,t)$ is differentiable with respect to u in $\mathcal{D}$ and this derivative $f_u = \partial f/\partial u$ is bounded then we can take

$$L = \max_{(u,t)\in\mathcal{D}} |f_u(u,t)|,$$

since

$$f(u,t) = f(u^*,t) + f_u(v,t)(u-u^*)$$

for some value v between u and u^*.

Example 5.2. For the linear problem $u'(t) = \lambda u(t) + g(t)$, $f'(u) \equiv \lambda$ and we can take $L = |\lambda|$. This problem of course has a unique solution for any initial data η given by (5.8) with $A = \lambda$.

In particular, if $\lambda = 0$ then $L = 0$. In this case $f(u,t) = g(t)$ is independent of u. The solution is then obtained by simply integrating the function $g(t)$, as in (5.10).

5.2.1 Existence and uniqueness of solutions

The basic existence and uniqueness theorem states that if f is Lipschitz continuous over some region $\mathcal{D}$ then there is a unique solution to the initial value problem (5.14) at least up to time $T^* = \min(t_1, t_0 + a/S)$, where

$$S = \max_{(u,t)\in\mathcal{D}} |f(u,t)|.$$

Note that this is the maximum modulus of the slope that the solution $u(t)$ can attain in this time interval, so that up to time $t_0 + a/S$ we know that $u(t)$ remains in the domain $\mathcal{D}$ where (5.15) holds.

Example 5.3. Consider the initial value problem

$$u'(t) = (u(t))^2, \qquad u(0) = \eta > 0.$$

The function $f(u) = u^2$ is independent of t and is Lipschitz continuous in u over any finite interval $|u - \eta| \le a$ with $L = 2(\eta + a)$, and the maximum slope over this interval is $S = (\eta + a)^2$. The theorem guarantees that a unique solution exists at least up to time $a/(\eta+a)^2$. Since a is arbitrary, we can choose a to maximize this expression, which yields $a = \eta$ and so there is a solution at least up to time $1/4\eta$.

In fact this problem can be solved analytically and the unique solution is

$$u(t) = \frac{1}{\eta^{-1} - t}.$$

Note that $u(t) \to \infty$ as $t \to 1/\eta$. There is no solution beyond time $1/\eta$.

If the function f is not Lipschitz continuous in any neighborhood of some point then the initial value problem may fail to have a unique solution over any time interval if this initial value is imposed.

Example 5.4. Consider the initial value problem

$$u'(t) = \sqrt{u(t)}$$

with initial condition

$$u(0) = 0.$$

The function $f(u) = \sqrt{u}$ is not Lipschitz continuous near $u = 0$ since $f'(u) = 1/(2\sqrt{u}) \to \infty$ as $u \to 0$. We cannot find a constant L so that the bound (5.15) holds for all u and u^* near 0.

As a result, this initial value problem does not have a unique solution. In fact it has two distinct solutions:

$$u(t) \equiv 0$$

and

$$u(t) = \frac{1}{4}t^2.$$

5.2.2 Systems of equations

For systems of $s > 1$ ordinary differential equations, $u(t) \in \mathbb{R}^s$ and $f(u, t)$ is a function mapping $\mathbb{R}^s \times \mathbb{R} \to \mathbb{R}^s$. We say the function f is Lipschitz continuous in u in some norm $\|\cdot\|$ if there is a constant L such that

$$\|f(u, t) - f(u^*, t)\| \le L\|u - u^*\| \tag{5.16}$$

for all (u, t) and $(u^*, , t)$ in some domain $\mathcal{D} = \{(u, t) : \|u - \eta\| \le a,\ t_0 \le t \le t_1\}$. By the equivalence of finite-dimensional norms (Appendix A), if f is Lipschitz continuous in one norm then it is Lipschitz continuous in any other norm, although the Lipschitz constant may depend on the norm chosen.

The theorems on existence and uniqueness carry over to systems of equations.

Example 5.5. Consider the pendulum problem from Section 2.16,

$$\theta''(t) = -\sin(\theta(t)),$$

which can be rewritten as a first order system of two equations by introducing $v(t) = \theta'(t)$:

$$u = \begin{bmatrix} \theta \\ v \end{bmatrix}, \qquad \frac{d}{dt}\begin{bmatrix} \theta \\ v \end{bmatrix} = \begin{bmatrix} v \\ -\sin(\theta) \end{bmatrix}.$$

Consider the max-norm. We have

$$\|u - u^*\|_\infty = \max(|\theta - \theta^*|,\ |v - v^*|)$$

and

$$\|f(u) - f(u^*)\|_\infty = \max(|v - v^*|,\ |\sin(\theta) - \sin(\theta^*)|).$$

To bound $\|f(u) - f(u^*)\|_\infty$, first note that $|v - v^*| \le \|u - u^*\|_\infty$. We also have

$$|\sin(\theta) - \sin(\theta^*)| \le |\theta - \theta^*| \le \|u - u^*\|_\infty$$

since the derivative of $\sin(\theta)$ is bounded by 1. So we have Lipschitz continuity with $L = 1$:

$$\|f(u) - f(u^*)\|_\infty \le \|u - u^*\|_\infty.$$

5.2.3 Significance of the Lipschitz constant

The Lipschitz constant measures how much $f(u, t)$ changes if we perturb u (at some fixed time t). Since $f(u, t) = u'(t)$, the slope of the line tangent to the solution curve through the value u, this indicates how the slope of the solution curve will vary if we perturb u. The significance of this is best seen through some examples.

Example 5.6. Consider the trivial equation $u'(t) = g(t)$, which has Lipschitz constant $L = 0$ and solutions given by (5.10). Several solution curves are sketched in Figure 5.1. Note that all these curves are "parallel"; they are simply shifted depending on the initial data. Tangent lines to the curves at any particular time are all parallel since $f(u, t) = g(t)$ is independent of u.

Example 5.7. Consider $u'(t) = \lambda u(t)$ with λ constant and $L = |\lambda|$. Then $u(t) = u(0)\exp(\lambda t)$. Two situations are shown in Figure 5.2 for negative and positive values of λ. Here the slope of the solution curve does vary depending on u. The variation in the slope with u (at fixed t) gives an indication of how rapidly the solution curves are converging toward one another (in the case $\lambda < 0$) or diverging away from one another (in the case $\lambda > 0$). If the magnitude of λ were increased, the convergence or divergence would clearly be more rapid.

The size of the Lipschitz constant is significant if we intend to solve the problem numerically since our numerical approximation will almost certainly produce a value U^n at time t_n that is not exactly equal to the true value $u(t_n)$. Hence we are on a different solution curve than the true solution. The best we can hope for in the future is that we stay close to the solution curve that we are now on. The size of the Lipschitz constant gives an indication of whether solution curves that start close together can be expected to stay close together or might diverge rapidly.

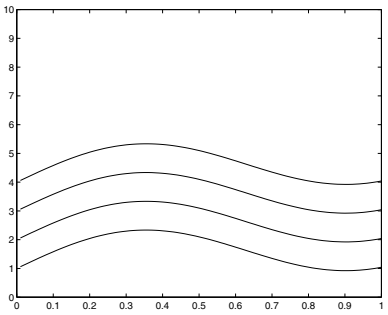

Figure 5.1. *Solution curves for Example* 5.6*, where* $L = 0$.

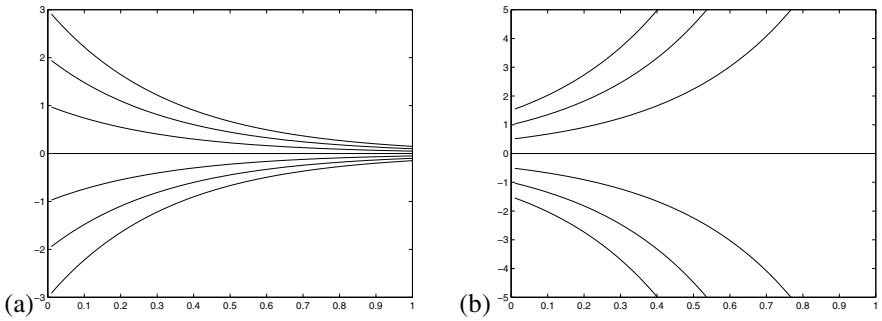

Figure 5.2. *Solution curves for Example* 5.7 *with* (a) $\lambda = -3$ *and* (b) $\lambda = 3$.

5.2.4 Limitations

Actually, the Lipschitz constant is not the perfect tool for this purpose, since it does not distinguish between rapid divergence and rapid convergence of solution curves. In both Figure 5.2(a) and Figure 5.2(b) the Lipschitz constant has the same value $L = |\lambda| = 3$. But we would expect that rapidly convergent solution curves as in Figure 5.2(a) should be easier to handle numerically than rapidly divergent ones. If we make an error at some stage, then the effect of this error should decay at later times rather than growing. To some extent this is true and as a result error bounds based on the Lipschitz constant may be orders of magnitude too large in this situation.

However, rapidly converging solution curves can also give serious numerical difficulties, which one might not expect at first glance. This is discussed in detail in Chapter 8, which covers stiff equations.

One should also keep in mind that a small value of the Lipschitz constant does not necessarily mean that two solution curves starting close together will stay close together forever.

Example 5.8. Consider two solutions to the pendulum problem from Example 5.5, one with initial data

$$\theta_1(0) = \pi - \epsilon, \qquad v_1(0) = 0,$$

and the other with

$$\theta_2(0) = \pi + \epsilon, \qquad v_2(0) = 0.$$

The Lipschitz constant is 1 and the data differ by 2ϵ, which can be arbitrarily small, and yet the solutions eventually diverge dramatically, as Solution 1 falls toward $\theta = 0$, while in Solution 2 the pendulum falls the other way, toward $\theta = 2\pi$.

In this case the IVP is very ill conditioned: small changes in the data can lead to order 1 changes in the solution. As always in numerical analysis, the solution of ill-conditioned problems can be very hard to compute accurately.

5.3 Some basic numerical methods

We begin by listing a few standard approaches to discretizing (5.1). Note that the IVP differs from the BVP considered before in that we are given all the data at the initial time $t_0 = 0$ and from this we should be able to march forward in time, computing approximations at successive times $t_1, t_2, \ldots$. We will use k to denote the time step, so $t_n = nk$ for $n \geq 0$. It is convenient to use the symbol k, which is different from the spatial grid size h, since we will soon study PDEs which involve both spatial and temporal discretizations. Often the symbols Δt and Δx are used.

We are given initial data

$$U^0 = \eta \tag{5.17}$$

and want to compute approximations $U^1,\ U^2,\ \ldots$ satisfying

$$U^n \approx u(t_n).$$

We will use superscripts to denote the time step index, again anticipating the notation of PDEs where we will use subscripts for spatial indices.

The simplest method is *Euler's method* (also called *forward Euler*), based on replacing $u'(t_n)$ with $D_+U^n = (U^{n+1} - U^n)/k$ from (1.1). This gives the method

$$\frac{U^{n+1} - U^n}{k} = f(U^n), \qquad n = 0,\ 1,\ \ldots. \tag{5.18}$$

Rather than viewing this as a system of simultaneous equations as we did for the BVP, it is possible to solve this explicitly for U^{n+1} in terms of U^n:

$$U^{n+1} = U^n + kf(U^n). \tag{5.19}$$

From the initial data U^0 we can compute U^1, then U^2, and so on. This is called a *time-marching method*.

The *backward Euler* method is similar but is based on replacing $u'(t_{n+1})$ with D_-U^{n+1}:

$$\frac{U^{n+1} - U^n}{k} = f(U^{n+1}) \tag{5.20}$$

or

$$U^{n+1} = U^n + kf(U^{n+1}). \tag{5.21}$$

Again we can march forward in time since computing U^{n+1} requires only that we know the previous value U^n. In the backward Euler method, however, (5.21) is an equation that

must be solved for U^{n+1}, and in general $f(u)$ is a nonlinear function. We can view this as looking for a zero of the function

$$g(u) = u - kf(u) - U^n,$$

which can be approximated using some iterative method such as *Newton's method*.

Because the backward Euler method gives an equation that must be solved for U^{n+1}, it is called an *implicit* method, whereas the forward Euler method (5.19) is an *explicit* method.

Another implicit method is the *trapezoidal method*, obtained by averaging the two Euler methods:

$$\frac{U^{n+1} - U^n}{k} = \frac{1}{2}(f(U^n) + f(U^{n+1})). \tag{5.22}$$

As one might expect, this symmetric approximation is second order accurate, whereas the Euler methods are only first order accurate.

The above methods are all *one-step methods*, meaning that U^{n+1} is determined from U^n alone and previous values of U are not needed. One way to get higher order accuracy is to use a *multistep* method that involves other previous values. For example, using the approximation

$$\frac{u(t+k) - u(t-k)}{2k} = u'(t) + \frac{1}{6}k^2u'''(t) + O(k^3)$$

yields the *midpoint method* (also called the *leapfrog method*),

$$\frac{U^{n+1} - U^{n-1}}{2k} = f(U^n) \tag{5.23}$$

or

$$U^{n+1} = U^{n-1} + 2kf(U^n), \tag{5.24}$$

which is a second order accurate explicit 2-step method. The approximation D_2u from (1.11), rewritten in the form

$$\frac{3u(t+k) - 4u(t) + u(t-k)}{2k} = u'(t+k) + \frac{1}{12}k^2u'''(t+k) + \cdots,$$

yields a second order implicit 2-step method

$$\frac{3U^{n+1} - 4U^n + U^{n-1}}{2k} = f(U^{n+1}). \tag{5.25}$$

This is one of the backward differentiation formula (*BDF*) methods that will be discussed further in Chapter 8.

5.4 Truncation errors

The truncation error for these methods is defined in the same way as in Chapter 2. We write the difference equation in the form that directly models the derivatives (e.g., in the form

(5.23) rather than (5.24)) and then insert the true solution to the ODE into the difference equation. We then use Taylor series expansion and cancel out common terms.

Example 5.9. The local truncation error (LTE) of the midpoint method (5.23) is defined by

$$\begin{aligned}\tau^n &= \frac{u(t_{n+1}) - u(t_{n-1})}{2k} - f(u(t_n)) \\ &= \left[u'(t_n) + \frac{1}{6}k^2u'''(t_n) + O(k^4)\right] - u'(t_n) \\ &= \frac{1}{6}k^2u'''(t_n) + O(k^4).\end{aligned}$$

Note that since $u(t)$ is the true solution of the ODE, $u'(t_n) = f(u(t_n))$. The $O(k^3)$ term drops out by symmetry. The truncation error is $O(k^2)$ and so we say the method is *second order accurate*, although it is not yet clear that the global error will have this behavior. As always, we need some form of *stability* to guarantee that the global error will exhibit the same rate of convergence as the local truncation error. This will be discussed below.

5.5 One-step errors

In much of the literature concerning numerical methods for ODEs, a slightly different definition of the local truncation error is used that is based on the form (5.24), for example, rather than (5.23). Denoting this value by $\mathcal{L}^n$, we have

$$\begin{aligned}\mathcal{L}^n &= u(t_{n+1}) - u(t_{n-1}) - 2kf(u(t_n)) \\ &= \frac{1}{3}k^3u'''(t_n) + O(k^5).\end{aligned} \tag{5.26}$$

Since $\mathcal{L}^n = 2k\tau^n$, this local error is $O(k^3)$ rather than $O(k^2)$, but of course the global error remains the same and will be $O(k^2)$. Using this alternative definition, many standard results in ODE theory say that a pth order accurate method should have an LTE that is $O(k^{p+1})$. With the notation we are using, a pth order accurate method has an LTE that is $O(k^p)$. The notation used here is consistent with the standard practice for PDEs and leads to a more coherent theory, but one should be aware of this possible source of confusion.

In this book $\mathcal{L}^n$ will be called the *one-step error*, since this can be viewed as the error that would be introduced in one time step if the past values U^n, U^{n-1}, ... were all taken to be the exact values from $u(t)$. For example, in the midpoint method (5.24) we suppose that

$$U^n = u(t_n) \quad \text{and} \quad U^{n-1} = u(t_{n-1})$$

and we now use these values to compute U^{n+1}, an approximation to $u(t_{n+1})$:

$$\begin{aligned}U^{n+1} &= u(t_{n-1}) + 2kf(u(t_n)) \\ &= u(t_{n-1}) + 2ku'(t_n).\end{aligned}$$

Then the error is

$$u(t_{n+1}) - U^{n+1} = u(t_{n+1}) - u(t_{n-1}) - 2ku'(t_n) = \mathcal{L}^n.$$

From (5.26) we see that in one step the error introduced is $O(k^3)$. This is consistent with second order accuracy in the global error if we think of trying to compute an approximation to the true solution $u(T)$ at some fixed time $T > 0$. To compute from time $t = 0$ up to time T, we need to take T/k time steps of length k. A rough estimate of the error at time T might be obtained by assuming that a new error of size $\mathcal{L}^n$ is introduced in the nth time step and is then simply carried along in later time steps without affecting the size of future local errors and without growing or diminishing itself. Then we would expect the resulting global error at time T to be simply the sum of all these local errors. Since each local error is $O(k^3)$ and we are adding up T/k of them, we end up with a global error that is $O(k^2)$.

This viewpoint is in fact exactly right for the simplest ODE (5.9), in which $f(u,t) = g(t)$ is independent of u and the solution is simply the integral of g, but it is a bit too simplistic for more interesting equations since the error at each time feeds back into the computation at the next step in the case where $f(u,t)$ depends on u. Nonetheless, it is essentially right in terms of the expected order of accuracy, provided the method is stable. In fact, it is useful to think of *stability* as exactly what is needed to make this naive analysis correct, by ensuring that the old errors from previous time steps do not grow too rapidly in future time steps. This will be investigated in detail in the following chapters.

5.6 Taylor series methods

The forward Euler method (5.19) can be derived using a Taylor series expansion of $u(t_{n+1})$ about $u(t_n)$:

$$u(t_{n+1}) = u(t_n) + ku'(t_n) + \frac{1}{2}k^2u''(t_n) + \cdots . \tag{5.27}$$

If we drop all terms of order k^2 and higher and use the differential equation to replace $u'(t_n)$ with $f(u(t_n), t_n)$, we obtain

$$u(t_{n+1}) \approx u(t_n) + kf(u(t_n), t_n).$$

This suggests the method (5.19). The 1-step error is $O(k^2)$ since we dropped terms of this order.

A *Taylor series method* of higher accuracy can be derived by keeping more terms in the Taylor series. If we keep the first $p + 1$ terms of the Taylor series expansion

$$u(t_{n+1}) \approx u(t_n) + ku'(t_n) + \frac{1}{2}k^2u''(t_n) + \cdots + \frac{1}{p!}k^pu^{(p)}(t_n)$$

we obtain a pth order accurate method. The problem is that we are given only

$$u'(t) = f(u(t), t)$$

and we must compute the higher derivatives by repeated differentiation of this function. For example, we can compute

$$\begin{aligned} u''(t) &= f_u(u(t),t)u'(t) + f_t(u(t),t) \\ &= f_u(u(t),t)f(u(t),t) + f_t(u(t),t). \end{aligned} \tag{5.28}$$

This can result in very messy expressions that must be worked out for each equation, and as a result this approach is not often used in practice. However, it is such an obvious approach that it is worth mentioning, and in some cases it may be useful. An example should suffice to illustrate the technique and its limitations.

Example 5.10. Suppose we want to solve the equation

$$u'(t) = t^2 \sin(u(t)). \tag{5.29}$$

Then we can compute

$$\begin{aligned} u''(t) &= 2t \sin(u(t)) + t^2 \cos(u(t))\, u'(t) \\ &= 2t \sin(u(t)) + t^4 \cos(u(t)) \sin(u(t)). \end{aligned}$$

A second order method is given by

$$U^{n+1} = U^n + k t_n^2 \sin(U^n) + \frac{1}{2}k^2[2t_n \sin(U^n) + t_n^4 \cos(U^n) \sin(U^n)].$$

Clearly higher order derivatives can be computed and used, but this is cumbersome even for this simple example. For systems of equations the method becomes still more complicated.

This Taylor series approach does get used in some situations, however—for example, in the derivation of the Lax–Wendroff method for hyperbolic PDEs; see Section 10.3. See also Section 11.3.

5.7 Runge–Kutta methods

Most methods used in practice do not require that the user explicitly calculate higher order derivatives. Instead a higher order finite difference approximation is designed that typically models these terms automatically.

A multistep method of the sort we will study in Section 5.9 can achieve high accuracy by using high order polynomial interpolation through several previous values of the solution and/or its derivative. To achieve the same effect with a 1-step method it is typically necessary to use a *multistage* method, where intermediate values of the solution and its derivative are generated and used within a single time step.

Example 5.11. A two-stage explicit Runge–Kutta method is given by

$$\begin{aligned} U^* &= U^n + \frac{1}{2}k f(U^n), \\ U^{n+1} &= U^n + k f(U^*). \end{aligned} \tag{5.30}$$

In the first stage an intermediate value is generated that approximates $u(t_{n+1/2})$ via Euler's method. In the second step the function f is evaluated at this midpoint to estimate the slope over the full time step. Since this now looks like a centered approximation to the derivative we might hope for second order accuracy, as we'll now verify by computing the LTE.

Combining the two steps above, we can rewrite the method as

$$U^{n+1} = U^n + k f\left(U^n + \frac{1}{2}k f(U^n)\right).$$

Viewed this way, this is clearly a 1-step explicit method. The truncation error is

$$\tau^n = \frac{1}{k}(u(t_{n+1}) - u(t_n)) - f\left(u(t_n) + \frac{1}{2}kf(u(t_n))\right). \tag{5.31}$$

Note that

$$\begin{aligned} f\left(u(t_n) + \frac{1}{2}kf(u(t_n))\right) &= f\left(u(t_n) + \frac{1}{2}ku'(t_n)\right) \\ &= f(u(t_n)) + \frac{1}{2}ku'(t_n)f'(u(t_n)) + \frac{1}{8}k^2(u'(t_n))^2 f''(u(t_n)) + \cdots. \end{aligned}$$

Since $f(u(t_n)) = u'(t_n)$ and differentiating gives $f'(u)u' = u''$, we obtain

$$f\left(u(t_n) + \frac{1}{2}kf(u(t_n))\right) = u'(t_n) + \frac{1}{2}ku''(t_n) + O(k^2).$$

Using this in (5.31) gives

$$\begin{aligned} \tau^n &= \frac{1}{k}\left(ku'(t_n) + \frac{1}{2}k^2u''(t_n) + O(k^3)\right) \\ &\qquad - \left(u'(t_n) + \frac{1}{2}ku''(t_n) + O(k^2)\right) \\ &= O(k^2) \end{aligned}$$

and the method is second order accurate. (Check the $O(k^2)$ term to see that this does not vanish.)

Remark: Another way to determine the order of accuracy of this simple method is to apply it to the special test equation $u' = \lambda u$, which has solution $u(t_{n+1}) = e^{\lambda k}u(t_n)$, and determine the error on this problem. Here we obtain

$$\begin{aligned} U^{n+1} &= U^n + k\lambda\left(U^n + \frac{1}{2}k\lambda U^n\right) \\ &= U^n + (k\lambda)U^n + \frac{1}{2}(k\lambda)^2U^n \\ &= e^{k\lambda}U^n + O(k^3). \end{aligned}$$

The one-step error is $O(k^3)$ and hence the LTE is $O(k^2)$. Of course we have checked only that the LTE is $O(k^2)$ on one particular function $u(t) = e^{\lambda t}$, not on all smooth solutions, and for general Runge–Kutta methods for nonautonomous problems this approach gives only an upper bound on the method's order of accuracy. Applying a method to this special equation is also a fundamental tool in stability analysis—see Chapter 7.

Example 5.12. The Runge–Kutta method (5.30) can be extended to nonautonomous equations of the form $u'(t) = f(u(t), t)$:

$$\begin{aligned} U^* &= U^n + \frac{1}{2}kf(U^n, t_n), \\ U^{n+1} &= U^n + kf\left(U^*, t_n + \frac{k}{2}\right). \end{aligned} \tag{5.32}$$

This is again second order accurate, as can be verified by expanding as above, but it is slightly more complicated since Taylor series in two variables must be used.

Example 5.13. One simple higher order Runge–Kutta method is the fourth order four-stage method given by

$$\begin{aligned}
Y_1 &= U^n,\\
Y_2 &= U^n + \frac{1}{2}kf(Y_1, t_n),\\
Y_3 &= U^n + \frac{1}{2}kf\left(Y_2, t_n + \frac{k}{2}\right),\\
Y_4 &= U^n + kf\left(Y_3, t_n + \frac{k}{2}\right),\\
U^{n+1} &= U^n + \frac{k}{6}\left[f(Y_1, t_n) + 2f\left(Y_2, t_n + \frac{k}{2}\right)\right.\\
&\qquad \left. + 2f\left(Y_3, t_n + \frac{k}{2}\right) + f(Y_4, t_n + k)\right].
\end{aligned} \tag{5.33}$$

Note that if $f(u, t) = f(t)$ does not depend on u, then this reduces to Simpson's rule for the integral. This method was particularly popular in the precomputer era, when computations were done by hand, because the coefficients are so simple. Today there is no need to keep the coefficients simple and other Runge–Kutta methods have advantages.

A general r-stage Runge–Kutta method has the form

$$\begin{aligned}
Y_1 &= U^n + k\sum_{j=1}^{r} a_{1j} f(Y_j, t_n + c_j k),\\
Y_2 &= U^n + k\sum_{j=1}^{r} a_{2j} f(Y_j, t_n + c_j k),\\
&\vdots\\
Y_r &= U^n + k\sum_{j=1}^{r} a_{rj} f(Y_j, t_n + c_j k),
\end{aligned} \tag{5.34}$$

$$U^{n+1} = U^n + k\sum_{j=1}^{r} b_j f(Y_j, t_n + c_j k).$$

Consistency requires

$$\begin{aligned}
&\sum_{j=1}^{r} a_{ij} = c_i, \quad i = 1,\ 2,\ \ldots,\ r,\\
&\sum_{j=1}^{r} b_j = 1.
\end{aligned} \tag{5.35}$$

If these conditions are satisfied, then the method will be at least first order accurate.

The coefficients for a Runge–Kutta method are often displayed in a so-called Butcher tableau:

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1r} \\ \vdots & \vdots & & \vdots \\ c_r & a_{r1} & \dots & a_{rr} \\ \hline & b_1 & \dots & b_r \end{array} \tag{5.36}$$

For example, the fourth order Runge–Kutta method given in (5.33) has the following tableau (entries not shown are all 0):

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

An important class of Runge–Kutta methods consists of the *explicit methods* for which $a_{ij} = 0$ for $j \geq i$. For an explicit method, the elements on and above the diagonal in the a_{ij} portion of the Butcher tableau are all equal to zero, as, for example, with the fourth order method displayed above. With an explicit method, each of the Y_i values is computed using only the previously computed Y_j.

Fully implicit Runge–Kutta methods, in which each Y_i depends on all the Y_j, can be expensive to implement on systems of ODEs. For a system of s equations (where each Y_i is in $\mathbb{R}^s$), a system of sr equations must be solved to compute the r vectors Y_i simultaneously.

One subclass of implicit methods that are simpler to implement are the *diagonally implicit* Runge–Kutta methods (DIRK methods) in which Y_i depends on Y_j for $j \leq i$, i.e., $a_{ij} = 0$ for $j > i$. For a system of s equations, DIRK methods require solving a sequence of r implicit systems, each of size s, rather than a coupled set of sr equations as would be required in a fully implicit Runge–Kutta method. DIRK methods are so named because their tableau has zero values above the diagonal but possibly nonzero diagonal elements.

Example 5.14. A second order accurate DIRK method is given by

$$\begin{aligned} Y_1 &= U^n, \\ Y_2 &= U^n + \frac{k}{4}\left[f(Y_1, t_n) + f\left(Y_2, t_n + \frac{k}{2}\right)\right], \\ Y_3 &= U^n + \frac{k}{3}\left[f(Y_1, t_n) + f\left(Y_2, t_n + \frac{k}{2}\right) + f(Y_3, t_n + k)\right], \\ U^{n+1} &= Y_3 = U^n + \frac{k}{3}\left[f(Y_1, t_n) + f\left(Y_2, t_n + \frac{k}{2}\right) + f(Y_3, t_n + k)\right]. \end{aligned} \tag{5.37}$$

This method is known as the TR-BDF2 method and is derived in a different form in Section 8.5. Its tableau is

0			
1/2	1/4	1/4	
1	1/3	1/3	1/3
	1/3	1/3	1/3

In addition to the conditions (5.35), a Runge–Kutta method is second order accurate if

$$\sum_{j=1}^{r} b_j c_j = \frac{1}{2}, \tag{5.38}$$

as is satisfied for the method (5.37). Third order accuracy requires two additional conditions:

$$\begin{aligned} \sum_{j=1}^{r} b_j c_j^2 &= \frac{1}{3}, \\ \sum_{i=1}^{r}\sum_{j=1}^{r} b_i a_{ij} c_j &= \frac{1}{6}. \end{aligned} \tag{5.39}$$

Fourth order accuracy requires an additional four conditions on the coefficients, and higher order methods require an exponentially growing number of conditions.

An r-stage explicit Runge–Kutta method can have order at most r, although for $r \geq 5$ the order is strictly less than the number of stages. Among implicit Runge–Kutta methods, r-stage methods of order $2r$ exist. There typically are many ways that the coefficients a_{ij} and b_j can be chosen to achieve a given accuracy, provided the number of stages is sufficiently large. Many different classes of Runge–Kutta methods have been developed over the years with various advantages. The order conditions are quite complicated for higher-order methods and an extensive theory has been developed by Butcher for analyzing these methods and their stability properties. For more discussion and details see, for example, [13], [43], [44].

Using more stages to increase the order of a method is an obvious strategy. For some problems, however, we will also see that it can be advantageous to use a large number of stages to increase the *stability* of the method while keeping the order of accuracy relatively low. This is the idea behind the *Runge–Kutta–Chebyshev methods*, for example, discussed in Section 8.6.

5.7.1 Embedded methods and error estimation

Most practical software for solving ODEs does not use a fixed time step but rather adjusts the time step during the integration process to try to achieve some specified error bound. One common way to estimate the error in the computation is to compute using two different methods and compare the results. Knowing something about the error behavior of each method often allows the possibility of estimating the error in at least one of the two results.

A simple way to do this for ODEs is to take a time step with two different methods, one of order p and one of a different order, say, $p + 1$. Assuming that the time step is small enough that the higher order method is really generating a better approximation, then

the difference between the two results will be an estimate of the one-step error in the lower order method. This can be used as the basis for choosing an appropriate time step for the lower order approximation. Often the time step is chosen in this manner, but then the higher order solution is used as the actual approximation at this time and as the starting point for the next time step. This is sometimes called *local extrapolation*. Once this is done there is no estimate of the error, but presumably it is even smaller than the error in the lower order method and so the approximation generated will be even better than the required tolerance. For more about strategies for time step selection, see, for example, [5], [43], [78].

Note, however, that the procedure of using two different methods in every time step could easily double the cost of the computation unless we choose the methods carefully. Since the main cost in a Runge–Kutta method is often in evaluating the function $f(u,t)$, it makes sense to reuse function values as much as possible and look for methods that provide two approximations to U^{n+1} of different order based on the same set of function evaluations, by simply taking different linear combinations of the $f(Y_j, t_n + c_j k)$ values in the final stage of the Runge–Kutta method (5.34). So in addition to the value U^{n+1} given there we would like to also compute a value

$$\hat{U}^{n+1} = U^n + k \sum_{j=1}^{r} \hat{b}_j f(Y_j, t_n + c_j k) \tag{5.40}$$

that gives an approximation of a different order that can be used for error estimation. These are called *embedded Runge–Kutta methods* and are often displayed in a tableau of the form

$$\begin{array}{c|ccc} c_1 & a_{11} & \dots & a_{1r} \\ \vdots & \vdots & & \vdots \\ c_r & a_{r1} & \dots & a_{rr} \\ \hline & b_1 & \dots & b_r \\ \hline & \hat{b}_1 & \dots & \hat{b}_r \end{array} \tag{5.41}$$

As a very simple example, the second order Runge–Kutta method (5.32) could be combined with the first order Euler method:

$$\begin{aligned} Y_1 &= U^n, \\ Y_2 &= U^n + \frac{1}{2} k f(Y_1, t_n), \\ U^{n+1} &= U^n + k f\left(Y_2, t_n + \frac{k}{2}\right), \\ \hat{U}^{n+1} &= U^n + k f(Y_1, t_n). \end{aligned} \tag{5.42}$$

Note that the computation of $\hat{U}^{n+1}$ reuses the value $f(Y_1, t_n)$ obtained in computing Y_2 and is essentially free. Also note that

$$\begin{aligned}\hat{U}^{n+1} - U^{n+1} &= k\left(f(Y_1, t_n) - f\left(Y_2, t_n + \frac{k}{2}\right)\right) \\ &\approx k\left(u'(t_n) - u'(t_{n+1/2})\right) \\ &\approx \frac{1}{2}k^2 u''(t_n),\end{aligned} \tag{5.43}$$

which is approximately the one-step error for Euler's method.

Most software based on Runge–Kutta methods uses embedded methods of higher order. For example, the `ode45` routine in MATLAB uses a pair of embedded Runge-Kutta methods of order 4 and 5 due to Dormand and Prince [25]. See Shampine and Reichelt [78] for implementation details (or `typeode45` in MATLAB).

5.8 One-step versus multistep methods

Taylor series and Runge–Kutta methods are *one-step methods*; the approximation U^{n+1} depends on U^n but not on previous values U^{n-1}, U^{n-2}, ... In the next section we will consider a class of multistep methods where previous values are also used (one example is the midpoint method (5.24)).

One-step methods have several advantages over multistep methods:

- The methods are *self-starting*: from the initial data U^0 the desired method can be applied immediately. Multistep methods require that some other method be used initially, as discussed in Section 5.9.3.

- The time step k can be changed at any point, based on an error estimate, for example. The time step can also be changed with a multistep method but more care is required since the previous values are assumed to be equally spaced in the standard form of these methods given below.

- If the solution $u(t)$ is not smooth at some isolated point t^* (for example, because $f(u, t)$ is discontinuous at t^*), then with a one-step method it is often possible to get full accuracy simply by ensuring that t^* is a grid point. With a multistep method that uses data from both sides of t^* in approximating derivatives, a loss of accuracy may occur.

On the other hand, one-step methods have some disadvantages. The disadvantage of Taylor series methods is that they require differentiating the given equation and are cumbersome and often expensive to implement. Runge–Kutta methods only use evaluations of the function f, but a higher order multistage method requires evaluating f several times each time step. For simple equations this may not be a problem, but if function values are expensive to compute, then high order Runge–Kutta methods may be quite expensive as well. This is particularly true for implicit methods, where an implicit nonlinear system must be solved in each stage.

An alternative is to use a multistep method in which values of f already computed in previous time steps are reused to obtain higher order accuracy. Typically only one new f evaluation is required in each time step. The popular class of *linear multistep methods* is discussed in the next section.

5.9 Linear multistep methods

All the methods introduced in Section 5.3 are members of a class of methods called linear multistep methods (LMMs). In general, an r-step LMM has the form

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = k \sum_{j=0}^{r} \beta_j f(U^{n+j}, t_{n+j}). \tag{5.44}$$

The value U^{n+r} is computed from this equation in terms of the previous values U^{n+r-1}, U^{n+r-2}, $\ldots$, U^n and f values at these points (which can be stored and reused if f is expensive to evaluate).

If $\beta_r = 0$, then the method (5.44) is explicit; otherwise it is implicit. Note that we can multiply both sides by any constant and have essentially the same method, although the coefficients α_j and β_j would change. The normalization $\alpha_r = 1$ is often assumed to fix this scale factor.

There are special classes of methods of this form that are particularly useful and have distinctive names. These will be written out for the autonomous case where $f(u,t) = f(u)$ to simplify the formulas, but each can be used more generally by replacing $f(U^{n+j})$ with $f(U^{n+j}, t_{n+j})$ in any of the formulas.

Example 5.15. The *Adams methods* have the form

$$U^{n+r} = U^{n+r-1} + k \sum_{j=0}^{r} \beta_j f(U^{n+j}). \tag{5.45}$$

These methods all have

$$\alpha_r = 1, \quad \alpha_{r-1} = -1, \quad \text{and } \alpha_j = 0 \text{ for } j < r-1.$$

The β_j coefficients are chosen to maximize the order of accuracy. If we require $\beta_r = 0$ so the method is explicit, then the r coefficients β_0, β_1, $\ldots$, β_{r-1} can be chosen so that the method has order r. This can be done by using Taylor series expansion of the local truncation error and then choosing the β_j to eliminate as many terms as possible. This gives the explicit *Adams–Bashforth methods*.

Another way to derive the Adams–Bashforth methods is by writing

$$\begin{aligned} u(t_{n+r}) &= u(t_{n+r-1}) + \int_{t_{n+r-1}}^{t_{n+r}} u'(t)\,dt \\ &= u(t_{n+r-1}) + \int_{t_{n+r-1}}^{t_{n+r}} f(u(t))\,dt \end{aligned} \tag{5.46}$$

and then applying a quadrature rule to this integral to approximate

$$\int_{t_{n+r-1}}^{t_{n+r}} f(u(t))\,dt \approx k \sum_{j=1}^{r-1} \beta_j f(u(t_{n+j})). \tag{5.47}$$

This quadrature rule can be derived by interpolating $f(u(t))$ by a polynomial $p(t)$ of degree $r-1$ at the points t_n, t_{n+1}, $\ldots$, t_{n+r-1} and then integrating the interpolating polynomial.

Either approach gives the same r-step explicit method. The first few are given below.

Explicit Adams–Bashforth methods

1-step: $U^{n+1} = U^n + kf(U^n)$ (forward Euler)

2-step: $U^{n+2} = U^{n+1} + \frac{k}{2}(-f(U^n) + 3f(U^{n+1}))$

3-step: $U^{n+3} = U^{n+2} + \frac{k}{12}(5f(U^n) - 16f(U^{n+1}) + 23f(U^{n+2}))$

4-step: $U^{n+4} = U^{n+3} + \frac{k}{24}(-9f(U^n) + 37f(U^{n+1}) - 59f(U^{n+2}) + 55f(U^{n+3}))$

If we allow β_r to be nonzero, then we have one more free parameter and so we can eliminate an additional term in the LTE. This gives an implicit method of order $r + 1$ called the r-step *Adams–Moulton*. These methods can again be derived by polynomial interpolation, now using a polynomial $p(t)$ of degree r that interpolates $f(u(t))$ at the points t_n, t_{n+1}, $\ldots$, t_{n+r} and then integrating the interpolating polynomial.

Implicit Adams–Moulton methods

1-step: $U^{n+1} = U^n + \frac{k}{2}(f(U^n) + f(U^{n+1}))$ (trapezoidal method)

2-step: $U^{n+2} = U^{n+1} + \frac{k}{12}(-f(U^n) + 8f(U^{n+1}) + 5f(U^{n+2}))$

3-step: $U^{n+3} = U^{n+2} + \frac{k}{24}(f(U^n) - 5f(U^{n+1}) + 19f(U^{n+2}) + 9f(U^{n+3}))$

4-step: $U^{n+4} = U^{n+3} + \frac{k}{720}(-19f(U^n) + 106f(U^{n+1}) - 264f(U^{n+2}) + 646f(U^{n+3}) + 251f(U^{n+4}))$

Example 5.16. The explicit *Nyström methods* have the form

$$U^{n+r} = U^{n+r-2} + k\sum_{j=0}^{r-1} \beta_j f(U^{n+j})$$

with the β_j chosen to give order r. The midpoint method (5.23) is a two-step explicit Nyström method. A two-step implicit Nyström method is *Simpson's rule*,

$$U^{n+2} = U^n + \frac{2k}{6}(f(U^n) + 4f(U^{n+1}) + f(U^{n+2})).$$

This reduces to Simpson's rule for quadrature if applied to the ODE $u'(t) = f(t)$.

5.9.1 Local truncation error

For LMMs it is easy to derive a general formula for the LTE. We have

$$\tau(t_{n+r}) = \frac{1}{k}\left(\sum_{j=0}^{r} \alpha_j u(t_{n+j}) - k\sum_{j=0}^{r} \beta_j u'(t_{n+j})\right).$$

We have used $f(u(t_{n+j})) = u'(t_{n+j})$ since $u(t)$ is the exact solution of the ODE. Assuming u is smooth and expanding in Taylor series gives

$$u(t_{n+j}) = u(t_n) + jku'(t_n) + \frac{1}{2}(jk)^2 u''(t_n) + \cdots,$$
$$u'(t_{n+j}) = u'(t_n) + jku''(t_n) + \frac{1}{2}(jk)^2 u'''(t_n) + \cdots,$$

and so

$$\begin{aligned}\tau(t_{n+r}) = \frac{1}{k}\left(\sum_{j=0}^{r}\alpha_j\right)u(t_n) + \left(\sum_{j=0}^{r}(j\alpha_j - \beta_j)\right)u'(t_n)\\ + k\left(\sum_{j=0}^{r}\left(\frac{1}{2}j^2\alpha_j - j\beta_j\right)\right)u''(t_n)\\ + \cdots + k^{q-1}\left(\sum_{j=0}^{r}\left(\frac{1}{q!}j^q\alpha_j - \frac{1}{(q-1)!}j^{q-1}\beta_j\right)\right)u^{(q)}(t_n) + \cdots.\end{aligned}$$

The method is *consistent* if $\tau \to 0$ as $k \to 0$, which requires that at least the first two terms in this expansion vanish:

$$\sum_{j=0}^{r}\alpha_j = 0 \qquad \text{and} \qquad \sum_{j=0}^{r} j\alpha_j = \sum_{j=0}^{r}\beta_j. \tag{5.48}$$

If the first $p+1$ terms vanish, then the method will be pth order accurate. Note that these conditions depend only on the coefficients α_j and β_j of the method and not on the particular differential equation being solved.

5.9.2 Characteristic polynomials

It is convenient at this point to introduce the so-called characteristic polynomials $\rho(\zeta)$ and $\sigma(\zeta)$ for the LMM:

$$\rho(\zeta) = \sum_{j=0}^{r}\alpha_j\zeta^j \qquad \text{and} \qquad \sigma(\zeta) = \sum_{j=0}^{r}\beta_j\zeta^j. \tag{5.49}$$

The first of these is a polynomial of degree r. So is $\sigma(\zeta)$ if the method is implicit; otherwise its degree is less than r. Note that $\rho(1) = \sum\alpha_j$ and also that $\rho'(\zeta) = \sum j\alpha_j\zeta^{j-1}$, so that the consistency conditions (5.48) can be written quite concisely as conditions on these two polynomials:

$$\rho(1) = 0 \qquad \text{and} \qquad \rho'(1) = \sigma(1). \tag{5.50}$$

This, however, is not the main reason for introducing these polynomials. The location of the roots of certain polynomials related to ρ and σ plays a fundamental role in stability theory as we will see in the next two chapters.

Example 5.17. The two-step Adams–Moulton method

$$U^{n+2} = U^{n+1} + \frac{k}{12}(-f(U^n) + 8f(U^{n+1}) + 5f(U^{n+2})) \tag{5.51}$$

has characteristic polynomials

$$\rho(\zeta) = \zeta^2 - \zeta, \qquad \sigma(\zeta) = \frac{1}{12}(-1 + 8\zeta + 5\zeta^2). \tag{5.52}$$

5.9.3 Starting values

One difficulty with using LMMs if $r > 1$ is that we need the values $U^0, U^1, \ldots, U^{r-1}$ before we can begin to apply the multistep method. The value $U^0 = \eta$ is known from the initial data for the problem, but the other values are not and typically must be generated by some other numerical method or methods.

Example 5.18. If we want to use the midpoint method (5.23), then we need to generate U^1 by some other method before we begin to apply (5.23) with $n = 1$. We can obtain U^1 from U^0 using any one-step method, such as Euler's method or the trapezoidal method, or a higher order Taylor series or Runge–Kutta method. Since the midpoint method is second order accurate we need to make sure that the value U^1 we generate is sufficiently accurate so that this second order accuracy will not be lost. Our first impulse might be to conclude that we need to use a second order accurate method such as the trapezoidal method rather than the first order accurate Euler method, but this is wrong. The overall method is second order in either case. The reason that we achieve second order accuracy even if Euler is used in the first step is exactly analogous to what was observed earlier for boundary value problems, where we found that we can often get away with one order of accuracy lower in the local error at a single point than what we have elsewhere.

In the present context this is easiest to explain in terms of the one-step error. The midpoint method has a one-step error that is $O(k^3)$ and because this method is applied in $O(T/k)$ time steps, the global error is expected to be $O(k^2)$. Euler's method has a one-step error that is $O(k^2)$, but we are applying this method only once.

If $U^0 = \eta = u(0)$, then the error in U^1 obtained with Euler will be $O(k^2)$. If the midpoint method is stable, then this error will not be magnified unduly in later steps and its contribution to the global error will be only $O(k^2)$. The overall second order accuracy will not be affected.

More generally, with an r-step method of order p, we need r starting values

$$U^0, U^1, \ldots, U^{r-1}$$

and we need to generate these values using a method that has a *one-step error* that is $O(k^p)$ (corresponding to an LTE that is $O(k^{p-1})$). Since the number of times we apply this method $(r-1)$ is independent of k as $k \to 0$, this is sufficient to give an $O(k^p)$ global error. Of course somewhat better accuracy (a smaller error constant) may be achieved by using a pth order accurate method for the starting values, which takes little additional work.

In software for the IVP, multistep methods generally are implemented in a form that allows changing the time step during the integration process, as is often required to efficiently solve the problem. Typically the order of the method is also allowed to vary,

depending on how the solution is behaving. In such software it is then natural to solve the starting-value problem by initially taking a small time step with a one-step method and then ramping up to higher order methods and longer time steps as the integration proceeds and more past data are available.

5.9.4 Predictor-corrector methods

The idea of comparing results obtained with methods of different order as a way to choose the time step, discussed in Section 5.7.1 for Runge–Kutta methods, is also used with LMMs. One approach is to use a *predictor-corrector method*, in which an explicit Adams–Bashforth method of some order is used to predict a value $\hat{U}^{n+1}$ and then the Adams–Moulton method of the same order is used to "correct" this value. This is done by using $\hat{U}^{n+1}$ on the right-hand side of the Adams–Moulton method inside the f evaluation, so that the Adams–Moulton formula is no longer implicit. For example, the one-step Adams–Bashforth (Euler's method) and the one-step Adams–Moulton method (the trapezoidal method) could be combined into

$$\begin{aligned} \hat{U}^{n+1} &= U^n + kf(U^n), \\ U^{n+1} &= U^n + \frac{1}{2}k(f(U^n) + f(\hat{U}^{n+1})). \end{aligned} \tag{5.53}$$

It can be shown that this method is second order accurate, like the trapezoidal method, but it also generates a lower order approximation and the difference between the two can be used to estimate the error. The MATLAB routine `ode113` uses this approach, with Adams–Bashforth–Moulton methods of orders 1–12; see [78].

Chapter 6

Zero-Stability and Convergence for Initial Value Problems

6.1 Convergence

To discuss the convergence of a numerical method for the initial value problem, we focus on a fixed (but arbitrary) time $T > 0$ and consider the error in our approximation to $u(T)$ computed with the method using time step k. The method converges on this problem if this error goes to zero as $k \to 0$. Note that the number of time steps that we need to take to reach time T increases as $k \to 0$. If we use N to denote this value ($N = T/k$), then convergence means that

$$\lim_{\substack{k \to 0 \\ Nk = T}} U^N = u(T). \tag{6.1}$$

In principle a method might converge on one problem but not on another, or converge with one set of starting values but not with another set. To speak of a *method* being *convergent* in general, we require that it converges on *all* problems in a reasonably large class with *all* reasonable starting values. For an r-step method we need r starting values. These values will typically depend on k, and to make this clear we will write them as $U^0(k),\ U^1(k),\ \ldots,\ U^{r-1}(k)$. While these will generally approximate $u(t)$ at the times $t_0 = 0,\ t_1 = k, \ldots,\ t_{r-1} = (r-1)k$, respectively, as $k \to 0$, each of these times approaches $t_0 = 0$. So the weakest condition we might put on our starting values is that they converge to the correct initial value η as $k \to 0$:

$$\lim_{k \to 0} U^\nu(k) = \eta \quad \text{for } \nu = 0,\ 1,\ \ldots,\ r-1. \tag{6.2}$$

We can now state the definition of convergence.

Definition 6.1. *An r-step method is said to be* convergent *if applying the method to any ODE* (5.1) *with $f(u,t)$ Lipschitz continuous in u, and with any set of starting values satisfying* (6.2)*, we obtain convergence in the sense of* (6.1) *for every fixed time $T > 0$ at which the ODE has a unique solution.*

To be convergent, a method must be *consistent*, meaning as before that the local truncation error (LTE) is $o(1)$ as $k \to 0$, and also *zero-stable*, as described later in this

chapter. We will begin to investigate these issues by first proving the convergence of one-step methods, which turn out to be zero-stable automatically. We start with Euler's method on linear problems, then consider Euler's method on general nonlinear problems and finally extend this to a wide class of one-step methods.

6.2 The test problem

Much of the theory presented below is based on examining what happens when a method is applied to a simple scalar linear equation of the form

$$u'(t) = \lambda u(t) + g(t) \tag{6.3}$$

with initial data

$$u(t_0) = \eta.$$

The solution is then given by Duhamel's principle (5.8),

$$u(t) = e^{\lambda(t-t_0)}\eta + \int_{t_0}^{t} e^{\lambda(t-\tau)} g(\tau)\, d\tau. \tag{6.4}$$

6.3 One-step methods

6.3.1 Euler's method on linear problems

If we apply Euler's method to (6.3), we obtain

$$\begin{aligned} U^{n+1} &= U^n + k(\lambda U^n + g(t_n)) \\ &= (1 + k\lambda)U^n + k g(t_n). \end{aligned} \tag{6.5}$$

The LTE for Euler's method is given by

$$\begin{aligned} \tau^n &= \left(\frac{u(t_{n+1}) - u(t_n)}{k}\right) - (\lambda u(t_n) + g(t_n)) \\ &= \left(u'(t_n) + \frac{1}{2}k u''(t_n) + O(k^2)\right) - u'(t_n) \\ &= \frac{1}{2}k u''(t_n) + O(k^2). \end{aligned} \tag{6.6}$$

Rewriting this equation as

$$u(t_{n+1}) = (1 + k\lambda)u(t_n) + k g(t_n) + k\tau^n$$

and subtracting this from (6.5) gives a difference equation for the global error E^n:

$$E^{n+1} = (1 + k\lambda)E^n - k\tau^n. \tag{6.7}$$

Note that this has exactly the same form as (6.5) but with a different nonhomogeneous term: $-\tau^n$ in place of $g(t_n)$. This is analogous to equation (2.15) in the boundary value theory

and again gives the relation we need between the local truncation error τ^n (which is easy to compute) and the global error E^n (which we wish to bound). Note again that *linearity* plays a critical role in making this connection. We will consider nonlinear problems below.

Because the equation and method we are now considering are both so simple, we obtain an equation (6.7) that we can explicitly solve for the global error E^n. Applying the recursion (6.7) repeatedly we see what form the solution should take:

$$\begin{aligned} E^n &= (1 + k\lambda)E^{n-1} - k\tau^{n-1} \\ &= (1 + k\lambda)[(1 + k\lambda)E^{n-2} - k\tau^{n-2}] - k\tau^{n-1} \\ &= \cdots . \end{aligned}$$

By induction we can easily confirm that in general

$$E^n = (1 + k\lambda)^n E^0 - k \sum_{m=1}^{n} (1 + k\lambda)^{n-m} \tau^{m-1}. \tag{6.8}$$

(Note that some of the superscripts are powers while others are indices!) This has a form that is very analogous to the solution (6.4) of the corresponding ordinary differential equation (ODE), where now $(1 + k\lambda)^{n-m}$ plays the role of the solution operator of the homogeneous problem—it transforms data at time t_m to the solution at time t_n. The expression (6.8) is sometimes called the discrete form of Duhamel's principle.

We are now ready to prove that Euler's method converges on (6.3). We need only observe that

$$|1 + k\lambda| \le e^{k|\lambda|} \tag{6.9}$$

and so

$$(1 + k\lambda)^{n-m} \le e^{(n-m)k|\lambda|} \le e^{nk|\lambda|} \le e^{|\lambda|T}, \tag{6.10}$$

provided that we restrict our attention to the finite time interval $0 \le t \le T$, so that $t_n = nk \le T$. It then follows from (6.8) that

$$\begin{aligned} |E^n| &\le e^{|\lambda|T} \left(|E^0| + k \sum_{m=1}^{n} |\tau^{m-1}| \right) \\ &\le e^{|\lambda|T} \left(|E^0| + nk \max_{1 \le m \le n} |\tau^{m-1}| \right). \end{aligned} \tag{6.11}$$

Let $N = T/k$ be the number of time steps needed to reach time T and set

$$\|\tau\|_\infty = \max_{0 \le n \le N-1} |\tau^n|.$$

From (6.6) we expect

$$\|\tau\|_\infty \approx \frac{1}{2} k \|u''\|_\infty = O(k),$$

where $\|u''\|_\infty$ is the maximum value of the function u'' over the interval $[0, T]$. Then for $t = nk \le T$, we have from (6.11) that

$$|E^n| \le e^{|\lambda|T} (|E^0| + T\|\tau\|_\infty).$$

If (6.2) is satisfied then $E^0 \to 0$ as $k \to 0$. In fact for this one-step method we would generally take $U^0 = u(0) = \eta$, in which case E^0 drops out and we are left with

$$|E^n| \le e^{|\lambda|T} T \|\tau\|_\infty = O(k) \quad \text{as } k \to 0 \tag{6.12}$$

and hence the method converges and is in fact first order accurate.

Note where *stability* comes into the picture. The one-step error $\mathcal{L}^{m-1} = k\tau^{m-1}$ introduced in the mth step contributes the term $(1 + k\lambda)^{n-m}\mathcal{L}^{m-1}$ to the global error. The fact that $|(1 + k\lambda)^{n-m}| < e^{|\lambda|T}$ is uniformly bounded as $k \to 0$ allows us to conclude that each contribution to the final error can be bounded in terms of its original size as a one-step error. Hence the "naive analysis" of Section 5.5 is valid, and the global error has the same order of magnitude as the local truncation error.

6.3.2 Relation to stability for boundary value problems

To see how this ties in with the definition of stability used in Chapter 2 for the BVP, it may be useful to view Euler's method as giving a linear system in matrix form, although this is not the way it is used computationally. If we view the equations (6.5) for $n = 0, 1, \ldots, N-1$ as a linear system $AU = F$ for $U = [U^1, U^2, \ldots, U^N]^T$, then

$$A = \frac{1}{k}\begin{bmatrix} 1 & & & & & \\ -(1+k\lambda) & 1 & & & & \\ & -(1+k\lambda) & 1 & & & \\ & & & \ddots & & \\ & & & -(1+k\lambda) & 1 & \\ & & & & -(1+k\lambda) & 1 \end{bmatrix}$$

and

$$U = \begin{bmatrix} U^1 \\ U^2 \\ U^3 \\ \vdots \\ U^{N-1} \\ U^N \end{bmatrix}, \qquad F = \begin{bmatrix} (1/k + \lambda)U^0 + g(t_0) \\ g(t_1) \\ g(t_2) \\ \vdots \\ g(t_{N-2}) \\ g(t_{N-1}) \end{bmatrix}.$$

We have divided both sides of (6.5) by k to conform to the notation of Chapter 2. Since the matrix A is lower triangular, this system is easily solved by forward substitution, which results in the iterative equation (6.5).

If we now let $\hat{U}$ be the vector obtained from the true solution as in Chapter 2, then subtracting $A\hat{U} = F + \tau$ from $AU = F$, we obtain (2.15) (the matrix form of (6.7)) with solution (6.8). We are then in exactly the same framework as in Chapter 2. So we have convergence and a global error with the same magnitude as the local error provided that the method is stable in the sense of Definition 2.1, i.e., that the inverse of the matrix A is bounded independent of k for all k sufficiently small.

The inverse of this matrix is easy to compute. In fact we can see from the solution (6.8) that

$$A^{-1} = k \begin{bmatrix} 1 & & & & & \\ (1+k\lambda) & 1 & & & & \\ (1+k\lambda)^2 & (1+k\lambda) & 1 & & & \\ (1+k\lambda)^3 & (1+k\lambda)^2 & (1+k\lambda) & 1 & & \\ \vdots & & & & \ddots & \\ (1+k\lambda)^{N-1} & (1+k\lambda)^{N-2} & (1+k\lambda)^{N-3} & \cdots & (1+k\lambda) & 1 \end{bmatrix}.$$

We easily compute using (A.10a) that

$$\|A^{-1}\|_\infty = k \sum_{m=1}^{N} |(1+k\lambda)^{N-m}|$$

and so

$$\|A^{-1}\|_\infty \le kNe^{|\lambda|T} = Te^{|\lambda|T}.$$

This is uniformly bounded as $k \to 0$ for fixed T. Hence the method is stable and $\|E\|_\infty \le \|A^{-1}\|_\infty \|\tau\|_\infty \le Te^{|\lambda|T}\|\tau\|_\infty$, which agrees with the bound (6.12).

6.3.3 Euler's method on nonlinear problems

So far we have focused entirely on linear equations. Practical problems are almost always nonlinear, but for the initial value problem it turns out that it is not significantly more difficult to handle this case if we assume that $f(u)$ is Lipschitz continuous, which is reasonable in light of the discussion in Section 5.2.

Euler's method on $u' = f(u)$ takes the form

$$U^{n+1} = U^n + kf(U^n) \tag{6.13}$$

and the truncation error is defined by

$$\begin{aligned} \tau^n &= \frac{1}{k}(u(t_{n+1}) - u(t_n)) - f(u(t_n)) \\ &= \frac{1}{2}ku''(t_n) + O(k^2), \end{aligned}$$

just as in the linear case. So the true solution satisfies

$$u(t_{n+1}) = u(t_n) + kf(u(t_n)) + k\tau^n$$

and subtracting this from (6.13) gives

$$E^{n+1} = E^n + k(f(U^n) - f(u(t_n))) - k\tau^n. \tag{6.14}$$

In the linear case $f(U^n) - f(u(t_n)) = \lambda E^n$ and we get the relation (6.7) for E^n. In the nonlinear case we cannot express $f(U^n) - f(u(t_n))$ directly in terms of the error E^n in general. However, using the Lipschitz continuity of f we can get a bound on this in terms of E^n:

$$|f(U^n) - f(u(t_n))| \le L|U^n - u(t_n)| = L|E^n|.$$

Using this in (6.14) gives

$$|E^{n+1}| \le |E^n| + kL|E^n| + k|\tau^n| = (1 + kL)|E^n| + k|\tau^n|. \tag{6.15}$$

From this inequality we can show by induction that

$$|E^n| \le (1 + kL)^n |E^0| + k \sum_{m=1}^{n} (1 + kL)^{n-m} |\tau^{m-1}|$$

and so, using the same steps as in obtaining (6.12) (and again assuming $E^0 = 0$), we obtain

$$|E^n| \le e^{LT} T \|\tau\|_\infty = O(k) \quad \text{as } k \to 0 \tag{6.16}$$

for all n with $nk \le T$, proving that the method converges. In the linear case $L = |\lambda|$ and this reduces to exactly (6.12).

6.3.4 General one-step methods

A general explicit one-step method takes the form

$$U^{n+1} = U^n + k\Psi(U^n, t_n, k) \tag{6.17}$$

for some function Ψ, which depends on f of course. We will assume that $\Psi(u, t, k)$ is continuous in t and k and Lipschitz continuous in u, with Lipschitz constant L' that is generally related to the Lipschitz constant of f.

Example 6.1. For the two-stage Runge–Kutta method of Example 5.11, we have

$$\Psi(u, t, k) = f\left(u + \frac{1}{2}kf(u)\right). \tag{6.18}$$

If f is Lipschitz continuous with Lipschitz constant L, then Ψ has Lipschitz constant $L' = L + \frac{1}{2}kL^2$.

The one-step method (6.17) is *consistent* if

$$\Psi(u, t, 0) = f(u, t)$$

for all u, t, and Ψ is continuous in k. The local truncation error is

$$\tau^n = \left(\frac{u(t_{n+1}) - u(t_n)}{k}\right) - \Psi(u(t_n), t_n, k).$$

We can show that any one-step method satisfying these conditions is convergent. We have

$$u(t_{n+1}) = u(t_n) + k\Psi(u(t_n), t_n, k) + k\tau^n$$

and subtracting this from (6.17) gives

$$E^{n+1} = E^n + k\left(\Psi(U^n, t_n, k) - \Psi(u(t_n), t_n, k)\right) - k\tau^n.$$

Using the Lipschitz condition we obtain

$$|E^{n+1}| \le |E^n| + kL'|E^n| + k|\tau^n|.$$

This has exactly the same form as (6.15) and the proof of convergence proceeds exactly as from there.

6.4 Zero-stability of linear multistep methods

The convergence proof of the previous section shows that for one-step methods, each one-step error $k\tau^{m-1}$ has an effect on the global error that is bounded by $e^{L'T}|k\tau^{m-1}|$. Although the error is possibly amplified by a factor $e^{L'T}$, this factor is bounded independent of k as $k \to 0$. Consequently the method is stable: the global error can be bounded in terms of the sum of all the one-step errors and hence has the same asymptotic behavior as the LTE as $k \to 0$. This form of stability is often called *zero-stability* in ODE theory, to distinguish it from other forms of stability that are of equal importance in practice. The fact that a method is zero-stable (and converges as $k \to 0$) is no guarantee that it will give reasonable results on the particular grid with $k > 0$ that we want to use in practice. Other "stability" issues of a different nature will be taken up in the next chapter.

But first we will investigate the issue of zero-stability for general LMMs, where the theory of the previous section does not apply directly. We begin with an example showing a consistent LMM that is *not* convergent. Examining what goes wrong will motivate our definition of zero-stability for LMMs.

Example 6.2. The LMM

$$U^{n+2} - 3U^{n+1} + 2U^n = -kf(U^n) \tag{6.19}$$

has an LTE given by

$$\tau^n = \frac{1}{k}[u(t_{n+2}) - 3u(t_{n+1}) + 2u(t_n) + ku'(t_n)] = \frac{5}{2}ku''(t_n) + O(k^2),$$

so the method is consistent and "first order accurate." But in fact the global error will not exhibit first order accuracy, or even convergence, in general. This can be seen even on the trivial initial-value problem

$$u'(t) = 0, \quad u(0) = 0 \tag{6.20}$$

with solution $u(t) \equiv 0$. In this problem, equation (6.19) takes the form

$$U^{n+2} - 3U^{n+1} + 2U^n = 0. \tag{6.21}$$

We need two starting values U^0 and U^1. If we take $U^0 = U^1 = 0$, then (6.21) generates $U^n = 0$ for all n and in this case we certainly converge to correct solution, and in fact we get the exact solution for any k.

But in general we will not have the exact value U^1 available and will have to approximate this, introducing some error into the computation. Table 6.1 shows results obtained by applying this method with starting data $U^0 = 0$, $U^1 = k$. Since $U^1(k) \to 0$ as $k \to 0$, this is valid starting data in the context of Definition 6.1 of convergence. If the method is convergent, we should see that U^N, the computed solution at time $T = 1$, converges to zero as $k \to 0$. Instead it blows up quite dramatically. Similar results would be seen if we applied this method to an arbitrary equation $u' = f(u)$ and used any one-step method to compute U^1 from U^0.

The homogeneous linear difference equation (6.21) can be solved explicitly for U^n in terms of the starting values U^0 and U^1. We obtain

$$U^n = 2U^0 - U^1 + 2^n(U^1 - U^0). \tag{6.22}$$

Table 6.1. *Solution U^N to* (6.21) *with $U^0 = 0$, $U^1 = k$ and various values of $k = 1/N$.*

N	U^N
5	6.2
10	1023
20	5.4×10^4

It is easy to verify that this satisfies (6.21) and also the starting values. (We'll see how to solve general linear difference equations in the next section.)

Since $u(t) = 0$, the error is $E^n = U^n$ and we see that any initial errors in U^1 or U^0 are magnified by a factor 2^n in the global error (except in the special case $U^1 = U^0$). This exponential growth of the error is the instability that leads to nonconvergence. To rule out this sort of growth of errors, we need to be able to solve a general linear difference equation.

6.4.1 Solving linear difference equations

We briefly review one solution technique for linear difference equations; see Section D.2.1 for a different approach. Consider the general homogeneous linear difference equation

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = 0. \tag{6.23}$$

Eventually we will look for a particular solution satisfying given initial conditions

$$U^0, U^1, \ldots, U^{r-1},$$

but to begin with we will find the general solution of the difference equation in terms of r free parameters. We will hypothesize that this equation has a solution of the form

$$U^n = \zeta^n \tag{6.24}$$

for some value of ζ (here ζ^n is the nth power!). Plugging this into (6.23) gives

$$\sum_{j=0}^{r} \alpha_j \zeta^{n+j} = 0$$

and dividing by ζ^n yields

$$\sum_{j=0}^{r} \alpha_j \zeta^j = 0. \tag{6.25}$$

We see that (6.24) is a solution of the difference equation if ζ satisfies (6.25), i.e., if ζ is a root of the polynomial

$$\rho(\zeta) = \sum_{j=0}^{r} \alpha_j \zeta^j.$$

Note that this is just the first characteristic polynomial of the LMM introduced in (5.49). In general $\rho(\zeta)$ has r roots $\zeta_1,\ \zeta_2,\ \ldots,\ \zeta_r$ and can be factored as

$$\rho(\zeta) = \alpha_r(\zeta - \zeta_1)(\zeta - \zeta_2)\cdots(\zeta - \zeta_r).$$

Since the difference equation is linear, any linear combination of solutions is again a solution. If $\zeta_1,\ \zeta_2,\ \ldots,\ \zeta_r$ are distinct ($\zeta_i \neq \zeta_j$ for $i \neq j$), then the r distinct solutions ζ_i^n are linearly independent and the general solution of (6.23) has the form

$$U^n = c_1\zeta_1^n + c_2\zeta_2^n + \cdots + c_r\zeta_r^n, \tag{6.26}$$

where $c_1,\ \ldots,\ c_r$ are arbitrary constants. In this case, every solution of the difference equation (6.23) has this form. If initial conditions $U^0,\ U^1,\ \ldots,\ U^{r-1}$ are specified, then the constants $c_1,\ \ldots,\ c_r$ can be uniquely determined by solving the $r \times r$ linear system

$$\begin{aligned} c_1 + c_2 + \cdots + c_r &= U^0, \\ c_1\zeta_1 + c_2\zeta_2 + \cdots + c_r\zeta_r &= U^1, \\ &\vdots \quad \vdots \\ c_1\zeta_1^{r-1} + c_2\zeta_2^{r-1} + \cdots + c_r\zeta_r^{r-1} &= U^{r-1}. \end{aligned} \tag{6.27}$$

Example 6.3. The characteristic polynomial for the difference equation (6.21) is

$$\rho(\zeta) = 2 - 3\zeta + \zeta^2 = (\zeta - 1)(\zeta - 2) \tag{6.28}$$

with roots $\zeta_1 = 1,\ \zeta_2 = 2$. The general solution has the form

$$U^n = c_1 + c_2 \cdot 2^n$$

and solving for c_1 and c_2 from U^0 and U^1 gives the solution (6.22).

This example indicates that if $\rho(\zeta)$ has any roots that are greater than one in modulus, the method will not be convergent. It turns out that the converse is nearly true: if all the roots have modulus no greater than one, then the method is convergent, with one proviso. There must be no *repeated* roots with modulus equal to one. The next two examples illustrate this.

If the roots are not distinct, say, $\zeta_1 = \zeta_2$ for simplicity, then ζ_1^n and ζ_2^n are not linearly independent and the U^n given by (6.26), while still a solution, is not the most general solution. The system (6.27) would be singular in this case. In addition to ζ_1^n there is also a solution of the form $n\zeta_1^n$ and the general solution has the form

$$U^n = c_1\zeta_1^n + c_2 n\zeta_1^n + c_3\zeta_3^n + \cdots + c_r\zeta_r^n.$$

If in addition $\zeta_3 = \zeta_1$, then the third term would be replaced by $c_3 n^2\zeta_1^n$. Similar modifications are made for any other repeated roots. Note how similar this theory is to the standard solution technique for an rth order linear ODE.

Example 6.4. Applying the consistent LMM

$$U^{n+2} - 2U^{n+1} + U^n = \frac{1}{2}k(f(U^{n+2}) - f(U^n)) \tag{6.29}$$

to the differential equation $u'(t) = 0$ gives the difference equation

$$U^{n+2} - 2U^{n+1} + U^n = 0.$$

The characteristic polynomial is

$$\rho(\zeta) = \zeta^2 - 2\zeta + 1 = (\zeta - 1)^2 \tag{6.30}$$

so $\zeta_1 = \zeta_2 = 1$. The general solution is

$$U^n = c_1 + c_2 n.$$

For particular starting values U^0 and U^1 the solution is

$$U^n = U^0 + (U^1 - U^0)n.$$

Again we see that the solution grows with n, although not as dramatically as in Example 6.2 (the growth is linear rather than exponential). But this growth is still enough to destroy convergence. If we take the same starting values as before, $U^0 = 0$ and $U^1 = k$, then $U^n = kn$ and so

$$\lim_{\substack{k \to 0 \\ Nk = T}} U^N = kN = T.$$

The method converges to the function $v(t) = t$ rather than to $u(t) = 0$, and hence the LMM (6.29) is not convergent.

This example shows that if $\rho(\zeta)$ has a *repeated* root of modulus 1, then the method cannot be convergent.

Example 6.5. Now consider the consistent LMM

$$U^{n+3} - 2U^{n+2} + \frac{5}{4}U^{n+1} - \frac{1}{4}U^n = \frac{1}{4}hf(U^n). \tag{6.31}$$

Applying this to (6.20) gives

$$U^{n+3} - 2U^{n+2} + \frac{5}{4}U^{n+1} - \frac{1}{4}U^n = 0$$

and the characteristic polynomial is

$$\rho(\zeta) = \zeta^3 - 2\zeta^2 + \frac{5}{4}\zeta - \frac{1}{4} = (\zeta - 1)(\zeta - 0.5)^2. \tag{6.32}$$

So $\zeta_1 = 1$, $\zeta_2 = \zeta_3 = 1/2$ and the general solution is

$$U^n = c_1 + c_2\left(\frac{1}{2}\right)^n + c_3 n\left(\frac{1}{2}\right)^n.$$

Here there is a repeated root but with modulus less than 1. The linear growth of n is then overwhelmed by the decay of $(1/2)^n$.

For this three-step method we need three starting values U^0, U^1, U^2 and we can find c_1, c_2, c_3 in terms of them by solving a linear system similar to (6.27). Each c_i will

be a linear combination of U^0, U^1, U^2 and so if $U^\nu(k) \to 0$ as $k \to 0$, then $c_i(k) \to 0$ as $k \to 0$ also. The value U^N computed at time T with step size k (where $kN = T$) has the form

$$U^N = c_1(k) + c_2(k)\left(\frac{1}{2}\right)^N + c_3(k)N\left(\frac{1}{2}\right)^N. \tag{6.33}$$

Now we see that

$$\lim_{\substack{k\to 0 \\ Nk=T}} U^N = 0$$

and so the method (6.31) converges on $u' = 0$ with arbitrary starting values $U^\nu(k)$ satisfying $U^\nu(k) \to 0$ as $k \to 0$. (In fact, this LMM is convergent in general.)

More generally, if $\rho(\zeta)$ has a root ζ_j that is repeated m times, then U^N will involve terms of the form $N^s\zeta_j^N$ for $s = 0, 1, \ldots, m-1$. This converges to zero as $N \to \infty$ provided $|\zeta_j| < 1$. The algebraic growth of N^s is overwhelmed by the exponential decay of ζ_j^N. This shows that repeated roots are not a problem as long as they have magnitude strictly less than 1.

With the above examples as motivation, we are ready to state the definition of zero-stability.

Definition 6.2. *An r-step LMM is said to be zero-stable if the roots of the characteristic polynomial $\rho(\zeta)$ defined by* (5.49) *satisfy the following conditions:*

$$\begin{aligned} &|\zeta_j| \le 1 \quad \text{for } j = 1, 2, , \ldots, r. \\ &\text{If } \zeta_j \text{ is a repeated root, then } |\zeta_j| < 1. \end{aligned} \tag{6.34}$$

If the conditions (6.34) are satisfied for all roots of ρ, then the polynomial is said to satisfy the *root condition*.

Example 6.6. The Adams methods have the form

$$U^{n+r} = U^{n+r-1} + k\sum_{j=1}^{r} \beta_j f(U^{n+j})$$

and hence

$$\rho(\zeta) = \zeta^r - \zeta^{r-1} = (\zeta - 1)\zeta^{r-1}.$$

The roots are $\zeta_1 = 1$ and $\zeta_2 = \cdots = \zeta_r = 0$. The root condition is clearly satisfied and all the Adams–Bashforth and Adams–Moulton methods are zero-stable.

The given examples certainly do not prove that zero-stability as defined above is a sufficient condition for convergence. We looked at only the simplest possible ODE $u'(t) = 0$ and saw that things could go wrong if the root condition is *not* satisfied. It turns out, however, that the root condition is all that is needed to prove convergence on the general initial value problem (in the sense of Definition 6.1).

Theorem 6.3 (Dahlquist [22]). *For LMMs applied to the initial value problem for $u'(t) = f(u(t), t)$,*

$$\textit{consistency} \;+\; \textit{zero-stability} \iff \textit{convergence}. \tag{6.35}$$

This is the analogue of the statement (2.21) for the BVP. A proof of this result can be found in [43].

Note: A consistent LMM always has one root equal to 1, say, $\zeta_1 = 1$, called the *principal root*. This follows from (5.50). Hence a consistent one-step LMM (such as Euler, backward Euler, trapezoidal) is certainly zero-stable. More generally we have proved in Section 6.3.4 that any consistent one-step method (that is a Lipschitz continuous) is convergent. Such methods are automatically "zero-stable" and behave well as $k \to 0$. We can think of zero-stability as meaning "stable in the limit as $k \to 0$."

Although a consistent zero-stable method is convergent, it may have other stability problems that show up if the time step k is chosen too large in an actual computation. Additional stability considerations are the subject of the next chapter.

Chapter 7
Absolute Stability for Ordinary Differential Equations

7.1 Unstable computations with a zero-stable method

In the last chapter we investigated zero-stability, the form of stability needed to guarantee convergence of a numerical method as the grid is refined ($k \to 0$). In practice, however, we are not able to compute this limit. Instead we typically perform a single calculation with some particular nonzero time step k (or some particular sequence of time steps with a variable step size method). Since the expense of the computation increases as k decreases, we generally want to choose the time step as large as possible consistent with our accuracy requirements. How can we estimate the size of k required?

Recall that if the method is stable in an appropriate sense, then we expect the global error to be bounded in terms of the local truncation errors at each step, and so we can often use the local truncation error to estimate the time step needed, as illustrated below. But the form of stability now needed is something stronger than zero-stability. We need to know that the error is well behaved for the particular time step we are now using. It is little help to know that things will converge in the limit "for k sufficiently small." The potential difficulties are best illustrated with some examples.

Example 7.1. Consider the initial value problem (IVP)

$$u'(t) = -\sin t, \qquad u(0) = 1$$

with solution

$$u(t) = \cos t.$$

Suppose we wish to use Euler's method to solve this problem up to time $T = 2$. The local truncation error (LTE) is

$$\begin{aligned} \tau(t) &= \frac{1}{2}ku''(t) + O(k^2) \\ &= -\frac{1}{2}k\cos(t) + O(k^2). \end{aligned} \tag{7.1}$$

Since the function $f(t) = -\sin t$ is independent of u, it is Lipschitz continuous with Lipschitz constant $L = 0$, and so the error estimate (6.12) shows that

$$|E^n| \le T\|\tau\|_\infty = k \max_{0\le t\le T} |\cos t| = k.$$

Suppose we want to compute a solution with $|E| \le 10^{-3}$. Then we should be able to take $k = 10^{-3}$ and obtain a suitable solution after $T/k = 2000$ time steps. Indeed, calculating using $k = 10^{-3}$ gives a computed value $U^{2000} = -0.415692$ with an error $E^{2000} = U^{2000} - \cos(2) = 0.4548 \times 10^{-3}$.

Example 7.2. Now suppose we modify the above equation to

$$u'(t) = \lambda(u - \cos t) - \sin t, \tag{7.2}$$

where λ is some constant. If we take the same initial data as before, $u(0) = 1$, then the solution is also the same as before, $u(t) = \cos t$. As a concrete example, let's take $\lambda = -10$. Now how small do we need to take k to get an error that is 10^{-3}? Since the LTE (7.1) depends only on the true solution $u(t)$, which is unchanged from Example 7.1, we might hope that we could use the same k as in that example, $k = 10^{-3}$. Solving the problem using Euler's method with this step size now gives $U^{2000} = -0.416163$ with an error $E^{2000} = 0.161 \times 10^{-4}$. We are again successful. In fact, the error is considerably smaller in this case than in the previous example, for reasons that will become clear later.

Example 7.3. Now consider the problem (7.2) with $\lambda = -2100$ and the same data as before. Again the solution is unchanged and so is the LTE. But now if we compute with the same step size as before, $k = 10^{-3}$, we obtain $U^{2000} = -0.2453 \times 10^{77}$ with an error of magnitude 10^{77}. The computation behaves in an "unstable" manner, with an error that grows exponentially in time. Since the method is zero-stable and $f(u,t)$ is Lipschitz continuous in u (with Lipschitz constant $L = 2100$), we know that the method is convergent, and indeed with sufficiently small time steps we achieve very good results. Table 7.1 shows the error at time $T = 2$ when Euler's method is used with various values of k. Clearly something dramatic happens between the values $k = 0.000976$ and $k = 0.000952$. For smaller values of k we get very good results, whereas for larger values of k there is no accuracy whatsoever.

The equation (7.2) is a linear equation of the form (6.3) and so the analysis of Section 6.3.1 applies directly to this problem. From (6.7) we see that the global error E^n satisfies the recursion relation

$$E^{n+1} = (1 + k\lambda)E^n - k\tau^n, \tag{7.3}$$

where the local error $\tau^n = \tau(t_n)$ from (7.1). The expression (7.3) reveals the source of the exponential growth in the error—in each time step the previous error is multiplied by a factor of $(1 + k\lambda)$. For the case $\lambda = -2100$ and $k = 10^{-3}$, we have $1 + k\lambda = -1.1$ and so we expect the local error introduced in step m to grow by a factor of $(-1.1)^{n-m}$ by the end of n steps (recall (6.8)). After 2000 steps we expect the truncation error introduced in the first step to have grown by a factor of roughly $(-1.1)^{2000} \approx 10^{82}$, which is consistent with the error actually seen.

Note that in Example 7.2 with $\lambda = -10$, we have $1 + k\lambda = 0.99$, causing a *decay* in the effect of previous errors in each step. This explains why we got a reasonable result in Example 7.2 and in fact a better result than in Example 7.1, where $1 + k\lambda = 1$.

Table 7.1. *Errors in the computed solution using Euler's method for Example* 7.3*, for different values of the time step* k*. Note the dramatic change in behavior of the error for* $k < 0.000952$*.*

k	Error
0.001000	0.145252E+77
0.000976	0.588105E+36
0.000950	0.321089E-06
0.000800	0.792298E-07
0.000400	0.396033E-07

Returning to the case $\lambda = -2100$, we expect to observe exponential growth in the error for any value of k greater than $2/2100 = 0.00095238$, since for any k larger than this we have $|1 + k\lambda| > 1$. For smaller time steps $|1 + k\lambda| < 1$ and the effect of each local error decays exponentially with time rather than growing. This explains the dramatic change in the behavior of the error that we see as we cross the value $k = 0.00095238$ in Table 7.1.

Note that the exponential growth of errors does not contradict zero-stability or convergence of the method in any way. The method does converge as $k \to 0$. In fact the bound (6.12),

$$|E^n| \le e^{|\lambda|T} T \|\tau\|_\infty = O(k) \quad \text{as } k \to 0,$$

that we used to prove convergence allows the possibility of exponential growth with time. The bound is valid for all k, but since $Te^{|\lambda|T} = 2e^{4200} = 10^{1825}$ while $\|\tau\|_\infty = \frac{1}{2}k$, this bound does not guarantee any accuracy whatsoever in the solution until $k < 10^{-1825}$. This is a good example of the fact that a mathematical convergence proof may be a far cry from what is needed in practice.

7.2 Absolute stability

To determine whether a numerical method will produce reasonable results with a given value of $k > 0$, we need a notion of stability that is different from zero-stability. There are a wide variety of other forms of "stability" that have been studied in various contexts. The one that is most basic and suggests itself from the above examples is *absolute stability*. This notion is based on the linear test equation (6.3), although a study of the absolute stability of a method yields information that is typically directly useful in determining an appropriate time step in nonlinear problems as well; see Section 7.4.3.

We can look at the simplest case of the test problem in which $g(t) = 0$ and we have simply

$$u'(t) = \lambda u(t).$$

Euler's method applied to this problem gives

$$U^{n+1} = (1 + k\lambda)U^n$$

and we say that this method is *absolutely stable* when $|1+k\lambda| \leq 1$; otherwise it is unstable. Note that there are two parameters k and λ, but only their product $z \equiv k\lambda$ matters. The method is stable whenever $-2 \leq z \leq 0$, and we say that the *interval of absolute stability* for Euler's method is $[-2, 0]$.

It is more common to speak of the *region of absolute stability* as a region in the complex z plane, allowing the possibility that λ is complex (of course the time step k should be real and positive). The region of absolute stability (or simply the *stability region*) for Euler's method is the disk of radius 1 centered at the point -1, since within this disk we have $|1 + k\lambda| \leq 1$ (see Figure 7.1a). Allowing λ to be complex comes from the fact that in practice we are usually solving a system of ordinary differential equations (ODEs). In the linear case it is the eigenvalues of the coefficient matrix that are important in determining stability. In the nonlinear case we typically linearize (see Section 7.4.3) and consider the eigenvalues of the Jacobian matrix. Hence λ represents a typical eigenvalue and these may be complex even if the matrix is real. For some problems, looking at the eigenvalues is not sufficient (see Section 10.12.1, for example), but eigenanalysis is generally very revealing.

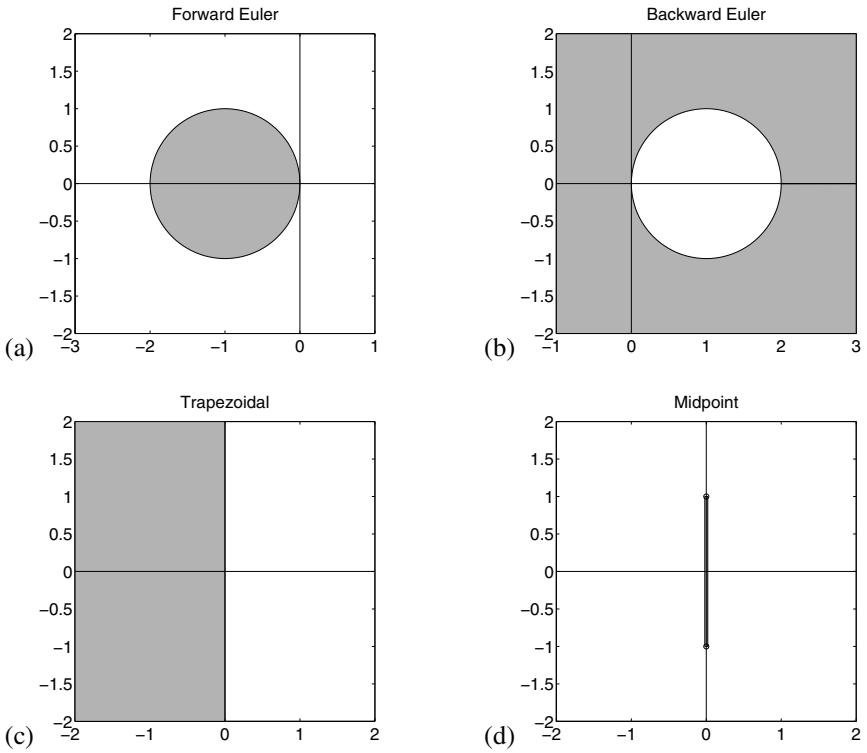

Figure 7.1. *Stability regions for* (a) *Euler,* (b) *backward Euler,* (c) *trapezoidal, and* (d) *midpoint (a segment on imaginary axis).*

7.3 Stability regions for linear multistep methods

For a general linear multistep method (LMM) of the form (5.44), the region of absolute stability is found by applying the method to $u' = \lambda u$, obtaining

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = k \sum_{j=0}^{r} \beta_j \lambda U^{n+j},$$

which can be rewritten as

$$\sum_{j=0}^{r} (\alpha_j - z\beta_j) U^{n+j} = 0. \tag{7.4}$$

Note again that it is only the product $z = k\lambda$ that is important, not the values of k or λ separately, and that this is a dimensionless quantity since the decay rate λ has dimensions time^{-1}, while the time step has dimensions of time. This makes sense—if we change the units of time (say, from seconds to milliseconds), then the parameter λ will decrease by a factor of 1000 and we may be able to increase the numerical value of k by a factor of 1000 and still be stable. But then we also have to solve out to time $1000T$ instead of to time T, so we haven't really changed the numerical problem or the number of time steps required.

The recurrence (7.4) is a homogeneous linear difference equation of the same form considered in Section 6.4.1. The solution has the general form (6.26), where the ζ_j are now the roots of the characteristic polynomial $\sum_{j=0}^{r} (\alpha_j - z\beta_j)\zeta^j$. This polynomial is often called the *stability polynomial* and denoted by $\pi(\zeta; z)$. It is a polynomial in ζ but its coefficients depend on the value of z. The stability polynomial can be expressed in terms of the characteristic polynomials for the LMM as

$$\pi(\zeta; z) = \rho(\zeta) - z\sigma(\zeta). \tag{7.5}$$

The LMM is absolutely stable for a particular value of z if errors introduced in one time step do not grow in future time steps. According to the theory of Section 6.4.1, this requires that the polynomial $\pi(\zeta; z)$ satisfy the root condition (6.34).

Definition 7.1. *The* region of absolute stability *for the LMM* (5.44) *is the set of points z in the complex plane for which the polynomial $\pi(\zeta; z)$ satisfies the root condition* (6.34).

Note that an LMM is zero-stable if and only if the origin $z = 0$ lies in the stability region.

Example 7.4. For Euler's method,

$$\pi(\zeta; z) = \zeta - (1 + z)$$

with the single root $\zeta_1 = 1 + z$. We have already seen that the stability region is the disk in Figure 7.1(a).

Example 7.5. For the backward Euler method (5.21),

$$\pi(\zeta; z) = (1 - z)\zeta - 1$$

with root $\zeta_1 = (1-z)^{-1}$. We have

$$|(1-z)^{-1}| \le 1 \iff |1-z| \ge 1$$

so the stability region is the *exterior* of the disk of radius 1 centered at $z = 1$, as shown in Figure 7.1(b).

Example 7.6. For the trapezoidal method (5.22),

$$\pi(\zeta; z) = \left(1 - \frac{1}{2}z\right)\zeta - \left(1 + \frac{1}{2}z\right)$$

with root

$$\zeta_1 = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}.$$

This is a linear fractional transformation and it can be shown that

$$|\zeta_1| \le 1 \iff \mathrm{Re}(z) \le 0,$$

where $\mathrm{Re}(z)$ is the real part. So the stability region is the left half-plane as shown in Figure 7.1(c).

Example 7.7. For the midpoint method (5.23),

$$\pi(\zeta; z) = \zeta^2 - 2z\zeta - 1.$$

The roots are $\zeta_{1,2} = z \pm \sqrt{z^2 + 1}$. It can be shown that if z is a pure imaginary number of the form $z = i\alpha$ with $|\alpha| < 1$, then $|\zeta_1| = |\zeta_2| = 1$ and $\zeta_1 \ne \zeta_2$, and hence the root condition is satisfied. For any other z the root condition is not satisfied. In particular, if $z = \pm i$, then $\zeta_1 = \zeta_2$ is a repeated root of modulus 1. So the stability region consists only of the open interval from $-i$ to i on the imaginary axis, as shown in Figure 7.1(d).

Since k is always real, this means the midpoint method is useful only on the test problem $u' = \lambda u$ if λ is pure imaginary. The method is not very useful for scalar problems where λ is typically real, but the method is of great interest in some applications with systems of equations. For example, if the matrix is real but skew symmetric ($A^T = -A$), then the eigenvalues are pure imaginary. This situation arises naturally in the discretization of hyperbolic partial differential equations (PDEs), as discussed in Chapter 10.

Example 7.8. Figures 7.2 and 7.3 show the stability regions for the r-step Adams–Bashforth and Adams–Moulton methods for various values of r. For an r-step method the polynomial $\pi(\zeta; z)$ has degree r and there are r roots. Determining the values of z for which the root condition is satisfied does not appear simple. However, there is a simple technique called the *boundary locus method* that makes it possible to determine the regions shown in the figures. This is briefly described in Section 7.6.1.

Note that for many methods the shape of the stability region near the origin $z = 0$ is directly related to the accuracy of the method. Recall that the stability polynomial $\rho(\zeta)$ for a consistent LMM always has a principal root $\zeta_1 = 1$. It can be shown that for z near 0 the polynomial $\pi(\zeta; z)$ has a corresponding principal root with behavior

$$\zeta_1(z) = e^z + O(z^{p+1}) \quad \text{as } z \to 0 \tag{7.6}$$

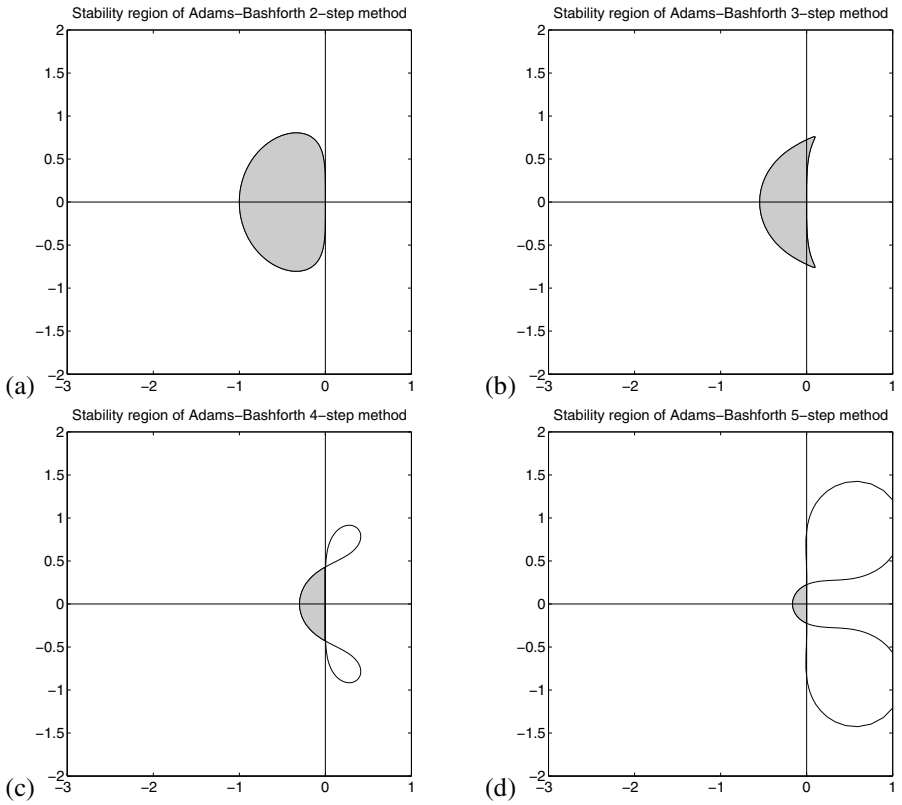

Figure 7.2. *Stability regions for some Adams–Bashforth methods. The shaded region just to the left of the origin is the region of absolute stability. See* Section 7.6.1 *for a discussion of the other loops seen in figures (c) and (d).*

if the method is pth order accurate. We can see this in the examples above for one-step methods, e.g., for Euler's method $\zeta_1(z) = 1 + z = e^z + O(z^2)$. It is this root that is giving the appropriate behavior $U^{n+1} \approx e^z U^n$ over a time step. Since this root is on the unit circle at the origin $z = 0$, and since $|e^z| < 1$ only when $\mathrm{Re}(z) < 0$, we expect the principal root to move inside the unit circle for small z with $\mathrm{Re}(z) < 0$ and outside the unit circle for small z with $\mathrm{Re}(z) > 0$. This suggests that if we draw a small circle around the origin, then the left half of this circle will lie inside the stability region (unless some other root moves outside, as happens for the midpoint method), while the right half of the circle will lie outside the stability region. Looking at the stability regions in Figure 7.1 we see that this is indeed true for all the methods except the midpoint method. Moreover, the higher the order of accuracy in general, the larger a circle around the origin where this will approximately hold, and so the boundary of the stability region tends to align with the imaginary axis farther and farther from the origin as the order of the method increases, as observed in Figures 7.2 and 7.3. (The trapezoidal method is a bit of an anomaly, as its stability region exactly agrees with that of e^z for all z.)

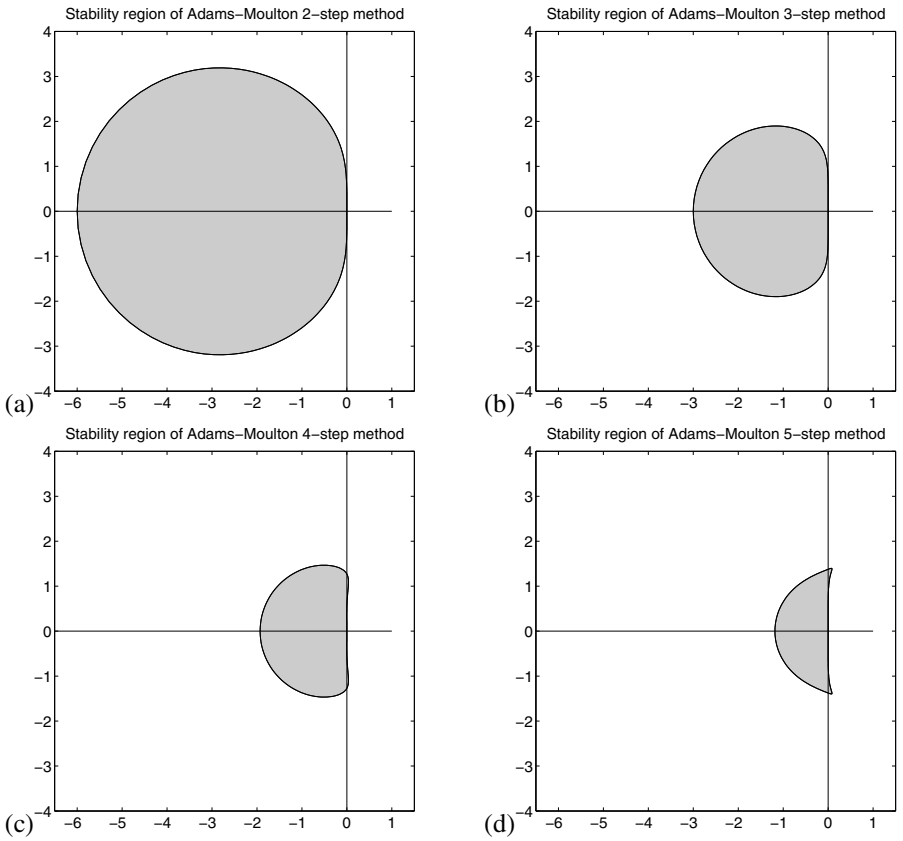

Figure 7.3. *Stability regions for some Adams–Moulton methods.*

See Section 7.6 for a discussion of ways in which stability regions can be determined and plotted.

7.4 Systems of ordinary differential equations

So far we have examined stability theory only in the context of a scalar differential equation $u'(t) = f(u(t))$ for a scalar function $u(t)$. In this section we will look at how this stability theory carries over to systems of m differential equations where $u(t) \in \mathbb{R}^m$. For a linear system $u' = Au$, where A is an $m \times m$ matrix, the solution can be written as $u(t) = e^{At}u(0)$ and the behavior is largely governed by the eigenvalues of A. A necessary condition for stability is that $k\lambda$ be in the stability region for each eigenvalue λ of A. For general nonlinear systems $u' = f(u)$, the theory is more complicated, but a good rule of thumb is that $k\lambda$ should be in the stability region for each eigenvalue λ of the Jacobian matrix $f'(u)$. This may not be true if the Jacobian is rapidly changing with time, or even for constant coefficient linear problems in some highly nonnormal cases (see [47] and Section 10.12.1 for an example), but most of the time eigenanalysis is surprisingly effective.

Before discussing this theory further we will review the theory of chemical kinetics, a field where the solution of systems of ODEs is very important, and where the eigenvalues of the Jacobian matrix often have a physical interpretation in terms of reaction rates.

7.4.1 Chemical kinetics

Let A and B represent chemical compounds and consider a reaction of the form

$$A \overset{K_1}{\to} B.$$

This represents a reaction in which A is transformed into B with rate $K_1 > 0$. If we let u_1 represent the concentration of A and u_2 represent the concentration of B (often denoted by $u_1 = [A],\ u_2 = [B]$), then the ODEs for u_1 and u_2 are

$$\begin{aligned} u_1' &= -K_1 u_1, \\ u_2' &= K_1 u_1. \end{aligned}$$

If there is also a reverse reaction at rate K_2, we write

$$A \underset{K_2}{\overset{K_1}{\rightleftharpoons}} B$$

and the equations then become

$$\begin{aligned} u_1' &= -K_1 u_1 + K_2 u_2, \\ u_2' &= K_1 u_1 - K_2 u_2. \end{aligned} \tag{7.7}$$

More typically, reactions involve combinations of two or more compounds, e.g.,

$$A + B \underset{K_2}{\overset{K_1}{\rightleftharpoons}} AB.$$

Since A and B must combine to form AB, the rate of the forward reaction is proportional to the *product* of the concentrations u_1 and u_2, while the backward reaction is proportional to $u_3 = [AB]$. The equations become

$$\begin{aligned} u_1' &= -K_1 u_1 u_2 + K_2 u_3, \\ u_2' &= -K_1 u_1 u_2 + K_2 u_3, \\ u_3' &= K_1 u_1 u_2 - K_2 u_3. \end{aligned} \tag{7.8}$$

Note that this is a nonlinear system of equations, while (7.7) are linear.

Often several reactions take place simultaneously, e.g.,

$$\begin{aligned} A + B &\underset{K_2}{\overset{K_1}{\rightleftharpoons}} AB, \\ 2A + C &\underset{K_4}{\overset{K_3}{\rightleftharpoons}} A_2C. \end{aligned}$$

If we now let $u_4 = [C],\ u_5 = [A_2C]$, then the equations are

$$\begin{aligned} u_1' &= -K_1u_1u_2 + K_2u_3 - 2K_3u_1^2u_4 + 2K_4u_5, \\ u_2' &= -K_1u_1u_2 + K_2u_3, \\ u_3' &= K_1u_1u_2 - K_2u_3, \\ u_4' &= -K_3u_1^2u_4 + K_4u_5, \\ u_5' &= K_3u_1^2u_4 - K_4u_5. \end{aligned} \tag{7.9}$$

Interesting kinetics problems can give rise to very large systems of ODEs. Frequently the rate constants $K_1,\ K_2,\ \ldots$ are of vastly different orders of magnitude. This leads to *stiff* systems of equations, as discussed in Chapter 8.

Example 7.9. One particularly simple system arises from the decay process

$$A \overset{K_1}{\to} B \overset{K_2}{\to} C.$$

Let $u_1 = [A],\ u_2 = [B],\ u_3 = [C]$. Then the system is linear and has the form $u' = Au$, where

$$A = \begin{bmatrix} -K_1 & 0 & 0 \\ K_1 & -K_2 & 0 \\ 0 & K_2 & 0 \end{bmatrix}. \tag{7.10}$$

Note that the eigenvalues are $-K_1,\ -K_2$, and 0. The general solution thus has the form (assuming $K_1 \neq K_2$)

$$u_j(t) = c_{j1}e^{-K_1t} + c_{j2}e^{-K_2t} + c_{j3}.$$

In fact, on physical grounds (since A decays into B which decays into C), we expect that u_1 simply decays to 0 exponentially,

$$u_1(t) = e^{-K_1t}u_1(0)$$

(which clearly satisfies the first ODE), and also that u_2 ultimately decays to 0 (although it may first grow if K_1 is larger than K_2), while u_3 grows and asymptotically approaches the value $u_1(0) + u_2(0) + u_3(0)$ as $t \to \infty$. A typical solution for $K_1 = 3$ and $K_2 = 1$ with $u_1(0) = 3,\ u_2(0) = 4$, and $u_3(0) = 2$ is shown in Figure 7.4.

7.4.2 Linear systems

Consider a linear system $u' = Au$, where A is a constant $m \times m$ matrix, and suppose for simplicity that A is diagonalizable, which means that it has a complete set of m linearly independent eigenvectors r_p satisfying $Ar_p = \lambda_p r_p$ for $p = 1,\ 2,\ \ldots,\ m$. Let $R = [r_1, r_2, \ldots, r_m]$ be the matrix of eigenvectors and $\Lambda = \text{diag}(\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_m)$ be the diagonal matrix of eigenvectors. Then we have

$$A = R\Lambda R^{-1} \quad \text{and} \quad \Lambda = R^{-1}AR.$$

Now let $v(t) = R^{-1}u(t)$. Multiplying $u' = Au$ by R^{-1} on both sides and introducing $I = RR^{-1}$ gives the equivalent equations

$$R^{-1}u'(t) = (R^{-1}AR)(R^{-1}u(t)),$$

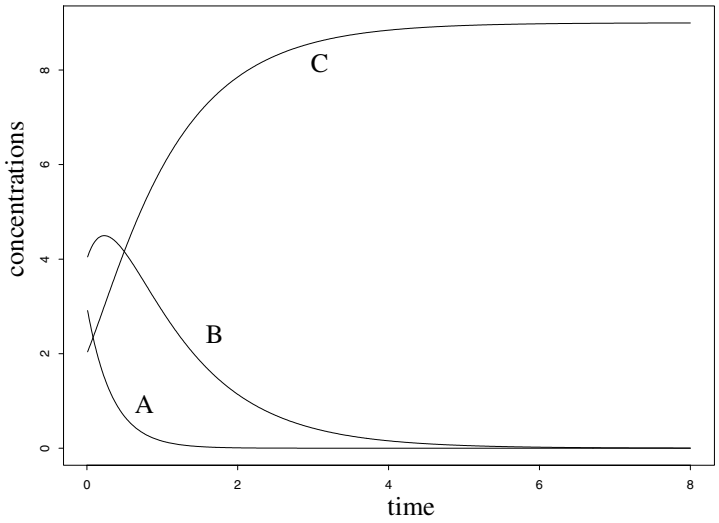

Figure 7.4. *Sample solution for the kinetics problem in Example* 7.9.

i.e.,

$$v'(t) = \Lambda v(t).$$

This is a diagonal system of equations that decouples into m independent scalar equations, one for each component of v. The pth such equation is

$$v_p'(t) = \lambda_p v_p(t).$$

A linear multistep method applied to the linear ODE can also be decoupled in the same way. For example, if we apply Euler's method, we have

$$U^{n+1} = U^n + kAU^n,$$

which, by the same transformation, can be rewritten as

$$V^{n+1} = V^n + k\Lambda V^n,$$

where $V^n = R^{-1}U^n$. This decouples into m independent numerical methods, one for each component of V^n. These take the form

$$V_p^{n+1} = (1 + k\lambda_p)V_p^n.$$

We can recover U^n from V^n using $U^n = RV^n$.

For the overall method to be stable, each of the scalar problems must be stable, and this clearly requires that $k\lambda_p$ be in the stability region of Euler's method for all values of p.

The same technique can be used more generally to show that an LMM can be absolutely stable only if $k\lambda_p$ is in the stability region of the method for each eigenvalue λ_p of the matrix A.

Example 7.10. Consider the linear kinetics problem with A given by (7.10). Since this matrix is upper triangular, the eigenvalues are the diagonal elements $\lambda_1 = -K_1$, $\lambda_2 = -K_2$, and $\lambda_3 = 0$. The eigenvalues are all real and we expect Euler's method to be stable provided $k \max(K_1, K_2) \le 2$. Numerical experiments easily confirm that this is exactly correct: when this condition is satisfied the numerical solution behaves well, and if k is slightly larger there is explosive growth of the error.

Example 7.11. Consider a linearized model for a swinging pendulum, this time with frictional forces added,

$$\theta''(t) = -a\theta(t) - b\theta'(t),$$

which is valid for small values of θ. If we introduce $u_1 = \theta$ and $u_2 = \theta'$ then we obtain a first order system $u' = Au$ with

$$A = \begin{bmatrix} 0 & 1 \\ -a & -b \end{bmatrix}. \tag{7.11}$$

The eigenvalues of this matrix are $\lambda = \frac{1}{2}\big(-b \pm \sqrt{b^2 - 4a}\big)$. Note in particular that if $b = 0$ (no damping), then $\lambda = \pm\sqrt{-a}$ are pure imaginary. For $b > 0$ the eigenvalues shift into the left half-plane. In the undamped case the midpoint method would be a reasonable choice, whereas Euler's method might be expected to have difficulties. In the damped case the opposite is true.

7.4.3 Nonlinear systems

Now consider a nonlinear system $u' = f(u)$. The stability analysis we have developed for the linear problem does not apply directly to this system. However, if the solution is slowly varying relative to the time step, then over a small time interval we would expect a linearized approximation to give a good indication of what is happening. Suppose the solution is near some value $\bar{u}$, and let $v(t) = u(t) - \bar{u}$. Then

$$v'(t) = u'(t) = f(u(t)) = f(v(t) + \bar{u}).$$

Taylor series expansion about $\bar{u}$ (assuming v is small) gives

$$v'(t) = f(\bar{u}) + f'(\bar{u})v(t) + O(\|v\|^2).$$

Dropping the $O(\|v\|^2)$ terms gives a linear system

$$v'(t) = Av(t) + b,$$

where $A = f'(\bar{u})$ is the Jacobian matrix evaluated at $\bar{u}$ and $b = f(\bar{u})$. Examining how the numerical method behaves on this linear system (for each relevant value of $\bar{u}$) gives a good indication of how it will behave on the nonlinear system.

Example 7.12. Consider the kinetics problem (7.8). The Jacobian matrix is

$$A = \begin{bmatrix} -K_1u_2 & -K_1u_1 & K_2 \\ -K_1u_2 & -K_1u_1 & K_2 \\ K_1u_2 & K_1u_1 & -K_2 \end{bmatrix}$$

with eigenvalues $\lambda_1 = -K_1(u_1 + u_2) - K_2$ and $\lambda_2 = \lambda_3 = 0$. Since $u_1 + u_2$ is simply the total quantity of species A and B present, this can be bounded for all time in terms of the initial data. (For example, we certainly have $u_1(t) + u_2(t) \le u_1(0) + u_2(0) + 2u_3(0)$.) So we can determine the possible range of λ_1 along the negative real axis and hence how small k must be chosen so that $k\lambda_1$ stays within the region of absolute stability.

7.5 Practical choice of step size

As the examples at the beginning of this chapter illustrated, obtaining computed results that are within some error tolerance requires two conditions:

1. The time step k must be small enough that the local truncation error is acceptably small. This gives a constraint of the form $k \le k_{\text{acc}}$, where k_{acc} depends on several things:
   - What method is being used, which determines the expansion for the local truncation error;
   - How smooth the solution is, which determines how large the high order derivatives occurring in this expansion are; and
   - What accuracy is required.
2. The time step k must be small enough that the method is absolutely stable on this particular problem. This gives a constraint of the form $k \le k_{\text{stab}}$ that depends on the magnitude and location of the eigenvalues of the Jacobian matrix $f'(u)$.

Typically we would like to choose our time step based on accuracy considerations, so we hope $k_{\text{stab}} > k_{\text{acc}}$. For a given method and problem, we would like to choose k so that the local error in each step is sufficiently small that the accumulated error will satisfy our error tolerance, assuming some "reasonable" growth of errors. If the errors grow exponentially with time because the method is not absolutely stable, however, then we would have to use a smaller time step to get useful results.

If stability considerations force us to use a *much* smaller time step than the local truncation error indicates should be needed, then this particular method is probably not optimal for this problem. This happens, for example, if we try to use an explicit method on a "stiff" problem as discussed in Chapter 8, for which special methods have been developed.

As already noted in Chapter 5, most software for solving initial value problems does a very good job of choosing time steps dynamically as the computation proceeds, based on the observed behavior of the solution and estimates of the local error. If a time step is chosen for which the method is unstable, then the local error estimate will typically indicate a large error and the step size will be automatically reduced. Details of the shape of the stability region and estimates of the eigenvalues are typically *not* used in the course of a computation to choose time steps.

However, the considerations of this chapter play a big role in determining whether a given method or class of methods is suitable for a particular problem. We will also see in Chapters 9 and 10 that a knowledge of the stability regions of ODE methods is necessary in order to develop effective methods for solving time-dependent PDEs.

7.6 Plotting stability regions

7.6.1 The boundary locus method for linear multistep methods

A point $z \in \mathbb{C}$ is in the stability region $\mathcal{S}$ of an LMM if the stability polynomial $\pi(\zeta; z)$ satisfies the root condition for this value of z. It follows that if z is on the *boundary* of the stability region, then $\pi(\zeta; z)$ must have at least one root ζ_j with magnitude exactly equal to 1. This ζ_j is of the form

$$\zeta_j = e^{i\theta}$$

for some value of θ in the interval $[0, 2\pi]$. (Beware of the two different uses of π.) Since ζ_j is a root of $\pi(\zeta; z)$, we have

$$\pi(e^{i\theta}; z) = 0$$

for this particular combination of z and θ. Recalling the definition of π, this gives

$$\rho(e^{i\theta}) - z\sigma(e^{i\theta}) = 0 \tag{7.12}$$

and hence

$$z = \frac{\rho(e^{i\theta})}{\sigma(e^{i\theta})}.$$

If we know θ, then we can find z from this.

Since every point z on the boundary of $\mathcal{S}$ must be of this form for some value of θ in $[0, 2\pi]$, we can simply plot the parametrized curve

$$\tilde{z}(\theta) \equiv \frac{\rho(e^{i\theta})}{\sigma(e^{i\theta})} \tag{7.13}$$

for $0 \le \theta \le 2\pi$ to find the locus of all points which are *potentially* on the boundary of $\mathcal{S}$. For simple methods this yields the region $\mathcal{S}$ directly.

Example 7.13. For Euler's method we have $\rho(\zeta) = \zeta - 1$ and $\sigma(\zeta) = 1$, and so

$$\tilde{z}(\theta) = e^{i\theta} - 1.$$

This function maps $[0, 2\pi]$ to the unit circle centered at $z = -1$, which is exactly the boundary of $\mathcal{S}$ as shown in Figure 7.1(a).

To determine which side of this curve is the *interior* of $\mathcal{S}$, we need only evaluate the roots of $\pi(\zeta; z)$ at some random point z on one side or the other and see if the polynomial satisfies the root condition.

Alternatively, as noted on page 155, most methods are stable just to the left of the origin on the negative real axis and unstable just to the right of the origin on the positive real axis. This is often enough information to determine where the stability region lies relative to the boundary locus.

For some methods the boundary locus may cross itself. In this case we typically find that at most one of the regions cut out of the plane corresponds to the stability region. We can determine which region is $\mathcal{S}$ by evaluating the roots at some convenient point z within each region.

Example 7.14. The five-step Adams–Bashforth method gives the boundary locus seen in Figure 7.2(d). The stability region is the small semicircular region to the left of the

origin where all roots are inside the unit circle. As we cross the boundary of this region one root moves outside. As we cross the boundary locus again into one of the loops in the right half-plane another root moves outside and the method is still unstable in these regions (two roots are outside the unit circle).

7.6.2 Plotting stability regions of one-step methods

If we apply a one-step method to the test problem $u' = \lambda u$, we typically obtain an expression of the form

$$U^{n+1} = R(z)U^n, \tag{7.14}$$

where $R(z)$ is some function of $z = k\lambda$ (typically a polynomial for an explicit method or a rational function for an implicit method). If the method is consistent, then $R(z)$ will be an approximation to e^z near $z = 0$, and if it is pth order accurate, then

$$R(z) - e^z = O(z^{p+1}) \quad \text{as } z \to 0. \tag{7.15}$$

Example 7.15. The pth order Taylor series method, when applied to $u' = \lambda u$, gives (since the jth derivative of u is $u^{(j)} = \lambda^j u$)

$$\begin{aligned} U^{n+1} &= U^n + k\lambda U^n + \frac{1}{2}k^2\lambda^2 U^n + \cdots + \frac{1}{p!}k^p\lambda^p U^n \\ &= \left(1 + z + \frac{1}{2}z^2 + \cdots + \frac{1}{p!}z^p\right) U^n. \end{aligned} \tag{7.16}$$

In this case $R(z)$ is the polynomial obtained from the first $p + 1$ terms of the Taylor series for e^z.

Example 7.16. If the fourth order Runge–Kutta method (5.33) is applied to $u' = \lambda u$, we find that

$$R(z) = 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4, \tag{7.17}$$

which agrees with $R(z)$ for the fourth order Taylor series method.

Example 7.17. For the trapezoidal method (5.22),

$$R(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z} \tag{7.18}$$

is a rational approximation to e^z with error $O(z^3)$ (the method is second order accurate). Note that this is also the root of the linear stability polynomial that we found by viewing this as an LMM in Example 7.6.

Example 7.18. The TR-BDF2 method (5.37) has

$$R(z) = \frac{1 + \frac{5}{12}z}{1 - \frac{7}{12}z + \frac{1}{12}z^2}. \tag{7.19}$$

This agrees with e^z to $O(z^3)$ near $z = 0$.

From the definition of absolute stability given at the beginning of this chapter, we see that the region of absolute stability for a one-step method is simply

$$\mathcal{S} = \{z \in \mathbb{C} : |R(z)| \le 1\}. \tag{7.20}$$

This follows from the fact that iterating a one-step method on $u' = \lambda u$ gives $|U^n| = |R(z)|^n |U^0|$ and this will be uniformly bounded in n if z lies in $\mathcal{S}$.

One way to attempt to compute $\mathcal{S}$ would be to compute the boundary locus as described in Section 7.6.1 by setting $R(z) = e^{i\theta}$ and solving for z as θ varies. This would give the set of z for which $|R(z)| = 1$, the boundary of $\mathcal{S}$. There's a problem with this, however: when $R(z)$ is a higher order polynomial or rational function there will be several solutions z for each θ and it is not clear how to connect these to generate the proper curve.

Another approach can be taken graphically that is more brute force, but effective. If we have a reasonable idea of what region of the complex z-plane contains the boundary of $\mathcal{S}$, we can sample $|R(z)|$ on a fine grid of points in this region and approximate the level set where this function has the value 1 and plot this as the boundary of $\mathcal{S}$. This is easily done with a contour plotter, for example, using the `contour` command in MATLAB. Or we can simply color each point depending on whether it is inside $\mathcal{S}$ or outside.

For example, Figure 7.5 shows the stability regions for the Taylor series methods of orders 2 and 4, for which

$$\begin{aligned} R(z) &= 1 + z + \frac{1}{2}z^2, \\ R(z) &= 1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4, \end{aligned} \tag{7.21}$$

respectively. These are also the stability regions of the second order Runge–Kutta method (5.30) and the fourth order accurate Runge–Kutta method (5.33), which are easily seen to have the same stability functions.

Note that for a one-step method of order p, the rational function $R(z)$ must agree with e^z to $O(z^{p+1})$. As for LMMs, we thus expect that points very close to the origin will lie in the stability region $\mathcal{S}$ for $\mathrm{Re}(z) < 0$ and outside of $\mathcal{S}$ for $\mathrm{Re}(z) > 0$.

7.7 Relative stability regions and order stars

Recall that for a one-step method the stability region $\mathcal{S}$ (more properly called the region of absolute stability) is the region $\mathcal{S} = \{z \in \mathbb{C} : |R(z)| \le 1\}$, where $U^{n+1} = R(z)U^n$ is the relation between U^n and U^{n+1} when the method is applied to the test problem $u' = \lambda u$. For $z = \lambda k$ in the stability region the numerical solution does not grow, and hence the method is absolutely stable in the sense that past errors will not grow in later time steps.

On the other hand, the true solution to this problem, $u(t) = e^{\lambda t}u(0)$, is itself exponentially growing or decaying. One might argue that if $u(t)$ is itself decaying, then it isn't good enough to simply have the past errors decaying, too—they should be decaying at a faster rate. Or conversely, if the true solution is growing exponentially, then perhaps it is fine for the error also to be growing, as long as it is not growing faster.

This suggests defining the *region of relative stability* as the set of $z \in \mathbb{C}$ for which $|R(z)| \le |e^z|$. In fact this idea has not proved to be particularly useful in terms of judging

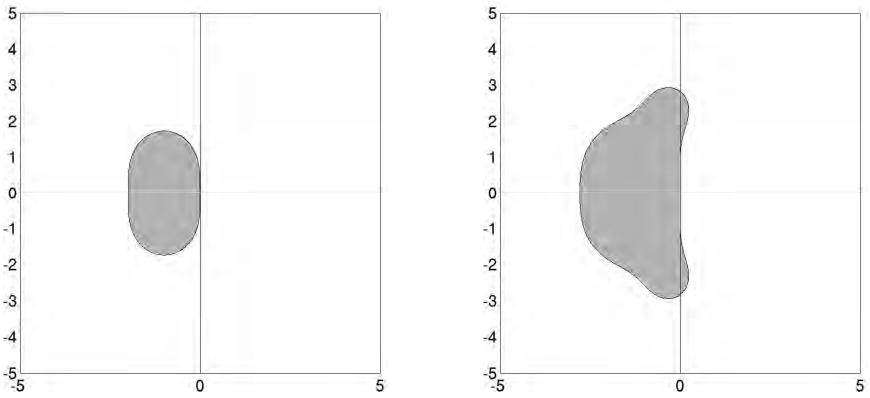

Figure 7.5. *Stability regions for the Taylor series methods of order 2 (left) and 4 (right).*

the practical stability of a method for finite-size time steps; absolute stability is the more useful concept in this regard.

Relative stability regions also proved hard to plot in the days before good computer graphics, and so they were not studied extensively. However, a pivotal 1978 paper by Wanner, Hairer, and Nørsett [99] showed that these regions are very useful in proving certain types of theorems about the relation between stability and the attainable order of accuracy for broad classes of methods. Rather than speaking in terms of regions of relative stability, the modern terminology concerns the *order star* of a rational function $R(z)$, which is the set of three regions $(\mathcal{A}_-,\ \mathcal{A}_0,\ \mathcal{A}_+)$:

$$\begin{aligned} \mathcal{A}_- &= \{z \in \mathbb{C}:\ |R(z)| < |e^z|\} = \{z \in \mathbb{C}:\ |e^{-z}R(z)| < 1\}, \\ \mathcal{A}_0 &= \{z \in \mathbb{C}:\ |R(z)| = |e^z|\} = \{z \in \mathbb{C}:\ |e^{-z}R(z)| = 1\}, \\ \mathcal{A}_+ &= \{z \in \mathbb{C}:\ |R(z)| > |e^z|\} = \{z \in \mathbb{C}:\ |e^{-z}R(z)| > 1\}. \end{aligned} \tag{7.22}$$

These sets turn out to be much more strange looking than regions of absolute stability. As their name implies, they have a star-like quality, as seen, for example, in Figure 7.6, which shows the order stars for the same two Taylor polynomials (7.21), and Figure 7.7, which shows the order stars for two implicit methods. In each case the shaded region is $\mathcal{A}_+$, while the white region is $\mathcal{A}_-$ and the boundary between them is $\mathcal{A}_0$. Their behavior near the origin is directly tied to the order of accuracy of the method, i.e., the degree to which $R(z)$ matches e^z at the origin. If $R(z) = e^z + Cz^{p+1} +$ higher order terms, then since $e^{-z} \approx 1$ near the origin,

$$e^{-z}R(z) \approx 1 + Cz^{p+1}. \tag{7.23}$$

As z traces out a small circle around the origin (say, $z = \delta e^{2\pi i\theta}$ for some small δ), the function $z^{p+1} = \delta^{p+1}e^{2(p+1)\pi i\theta}$ goes around a smaller circle about the origin $p+1$ times and hence crosses the imaginary axis $2(p+1)$ times. Each of these crossings corresponds to z moving across $\mathcal{A}_0$. So in a disk very close to the origin the order star must consist of $p+1$ wedgelike sectors of $\mathcal{A}_+$ separated by $p+1$ sectors of $\mathcal{A}_-$. This is apparent in Figures 7.6 and 7.7.

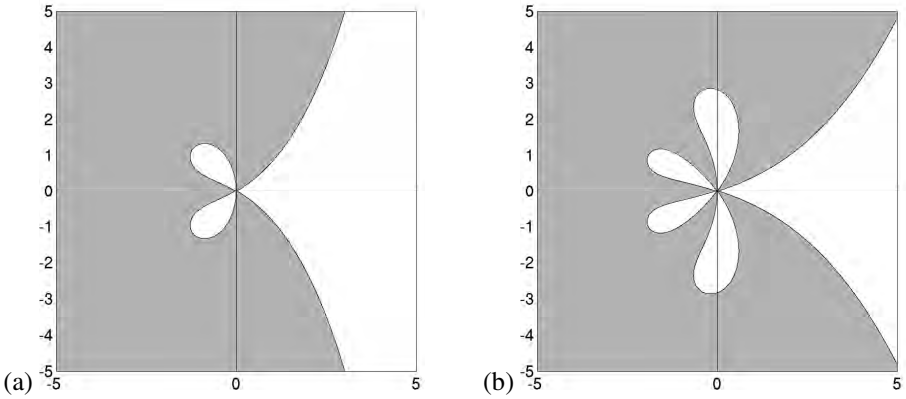

Figure 7.6. *Order stars for the Taylor series methods of order* (a) 2 *and* (b) 4.

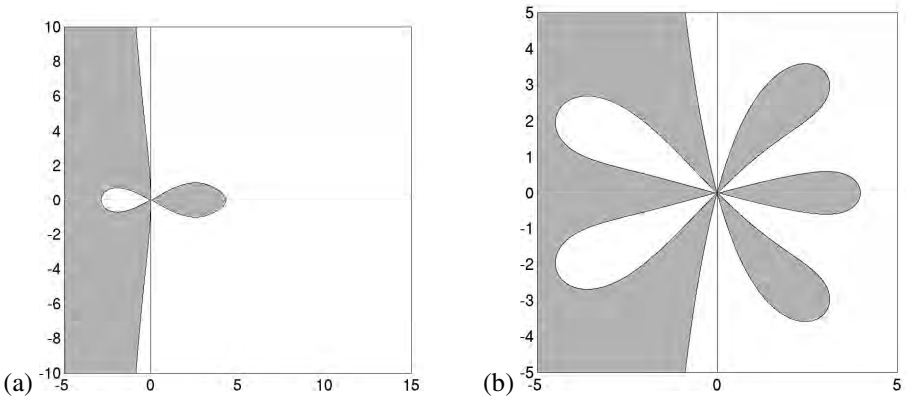

Figure 7.7. *Order stars for two A-stable implicit methods,* (a) *the TR-BDF2 method* (5.37) *with* $R(z)$ *given by* (7.19)*, and* (b) *the fifth-order accurate Radau*5 *method* [44]*, for which* $R(z)$ *is a rational function with degree* 2 *in the numerator and* 3 *in the denominator.*

It can also be shown that each bounded finger of $\mathcal{A}_-$ contains at least one root of the rational function $R(z)$ and each bounded finger of $\mathcal{A}_+$ contains at least one pole. (There are no poles for an explicit method; see Figure 7.6.) Moreover, certain stability properties of the method can be related to the geometry of the order star, facilitating the proof of some "barrier theorems" on the possible accuracy that might be obtained.

This is just a hint of the sort of question that can be tackled with order stars. For a better introduction to their power and beauty, see, for example, [44], [51], [98].

Chapter 8
Stiff Ordinary Differential Equations

The problem of *stiffness* leads to computational difficulty in many practical problems. The classic example is the case of a stiff ordinary differential equation (ODE), which we will examine in this chapter. In general a problem is called *stiff* if, roughly speaking, we are attempting to compute a particular solution that is smooth and slowly varying (relative to the time interval of the computation), but in a context where the nearby solution curves are much more rapidly varying. In other words, if we perturb the solution slightly at any time, the resulting solution curve through the perturbed data has rapid variation. Typically this takes the form of a short-lived "transient" response that moves the solution back toward a smooth solution.

Example 8.1. Consider the ODE (7.2) from the previous chapter,

$$u'(t) = \lambda(u - \cos t) - \sin t. \tag{8.1}$$

One particular solution is the function $u(t) = \cos t$, and this is the solution with the initial data $u(0) = 1$ considered previously. This smooth function is a solution for any value of λ. If we consider initial data of the form $u(t_0) = \eta$ that does not lie on this curve, then the solution through this point is a different function, of course. However, if $\lambda < 0$ (or $\mathrm{Re}(\lambda) < 0$ more generally), this function approaches $\cos t$ exponentially quickly, with decay rate λ. It is easy to verify that the solution is

$$u(t) = e^{\lambda(t-t_0)}(\eta - \cos(t_0)) + \cos t. \tag{8.2}$$

Figure 8.1 shows a number of different solution curves for this equation with different choices of t_0 and η, with the fairly modest value $\lambda = -1$. Figure 8.1b shows the corresponding solution curves when $\lambda = -10$.

In this scalar example, when we perturb the solution at some point it quickly relaxes toward the particular solution $u(t) = \cos t$. In other stiff problems the solution might move quickly toward some different smooth solution, as seen in the next example.

Example 8.2. Consider the kinetics model $A \to B \to C$ developed in Example 7.9. The system of equations is given by (7.10). Suppose that $K_1 \gg K_2$ so that a typical solution appears as in Figure 8.2(a). (Here $K_1 = 20$ and $K_2 = 1$. Compare this to

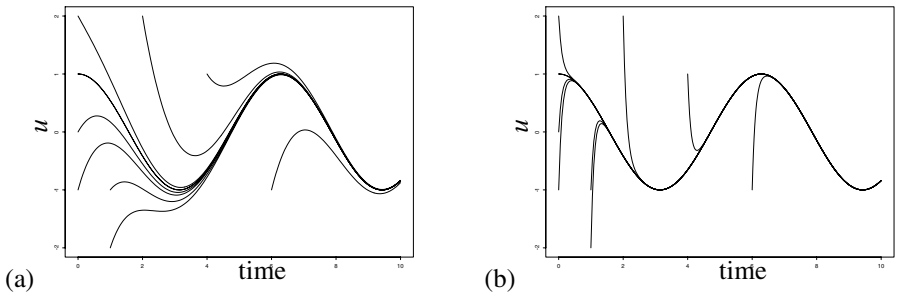

Figure 8.1. *Solution curves for the ODE* (8.1) *for various initial values.* (a) *With* $\lambda = -1$. (b) *With* $\lambda = -10$ *and the same set of initial values.*

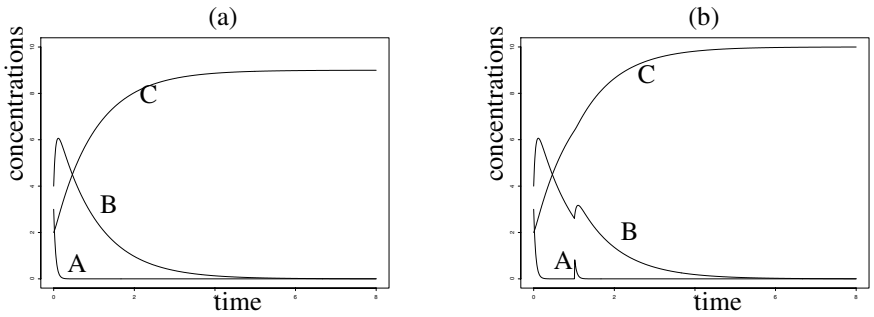

Figure 8.2. *Solution curves for the kinetics problem in Example* 7.9 *with* $K_1 = 20$ *and* $K_2 = 1$. *In* (b) *a perturbation has been made by adding one unit of species* A *at time* $t = 1$. *Figure 8.3 shows similar solutions for the case* $K_1 = 10^6$.

Figure 7.4.) Now suppose at time $t = 1$ we perturb the system by adding more of species A. Then the solution behaves as shown in Figure 8.2(b). The additional A introduced is rapidly converted into B (the fast transient response) and then slowly from B into C. After the rapid transient the solution is again smooth, although it differs from the original solution since the final asymptotic value of C must be higher than before by the same magnitude as the amount of A introduced.

8.1 Numerical difficulties

Stiffness causes numerical difficulties because any finite difference method is constantly introducing errors. The local truncation error acts as a perturbation to the system that moves us away from the smooth solution we are trying to compute. Why does this cause more difficulty in a stiff system than in other systems? At first glance it seems like the stiffness might work to our advantage. If we are trying to compute the solution $u(t) = \cos t$ to the ODE (8.1) with initial data $u(0) = 1$, for example, then the fact that any errors introduced decay exponentially should help us. The true solution is very robust and the solution is almost completely insensitive to errors made in the past. In fact, this *stability* of the true

solution does help us, as long as the numerical method is also stable. (Recall that the results in Example 7.2 were much better than those in Example 7.1.)

The difficulty arises from the fact that many numerical methods, including all explicit methods, are unstable (in the sense of absolute stability) *unless the time step is small relative to the time scale of the rapid transient*, which in a stiff problem is much smaller than the time scale of the smooth solution we are trying to compute. In the terminology of Section 7.5, this means that $k_{\text{stab}} \ll k_{\text{acc}}$. Although the true solution is smooth and it seems that a reasonably large time step would be appropriate, the numerical method must always deal with the rapid transients introduced by truncation error in every time step and may need a very small time step to do so stably.

8.2 Characterizations of stiffness

A stiff ODE can be characterized by the property that $f'(u)$ is much larger (in absolute value or norm) than $u'(t)$. The latter quantity measures the smoothness of the solution being computed, while $f'(u)$ measures how rapidly f varies as we move away from this particular solution. Note that stiff problems typically have large Lipschitz constants too.

For *systems* of ODEs, stiffness is sometimes defined in terms of the "stiffness ratio" of the system, which is the ratio

$$\frac{\max |\lambda_p|}{\min |\lambda_p|} \tag{8.3}$$

over all eigenvalues of the Jacobian matrix $f'(u)$. If this is large, then a large range of time scales is present in the problem, a necessary component for stiffness to arise. While this is often a useful quantity, one should not rely entirely on this measure to determine whether a problem is stiff.

For one thing, it is possible even for a scalar problem to be stiff (as we have seen in Example 8.1), although for a scalar problem the stiffness ratio is always 1 since there is only one eigenvalue. Still, more than one time scale can be present. In (8.1) the fast time scale is determined by λ, the eigenvalue, and the slow time scale is determined by the inhomogeneous term $\sin(t)$. For systems of equations there also may be additional time scales arising from inhomogeneous forcing terms or other time-dependent coefficients that are distinct from the scales imposed by the eigenvalues.

We also know that for highly nonnormal matrices the eigenvalues don't always tell the full story (see Section D.4). Often they give adequate guidance, but there are examples where the problem is more stiff than (8.3) would suggest. An example arises when certain spectral approximations to spatial derivatives are used in discretizing hyperbolic equations; see Section 10.13.

On the other hand, it is also important to note that a system of ODEs which has a large "stiffness ratio" is not necessarily stiff! If the eigenvalue with large amplitude lies close to the imaginary axis, then it leads to highly oscillatory behavior in the solution rather than rapid damping. If the solution is rapidly oscillating, then it will probably be necessary to take small time steps for accuracy reasons and k_{acc} may be roughly the same magnitude as k_{stab} even for explicit methods, at least with the sort of methods discussed here. (Special methods for highly oscillatory problems have been developed that allow one to take larger time steps; see, e.g., [50], [74].)

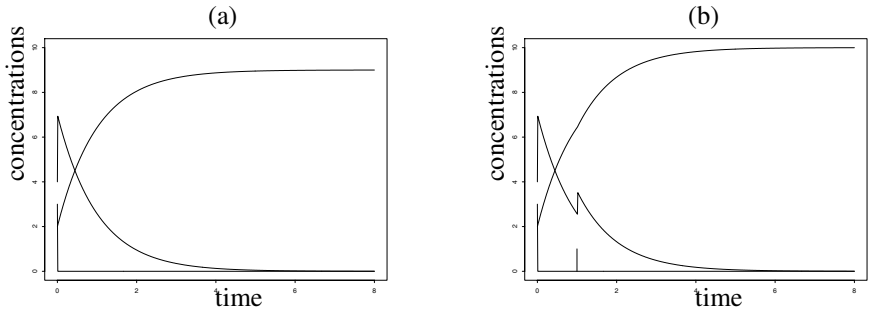

Figure 8.3. *Solution curves for the kinetics problem in Example* 7.9 *with* $K_1 = 10^6$ *and* $K_2 = 1$. *In* (b) *a perturbation has been made by adding one unit of species A at time* $t = 1$.

Finally, note that a particular problem may be stiff over some time intervals and nonstiff elsewhere. In particular, if we are computing a solution that has a rapid transient, such as the kinetics problem shown in Figure 8.3(a), then the problem is not stiff over the initial transient period where the true solution is as rapidly varying as nearby solution curves. Only for times greater than 10^{-6} or so does the problem become stiff, once the desired solution curve is much smoother than nearby curves.

For the problem shown in Figure 8.3(b), there is another time interval just after $t = 1$ over which the problem is again not stiff since the solution again exhibits rapid transient behavior and a small time step would be needed on the basis of accuracy considerations.

8.3 Numerical methods for stiff problems

Over time intervals where a problem is stiff, we would like to use a numerical method that has a large region of absolute stability, extending far into the left half-plane. The problem with a method like Euler's, with a stability region that extends only out to $\text{Re}(\lambda) = -2$, is that the time step k is severely limited by the eigenvalue with largest magnitude, and we need to take $k \approx 2/|\lambda_{\max}|$. Over time intervals where this fastest time scale does not appear in the solution, we would like to be able to take much larger time steps. For example, in the problems shown in Figure 8.3, where $K_1 = 10^6$, we would need to take $k \approx 2 \times 10^{-6}$ with Euler's method, requiring 4 million time steps to compute over the time interval shown in the figure, although the solution is very smooth over most of this time.

An analysis of stability regions shows that there are basically two different classes of methods: those for which the stability region is bounded and extends a distance $O(1)$ from the origin, such as Euler's method, the midpoint method, or any of the Adams methods (see Figures 7.1 and 8.5), and those for which the stability region is unbounded, such as backward Euler or trapezoidal. Clearly the first class of methods is inappropriate for stiff problems.

Unfortunately, all explicit methods have bounded stability regions and hence are generally quite inefficient on stiff problems. An exception is methods such as the Runge–Kutta–Chebyshev methods described in Section 8.6 that may work well for mildly stiff problems. Some implicit methods also have bounded stability regions, such as the Adams–Moulton methods, and are not useful for stiff problems.

8.3.1 A-stability and A(α)-stability

It seems like it would be optimal to have a method whose stability region contains the entire left half-plane. Then any time step would be allowed, provided that all the eigenvalues have negative real parts, as is often the case in practice. The backward Euler and trapezoidal methods have this property, for example. We give methods with this property a special name, as follows.

Definition 8.1. *An ODE method is said to be* A-stable *if its region of absolute stability $\mathcal{S}$ contains the entire left half-plane* $\{z \in \mathbb{C} : Re(z) \le 0\}$.

For LMMs it turns out that this is quite restrictive. A theorem of Dahlquist [21] (the paper in which the term *A-stability* was introduced) states that any A-stable LMM is at most second order accurate, and in fact the trapezoidal method is the A-stable method with smallest truncation error. This is *Dahlquist's second barrier theorem*, proved, for example, in [44] and by using order stars in [51].

Higher order A-stable implicit Runge–Kutta methods do exist, including diagonally implicit (DIRK) methods; see, for example, [13], [44].

For many stiff problems the eigenvalues are far out in the left half-plane but near (or even exactly on) the real axis. For such problems there is no reason to require that the entire left half-plane lie in the region of absolute stability. If $\arg(z)$ represents the argument of z with $\arg(z) = \pi$ on the negative real axis, and if the wedge $\pi - \alpha \le \arg(z) \le \pi + \alpha$ is contained in the stability region, then we say the method is A(α)-stable. An A-stable method is A($\pi/2$)-stable. A method is A(0)-stable if the negative real axis itself lies in the stability region. Note that in general it makes sense to require a wedge to lie in the stability regions, since adjusting the time step k causes $z = k\lambda$ to move in toward the origin on a ray through each eigenvalue.

8.3.2 L-stability

Notice a major difference between the stability regions for trapezoidal and backward Euler: the trapezoidal method is stable only in the left half-plane, whereas backward Euler is also stable over much of the right half-plane. The point at infinity (if we view these stability regions on the Riemann sphere) lies on the boundary of the stability region for the trapezoidal method but in the interior of the stability region for backward Euler.

These are both one-step methods and so on the test problem $u' = \lambda u$ we have $U^{n+1} = R(z)U^n$, where

$$R(z) = \frac{1}{(1-z)} \quad \text{and } |R(z)| \to 0 \text{ as } |z| \to \infty$$

for backward Euler, while

$$R(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z} \quad \text{and } |R(z)| \to 1 \text{ as } |z| \to \infty$$

for the trapezoidal method.

This difference can have a significant effect on the quality of solutions in some situations, in particular if there are rapid transients in the solution that we are not interested in resolving accurately with very small time steps. For these transients we want more than just stability—we want them to be effectively damped in a single time step since we are planning to use a time step that is much larger than the true decay time of the transient. For this purpose a method like backward Euler will perform better than the trapezoidal method. The backward Euler method is said to be L-stable.

Definition 8.2. *A one-step method is* L-stable *if it is A-stable and* $\lim_{z\to\infty} |R(z)| = 0$, *where the stability function* $R(z)$ *is defined in* Section 7.6.2.

The value of L-stability is best illustrated with an example.

Example 8.3. Consider the problem (8.1) with $\lambda = -10^6$. We will see how the trapezoidal and backward Euler methods behave in two different situations.

Case 1: Take data $u(0) = 1$, so that $u(t) = \cos(t)$ and there is no initial transient. Then both trapezoidal and backward Euler behave reasonably and the trapezoidal method gives smaller errors since it is second order accurate. Table 8.1 shows the errors at $T = 3$ with various values of k.

Case 2: Now take data $u(0) = 1.5$ so there is an initial rapid transient toward $u = \cos(t)$ on a time scale of about 10^{-6}. Both methods are still absolutely stable, but the results in Table 8.1 show that backward Euler works much better in this case.

To understand what is happening, see Figure 8.4, which shows the true and computed solutions with each method if we use $k = 0.1$. The trapezoidal method is stable and the results stay bounded, but since $k\lambda = -10^5$ we have $\frac{1-\frac{1}{2}k\lambda}{1+\frac{1}{2}k\lambda} = -.99996 \approx -1$ and the initial deviation from the smooth curve $\cos(t)$ is essentially negated in each time step.

Backward Euler, on the other hand, damps the deviation very effectively in the first time step, since $(1 + k\lambda)^{-1} \approx -10^{-6}$. This is the proper behavior since the true rapid transient decays in a period much shorter than a single time step.

If we are solving a stiff equation with initial data for which the solution is smooth from the beginning (no rapid transients), or if we plan to compute rapid transients accurately by taking suitably small time steps in these regions, then it may be fine to use a method such as the trapezoidal method that is not L-stable.

Table 8.1. *Errors at time* $T = 3$ *for Example* 8.3.

	k	Backward Euler	Trapezoidal
Case 1	0.4	4.7770e−02	4.7770e−02
	0.2	9.7731e−08	4.7229e−10
	0.1	4.9223e−08	1.1772e−10
Case 2	0.4	4.7770e−02	4.5219e−01
	0.2	9.7731e−08	4.9985e−01
	0.1	4.9223e−08	4.9940e−01

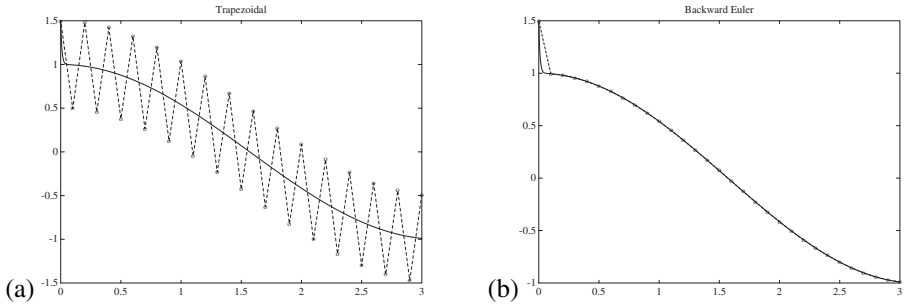

Figure 8.4. *Comparison of* (a) *trapezoidal method and* (b) *backward Euler on a stiff problem with an initial transient (Case* 2 *of Example* 8.3*).*

8.4 BDF methods

One class of very effective methods for stiff problems consists of the backward differentiation formula (BDF) methods. These were introduced by Curtiss and Hirschfelder [20]. See e.g., [33], [44], or [59] for more about these methods.

These methods result from taking $\sigma(\zeta) = \beta_r \zeta^r$, which has all its roots at the origin, so that the point at infinity is in the interior of the stability region. The method thus has the form

$$\alpha_0 U^n + \alpha_1 U^{n+1} + \cdots + \alpha_r U^{n+r} = k\beta_r f(U^{n+r}) \tag{8.4}$$

with $\beta_0 = \beta_1 = \cdots = \beta_{r-1} = 0$. Since $f(u) = u'$, this form of method can be derived by approximating $u'(t_{n+r})$ by a backward difference approximation based on $u(t_{n+r})$ and r additional points going backward in time.

It is possible to derive an r-step method that is rth order accurate. The one-step BDF method is simply the backward Euler method, $U^{n+1} = U^n + kf(U^{n+1})$, which is first order accurate. The other useful BDF methods are below:

$$\begin{aligned}
r = 2: \quad & 3U^{n+2} - 4U^{n+1} + U^n = 2kf(U^{n+2}) \\
r = 3: \quad & 11U^{n+3} - 18U^{n+2} + 9U^{n+1} - 2U^n = 6kf(U^{n+3}) \\
r = 4: \quad & 25U^{n+4} - 48U^{n+3} + 36U^{n+2} - 16U^{n+1} + 3U^n = 12kf(U^{n+4}) \\
r = 5: \quad & 137U^{n+5} - 300U^{n+4} + 300U^{n+3} - 200U^{n+2} \\
& \qquad + 75U^{n+1} - 12U^n = 60kf(U^{n+5}) \\
r = 6: \quad & 147U^{n+6} - 360U^{n+5} + 450U^{n+4} - 400U^{n+3} \\
& \qquad + 225U^{n+2} - 72U^{n+1} + 10U^n = 60kf(U^{n+6})
\end{aligned}$$

These methods have the proper behavior on eigenvalues for which $\mathrm{Re}(\lambda)$ is very negative, but of course we also have other eigenvalues for which $z = k\lambda$ is closer to the origin, corresponding to the active time scales in the problem. So deciding its suitability for a particular problem requires looking at the full stability region of a method. This is shown in Figure 8.5 for each of the BDF methods.

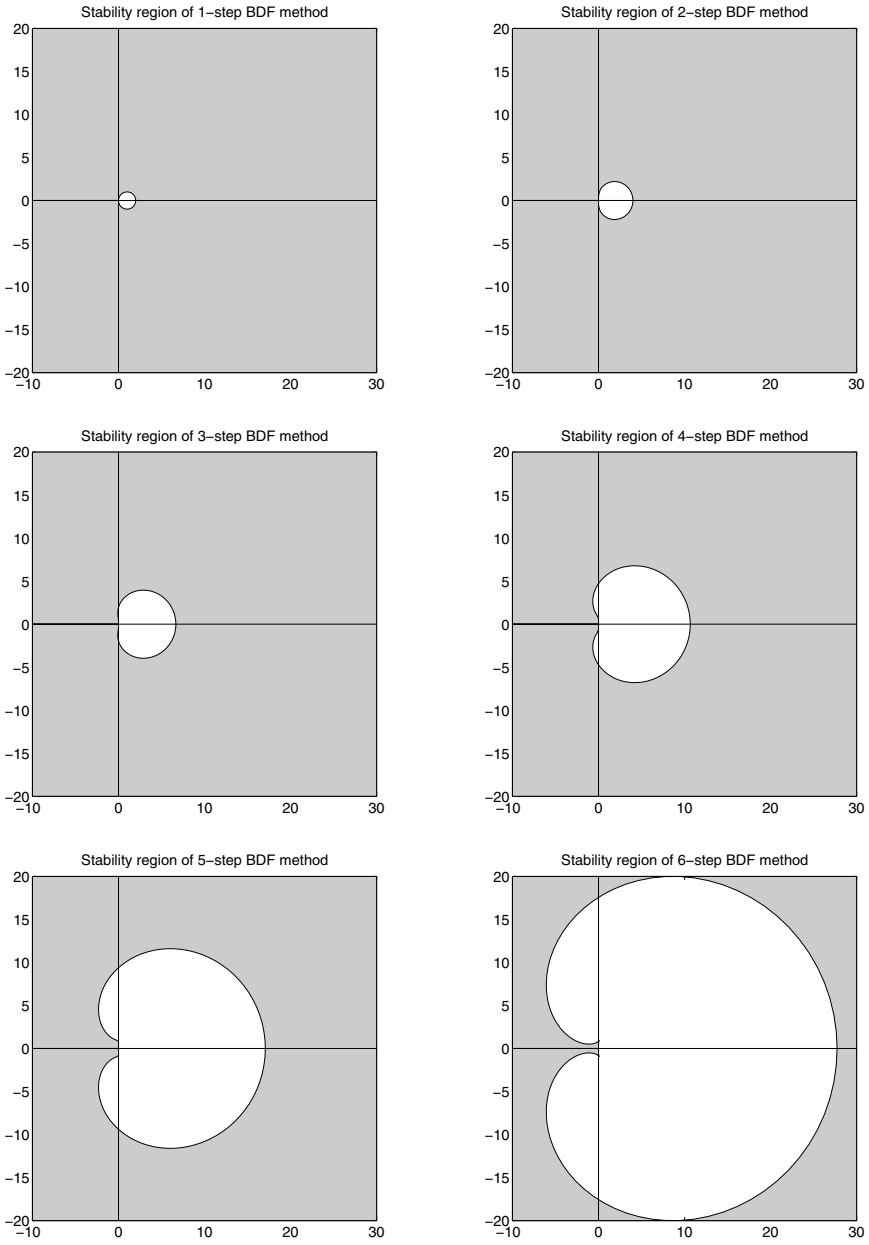

Figure 8.5. *Stability regions for the BDF methods. The stability region is the shaded region* exterior *to the curves.*

In particular, we need to make sure that the method is zero-stable. Otherwise it would not be convergent. This is not guaranteed from our derivation of the methods, since zero-stability depends only on the polynomial $\rho(\zeta)$, whose coefficients α_j are determined by considering the local truncation error and not stability considerations. It turns out that the BDF methods are zero-stable only for $r \le 6$. Higher order BDF methods cannot be used in practice. For $r > 2$ the methods are not A-stable but are A(α)-stable for the following values of α:

$$\begin{aligned} r = 1: &\ \alpha = 90^\circ, & r = 4: &\ \alpha = 73^\circ \\ r = 2: &\ \alpha = 90^\circ, & r = 5: &\ \alpha = 51^\circ, \\ r = 3: &\ \alpha = 88^\circ, & r = 6: &\ \alpha = 18^\circ. \end{aligned} \tag{8.5}$$

8.5 The TR-BDF2 method

There are often situations in which it is useful to have a one-step method that is L-stable. The backward Euler method is one possibility, but it is only first order accurate. It is possible to derive higher order implicit Runge–Kutta methods that are L-stable. As one example, we mention the two-stage second order accurate diagonally implicit method

$$\begin{aligned} U^* &= U^n + \frac{k}{4}(f(U^n) + f(U^*)), \\ U^{n+1} &= \frac{1}{3}(4U^* - U^n + kf(U^{n+1})). \end{aligned} \tag{8.6}$$

Each stage is implicit. The first stage is simply the trapezoidal method (or trapezoidal rule, hence *TR* in the name) applied over time $k/2$. This generates a value U^* at $t_{n+1/2}$. Then the two-step BDF method is applied to the data U^n and U^* with time step $k/2$ to obtain U^{n+1}. This method is written in a different form in (5.37).

8.6 Runge–Kutta–Chebyshev explicit methods

While conventional wisdom says that implicit methods should be used for stiff problems, there's an important class of "mildly stiff" problems for which special explicit methods have been developed with some advantages. In this section we take a brief look at the Runge–Kutta–Chebyshev methods that are applicable to problems where the eigenvalues of the Jacobian matrix are on the negative real axis and not too widely distributed. This type of problem arises, for example, when a parabolic heat equation or diffusion equation is discretized in space, giving rise to a large system of ordinary differential equations in time. This problem is considered in detail in Chapter 9.

The idea is to develop an explicit method whose stability region stretches out along the negative real axis as far as possible, without much concern with what's happening away from the real axis. We want the region of absolute stability $\mathcal{S}$ to contain a "stability interval" $[-\beta, 0]$ that is as long as possible. To do this we will consider multistage explicit Runge–Kutta methods as introduced in Section 5.7, but now as we increase the number of stages we will choose the coefficients to make β as large as possible rather than to increase the order of accuracy of the method.

We will consider only first order methods, because they are easiest to study. Second order methods are commonly used in practice, and this is often quite sufficient for the applications we have in mind, such as integrating method of lines (MOL) discretizations of the heat equation. The system of ODEs is obtained by discretizing the original partial differential equations PDE in space, and often this discretization is only second order accurate spatially. Second order methods similar to those described here can be found in [80], [97], [2] and higher order methods in [1], [69], for example.

Consider an explicit r-stage Runge–Kutta method of the form (5.34) with $a_{ij} = 0$ for $i \le j$. If we apply this method to $u' = \lambda u$ we obtain

$$U^{n+1} = R(z)U^n,$$

where $z = k\lambda$ and $R(z)$ is a polynomial of degree r in z, the *stability polynomial*. We will write this as

$$R(z) = d_0 + d_1 z + d_2 z^2 + \cdots + d_r z^r. \tag{8.7}$$

Consistency and first order accuracy require that

$$d_0 = 1, \quad d_1 = 1. \tag{8.8}$$

The region of absolute stability is

$$\mathcal{S} = \{z \in \mathbb{C} : |R(z)| \le 1\}. \tag{8.9}$$

The coefficients d_j depend on the coefficients A and b defining the Runge–Kutta method, but for the moment suppose that for any choice of d_j we can find suitable coefficients A and b that define an explicit r-stage method with stability polynomial (8.7). Then our goal is to choose the d_j coefficients so that $\mathcal{S}$ contains an interval $[-\beta, 0]$ of the negative real axis with β as large as possible. We also require (8.8) for a first order accurate method.

Note that the condition (8.8) amounts to requiring

$$R(0) = 1 \quad \text{and} \quad R'(0) = 1. \tag{8.10}$$

Then we want to choose $R(z)$ as a polynomial of degree r that satisfies $|R(z)| \le 1$ for as long as possible as we go out on the negative real axis. The solution is simply a scaled and shifted Chebyshev polynomial. Figure 8.6 shows the optimal polynomials of degrees 3 and 6. Note that these are plots of $R(x)$ for x real.

We have seen several other examples where Chebyshev polynomials solve problems of this sort, and as usual the optimal solution equioscillates at a set of points, in this case $r + 2$ points (including $x = -\beta$ and $x = 0$), where $|R(x)| = 1$. If we try to perturb the polynomial so that $|R(-\beta)| < 1$ (and hence the method will be stable further out on the negative real axis), one of the other extrema of the polynomial will move above 1, losing stability for some z closer to the origin.

The optimal polynomial is

$$R(z) = T_r\left(1 + z/r^2\right), \tag{8.11}$$

where $T_r(z)$ is the Chebyshev polynomial of degree r, as in Section B.3.2. For this choice $R(-2r^2) = T_r(-1) = \pm 1$, while $|R(x)| > 1$ for $x < -2r^2$, and so

$$\beta(r) = 2r^2. \tag{8.12}$$

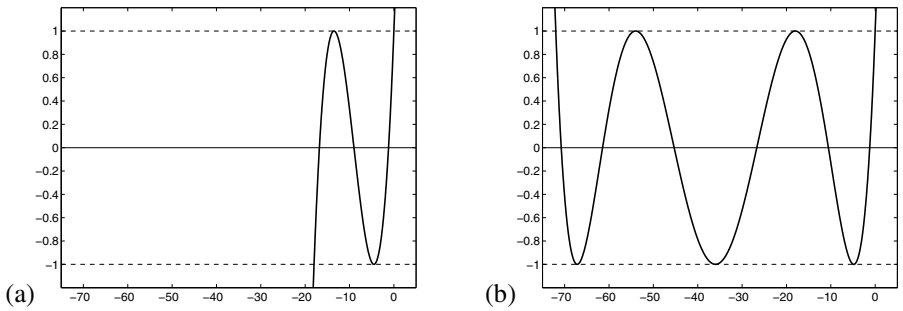

Figure 8.6. *Shifted Chebyshev polynomials corresponding to $R(x)$ along the real axis for the first order Runge–Kutta–Chebyshev methods.* (a) $r = 3$. (b) $r = 6$.

We have optimized along the negative real axis, but it is interesting to see what the stability region $\mathcal{S}$ from (8.9) looks like in the complex plane. Figure 8.7 shows the stability region for the two polynomials shown in Figure 8.6 of degrees 3 and 6. They do include the negative real axis out to $-\beta(r)$, but notice that everywhere $|R(x)| = 1$ on the real axis is a point where the region $\mathcal{S}$ contracts to the axis. Perturbing such an x away from the axis into the complex plane gives a point z where $|R(z)| > 1$. So these methods are suitable only for problems with real eigenvalues, such as MOL discretizations of the heat equation.

Even for these problems it is generally preferable to perturb the polynomial $R(z)$ a bit so that $|R(z)|$ is bounded slightly below 1 on the real axis between $-\beta$ and the origin, which can be done at the expense of decreasing β slightly. This is done by using the shifted Chebyshev polynomial

$$R(z) = \frac{T_r(w_0 + w_1 z)}{T_r(w_0)}, \qquad \text{where } w_1 = \frac{T_r(w_0)}{T_r'(w_0)}.$$

Here $w_0 > 1$ is the "damping parameter" and w_1 is chosen so that (8.8) holds and hence the method remains first order accurate. Now $R(x)$ alternates between $\pm(T_r(w_0))^{-1} < 1$ on the interval $[-\beta, 0]$, where β is now found by solving $w_0 - \beta w_1 = -1$, so

$$\beta = \frac{(w_0 + 1)T_r'(w_0)}{T_r(w_0)}.$$

Verwer [97] suggests taking $w_0 = 1 + \epsilon/r^2$ with $\epsilon = 0.05$, for which

$$\beta \approx \left(2 - \frac{4}{3}\epsilon\right) r^2 \approx 1.93 r^2.$$

This gives about 5% damping with little reduction in β. Figure 8.8 shows the stability regions for the damped first order methods, again for $r = 3$ and $r = 6$.

Once we have determined the desired stability polynomial $R(z)$, we must still develop an r-stage Runge–Kutta method with this stability polynomial. In general there are infinitely many such Runge–Kutta methods. Here we discuss one approach that has several desirable properties, the *Runge–Kutta–Chebyshev* method introduced by van der Houwen

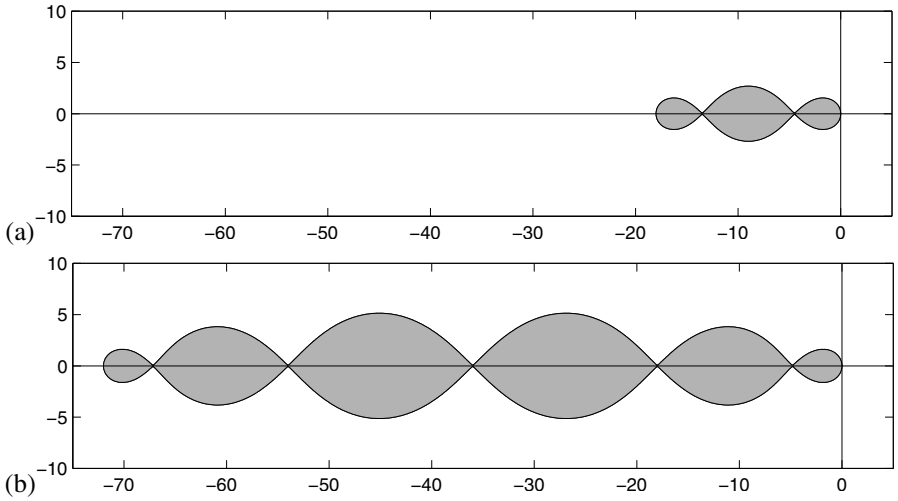

Figure 8.7. *Absolute stability regions for the first order Runge–Kutta–Chebyshev methods.* (a) $r = 3$. (b) $r = 6$.

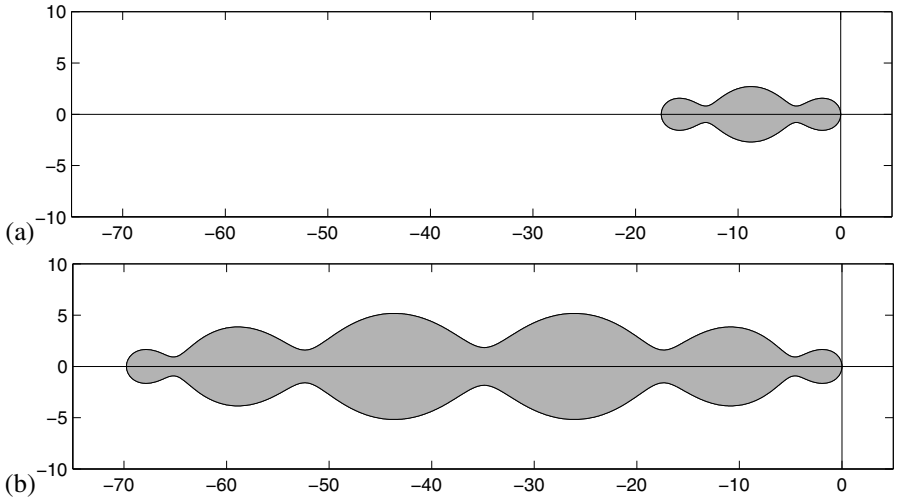

Figure 8.8. *Absolute stability regions for the first order Runge–Kutta–Chebyshev methods with damping.* (a) $r = 3$. (b) $r = 6$.

and Sommeijer [94]. See the review paper of Verwer [97] for a derivation of this method and a discussion and comparison of some other approaches, and see [80] for a discussion of software that implements these methods.

The Runge–Kutta–Chebyshev methods are based on the recurrence relation of Chebyshev polynomials and have the general form

$$
\begin{aligned}
Y_0 &= U^n, \\
Y_1 &= Y_0 + \tilde{\mu}_1 k f(Y_0, t_n), \\
Y_j &= (1 - \mu_j - \nu_j) Y_0 + \mu_j Y_{j-1} \\
&\quad + \nu_j Y_{j-2} + \tilde{\mu}_j k f(Y_{j-1}, t_n + c_{j-1}k) + \tilde{\gamma} k f(Y_0, t_n), \quad j = 2, \ \ldots, \ r, \\
U^{n+1} &= Y_r.
\end{aligned} \tag{8.13}
$$

The various parameters in these formulas are given in the references above. This is an r-stage explicit Runge–Kutta method, although written in a different form than (5.34). An advantage of this recursive form is that only three intermediate solution vectors Y_0, Y_{j-1}, and Y_{j-2} are needed to compute the next Y_j, no matter how many stages r are used. This is important since large values of r (e.g., $r = 50$) are often used in practice, and for PDE applications each Y_j is a grid function over the entire (possibly multidimensional) spatial domain.

Another advantage of the Runge–Kutta–Chebyshev methods is that they have good *internal stability* properties, as discussed in the references above. We know the stability region of the overall stability polynomial $R(z)$, but some r-stage methods for large r suffer from exponential growth of the solution from one stage to the next in the early stages before ultimately decaying even when $|R(z)| < 1$ (analogous to the transient growth sometimes seen when iterating with a nonnormal matrix even if it is asymptotically stable; see Section D.4). This growth can cause numerical difficulties if methods are not carefully designed.

Chapter 9

Diffusion Equations and Parabolic Problems

We now begin to study finite difference methods for time-dependent partial differential equations (PDEs), where variations in space are related to variations in time. We begin with the heat equation (or diffusion equation) introduced in Appendix E,

$$u_t = \kappa u_{xx}. \tag{9.1}$$

This is the classical example of a *parabolic* equation, and many of the general properties seen here carry over to the design of numerical methods for other parabolic equations. We will assume $\kappa = 1$ for simplicity, but some comments will be made about how the results scale to other values of $\kappa > 0$. (If $\kappa < 0$, then (9.1) would be a "backward heat equation," which is an ill-posed problem.)

Along with this equation we need initial conditions at some time t_0, which we typically take to be $t_0 = 0$,

$$u(x, 0) = \eta(x), \tag{9.2}$$

and also boundary conditions if we are working on a bounded domain, e.g., the Dirichlet conditions

$$\begin{aligned} u(0, t) &= g_0(t) \quad \text{for } t > 0, \\ u(1, t) &= g_1(t) \quad \text{for } t > 0 \end{aligned} \tag{9.3}$$

if $0 \le x \le 1$.

We have already studied the steady-state version of this equation and spatial discretizations of u_{xx} in Chapter 2. We have also studied discretizations of the time derivatives and some of the stability issues that arise with these discretizations in Chapters 5 through 8. Next we will put these two types of discretizations together.

In practice we generally apply a set of finite difference equations on a discrete grid with grid points (x_i, t_n), where

$$x_i = ih, \qquad t_n = nk.$$

Here $h = \Delta x$ is the mesh spacing on the x-axis and $k = \Delta t$ is the time step. Let $U_i^n \approx u(x_i, t_n)$ represent the numerical approximation at grid point (x_i, t_n).

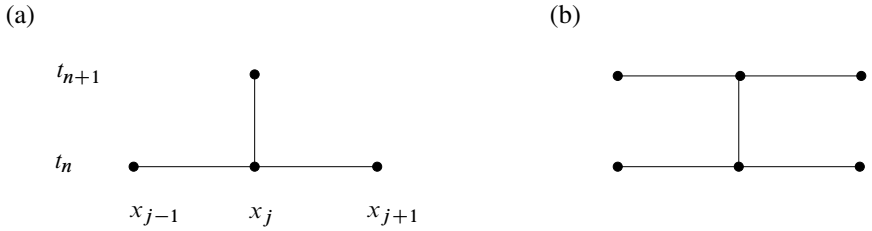

Figure 9.1. *Stencils for the methods* (9.5) *and* (9.7).

Since the heat equation is an evolution equation that can be solved forward in time, we set up our difference equations in a form where we can march forward in time, determining the values U_i^{n+1} for all i from the values U_i^n at the previous time level, or perhaps using also values at earlier time levels with a multistep formula.

As an example, one natural discretization of (9.1) would be

$$\frac{U_i^{n+1} - U_i^n}{k} = \frac{1}{h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n). \tag{9.4}$$

This uses our standard centered difference in space and a forward difference in time. This is an *explicit* method since we can compute each U_i^{n+1} explicitly in terms of the previous data:

$$U_i^{n+1} = U_i^n + \frac{k}{h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n). \tag{9.5}$$

Figure 9.1(a) shows the *stencil* of this method. This is a one-step method in time, which is also called a two-level method in the context of PDEs since it involves the solution at two different time levels.

Another one-step method, which is much more useful in practice, as we will see below, is the *Crank–Nicolson* method,

$$\begin{aligned} \frac{U_i^{n+1} - U_i^n}{k} &= \frac{1}{2}(D^2 U_i^n + D^2 U_i^{n+1}) \\ &= \frac{1}{2h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n + U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}), \end{aligned} \tag{9.6}$$

which can be rewritten as

$$U_i^{n+1} = U_i^n + \frac{k}{2h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n + U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}) \tag{9.7}$$

or

$$-rU_{i-1}^{n+1} + (1+2r)U_i^{n+1} - rU_{i+1}^{n+1} = rU_{i-1}^n + (1-2r)U_i^n + rU_{i+1}^n, \tag{9.8}$$

where $r = k/2h^2$. This is an *implicit* method and gives a tridiagonal system of equations to solve for all the values U_i^{n+1} simultaneously. In matrix form this is

$$\begin{bmatrix} (1+2r) & -r & & & & & \\ -r & (1+2r) & -r & & & & \\ & -r & (1+2r) & -r & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -r & (1+2r) & -r & \\ & & & & -r & (1+2r) \end{bmatrix} \begin{bmatrix} U_1^{n+1} \\ U_2^{n+1} \\ U_3^{n+1} \\ \vdots \\ U_{m-1}^{n+1} \\ U_m^{n+1} \end{bmatrix} \tag{9.9}$$

$$= \begin{bmatrix} r(g_0(t_n) + g_0(t_{n+1})) + (1-2r)U_1^n + rU_2^n \\ rU_1^n + (1-2r)U_2^n + rU_3^n \\ rU_2^n + (1-2r)U_3^n + rU_4^n \\ \vdots \\ rU_{m-2}^n + (1-2r)U_{m-1}^n + rU_m^n \\ rU_{m-1}^n + (1-2r)U_m^n + r(g_1(t_n) + g_1(t_{n+1})) \end{bmatrix}.$$

Note how the boundary conditions $u(0,t) = g_0(t)$ and $u(1,t) = g_1(t)$ come into these equations.

Since a tridiagonal system of m equations can be solved with $O(m)$ work, this method is essentially as efficient per time step as an explicit method. We will see in Section 9.4 that the heat equation is "stiff," and hence this implicit method, which allows much larger time steps to be taken than an explicit method, is a very efficient method for the heat equation.

Solving a parabolic equation with an implicit method requires solving a system of equations with the same structure as the two-point boundary value problem we studied in Chapter 2. Similarly, a multidimensional parabolic equation requires solving a problem with the structure of a multidimensional elliptic equation in each time step; see Section 9.7.

9.1 Local truncation errors and order of accuracy

We can define the local truncation error as usual—we insert the exact solution $u(x,t)$ of the PDE into the finite difference equation and determine by how much it fails to satisfy the discrete equation.

Example 9.1. The local truncation error of the method (9.5) is based on the form (9.4): $\tau_i^n = \tau(x_i, t_n)$, where

$$\tau(x,t) = \frac{u(x,t+k) - u(x,t)}{k} - \frac{1}{h^2}(u(x-h,t) - 2u(x,t) + u(x+h,t)).$$

Again we should be careful to use the form that directly models the differential equation in order to get powers of k and h that agree with what we hope to see in the global error. Although we don't know $u(x,t)$ in general, if we assume it is smooth and use Taylor series expansions about $u(x,t)$, we find that

$$\tau(x,t) = \left(u_t + \frac{1}{2}ku_{tt} + \frac{1}{6}k^2u_{ttt} + \cdots\right) - \left(u_{xx} + \frac{1}{12}h^2u_{xxxx} + \cdots\right).$$

Since $u_t = u_{xx}$, the $O(1)$ terms drop out. By differentiating $u_t = u_{xx}$ we find that $u_{tt} = u_{txx} = u_{xxxx}$ and so

$$\tau(x,t) = \left(\frac{1}{2}k - \frac{1}{12}h^2\right) u_{xxxx} + O(k^2 + h^4).$$

This method is said to be *second order accurate in space* and *first order accurate in time* since the truncation error is $O(h^2 + k)$.

The Crank–Nicolson method is centered in both space and time, and an analysis of its local truncation error shows that it is second order accurate in both space and time,

$$\tau(x,t) = O(k^2 + h^2).$$

A method is said to be *consistent* if $\tau(x,t) \to 0$ as $k,\ h \to 0$. Just as in the other cases we have studied (boundary value problems and initial value problems for ordinary differential equations (ODEs)), we expect that consistency, plus some form of stability, will be enough to prove that the method converges at each fixed point (X, T) as we refine the grid in both space and time. Moreover, we expect that for a stable method the global order of accuracy will agree with the order of the local truncation error, e.g., for Crank–Nicolson we expect that

$$U_i^n - u(X,T) = O(k^2 + h^2)$$

as $k,\ h \to 0$ when $ih \equiv X$ and $nk \equiv T$ are fixed.

For linear PDEs, the fact that consistency plus stability is equivalent to convergence is known as the *Lax equivalence theorem* and is discussed in Section 9.5 after an introduction of the proper concept of stability. As usual, it is the definition and study of stability that is the hard (and interesting) part of this theory.

9.2 Method of lines discretizations

To understand how stability theory for time-dependent PDEs relates to the stability theory we have already developed for time-dependent ODEs, it is easiest to first consider the so-called method of lines (MOL) discretization of the PDE. In this approach we first discretize in space alone, which gives a large system of ODEs with each component of the system corresponding to the solution at some grid point, as a function of time. The system of ODEs can then be solved using one of the methods for ODEs that we have previously studied.

This system of ODEs is also often called a *semidiscrete* method, since we have discretized in space but not yet in time.

For example, we might discretize the heat equation (9.1) in space at grid point x_i by

$$U_i'(t) = \frac{1}{h^2}(U_{i-1}(t) - 2U_i(t) + U_{i+1}(t)) \quad \text{for } i = 1,\ 2,\ \ldots,\ m, \tag{9.10}$$

where prime now means differentiation with respect to time. We can view this as a coupled system of m ODEs for the variables $U_i(t)$, which vary continuously in time along the lines shown in Figure 9.2. This system can be written as

$$U'(t) = AU(t) + g(t), \tag{9.11}$$

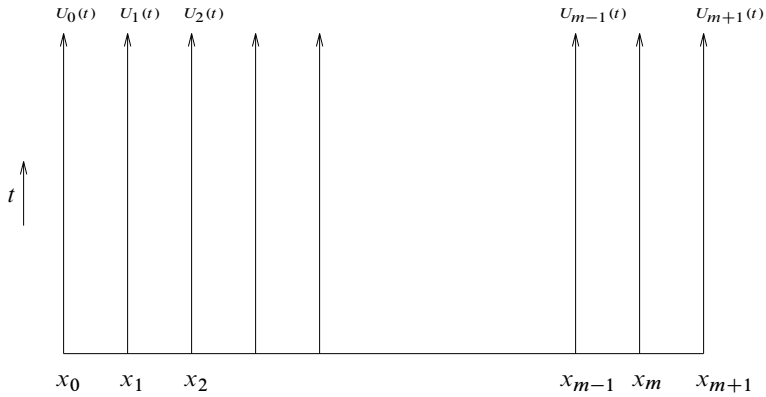

Figure 9.2. *Method of lines interpretation. $U_i(t)$ is the solution along the line forward in time at the grid point x_i.*

where the tridiagonal matrix A is exactly as in (2.9) and $g(t)$ includes the terms needed for the boundary conditions, $U_0(t) \equiv g_0(t)$ and $U_{m+1}(t) \equiv g_1(t)$,

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 \end{bmatrix}, \qquad g(t) = \frac{1}{h^2}\begin{bmatrix} g_0(t) \\ 0 \\ 0 \\ \vdots \\ 0 \\ g_1(t) \end{bmatrix}. \tag{9.12}$$

This MOL approach is sometimes used in practice by first discretizing in space and then applying a software package for systems of ODEs. There are also packages that are specially designed to apply MOL. This approach has the advantage of being relatively easy to apply to a fairly general set of time-dependent PDEs, but the resulting method is often not as efficient as specially designed methods for the PDE. See Section 11.2 for more discussion of this.

As a tool in understanding stability theory, however, the MOL discretization is extremely valuable, and this is the main use we will make of it. We know how to analyze the stability of ODE methods applied to a linear system of the form (9.11) based on the eigenvalues of the matrix A, which now depend on the spatial discretization.

If we apply an ODE method to discretize the system (9.11), we will obtain a fully discrete method which produces approximations $U_i^n \approx U_i(t_n)$ at discrete points in time which are exactly the points (x_i, t_n) of the grid that we introduced at the beginning of this chapter.

For example, applying Euler's method $U^{n+1} = U^n + kf(U^n)$ to this linear system results in the fully discrete method (9.5). Applying instead the trapezoidal method (5.22) results in the Crank–Nicolson method (9.7). Applying a higher order linear multistep or Runge–Kutta method would give a different method, although with the spatial discretization (9.10) the overall method would be only second order accurate in space. Replacing

the right-hand side of (9.10) with a higher order approximation to $u_{xx}(x_i)$ and then using a higher order time discretization would give a more accurate method.

9.3 Stability theory

We can now investigate the stability of schemes like (9.5) or (9.7) since these can be interpreted as standard ODE methods applied to the linear system (9.11). We expect the method to be stable if $k\lambda \in \mathcal{S}$, i.e., if the time step k multiplied by any eigenvalue λ of A lies in the stability region of the ODE method, as discussed in Chapter 7. (Note that A is symmetric and hence normal, so eigenvalues are the right thing to look at.)

We have determined the eigenvalues of A in (2.23),

$$\lambda_p = \frac{2}{h^2}(\cos(p\pi h) - 1) \quad \text{for } p = 1, 2, \ldots, m, \tag{9.13}$$

where again m and h are related by $h = 1/(m+1)$. Note that there is a new wrinkle here relative to the ODEs we considered in Chapter 7: the eigenvalues λ_p depend on the mesh width h. As we refine the grid and $h \to 0$, the dimension of A increases, the number of eigenvalues we must consider increases, and the values of the eigenvalues change.

This is something we must bear in mind when we attempt to prove convergence as $k, h \to 0$. To begin, however, let's consider the simpler question of how the method behaves for some fixed k and h, i.e., the question of absolute stability in the ODE sense. Then it is clear that the method is absolutely stable (i.e., the effect of past errors will not grow exponentially in future time steps) provided that $k\lambda_p \in \mathcal{S}$ for each p, where $\mathcal{S}$ is the stability region of the ODE method, as discussed in Chapter 7.

For the matrix (9.12) coming from the heat equation, the eigenvalues lie on the negative real axis and the one farthest from the origin is $\lambda_m \approx -4/h^2$. Hence we require that $-4k/h^2 \in \mathcal{S}$ (assuming the stability region is connected along the negative real axis up to the origin, as is generally the case).

Example 9.2. If we use Euler's method to obtain the discretization (9.5), then we must require $|1 + k\lambda| \le 1$ for each eigenvalue (see Chapter 7) and hence we require $-2 \le -4k/h^2 \le 0$. This limits the time step allowed to

$$\frac{k}{h^2} \le \frac{1}{2}. \tag{9.14}$$

This is a severe restriction: the time step must decrease at the rate of h^2 as we refine the grid, which is much smaller than the spatial width h when h is small.

Example 9.3. If we use the trapezoidal method, we obtain the Crank–Nicolson discretization (9.6). The trapezoidal method for the ODE is absolutely stable in the whole left half-plane and the eigenvalues (9.13) are always negative. Hence the Crank–Nicolson method is stable for *any* time step $k > 0$. Of course it may not be accurate if k is too large. Generally we must take $k = O(h)$ to obtain a reasonable solution, and the unconditional stability allows this.

9.4 Stiffness of the heat equation

Note that the system of ODEs we are solving is quite stiff, particularly for small h. The eigenvalues of A lie on the negative real axis with one fairly close to the origin, $\lambda_1 \approx -\pi^2$

for all h, while the largest in magnitude is $\lambda_m \approx -4/h^2$. The "stiffness ratio" of the system is $4/\pi^2 h^2$, which grows rapidly as $h \to 0$. As a result the explicit Euler method is stable only for very small time steps $k \leq \frac{1}{2}h^2$. This is typically much smaller than what we would like to use over physically meaningful times, and a method designed for stiff problems will be more efficient.

The stiffness is a reflection of the very different time scales present in solutions to the physical problem modeled by the heat equation. High frequency spatial oscillations in the initial data will decay very rapidly due to rapid diffusion over very short distances, while smooth data decay much more slowly since diffusion over long distances takes much longer. This is apparent from the Fourier analysis of Section E.3.3 or is easily seen by writing down the exact solution to the heat equation on $0 \leq x \leq 1$ with $g_0(t) = g_1(t) \equiv 0$ as a Fourier sine series:

$$u(x,t) = \sum_{j=1}^{\infty} \hat{u}_j(t) \sin(j\pi x).$$

Inserting this in the heat equation gives the ODEs

$$\hat{u}_j'(t) = -j^2\pi^2 \hat{u}_j(t) \qquad \text{for } j = 1, 2, \ldots. \tag{9.15}$$

and so

$$\hat{u}_j(t) = e^{-j^2\pi^2 t} \hat{u}_j(0)$$

with the $\hat{u}_j(0)$ determined as the Fourier coefficients of the initial data $\eta(x)$.

We can view (9.15) as an infinite system of ODEs, but which are decoupled so that the coefficient matrix is diagonal, with eigenvalues $-j^2\pi^2$ for $j = 1, 2, \ldots$. By choosing data with sufficiently rapid oscillation (large j), we can obtain arbitrarily rapid decay. For general initial data there may be some transient period when any high wave numbers are rapidly damped, but then the long-time behavior is dominated by the slower decay rates. See Figure 9.3 for some examples of the time evolution with different sets of data.

If we are solving the problem over the long periods needed to track this slow diffusion, then we would ultimately (after any physical transients have decayed) like to use rather large time steps, since typically the variation in time is then on roughly the same scale as variations in space. We would generally like to have $k \approx h$ so that we have roughly the same resolution in time as we do in space. A method that requires $k \approx h^2$ forces us to take a much finer temporal discretization than we should need to represent smooth solutions. If $h = 0.001$, for example, then if we must take $k = h^2$ rather than $k = h$ we would need to take 1000 time steps to cover each time interval that should be well modeled by a single time step. This is the same difficulty we encountered with stiff ODEs in Chapter 8.

Note: The remark above that we want $k \approx h$ is reasonable assuming the method we are using has the same order of accuracy in both space and time. The method (9.5) does not have this property. Since the error is $O(k + h^2)$ we might want to take $k = O(h^2)$ just to get the same level of accuracy in both space and time. In this sense the stability restriction $k = O(h^2)$ may not seem unreasonable, but this is simply another reason for not wanting to use this particular method in practice.

Note: The general diffusion equation is $u_t = \kappa u_{xx}$ and in practice the diffusion coefficient κ may be different from 1 by many orders of magnitude. How does this affect our conclusions above? We would expect by scaling considerations that we should take

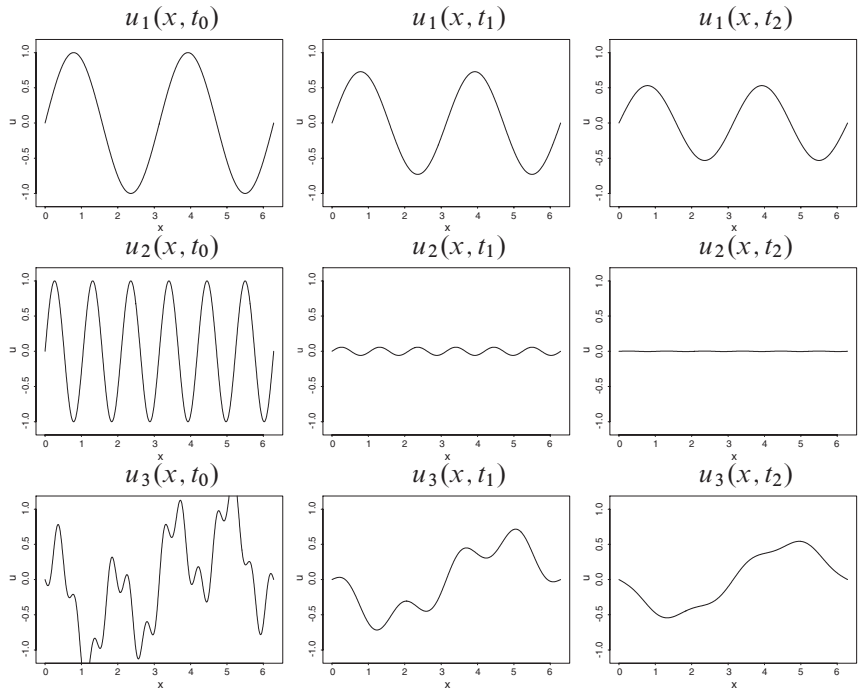

Figure 9.3. *Solutions to the heat equation at three different times (columns) shown for three different sets of initial conditions (rows). In the top row $u_1(x,t_0)$ consists of only a low wave number, which decays slowly. The middle row shows data consisting of a higher wave number, which decays more quickly. The bottom row shows data $u_3(x,t_0)$ that contains a mixture of wave numbers. The high wave numbers are most rapidly damped (an initial rapid transient), while at later times only the lower wave numbers are still visible and decaying slowly.*

$k \approx h/\kappa$ in order to achieve comparable resolution in space and time, i.e., we would like to take $\kappa k/h \approx 1$. (Note that $\hat{u}_j(t) = \exp(-j^2\pi^2\kappa t)\hat{u}_j(0)$ in this case.) With the MOL discretization we obtain the system (9.11) but A now has a factor κ in front. For stability we thus require $-4\kappa k/h^2 \in \mathcal{S}$, which requires $\kappa k/h^2$ to be order 1 for any explicit method. This is smaller than what we wish to use by a factor of h, regardless of the magnitude of κ. So our conclusions on stiffness are unchanged by κ. In particular, even when the diffusion coefficient is very small it is best to use an implicit method because we then want to take very long time steps $k \approx h/\kappa$.

These comments apply to the case of pure diffusion. If we are solving an advection-diffusion or reaction-diffusion equation where there are other time scales determined by other phenomena, then if the diffusive term has a very small coefficient we may be able to use an explicit method efficiently because of other restrictions on the time step.

Note: The physical problem of diffusion is "infinitely stiff" in the sense that (9.15) has eigenvalues $-j^2\pi^2$ with arbitrarily large magnitude, since j can be any integer. Luckily the discrete problem is not this stiff. It is not stiff because, once we discretize in space, only a finite number of spatial wave numbers can be represented and we obtain the finite

set of eigenvalues (9.13). As we refine the grid we can represent higher and higher wave numbers, leading to the increasing stiffness ratio as $h \to 0$.

9.5 Convergence

So far we have only discussed absolute stability and determined the relation between k and h that must be satisfied to ensure that errors do not grow exponentially as we march forward in time on this fixed grid. We now address the question of convergence at a fixed point (X, T) as the grid is refined. It turns out that in general exactly the same relation between k and h must now be required to hold as we vary k and h, letting both go to zero.

In other words, we cannot let k and h go to zero at arbitrary independent rates and necessarily expect the resulting approximations to converge to the solution of the PDE. For a particular sequence of grids (k_1, h_1), (k_2, h_2), ..., with $k_j \to 0$ and $h_j \to 0$, we will expect convergence only if the proper relation ultimately holds for each pair. For the method (9.5), for example, the sequence of approximations will converge only if $k_j/h_j^2 \leq 1/2$ for all j sufficiently large.

It is sometimes easiest to think of k and h as being related by some fixed rule (e.g., we might choose $k = 0.4h^2$ for the method (9.5)), so that we can speak of convergence as $k \to 0$ with the understanding that this relation holds on each grid.

The methods we have studied so far can be written in the form

$$U^{n+1} = B(k)U^n + b^n(k) \tag{9.16}$$

for some matrix $B(k) \in \mathbb{R}^{m\times m}$ on a grid with $h = 1/(m+1)$ and $b^n(k) \in \mathbb{R}^m$. In general these depend on both k and h, but we will assume some fixed rule is specified relating h to k as $k \to 0$.

For example, applying forward Euler to the MOL system (9.11) gives

$$B(k) = I + kA, \tag{9.17}$$

where A is the tridiagonal matrix in (9.12). The Crank–Nicolson method results from applying the trapezoidal method to (9.11), which gives

$$B(k) = \left(I - \frac{k}{2}A\right)^{-1}\left(I + \frac{k}{2}A\right). \tag{9.18}$$

To prove convergence we need consistency and a suitable form of stability. As usual, consistency requires that the local truncation error vanishes as $k \to 0$. The form of stability that we need is often called Lax–Richtmyer stability.

Definition 9.1. *A linear method of the form* (9.16) *is* Lax–Richtmyer stable *if, for each time T, there is a constant $C_T > 0$ such that*

$$\|B(k)^n\| \leq C_T \tag{9.19}$$

for all $k > 0$ and integers n for which $kn \leq T$.

Theorem 9.2 (Lax Equivalence Theorem). *A consistent linear method of the form* (9.16) *is convergent if and only if it is Lax–Richtmyer stable.*

For more discussion and a proof see [75]. The main idea is the same as our proof in Section 6.3.1 that Euler's method converges on a linear problem. If we apply the numerical method to the exact solution $u(x,t)$, we obtain

$$u^{n+1} = Bu^n + b^n + k\tau^n, \tag{9.20}$$

where we suppress the dependence on k for clarity and where

$$u^n = \begin{bmatrix} u(x_1,t_n) \\ u(x_2,t_n) \\ \vdots \\ u(x_m,t_n) \end{bmatrix}, \qquad \tau^n = \begin{bmatrix} \tau(x_1,t_n) \\ \tau(x_2,t_n) \\ \vdots \\ \tau(x_m,t_n) \end{bmatrix}.$$

Subtracting (9.20) from (9.16) gives the difference equation for the global error $E^n = U^n - u^n$:

$$E^{n+1} = BE^n - k\tau^n,$$

and hence, after N time steps,

$$E^N = B^N E^0 - k\sum_{n=1}^{N} B^{N-n}\tau^{n-1},$$

from which we obtain

$$\|E^N\| \le \|B^N\|\|E^0\| + k\sum_{n=1}^{N} \|B^{N-n}\|\,\|\tau^{n-1}\|. \tag{9.21}$$

If the method is Lax–Richtmyer stable, then for $Nk \le T$,

$$\begin{aligned} \|E^N\| &\le C_T\|E^0\| + TC_T \max_{1\le n\le N} \|\tau^{n-1}\| \\ &\to 0 \text{ as } k \to 0 \text{ for } Nk \le T, \end{aligned}$$

provided the method is consistent ($\|\tau\| \to 0$) and we use appropriate initial data ($\|E^0\| \to 0$ as $k \to 0$).

Example 9.4. For the heat equation the matrix A from (9.12) is symmetric, and hence the 2-norm is equal to the spectral radius, and the same is true of the matrix B from (9.17). From (9.13) we see that $\|B(k)\|_2 \le 1$, provided (9.14) is satisfied, and so the method is Lax–Richtmyer stable and hence convergent under this restriction on the time step. Similarly, the matrix B of (9.18) is symmetric and has eigenvalues $(1 + k\lambda_p/2)/(1 - k\lambda_p/2)$, and so the Crank–Nicolson method is stable in the 2-norm for any $k > 0$.

For the methods considered so far we have obtained $\|B\| \le 1$. This is called *strong stability*. But note that this is not necessary for Lax–Richtmyer stability. If there is a constant α so that a bound of the form

$$\|B(k)\| \le 1 + \alpha k \tag{9.22}$$

holds in some norm (at least for all k sufficiently small), then we will have Lax–Richtmyer stability in this norm, since

$$\|B(k)^n\| \le (1 + \alpha k)^n \le e^{\alpha T}$$

for $nk \le T$. Note that the matrix $B(k)$ depends on k, and its dimension $m = O(1/h)$ grows as $k, h \to 0$. The general theory of stability in the sense of uniform power boundedness of such families of matrices is often nontrivial.

9.5.1 PDE versus ODE stability theory

It may bother you that the stability we need for convergence now seems to depend on absolute stability, and on the shape of the stability region for the time-discretization, which determines the required relationship between k and h. Recall that in the case of ODEs all we needed for convergence was "zero-stability," which does not depend on the shape of the stability region except for the requirement that the point $z = 0$ must lie in this region.

Here is the difference: with ODEs we were studying a fixed system of ODEs of fixed dimension, and the fixed set of eigenvalues λ was independent of k. For convergence we needed $k\lambda$ in the stability region as $k \to 0$, but since these values all converge to 0 it is only the origin that is important, at least to prove convergence as $k \to 0$. Hence the need for zero-stability. With PDEs, on the other hand, in our MOL discretization the system of ODEs grows as we refine the grid, and the eigenvalues λ grow in magnitude as k and h go to zero. So it is not clear that $k\lambda$ will go to zero, and zero-stability is not sufficient. For the heat equation with k/h^2 fixed, these values do not go to zero as $k \to 0$. For convergence we must now require that these values at least lie in the region of absolute stability as $k \to 0$, and this gives the stability restriction relating k and h. If we keep k/h fixed as $k, h \to 0$, then $k\lambda \to -\infty$ for the eigenvalues of the matrix A from (9.12). We must use an implicit method that includes the entire negative real axis in its stability region.

We also notice another difference between stability theory for ODEs and PDEs that for the ODE $u'(t) = f(u(t))$ we could prove convergence of standard methods for any Lipschitz continuous function $f(u)$. For example, the proof of convergence of Euler's method for the linear case, found in Section 6.3.1, was easily extended to nonlinear functions in Section 6.3.3. In the PDE case, the Lax equivalence theorem is much more limited: it applies only to linear methods (9.16), and such methods typically only arise when discretizing linear PDEs such as the heat equation. It is possible to prove stability of many methods for nonlinear PDEs by showing that a suitable form of stability holds, but a variety of different techniques must be used, depending on the character of the differential equation, and there is no general theory of the sort obtained for ODEs.

The essential difficulty is that even a linear PDE such as the heat equation $u_t = \partial_x^2 u$ involves an operator on the right-hand side that is not Lipschitz continuous in a function space norm of the sort introduced in Section A.4. Discretizing on a grid replaces $\partial_x^2 u$ by $f(U) = AU$, which is Lipschitz continuous, but the Lipschitz constant $\|A\|$ grows at the rate of $1/h^2$ as the grid is refined. In the nonlinear case it is often difficult to obtain the sort of bounds needed to prove convergence. See [40], [68], [75], or [84] for further discussions of stability.

9.6 Von Neumann analysis

Although it is useful to go through the MOL formulation to understand how stability theory for PDEs is related to the theory for ODEs, in practice there is another approach that will sometimes give the proper stability restrictions more easily.

The von Neumann approach to stability analysis is based on Fourier analysis and hence is generally limited to constant coefficient linear PDEs. For simplicity it is usually applied to the *Cauchy problem*, which is the PDE on all space with no boundaries, $-\infty < x < \infty$ in the one-dimensional case. Von Neumann analysis can also be used to study the stability of problems with *periodic boundary conditions*, e.g., in $0 \le x \le 1$ with $u(0,t) = u(1,t)$ imposed. This is generally equivalent to a Cauchy problem with periodic initial data.

Stability theory for PDEs with more general boundary conditions can often be quite difficult, as the coupling between the discretization of the boundary conditions and the discretization of the PDE can be very subtle. Von Neumann analysis addresses the issue of stability of the PDE discretization alone. Some discussion of stability theory for initial boundary value problems can be found in [84], [75]. See also Section 10.12.

The Cauchy problem for linear PDEs can be solved using Fourier transforms—see Section E.3 for a review. The basic reason this works is that the functions $e^{i\xi x}$ with wave number $\xi =$ constant are eigenfunctions of the differential operator ∂_x,

$$\partial_x e^{i\xi x} = i\xi e^{i\xi x},$$

and hence of any constant coefficient linear differential operator. Von Neumann analysis is based on the fact that the related grid function $W_j = e^{ijh\xi}$ is an eigenfunction of any translation-invariant finite difference operator.[1] For example, if we approximate $v'(x_j)$ by $D_0 V_j = \frac{1}{2h}(V_{j+1} - V_{j-1})$, then in general the grid function $D_0 V$ is not a scalar multiple of V. But for the special case of W, we obtain

$$\begin{aligned} D_0 W_j &= \frac{1}{2h}\left(e^{i(j+1)h\xi} - e^{i(j-1)h\xi}\right) \\ &= \frac{1}{2h}\left(e^{ih\xi} - e^{-ih\xi}\right) e^{ijh\xi} \\ &= \frac{i}{h}\sin(h\xi) e^{ijh\xi} \\ &= \frac{i}{h}\sin(h\xi) W_j. \end{aligned} \tag{9.23}$$

So W is an "eigengridfunction" of the operator D_0, with eigenvalue $\frac{i}{h}\sin(h\xi)$.

Note the relation between these and the eigenfunctions and eigenvalues of the operator ∂_x found earlier: W_j is simply the eigenfunction $w(x)$ of ∂_x evaluated at the point x_j, and for small $h\xi$ we can approximate the eigenvalue of D_0 by

[1] In this section $i = \sqrt{-1}$ and the index j is used on the grid functions.

$$\frac{i}{h}\sin(h\xi) = \frac{i}{h}\left(h\xi - \frac{1}{6}h^3\xi^3 + O(h^5\xi^5)\right)$$
$$= i\xi - \frac{i}{6}h^2\xi^3 + \cdots.$$

This agrees with the eigenvalue $i\xi$ of ∂_x to $O(h^2\xi^3)$.

Suppose we have a grid function V_j defined at grid points $x_j = jh$ for $j = 0, \pm 1, \pm 2, \ldots$, which is an l_2 function in the sense that the 2-norm

$$\|U\|_2 = \left(h \sum_{j=-\infty}^{\infty} |U_j|^2\right)^{1/2}$$

is finite. Then we can express V_j as a linear combination of the grid functions $e^{ijh\xi}$ for all ξ in the range $-\pi/h \le \xi \le \pi/h$. Functions with larger wave number ξ cannot be resolved on this grid. We can write

$$V_j = \frac{1}{\sqrt{2\pi}} \int_{-\pi/h}^{\pi/h} \hat{V}(\xi) e^{ijh\xi}\, d\xi,$$

where

$$\hat{V}(\xi) = \frac{h}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} V_j e^{-ijh\xi}.$$

These are direct analogue of the formulas for a function $v(x)$ in the discrete case.

Again we have *Parseval's relation*, $\|\hat{V}\|_2 = \|V\|_2$, although the 2-norms used for the grid function V_j and the function $\hat{V}(\xi)$ defined on $[-\pi/h, \pi/h]$ are different:

$$\|V\|_2 = \left(h \sum_{j=-\infty}^{\infty} |V_j|^2\right)^{1/2}, \qquad \|\hat{V}\|_2 = \left(\int_{-\pi/h}^{\pi/h} |\hat{V}(\xi)|^2\, d\xi\right)^{1/2}.$$

To show that a finite difference method is stable in the 2-norm by the techniques discussed earlier in this chapter, we would have to show that $\|B\|_2 \le 1 + \alpha k$ in the notation of (9.22). This amounts to showing that there is a constant α such that

$$\|U^{n+1}\|_2 \le (1 + \alpha k)\|U^n\|_2$$

for all U^n. This can be difficult to attack directly because of the fact that computing $\|U\|_2$ requires summing over all grid points, and each U_j^{n+1} depends on values of U^n at neighboring grid points so that all grid points are coupled together. In some cases one can work with these infinite sums directly, but it is rare that this can be done. Alternatively one can work with the matrix B itself, as we did above in Section 9.5, but this matrix is growing as we refine the grid.

Using Parseval's relation, we see that it is sufficient to instead show that

$$\|\hat{U}^{n+1}\|_2 \le (1 + \alpha k)\|\hat{U}^n\|_2,$$

where $\hat{U}^n$ is the Fourier transform of the grid function U^n. The utility of Fourier analysis now stems from the fact that after Fourier transforming the finite difference method, we obtain a recurrence relation for each $\hat{U}^n(\xi)$ that is decoupled from all other wave numbers. For a two-level method this has the form

$$\hat{U}^{n+1}(\xi) = g(\xi)\hat{U}^n(\xi). \tag{9.24}$$

The factor $g(\xi)$, which depends on the method, is called the *amplification factor* for the method at wave number ξ. If we can show that

$$|g(\xi)| \le 1 + \alpha k,$$

where α is independent of ξ, then it follows that the method is stable, since then

$$|\hat{U}^{n+1}(\xi)| \le (1 + \alpha k)|\hat{U}^n(\xi)| \quad \text{for all } \xi$$

and so

$$\|\hat{U}^{n+1}\|_2 \le (1 + \alpha k)\|\hat{U}^n\|_2.$$

Fourier analysis allows us to obtain simple scalar recursions of the form (9.24) for each wave number separately, rather than dealing with a system of equations for U_j^n that couples together all values of j.

Note: Here we are assuming that $u(x, t)$ is a scalar, so that $g(\xi)$ is a scalar. For a system of s equations we would find that $g(\xi)$ is an $s \times s$ matrix for each value of ξ, so some analysis of matrix eigenvalues is still required to investigate stability. But the dimension of the matrices is s, independent of the grid spacing, unlike the MOL analysis, where the matrix dimension increases as $k \to 0$.

Example 9.5. Consider the method (9.5). To apply von Neumann analysis we consider how this method works on a single wave number ξ, i.e., we set

$$U_j^n = e^{ijh\xi}. \tag{9.25}$$

Then we expect that

$$U_j^{n+1} = g(\xi)e^{ijh\xi}, \tag{9.26}$$

where $g(\xi)$ is the amplification factor for this wave number. Inserting these expressions into (9.5) gives

$$\begin{aligned} g(\xi)e^{ijh\xi} &= e^{ijh\xi} + \frac{k}{h^2}\left(e^{i\xi(j-1)h} - 2e^{ijh\xi} + e^{i\xi(j+1)h}\right) \\ &= \left(1 + \frac{k}{h^2}\left(e^{-i\xi h} - 2 + e^{i\xi h}\right)\right)e^{ijh\xi}, \end{aligned}$$

and hence

$$g(\xi) = 1 + 2\frac{k}{h^2}(\cos(\xi h) - 1).$$

Since $-1 \le \cos(\xi h) \le 1$ for any value of ξ, we see that

$$1 - 4\frac{k}{h^2} \le g(\xi) \le 1$$

for all ξ. We can guarantee that $|g(\xi)| \le 1$ for all ξ if we require

$$4\frac{k}{h^2} \le 2.$$

This is exactly the stability restriction (9.14) we found earlier for this method. If this restriction is violated, then the Fourier components with some wave number ξ will be amplified (and, as expected, it is the largest wave numbers that become unstable first as k is increased).

Example 9.6. The fact that the Crank–Nicolson method is stable for all k and h can also be shown using von Neumann analysis. Substituting (9.25) and (9.26) into the difference equations (9.7) and canceling the common factor of $e^{ijh\xi}$ gives the following relation for $g \equiv g(\xi)$:

$$g = 1 + \frac{k}{2h^2}\left(e^{-i\xi h} - 2 + e^{i\xi h}\right)(1 + g),$$

and hence

$$g = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z}, \tag{9.27}$$

where

$$\begin{aligned} z &= \frac{k}{h^2}(e^{-i\xi h} - 2 + e^{i\xi h}) \\ &= \frac{2k}{h^2}(\cos(\xi h) - 1). \end{aligned} \tag{9.28}$$

Since $z \le 0$ for all ξ, we see that $|g| \le 1$ and the method is stable for any choice of k and h.

Note that (9.27) agrees with the root ζ_1 found for the trapezoidal method in Example 7.6, while the z determined in (9.28), for certain values of ξ, is simply k times an eigenvalue λ_p from (9.13), the eigenvalues of the MOL matrix (9.11). In general there is a close connection between the von Neumann approach and the MOL reduction of a periodic problem to a system of ODEs.

9.7 Multidimensional problems

In two space dimensions the heat equation takes the form

$$u_t = u_{xx} + u_{yy} \tag{9.29}$$

with initial conditions $u(x, y, 0) = \eta(x, y)$ and boundary conditions all along the boundary of our spatial domain Ω. We can discretize in space using a discrete Laplacian of the form considered in Chapter 3, say, the 5-point Laplacian from Section 3.2:

$$\nabla_h^2 U_{ij} = \frac{1}{h^2}(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{ij}). \tag{9.30}$$

If we then discretize in time using the trapezoidal method, we will obtain the two-dimensional version of the Crank–Nicolson method,

$$U_{ij}^{n+1} = U_{ij}^n + \frac{k}{2}\left[\nabla_h^2 U_{ij}^n + \nabla_h^2 U_{ij}^{n+1}\right]. \tag{9.31}$$

Since this method is implicit, we must solve a system of equations for all the U_{ij} where the matrix has the same nonzero structure as for the elliptic systems considered in Chapters 3 and 4. This matrix is large and sparse, and we generally do not want to solve the system by a direct method such as Gaussian elimination. This is even more true for the systems we are now considering than for the elliptic equation, because of the slightly different nature of this system, which makes other approaches even more efficient relative to direct methods. It is also extremely important now that we use the most efficient method possible, because we must now solve a linear system of this form *in every time step*, and we may need to take thousands of time steps to solve the time-dependent problem.

We can rewrite the equations (9.31) as

$$\left(I - \frac{k}{2}\nabla_h^2\right) U_{ij}^{n+1} = \left(I + \frac{k}{2}\nabla_h^2\right) U_{ij}^n. \tag{9.32}$$

The matrix for this linear system has the same pattern of nonzeros as the matrix for ∇_h^2 (see Chapter 3), but the values are scaled by $k/2$ and then subtracted from the identity matrix, so that the diagonal elements are fundamentally different. If we call this matrix A,

$$A = I - \frac{k}{2}\nabla_h^2,$$

then we find that the eigenvalues of A are

$$\lambda_{p,q} = 1 - \frac{k}{h^2}\big[(\cos(p\pi h) - 1) + (\cos(q\pi h) - 1)\big]$$

for $p,\ q = 1,\ 2,\ \ldots,\ m$, where we have used the expression for the eigenvalues of ∇_h^2 from Section 3.4. Now the largest eigenvalue of the matrix A thus has magnitude $O(k/h^2)$ while the ones closest to the origin are at $1 + O(k)$. As a result the condition number of A is $O(k/h^2)$. By contrast, the discrete Laplacian ∇_h^2 alone has condition number $O(1/h^2)$ as we found in Section 3.4. The smaller condition number in the present case can be expected to lead to faster convergence of iterative methods.

Moreover, we have an excellent starting guess for the solution U^{n+1} to (9.31), a fact that we can use to good advantage with iterative methods but not with direct methods. Since $U_{ij}^{n+1} = U_{ij}^n + O(k)$, we can use U_{ij}^n, the values from the previous time step, as initial values $U_{ij}^{[0]}$ for an iterative method. We might do even better by extrapolating forward in time, using, say, $U_{ij}^{[0]} = 2U_{ij}^n - U_{ij}^{n-1}$, or by using an explicit method, say,

$$U_{ij}^{[0]} = (I + k\nabla_h^2)U_{ij}^n.$$

This explicit method (forward Euler) would probably be unstable as a time-marching procedure if we used only this with the value of k we have in mind, but it can sometimes be used successfully as a way to generate initial data for an iterative procedure.

Because of the combination of a reasonably well-conditioned system and very good initial guess, we can often get away with taking only one or two iterations in each time step, and still get global second order accuracy.

9.8 The locally one-dimensional method

Rather than solving the coupled sparse matrix equation for all the unknowns on the grid simultaneously as in (9.32), an alternative approach is to replace this fully coupled single time step with a sequence of steps, each of which is coupled in only one space direction, resulting in a set of tridiagonal systems which can be solved much more easily. One example is the *locally one-dimensional (LOD)* method:

$$U_{ij}^* = U_{ij}^n + \frac{k}{2}(D_x^2 U_{ij}^n + D_x^2 U_{ij}^*), \tag{9.33}$$

$$U_{ij}^{n+1} = U_{ij}^* + \frac{k}{2}(D_y^2 U_{ij}^* + D_y^2 U_{ij}^{n+1}), \tag{9.34}$$

or, in matrix form,

$$\left(I - \frac{k}{2}D_x^2\right) U^* = \left(I + \frac{k}{2}D_x^2\right) U^n, \tag{9.35}$$

$$\left(I - \frac{k}{2}D_y^2\right) U^{n+1} = \left(I + \frac{k}{2}D_y^2\right) U^*. \tag{9.36}$$

In (9.33) we apply Crank–Nicolson in the x-direction only, solving $u_t = u_{xx}$ alone over time k, and we call the result U^*. Then in (9.34) we take this result and apply Crank–Nicolson in the y-direction to it, solving $u_t = u_{yy}$ alone, again over time k. Physically this corresponds to modeling diffusion in the x- and y-directions over time k as a decoupled process in which we first allow u to diffuse only in the x-direction and then only in the y-direction. If the time steps are very short, then this might be expected to give similar physical behavior and hence convergence to the correct behavior as $k \to 0$. In fact, for the constant coefficient diffusion problem, it can even be shown that (in the absence of boundaries at least) this alternating diffusion approach gives *exactly* the same behavior as the original two-dimensional diffusion. Diffusing first in x alone over time k and then in y alone over time k gives the same result as if the diffusion occurs simultaneously in both directions. (This is because the differential operators ∂_x^2 and ∂_y^2 commute, as discussed further in Example 11.1.)

Numerically there is a great advantage in using (9.35) and (9.36) rather than the fully coupled (9.32). In (9.35) the unknowns U_{ij}^* are coupled together only across each row of the grid. For any fixed value of j we have a *tridiagonal system* of equations to solve for $U_{ij}^*(i = 1, 2, \ldots, m)$. The system obtained for each value of j is completely decoupled from the system obtained for other values of j. Hence we have a set of $m + 2$ tridiagonal systems to solve (for $j = 0, 1, \ldots, m + 1$), each of dimension m, rather than a single coupled system with m^2 unknowns as in (9.32). Since each of these systems is tridiagonal, it is easily solved in $O(m)$ operations by Gaussian elimination and there is no need for iterative methods. (In the next section we will see why we need to solve these for $j = 0$ and $j = m + 1$ as well as at the interior grid points.)

Similarly, (9.34) decouples into a set of m tridiagonal systems in the y-direction for $i = 1, 2, \ldots, m$. Hence taking a single time step requires solving $2m + 2$ tridiagonal systems of size m, and thus $O(m^2)$ work. Since there are m^2 grid points, this is the optimal order and no worse than an explicit method, except for a constant factor.

9.8.1 Boundary conditions

In solving the second set of systems (9.34), we need boundary values U^*_{i0} and U^{n+1}_{i0} along the bottom boundary and $U^*_{i,m+1}$ and $U^{n+1}_{i,m+1}$ along the top boundary, for terms that go on the right-hand side of each tridiagonal system. The values at level $n+1$ are available from the given boundary data for the heat equation, by evaluating the boundary conditions at time t_{n+1} (assuming Dirichlet boundary conditions are given). To obtain the values U^*_{i0} we solve (9.33) for $j = 0$ and $j = m+1$ (along the boundaries) in addition to the systems along each row interior to the grid.

To solve the first set of systems (9.33), we need boundary values U^n_{0j} and U^*_{0j} along the left boundary and values $U^n_{m+1,j}$ and $U^*_{m+1,j}$ along the right boundary. The values at level n come from the given boundary conditions, but we must determine the intermediate boundary conditions at level $*$ along these boundaries. It is not immediately clear what values should be used. One might be tempted to think of level $*$ as being halfway between t_n and t_{n+1}, since U^* is generated in the middle of the two-step procedure used to obtain U^{n+1} from U^n. If this were valid, then evaluating the given boundary data at time $t_{n+1/2} = t_n + k/2$ might provide values for U^* on the boundary. This is not a good idea, however, and would lead to a degradation of accuracy. The problem is that in the first step, (9.33) does not model the full heat equation over time $k/2$ but rather models part of the equation (diffusion in x alone) over the full time step k. The values along the boundary will in general evolve quite differently in the two different cases.

To determine proper values for U^*_{0j} and $U^*_{m+1,j}$, we can use (9.34) along the left and right boundaries. At $i = 0$, for example, this equation gives a system of equations along the left boundary that can be viewed as a tridiagonal linear system or the unknowns U^*_{0j} in terms of the values U^{n+1}_{0j}, which are already known from the boundary conditions at time t_{n+1}. Note that we are solving this equation backward from the way it will be used in the second step of the LOD process on the interior of the grid, and this works only because we already know U^{n+1}_{0j} from boundary data.

Since we are solving this equation backward, we can view this as solving the diffusion equation $u_t = u_{yy}$ over a time step of length $-k$, backward in time. This makes sense physically—the intermediate solution U^* represents what is obtained from U^n by doing diffusion in x alone, with no diffusion yet in y. There are in principle two ways to get this, either by starting with U^n and diffusing in x or by starting with U^{n+1} and "undiffusing" in y. We are using the latter approach along the boundaries to generate data for U^*.

Equivalently we can view this as solving the *backward heat equation* $u_t = -u_{yy}$ over time k. This may be cause for concern, since the backward heat equation is ill posed (see Section E.3.4). However, since we are doing this only over one time step starting with given values U^{n+1}_{0j} in each time step, this turns out to be a stable procedure.

There is still a difficulty at the corners. To solve (9.34) for U^*_{0j}, $j = 1, 2, \ldots, m$, we need to know the values of U^*_{00} and $U^*_{0,m+1}$ that are the boundary values for this system. These can be approximated using some sort of explicit and uncentered approximation to either $u_t = u_{xx}$ starting with U^n, or to $u_t = -u_{yy}$ starting with U^{n+1}. For example, we might use

$$U_{00}^* = U_{00}^{n+1} - \frac{k}{h^2}(U_{00}^{n+1} - 2U_{01}^{n+1} + U_{02}^{n+1}),$$

which uses the approximation to u_{yy} centered at (x_0, y_1).

Alternatively, rather than solving the tridiagonal systems obtained from (9.34) for U_{0j}^*, we could simply use an explicit approximation to the backward heat equation along this boundary,

$$U_{0j}^* = U_{0j}^{n+1} - \frac{k}{h^2}(U_{0,j-1}^{n+1} - 2U_{0j}^{n+1} + U_{0,j+1}^{n+1}) \tag{9.37}$$

for $j = 1, 2, \ldots, m$. This eliminates the need for values of U^* in the corners. Again, since this is not iterated but done only starting with given (and presumably smooth) boundary data U^{n+1} in each time step, this yields a stable procedure.

With proper treatment of the boundary conditions, it can be shown that the LOD method is second order accurate (see Example 11.1). It can also be shown that this method, like full Crank–Nicolson, is unconditionally stable for any time step.

9.8.2 The alternating direction implicit method

A modification of the LOD method is also often used, in which the two steps each involve discretization in only one spatial direction at the advanced time level (giving decoupled tridiagonal systems again) but coupled with discretization in the *opposite* direction at the old time level. The classical method of this form is

$$U_{ij}^* = U_{ij}^n + \frac{k}{2}(D_y^2 U_{ij}^n + D_x^2 U_{ij}^*), \tag{9.38}$$

$$U_{ij}^{n+1} = U_{ij}^* + \frac{k}{2}(D_x^2 U_{ij}^* + D_y^2 U_{ij}^{n+1}). \tag{9.39}$$

This is called the *alternating direction implicit (ADI)* method and was first introduced by Douglas and Rachford [26]. This again gives decoupled tridiagonal systems to solve in each step:

$$\left(I - \frac{k}{2}D_x^2\right) U^* = \left(I + \frac{k}{2}D_y^2\right) U^n, \tag{9.40}$$

$$\left(I - \frac{k}{2}D_y^2\right) U^{n+1} = \left(I + \frac{k}{2}D_x^2\right) U^*. \tag{9.41}$$

With this method, each of the two steps involves diffusion in both the x- and the y-direction. In the first step the diffusion in x is modeled implicitly, while diffusion in y is modeled explicitly, with the roles reversed in the second step. In this case each of the two steps can be shown to give a *first order accurate* approximation to the full heat equation over time $k/2$, so that U^* represents a first order accurate approximation to the solution at time $t_{n+1/2}$. Because of the symmetry of the two steps, however, the local error introduced in the second step almost exactly cancels the local error introduced in the first step, so that the combined method is in fact *second order accurate* over the full time step.

Because U^* does approximate the solution at time $t_{n+1/2}$ in this case, it is possible to simply evaluate the given boundary conditions at time $t_{n+1/2}$ to generate the necessary boundary values for U^*. This will maintain second order accuracy. A better error constant

can be achieved by using slightly modified boundary data which introduces the expected error in U^* into the boundary data that should be canceled out by the second step.

9.9 Other discretizations

For illustration purposes we have considered only the classic Crank–Nicolson method consisting of second order centered approximation to u_{xx} coupled with the trapezoidal method for time stepping. However, an infinite array of other combinations of spatial approximation and time stepping methods could be considered, some of which may be preferable. The following are a few possibilities:

- The second order accurate spatial difference operator could be replaced by a higher order method, such as the fourth order accurate approximations of Section 2.20.1 in one dimension of Section 3.5 in more dimensions.

- A spectral method could be used in the spatial dimension(s), as discussed in Section 2.21. Note that in this case the linear system that must be solved in each time step will be dense. On the other hand, for many problems it is possible to use a much coarser grid for spectral methods, leading to relatively small linear algebra problems.

- The time-stepping procedure could be replaced by a different implicit method suitable for stiff equations, of the sort discussed in Chapter 8. In particular, for some problems it is desirable to use an L-stable method. While the trapezoidal method is stable, it does not handle underresolved transients well (recall Figure 8.4). For some problems where diffusion is coupled with other processes there are constantly high-frequency oscillations or discontinuities introduced that should be smoothed by diffusion, and Crank–Nicolson can suffer from oscillations in time.

- The time stepping could be done by using a method such as the Runge–Kutta–Chebyshev method described in Section 8.6. This is an explicit method that works for mildly stiff problems with real eigenvalues, such as the heat equation.

- The time stepping could be done using the exponential time differencing (ETD) methods described in Section 11.6. The heat equation with constant coefficients and time-varying boundary conditions leads to a MOL discretization of the form (9.11), where A is a constant matrix. If the centered difference approximation is used in one dimension, then (9.12) holds, but even with other discretizations, or in more dimensions, the semidiscrete system still has the form $U'(t) = AU(t) + g(t)$. The exact solution can be written in terms of the matrix exponential e^{At} and this form is used in the ETD methods. The manner in which this is computed depends on whether A is large and sparse (the typical case with a finite difference discretization) or small and dense (as it might be if a spectral discretization is used in space). See Section 11.6.1 for more discussion of this.

Chapter 10
Advection Equations and Hyperbolic Systems

Hyperbolic partial differential equations (PDEs) arise in many physical problems, typically whenever wave motion is observed. Acoustic waves, electromagnetic waves, seismic waves, shock waves, and many other types of waves can be modeled by hyperbolic equations. Often these are modeled by linear hyperbolic equations (for the propagation of sufficiently small perturbations), but modeling large motions generally requires solving nonlinear hyperbolic equations. Hyperbolic equations also arise in advective transport, when a substance is carried along with a flow, giving rise to an advection equation. This is a scalar linear first order hyperbolic PDE, the simplest possible case. See Appendix E for more discussion of hyperbolic problems and a derivation of the advection equation in particular.

In this chapter we will primarily consider the advection equation. This is sufficient to illustrate many (although certainly not all) of the issues that arise in the numerical solution of hyperbolic equations. Section 10.10 contains a very brief introduction to hyperbolic systems, still in the linear case. A much more extensive discussion of hyperbolic problems and numerical methods, including nonlinear problems and multidimensional methods, can be found in [66]. Those interested in solving more challenging hyperbolic problems may also look at the CLAWPACK software [64], which was designed primarily for hyperbolic problems. There are also a number of other books devoted to nonlinear hyperbolic equations and their solution, e.g., [58], [88].

10.1 Advection

In this section we consider numerical methods for the scalar advection equation

$$u_t + au_x = 0, \tag{10.1}$$

where a is a constant. See Section E.2.1 for a discussion of this equation. For the Cauchy problem we also need initial data

$$u(x, 0) = \eta(x).$$

This is the simplest example of a *hyperbolic* equation, and it is so simple that we can write down the exact solution,

$$u(x,t) = \eta(x - at). \tag{10.2}$$

One can verify directly that this is the solution (see also Appendix E). However, many of the issues that arise more generally in discretizing hyperbolic equations can be most easily seen with this equation.

The first approach we might consider is the analogue of the method (9.4) for the heat equation. Using the centered difference in space,

$$u_x(x,t) = \frac{u(x+h,t) - u(x-h,t)}{2h} + O(h^2) \tag{10.3}$$

and the forward difference in time results in the numerical method

$$\frac{U_j^{n+1} - U_j^n}{k} = -\frac{a}{2h}(U_{j+1}^n - U_{j-1}^n), \tag{10.4}$$

which can be rewritten as

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n). \tag{10.5}$$

This again has the stencil shown in Figure 9.1(a). In practice this method is not useful because of stability considerations, as we will see in the next section.

A minor modification gives a more useful method. If we replace U_j^n on the right-hand side of (10.5) by the average $\frac{1}{2}(U_{j-1}^n + U_{j+1}^n)$, then we obtain the *Lax–Friedrichs method*,

$$U_j^{n+1} = \frac{1}{2}(U_{j-1}^n + U_{j+1}^n) - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n). \tag{10.6}$$

Because of the low accuracy, this method is not commonly used in practice, but it serves to illustrate some stability issues and so we will study this method along with (10.5) before describing higher order methods, such as the well-known Lax–Wendroff method.

We will see in the next section that Lax–Friedrichs is Lax–Richtmyer stable (see Section 9.5) and convergent provided

$$\left|\frac{ak}{h}\right| \le 1. \tag{10.7}$$

Note that this stability restriction allows us to use a time step $k = O(h)$ although the method is explicit, unlike the case of the heat equation. The basic reason is that the advection equation involves only the first order derivative u_x rather than u_{xx} and so the difference equation involves $1/h$ rather than $1/h^2$.

The time step restriction (10.7) is consistent with what we would choose anyway based on accuracy considerations, and in this sense the advection equation is *not stiff*, unlike the heat equation. This is a fundamental difference between hyperbolic equations and parabolic equations more generally and accounts for the fact that hyperbolic equations are typically solved with explicit methods, while the efficient solution of parabolic equations generally requires implicit methods.

To see that (10.7) gives a reasonable time step, note that

$$u_x(x,t) = \eta'(x - at),$$

while

$$u_t(x,t) = -au_x(x,t) = -a\eta'(x - at).$$

The time derivative u_t is larger in magnitude than u_x by a factor of a, and so we would expect the time step required to achieve temporal resolution consistent with the spatial resolution h to be smaller by a factor of a. This suggests that the relation $k \approx h/a$ would be reasonable in practice. This is completely consistent with (10.7).

10.2 Method of lines discretization

To investigate stability further we will again introduce the method of lines (MOL) discretization as we did in Section 9.2 for the heat equation. To obtain a system of equations with finite dimension we must solve the equation on some bounded domain rather than solving the Cauchy problem. However, in a bounded domain, say, $0 \le x \le 1$, the advection equation can have a boundary condition specified on only one of the two boundaries. If $a > 0$, then we need a boundary condition at $x = 0$, say,

$$u(0,t) = g_0(t), \tag{10.8}$$

which is the *inflow* boundary in this case. The boundary at $x = 1$ is the *outflow* boundary and the solution there is completely determined by what is advecting to the right from the interior. If $a < 0$, we instead need a boundary condition at $x = 1$, which is the inflow boundary in this case.

The symmetric 3-point methods defined above can still be used near the inflow boundary but not at the outflow boundary. Instead the discretization will have to be coupled with some "numerical boundary condition" at the outflow boundary, say, a one-sided discretization of the equation. This issue complicates the stability analysis and will be discussed in Section 10.12.

For analysis purposes we can obtain a nice MOL discretization if we consider the special case of *periodic boundary conditions*,

$$u(0,t) = u(1,t) \quad \text{for } t \ge 0.$$

Physically, whatever flows out at the outflow boundary flows back in at the inflow boundary. This also models the Cauchy problem in the case where the initial data is periodic with period 1, in which case the solution remains periodic and we need to model only a single period $0 \le x \le 1$.

In this case the value $U_0(t) = U_{m+1}(t)$ along the boundaries is another unknown, and we must introduce one of these into the vector $U(t)$. If we introduce $U_{m+1}(t)$, then we have the vector of grid values

$$U(t) = \begin{bmatrix} U_1(t) \\ U_2(t) \\ \vdots \\ U_{m+1}(t) \end{bmatrix}.$$

For $2 \le j \le m$ we have the ordinary differential equation (ODE)

$$U_j'(t) = -\frac{a}{2h}(U_{j+1}(t) - U_{j-1}(t)),$$

while the first and last equations are modified using the periodicity:

$$U_1'(t) = -\frac{a}{2h}(U_2(t) - U_{m+1}(t)),$$
$$U_{m+1}'(t) = -\frac{a}{2h}(U_1(t) - U_m(t)).$$

This system can be written as

$$U'(t) = AU(t) \tag{10.9}$$

with

$$A = -\frac{a}{2h}\begin{bmatrix} 0 & 1 & & & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ 1 & & & & -1 & 0 \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}. \tag{10.10}$$

Note that this matrix is skew-symmetric ($A^T = -A$) and so its eigenvalues must be pure imaginary. In fact, the eigenvalues are

$$\lambda_p = -\frac{ia}{h}\sin(2\pi ph) \quad \text{for } p = 1,\ 2,\ \ldots,\ m+1. \tag{10.11}$$

The corresponding eigenvector u^p has components

$$u_j^p = e^{2\pi ipjh} \quad \text{for } j = 1,\ 2,\ \ldots,\ m+1. \tag{10.12}$$

The eigenvalues lie on the imaginary axis between $-ia/h$ and ia/h.

For absolute stability of a time discretization we need the stability region $\mathcal{S}$ to include this interval. Any method that includes some interval iy, $|y| < b$ of the imaginary axis will lead to a stable method for the advection equation provided $|ak/h| \le b$. For example, looking again at the stability regions plotted in Figures 7.1 through 7.3 and Figure 8.5 shows that the midpoint method or certain Adams methods may be suitable for this problem, whereas the backward differentiation formula (BDF) methods are not.

10.2.1 Forward Euler time discretization

The method (10.5) can be viewed as the forward Euler time discretization of the MOL system of ODEs (10.9). We found in Section 7.3 that this method is stable only if $|1+k\lambda| \le 1$ and the stability region $\mathcal{S}$ is the unit circle centered at -1. No matter how small the ratio k/h is, since the eigenvalues λ_p from (10.11) are imaginary, the values $k\lambda_p$ will not lie in $\mathcal{S}$. Hence the method (10.5) is *unstable* for any fixed mesh ratio k/h; see Figure 10.1(a).

The method (10.5) will be convergent if we let $k \to 0$ faster than h, since then $k\lambda_p \to 0$ for all p and the zero-stability of Euler's method is enough to guarantee convergence. Taking k much smaller than h is generally not desirable and the method is not used in practice. However, it is interesting to analyze this situation also in terms of Lax–Richtmyer stability, since it shows an example where the Lax–Richtmyer stability uses a weaker bound of the form (9.22), $\|B\| \le 1 + \alpha k$, rather than $\|B\| \le 1$. Here $B = I + kA$. Suppose we take $k = h^2$, for example. Then we have

$$|1 + k\lambda_p|^2 \le 1 + (ka/h)^2$$

for each p (using the fact that λ_p is pure imaginary) and so

$$|1 + k\lambda_p|^2 \le 1 + a^2h^2 = 1 + a^2k.$$

Hence $\|I + kA\|_2^2 \le 1 + a^2k$ and if $nk \le T$, we have

$$\|(I + kA)^n\|_2 \le (1 + a^2k)^{n/2} \le e^{a^2T/2},$$

showing the uniform boundedness of $\|B^n\|$ (in the 2-norm) needed for Lax–Richtmyer stability.

10.2.2 Leapfrog

A better time discretization is to use the midpoint method (5.23),

$$U^{n+1} = U^{n-1} + 2kAU^n,$$

which gives the *leapfrog method* for the advection equation,

$$U_j^{n+1} = U_j^{n-1} - \frac{ak}{h}(U_{j+1}^n - U_{j-1}^n). \tag{10.13}$$

This is a 3-level explicit method and is second order accurate in both space and time.

Recall from Section 7.3 that the stability region of the midpoint method is the interval $i\alpha$ for $-1 < \alpha < 1$ of the imaginary axis. This method is hence stable on the advection equation provided $|ak/h| < 1$ is satisfied.

On the other hand, note that the $k\lambda_p$ will always be on the *boundary* of the stability region (the stability region for midpoint has no interior). This means the method is only marginally stable—there is no growth but also no decay of any eigenmode. The difference equation is said to be *nondissipative*. In some ways this is good—the true advection equation is also nondissipative, and any initial condition simply translates unchanged, no matter how oscillatory. Leapfrog captures this qualitative behavior well.

However, there are problems with this. All modes translate without decay, but they do not all propagate at the correct velocity, as will be explained in Example 10.12. As a result initial data that contains high wave number components (e.g., if the data contains steep gradients) will disperse and can result in highly oscillatory numerical approximations.

The marginal stability of leapfrog can also turn into instability if a method of this sort is applied to a more complicated problem with variable coefficients or nonlinearities.

10.2.3 Lax–Friedrichs

Again consider the Lax–Friedrichs method (10.6). Note that we can rewrite (10.6) using the fact that

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

to obtain

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{1}{2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n). \tag{10.14}$$

This can be rearranged to give

$$\frac{U_j^{n+1} - U_j^n}{k} + a\left(\frac{U_{j+1}^n - U_{j-1}^n}{2h}\right) = \frac{h^2}{2k}\left(\frac{U_{j-1}^n - 2U_j^n + U_{j+1}^n}{h^2}\right).$$

If we compute the local truncation error from this form we see, as expected, that it is consistent with the advection equation $u_t + au_x = 0$, since the term on the right-hand side vanishes as $k,\ h \to 0$ (assuming k/h is fixed). However, it looks more like a discretization of the advection-diffusion equation

$$u_t + au_x = \epsilon u_{xx},$$

where $\epsilon = h^2/2k$.

Later in this chapter we will study the diffusive nature of many methods for the advection equation. For our present purposes, however, the crucial part is that we can now view (10.14) as resulting from a forward Euler discretization of the system of ODEs

$$U'(t) = A_\epsilon U(t)$$

with

$$\begin{aligned} A_\epsilon = &-\frac{a}{2h}\begin{bmatrix} 0 & 1 & & & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ 1 & & & & -1 & 0 \end{bmatrix} \\ &+ \frac{\epsilon}{h^2}\begin{bmatrix} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ 1 & & & & 1 & -2 \end{bmatrix}, \end{aligned} \tag{10.15}$$

where $\epsilon = h^2/2k$. The matrix A_ϵ differs from the matrix A of (10.10) by the addition of a small multiple of the second difference operator, which is symmetric rather than skew-symmetric. As a result the eigenvalues of A_ϵ are shifted off the imaginary axis and now lie

in the left half-plane. There is now some hope that each $k\lambda$ will lie in the stability region of Euler's method if k is small enough relative to h.

It can be verified that the eigenvectors (10.12) of the matrix A are also eigenvectors of the second difference operator (with periodic boundary conditions) that appears in (10.15), and hence these are also the eigenvectors of the full matrix A_ϵ. We can easily compute that the eigenvalues of A_ϵ are

$$\mu_p = -\frac{ia}{h}\sin(2\pi ph) - \frac{2\epsilon}{h^2}(1-\cos(2\pi ph)). \tag{10.16}$$

The values $k\mu_p$ are plotted in the complex plane for various different values of ϵ in Figure 10.1. They lie on an ellipse centered at $-2k\epsilon/h^2$ with semi-axes of length $2k\epsilon/h^2$ in the x-direction and ak/h in the y-direction. For the special case $\epsilon = h^2/2k$ used in Lax–Friedrichs, we have $-2k\epsilon/h^2 = -1$ and this ellipse lies entirely inside the unit circle centered at -1, provided that $|ak/h| \le 1$. (If $|ak/h| > 1$, then the top and bottom of the ellipse would extend outside the circle.) The forward Euler method is stable as a time-discretization, and hence the Lax–Friedrichs method is Lax–Richtmyer stable, provided $|ak/h| \le 1$.

10.3 The Lax–Wendroff method

One way to achieve second order accuracy on the advection equation is to use a second order temporal discretization of the system of ODEs (10.9), since this system is based on a second order spatial discretization. This can be done with the midpoint method, for example, which gives rise to the leapfrog scheme (10.13) already discussed. However, this is a three-level method and for various reasons it is often much more convenient to use two-level methods for PDEs whenever possible—in more than one dimension the need to store several levels of data may be restrictive, boundary conditions can be harder to impose, and combining methods using fractional step procedures (as discussed in Chapter 11) may require two-level methods for each step, to name a few reasons. Moreover, the leapfrog method is nondissipative, leading to potential stability problems if the method is extended to variable coefficient or nonlinear problems.

Another way to achieve second order accuracy in time would be to use the trapezoidal method to discretize the system (10.9), as was done to derive the Crank–Nicolson method for the heat equation. But this is an implicit method and for hyperbolic equations there is generally no need to introduce this complication and expense.

Another possibility is to use a two-stage Runge–Kutta method such as the one in Example 5.11 for the time discretization. This can be done, although some care must be exercised near boundaries, and the use of a multistage method again typically requires additional storage.

One simple way to achieve a two-level explicit method with higher accuracy is to use the idea of Taylor series methods, as described in Section 5.6. Applying this directly to the linear system of ODEs $U'(t) = AU(t)$ (and using $U'' = AU' = A^2U$) gives the second order method

$$U^{n+1} = U^n + kAU^n + \frac{1}{2}k^2A^2U^n.$$

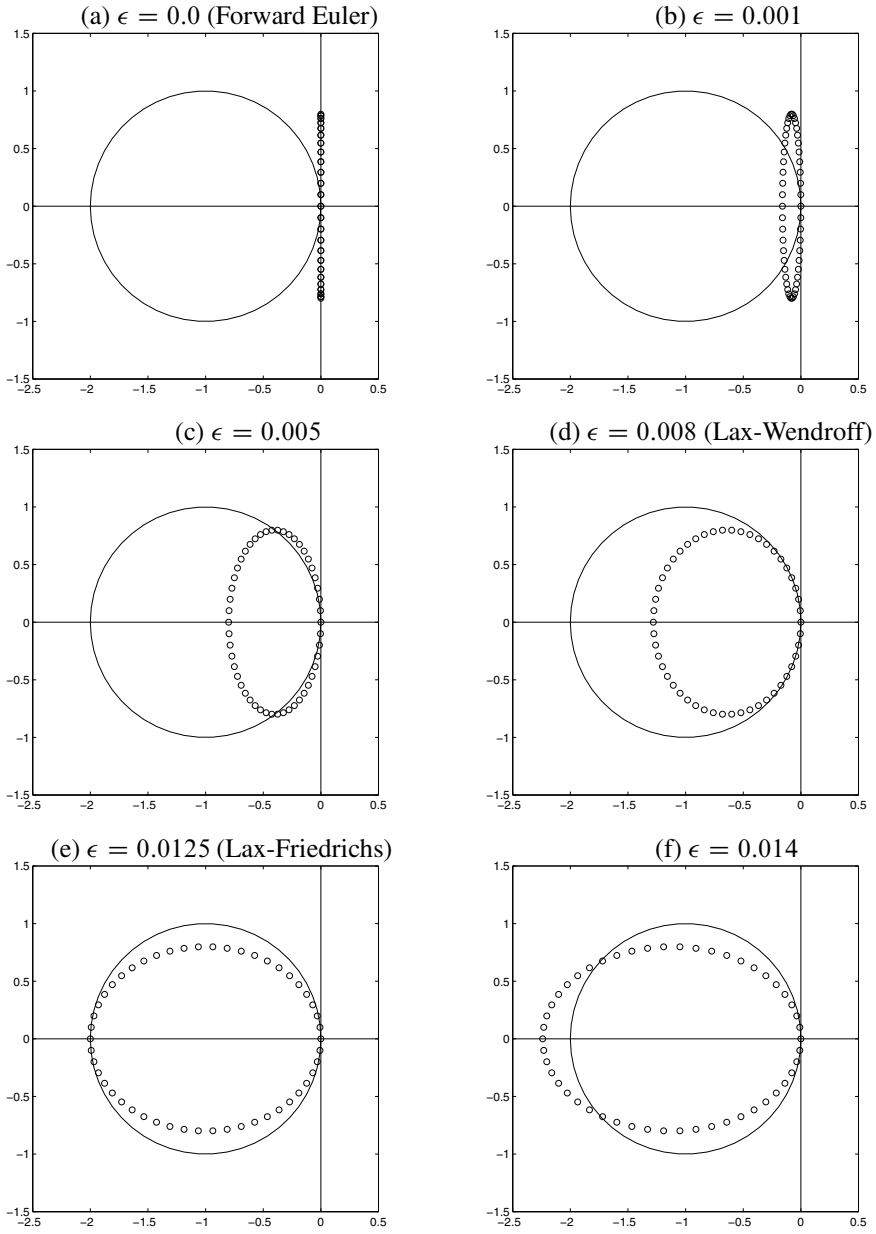

Figure 10.1. *Eigenvalues of the matrix A_ϵ in* (10.15)*, for various values of ϵ, in the case $h = 1/50$ and $k = 0.8h$, $a = 1$, so $ak/h = 0.8$.* (a) *shows the case $\epsilon = 0$ which corresponds to the forward Euler method* (10.5). (d) *shows the case $\epsilon = a^2k/2$, the Lax–Wendroff method* (10.18). (e) *shows the case $\epsilon = h^2/2k$, the Lax–Friedrichs method* (10.6). *The method is stable for ϵ between $a^2k/2$ and $h^2/2k$, as in* (d) *through* (e).

Here A is the matrix (10.10), and computing A^2 and writing the method at the typical grid point then gives

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{a^2k^2}{8h^2}(U_{j-2}^n - 2U_j^n + U_{j+2}^n). \tag{10.17}$$

This method is second order accurate and explicit but has a 5-point stencil involving the points U_{j-2}^n and U_{j+2}^n. With periodic boundary conditions this is not a problem, but with other boundary conditions this method needs more numerical boundary conditions than a 3-point method. This makes it less convenient to use and potentially more prone to numerical instability.

Note that the last term in (10.17) is an approximation to $\frac{1}{2}a^2k^2u_{xx}$ using a centered difference based on step size $2h$. A simple way to achieve a second order accurate 3-point method is to replace this term by the more standard 3-point formula. We then obtain the standard *Lax–Wendroff method*:

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{a^2k^2}{2h^2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n). \tag{10.18}$$

A cleaner way to derive this method is to use Taylor series expansions directly on the PDE $u_t + au_x = 0$, to obtain

$$u(x,t+k) = u(x,t) + ku_t(x,t) + \frac{1}{2}k^2u_{tt}(x,t) + \cdots.$$

Replacing u_t by $-au_x$ and u_{tt} by a^2u_{xx} gives

$$u(x,t+k) = u(x,t) - kau_x(x,t) + \frac{1}{2}k^2a^2u_{xx}(x,t) + \cdots.$$

If we now use the standard centered approximations to u_x and u_{xx} and drop the higher order terms, we obtain the Lax–Wendroff method (10.18). It is also clear how we could obtain higher order accurate explicit two-level methods by this same approach, by retaining more terms in the series and approximating the spatial derivatives (including the higher order spatial derivatives that will then arise) by suitably high order accurate finite difference approximations. The same approach can also be used with other PDEs. The key is to replace the time derivatives arising in the Taylor series expansion with spatial derivatives, using expressions obtained by differentiating the original PDE.

10.3.1 Stability analysis

We can analyze the stability of Lax–Wendroff following the same approach used for Lax–Friedrichs in Section 10.2. Note that with periodic boundary conditions, the Lax–Wendroff method (10.18) can be viewed as Euler's method applied to the linear system of ODEs $U'(t) = A_\epsilon U(t)$, where A_ϵ is given by (10.15) with $\epsilon = a^2k/2$ (instead of the value $\epsilon = h^2/2k$ used in Lax–Friedrichs). The eigenvalues of A_ϵ are given by (10.16) with the appropriate value of ϵ, and multiplying by the time step k gives

$$k\mu_p = -i\left(\frac{ak}{h}\right)\sin(p\pi h) + \left(\frac{ak}{h}\right)^2(\cos(p\pi h) - 1).$$

These values all lie on an ellipse centered at $-(ak/h)^2$ with semi-axes of length $(ak/h)^2$ and $|ak/h|$. If $|ak/h| \leq 1$, then all of these values lie inside the stability region of Euler's method. Figure 10.1(d) shows an example in the case $ak/h = 0.8$. The Lax–Wendroff method is stable with exactly the same time step restriction (10.7) as required for Lax–Friedrichs. In Section 10.7 we will see that this is a very natural stability condition to expect for the advection equation and is the best we could hope for when a 3-point method is used.

A close look at Figure 10.1 shows that the values $k\mu_p$ near the origin lie much closer to the boundary of the stability region for the Lax–Wendroff method (Figure 10.1(d)) than for the other methods illustrated in this figure. This is a reflection of the fact that Lax–Wendroff is second order accurate, while the others are only first order accurate. Note that a value $k\mu_p$ lying inside the stability region indicates that this eigenmode will be damped as the wave propagates, which is unphysical behavior since the true solution advects with no dissipation. For small values of μ_p (low wave numbers, smooth components) the Lax–Wendroff method has relatively little damping and the method is more accurate. Higher wave numbers are still damped with Lax–Wendroff (unless $|ak/h| = 1$, in which case all the $k\mu_p$ lie on the boundary of $\mathcal{S}$) and resolving the behavior of these modes properly would require a finer grid.

Comparing Figures 10.1(c), (d), and (e) shows that Lax–Wendroff has the minimal amount of numerical damping needed to bring the values $k\mu_p$ within the stability region. Any less damping, as in Figure 10.1(c) would lead to instability, while more damping as in Figure 10.1(e) gives excessive smearing of low wave numbers. Recall that the value of ϵ used in Lax–Wendroff was determined by doing a Taylor series expansion and requiring second order accuracy, so this makes sense.

10.4 Upwind methods

So far we have considered methods based on symmetric approximations to derivatives. Alternatively, one might use a nonsymmetric approximation to u_x in the advection equation, e.g.,

$$u_x(x_j, t) \approx \frac{1}{h}(U_j - U_{j-1}) \tag{10.19}$$

or

$$u_x(x_j, t) \approx \frac{1}{h}(U_{j+1} - U_j). \tag{10.20}$$

These are both *one-sided approximations*, since they use data only to one side or the other of the point x_j. Coupling one of these approximations with forward differencing in time gives the following methods for the advection equation:

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n) \tag{10.21}$$

or

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_{j+1}^n - U_j^n). \tag{10.22}$$

These methods are first order accurate in both space and time. One might wonder why we would want to use such approximations, since centered approximations are more accurate.

For the advection equation, however, there is an asymmetry in the equations because the equation models translation at speed a. If $a > 0$, then the solution moves to the right, while if $a < 0$ it moves to the left. There are situations where it is best to acknowledge this asymmetry and use one-sided differences in the appropriate direction.

The choice between the two methods (10.21) and (10.22) should be dictated by the sign of a. Note that the true solution over one time step can be written as

$$u(x_j, t + k) = u(x_j - ak, t)$$

so that the solution at the point x_j at the next time level is given by data to the *left* of x_j if $a > 0$, whereas it is determined by data to the *right* of x_j if $a < 0$. This suggests that (10.21) might be a better choice for $a > 0$ and (10.22) for $a < 0$.

In fact the stability analysis below shows that (10.21) is stable only if

$$0 \le \frac{ak}{h} \le 1. \tag{10.23}$$

Since k and h are positive, we see that this method can be used only if $a > 0$. This method is called the *upwind method* when used on the advection equation with $a > 0$. If we view the equation as modeling the concentration of some tracer in air blowing past us at speed a, then we are looking in the correct upwind direction to judge how the concentration will change with time. (This is also referred to as an *upstream differencing* method in some literature.)

Conversely, (10.22) is stable only if

$$-1 \le \frac{ak}{h} \le 0 \tag{10.24}$$

and can be used only if $a < 0$. In this case (10.22) is the proper upwind method to use.

10.4.1 Stability analysis

The method (10.21) can be written as

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{ak}{2h}(U_{j+1}^n - 2U_j^n + U_{j-1}^n), \tag{10.25}$$

which puts it in the form (10.15) with $\epsilon = ah/2$. We have seen previously that methods of this form are stable provided $|ak/h| \le 1$ and also $-2 < -2\epsilon k/h^2 < 0$. Since $k,\ h > 0$, this requires in particular that $\epsilon > 0$. For Lax–Friedrichs and Lax–Wendroff, this condition was always satisfied, but for upwind the value of ϵ depends on a and we see that $\epsilon > 0$ only if $a > 0$. If $a < 0$, then the eigenvalues of the MOL matrix lie on a circle that lies entirely in the right half-plane, and the method will certainly be unstable. If $a > 0$, then the above requirements lead to the stability restriction (10.23).

If we think of (10.25) as modeling an advection-diffusion equation, then we see that $a < 0$ corresponds to a negative diffusion coefficient. This leads to an ill-posed equation, as in the "backward heat equation" (see Section E.3.4).

The method (10.22) can also be written in a form similar to (10.25), but the last term will have a minus sign in front of it. In this case we need $a < 0$ for any hope of stability and then easily derive the stability restriction (10.24).

The three methods, Lax–Wendroff, upwind, and Lax–Friedrichs, can all be written in the same form (10.15) with different values of ϵ. If we call these values ϵ_{LW}, ϵ_{up}, and ϵ_{LF}, respectively, then we have

$$\epsilon_{LW} = \frac{a^2k}{2} = \frac{ah\nu}{2}, \qquad \epsilon_{up} = \frac{ah}{2}, \qquad \epsilon_{LF} = \frac{h^2}{2k} = \frac{ah}{2\nu},$$

where $\nu = ak/h$. Note that

$$\epsilon_{LW} = \nu\epsilon_{up} \qquad \text{and} \qquad \epsilon_{up} = \nu\epsilon_{LF}.$$

If $0 < \nu < 1$, then $\epsilon_{LW} < \epsilon_{up} < \epsilon_{LF}$ and the method is stable for any value of ϵ between ϵ_{LW} and ϵ_{LF}, as suggested by Figure 10.1.

10.4.2 The Beam–Warming method

The upwind method is only first order accurate. A second order accurate method with the same one-sided character can be derived by following the derivation of the Lax–Wendroff method, but using one-sided approximations to the spatial derivatives. This results in the *Beam–Warming* method, which for $a > 0$ takes the form

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(3U_j^n - 4U_{j-1}^n + U_{j-2}^n) + \frac{a^2k^2}{2h^2}(U_j^n - 2U_{j-1}^n + U_{j-2}^n). \quad (10.26)$$

For $a < 0$ the Beam–Warming method is one-sided in the other direction:

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(-3U_j^n + 4U_{j+1}^n - U_{j+2}^n) + \frac{a^2k^2}{2h^2}(U_j^n - 2U_{j+1}^n + U_{j+2}^n). \quad (10.27)$$

These methods are stable for $0 \le \nu \le 2$ and $-2 \le \nu \le 0$, respectively.

10.5 Von Neumann analysis

We have analyzed the stability of various algorithms for the advection equation by viewing them as ODE methods applied to the MOL system (10.9). The same stability criteria can be obtained by using von Neumann analysis as described in Section 9.6. Recall that this is done by replacing U_j^n by $g(\xi)^n e^{i\xi jh}$ (where $i = \sqrt{-1}$ in this section). Canceling out common factors results in an expression for the amplification factor $g(\xi)$, and requiring that this be bounded by 1 in magnitude gives the stability bounds for the method.

Also recall from Section 9.6 that this can be expected to give the same result as our MOL analysis because of the close relation between the $e^{i\xi jh}$ factor and the eigenvectors of the matrix A. In a sense von Neumann analysis simply combines the computation of the eigenvalues of A together with the absolute stability analysis of the time-stepping method being used. Nonetheless we will go through this analysis explicitly for several of the methods already considered to show how it works, since for other methods it may be more convenient to work with this approach than to interpret the method as an MOL method.

For the von Neumann analysis in this section we will simplify notation slightly by setting $\nu = ak/h$, the Courant number.

Example 10.1. Following the procedure of Example 9.6 for the upwind method (10.21) gives

$$g(\xi) = 1 - \nu\left(1 - e^{-i\xi h}\right) = (1-\nu) + \nu e^{-i\xi h}. \tag{10.28}$$

As the wave number ξ varies, $g(\xi)$ moves around a circle of radius ν centered at $1-\nu$. These values stay within the unit circle if and only if $0 \le \nu \le 1$, the stability limit that was also found in Section 10.4.1.

Example 10.2. Going through the same procedure for Lax–Friedrichs (10.6) gives

$$\begin{aligned} g(\xi) &= \frac{1}{2}\left(e^{-i\xi h} + e^{i\xi h}\right) - \nu\left(e^{i\xi h} - e^{-i\xi h}\right) \\ &= \cos(\xi h) - \nu i \sin(\xi h) \end{aligned} \tag{10.29}$$

and so

$$|g(\xi)|^2 = \cos^2(\xi h) + \nu^2 \sin^2(\xi h), \tag{10.30}$$

which is bounded by 1 for all ξ only if $|\nu| \le 1$.

Example 10.3. For the Lax–Wendroff method (10.18) we obtain

$$\begin{aligned} g(\xi) &= 1 - \frac{1}{2}\nu\left(e^{i\xi h} - e^{-i\xi h}\right) + \frac{1}{2}\nu^2\left(e^{i\xi h} - 2 + e^{-i\xi h}\right) \\ &= 1 - i\nu\sin(\xi h) + \nu^2(\cos(\xi h) - 1) \\ &= 1 - i\nu[2\sin(\xi h/2)\cos(\xi h/2)] + \nu^2[2\sin^2(\xi h/2)], \end{aligned} \tag{10.31}$$

where we have used two trigonometric identities to obtain the last line. This complex number has modulus

$$\begin{aligned} |g(\xi)|^2 &= [1 - 2\nu^2\sin^2(\xi h/2)]^2 + 4\sin^2(\xi h/2)\cos^2(\xi h/2) \\ &= 1 - 4\nu^2(1-\nu^2)\sin^4(\xi h/2). \end{aligned} \tag{10.32}$$

Since $0 \le \sin^4(\xi h/2) \le 1$ for all values of ξ, we see that $|g(\xi)|^2 \le 1$ for all ξ, and hence the method is stable provided that $|\nu| \le 1$, which again gives the expected stability bound (10.7).

Example 10.4. The leapfrog method (10.13) involves three time levels but can still be handled by the same basic approach. If we set $U_j^n = g(\xi)^n e^{i\xi jh}$ in the leapfrog method we obtain

$$g(\xi)^{n+1} e^{i\xi jh} = g(\xi)^{n-1} e^{i\xi jh} - \nu g(\xi)^n \left(e^{i\xi(j+1)h} - e^{-i\xi(j-1)h}\right). \tag{10.33}$$

If we now divide by $g(\xi)^{n-1} e^{i\xi jh}$ we obtain a quadratic equation for $g(\xi)$,

$$g(\xi)^2 = 1 - 2\nu i \sin(\xi h) g(\xi). \tag{10.34}$$

Examining this in the same manner as the analysis of the stability region for the midpoint method in Example 7.7 yields the stability limit $|\nu| < 1$.

It is important to note the severe limitations of the von Neumann approach just presented. It is strictly applicable only in the constant coefficient linear case (with periodic boundary conditions or on the Cauchy problem). Applying von Neumann analysis to the "frozen coefficient" problem locally often gives good guidance to the stability properties of a method more generally, but it cannot always be relied on. A great deal of work has been done on proving that methods stable for frozen coefficient problems remain stable for variable coefficient or nonlinear problems when everything is sufficiently smooth; see, for example, [40], [75]. In the nonlinear case, where the solution can contain shocks, a nonlinear stability theory is needed that employs techniques very different from von Neumann analysis; see, e.g., [66].

10.6 Characteristic tracing and interpolation

The solution to the advection equation is given by (10.2). The value of u is constant along each characteristic, which for this example is a straight line with constant slope. Over a single time step we have

$$u(x_j, t_{n+1}) = u(x_j - ak, t_n). \tag{10.35}$$

Tracing this characteristic back over time step k from the grid point x_j results in the picture shown in Figure 10.2(a). Note that if $0 < ak/h < 1$, then the point $x_j - ak$ lies between x_{j-1} and x_j. If we carefully choose k and h so that $ak/h = 1$ exactly, then $x_j - ak = x_{j-1}$ and we would find that $u(x_j, t_{n+1}) = u(x_{j-1}, t_n)$. The solution should just shift one grid cell to the right in each time step. We could compute the *exact* solution numerically with the method

$$U_j^{n+1} = U_{j-1}^n. \tag{10.36}$$

Actually, all the two-level methods that we have considered so far reduce to the formula (10.36) in this special case $ak = h$, and each of these methods happens to be exact in this case.

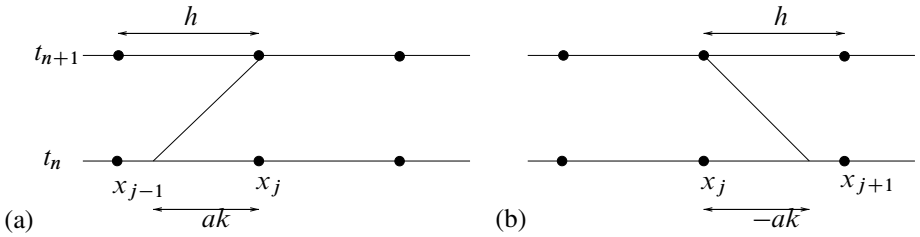

Figure 10.2. *Tracing the characteristic of the advection equation back in time from the point* (x_j, t_{n+1}) *to compute the solution according to* (10.35)*. Interpolating the value at this point from neighboring grid values gives the upwind method (for linear interpolation) or the Lax–Wendroff or Beam–Warming methods (quadratic interpolation).* (a) *shows the case* $a > 0$*,* (b) *shows the case* $a < 0$*.*

If $ak/h < 1$, then the point $x_j - ak$ is not exactly at a grid point, as illustrated in Figure 10.2. However, we might attempt to use the relation (10.35) as the basis for a numerical method by computing an approximation to $u(x_j - ak, t_n)$ based on interpolation from the grid values U_i^n at nearby grid points. For example, we might perform simple linear interpolation between U_{j-1}^n and U_j^n. Fitting a linear function to these points gives the function

$$p(x) = U_j^n + (x - x_j)\left(\frac{U_j^n - U_{j-1}^n}{h}\right). \tag{10.37}$$

Evaluating this at $x_j - ak$ and using this to define U_j^{n+1} gives

$$U_j^{n+1} = p(x_j - ak) = U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n).$$

This is precisely the first order upwind method (10.21). Note that this also can be interpreted as a linear combination of the two values U_{j-1}^n and U_j^n:

$$U_j^{n+1} = \left(1 - \frac{ak}{h}\right)U_j^n + \frac{ak}{h}U_{j-1}^n. \tag{10.38}$$

Moreover, this is a *convex combination* (i.e., the coefficients of U_j^n and U_{j-1}^n are both nonnegative and sum to 1) provided the stability condition (10.23) is satisfied, which is also the condition required to ensure that $x_j - ak$ lies between the two points x_{j-1} and x_j. In this case we are *interpolating* between these points with the function $p(x)$. If the stability condition is violated, then we would be using $p(x)$ to *extrapolate* outside of the interval where the data lies. It is easy to see that this sort of extrapolation can lead to instability—consider what happens if the data U^n is oscillatory with $U_j^n = (-1)^j$, for example.

To obtain better accuracy, we might try using a higher order interpolating polynomial based on more data points. If we define a quadratic polynomial $p(x)$ by interpolating the values U_{j-1}^n, U_j^n, and U_{j+1}^n, and then define U_j^{n+1} by evaluating $p(x_j - ak)$, we simply obtain the Lax–Wendroff method (10.18). Note that in this case we are properly interpolating provided that the stability restriction $|ak/h| \le 1$ is satisfied. If we instead base our quadratic interpolation on the three points U_{j-2}^n, U_{j-1}^n, and U_j^n, then we obtain the Beam–Warming method (10.26), and we are properly interpolating provided $0 \le ak/h \le 2$.

10.7 The Courant–Friedrichs–Lewy condition

The discussion of Section 10.6 suggests that for the advection equation, the point $x_j - ak$ must be bracketed by points used in the stencil of the finite difference method if the method is to be stable and convergent. This turns out to be a *necessary* condition in general for any method developed for the advection equation: if U_j^{n+1} is computed based on values U_{j+p}^n, U_{j+p+1}^n, $\ldots$, U_{j+q}^n with $p \le q$ (negative values are allowed for p and q), then we must have $x_{j+p} \le x_j - ak \le x_{j+q}$ or the method cannot be convergent. Since $x_i = ih$, this requires

$$-q \le \frac{ak}{h} \le -p.$$

This result for the advection equation is one special case of a much more general principle that is called the *CFL condition*. This condition is named after Courant, Friedrichs, and Lewy, who wrote a fundamental paper in 1928 that was the first paper on the stability and convergence of finite difference methods for PDEs. (The original paper [17] is in German but an English translation is available in [18].) The value $\nu = ak/h$ is often called the *Courant number*.

To understand this general condition, we must discuss the *domain of dependence* of a time-dependent PDE. (See, e.g., [55], [66] for more details.) For the advection equation, the solution $u(X, T)$ at some fixed point (X, T) depends on the initial data η at only a single point: $u(X, T) = u(X - aT)$. We say that the domain of dependence of the point (X, T) is the point $X - aT$:

$$\mathcal{D}(X, T) = \{X - aT\}.$$

If we modify the data η at this point, then the solution $u(X, T)$ will change, while modifying the data at any other point will have no effect on the solution at this point.

This is a rather unusual situation for a PDE. More generally we might expect the solution at (X, T) to depend on the data at several points or over a whole interval. In Section 10.10 we consider hyperbolic systems of equations of the form $u_t + Au_x = 0$, where $u \in \mathbb{R}^s$ and $A \in \mathbb{R}^{s\times s}$ is a matrix with real eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_s$. If these values are distinct then we will see that the solution $u(X, T)$ depends on the data at the s distinct points $X - \lambda_1 T, \ \ldots, X - \lambda_s T$, and hence

$$\mathcal{D}(X, T) = \{X - \lambda_p T \text{ for } p = 1, 2, \ \ldots, \ s\}. \tag{10.39}$$

The heat equation $u_t = u_{xx}$ has a much larger domain of dependence. For this equation the solution at any point (X, T) depends on the data *everywhere* and the domain of dependence is the whole real line,

$$\mathcal{D}(X, T) = (-\infty, \infty).$$

This equation is said to have infinite propagation speed, since data at any point affects the solution everywhere at any small time in the future (although its effect of course decays exponentially away from this point, as seen from the Green's function (E.37)).

A finite difference method also has a domain of dependence. On a particular fixed grid we define the domain of dependence of a grid point (x_j, t_n) to be the set of grid points x_i at the initial time $t = 0$ with the property that the data U_i^0 at x_i has an effect on the solution U_j^n. For example, with the Lax–Wendroff method (10.18) or any other 3-point method, the value U_j^n depends on U_{j-1}^{n-1}, U_j^{n-1}, and U_{j+1}^{n-1}. These values depend in turn on U_{j-2}^{n-2} through U_{j+2}^{n-2}. Tracing back to the initial time we obtain a triangular array of grid points as seen in Figure 10.3(a), and we see that U_j^n depends on the initial data at the points $x_{j-n}, \ \ldots, x_{j+n}$.

Now consider what happens if we refine the grid, keeping k/h fixed. Figure 10.3(b) shows the situation when k and h are reduced by a factor of 2, focusing on the same value of (X, T) which now corresponds to U_{2j}^{2n} on the finer grid. This value depends on twice as many values of the initial data, but these values all lie within the same interval and are merely twice as dense.

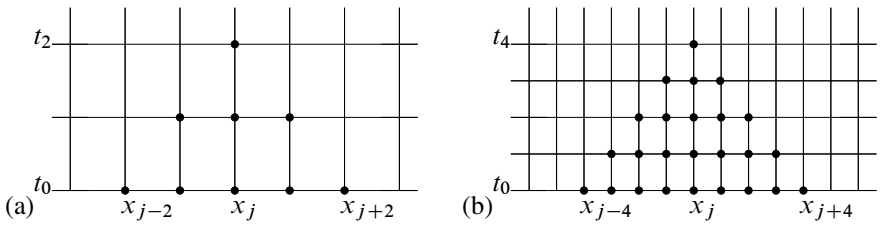

Figure 10.3. (a) *Numerical domain of dependence of a grid point when using a 3-point explicit method.* (b) *On a finer grid.*

If the grid is refined further with $k/h \equiv r$ fixed, then clearly the numerical domain of dependence of the point (X, T) will fill in the interval $[X - T/r,\ X + T/r]$. As we refine the grid, we hope that our computed solution at (X, T) will converge to the true solution $u(X, T) = \eta(X - aT)$. Clearly this can be possible only if

$$X - T/r \le X - aT \le X + T/r. \tag{10.40}$$

Otherwise, the true solution will depend only on a value $\eta(X - aT)$ that is never seen by the numerical method, no matter how fine a grid we take. We could change the data at this point and hence change the true solution without having any effect on the numerical solution, so the method cannot be convergent for general initial data.

Note that the condition (10.40) translates into $|a| \le 1/r$ and hence $|ak/h| \le 1$. This can also be written as $|ak| \le h$, which just says that over a single time step the characteristic we trace back must lie within one grid point of x_j. (Recall the discussion of interpolation versus extrapolation in Section 10.6.)

The CFL condition generalizes this idea:

The CFL condition: A numerical method can be convergent only if its numerical domain of dependence contains the true domain of dependence of the PDE, at least in the limit as k and h go to zero.

For the Lax–Friedrichs, leapfrog, and Lax–Wendroff methods the condition on k and h required by the CFL condition is exactly the stability restriction we derived earlier in this chapter. But it is important to note that in general the CFL condition is only a *necessary* condition. If it is violated, then the method cannot be convergent. If it is satisfied, then the method *might* be convergent, but a proper stability analysis is required to prove this or to determine the proper stability restriction on k and h. (And of course consistency is also required for convergence—stability alone is not enough.)

Example 10.5. The 3-point method (10.5) has the same stencil and numerical domain of dependence as Lax–Wendroff but is unstable for any fixed value of k/h even though the CFL condition is satisfied for $|ak/h| \le 1$.

Example 10.6. The upwind methods (10.21) and (10.22) each have a 2-point stencil and the stability restrictions of these methods, (10.23) and (10.24), respectively, agree precisely with what the CFL condition requires.

Example 10.7. The Beam–Warming method (10.26) has a 3-point one-sided stencil. The CFL condition is satisfied if $0 \le ak/h \le 2$. When $a < 0$ the method (10.27) is used and the CFL condition requires $-2 \le ak/h \le 0$. These are also the stability regions for the methods, which must be verified by appropriate stability analysis.

Example 10.8. For the heat equation the true domain of dependence is the whole real line. It appears that any 3-point explicit method violates the CFL condition, and indeed it does if we fix k/h as the grid is refined. However, recall from Section 10.2.1 that the 3-point explicit method (9.5) is convergent as we refine the grid, provided we have $k/h^2 \le 1/2$. In this case when we make the grid finer by a factor of 2 in space it will become finer by a factor of 4 in time, and hence the numerical domain of dependence will cover a wider interval at time $t = 0$. As $k \to 0$ the numerical domain of dependence will spread to cover the entire real line, and hence the CFL condition is satisfied in this case.

An implicit method such as the Crank–Nicolson method (9.7) satisfies the CFL condition for any time step k. In this case the numerical domain of dependence is the entire real line because the tridiagonal linear system couples together all points in such a manner that the solution at each point depends on the data at all points (i.e., the inverse of a tridiagonal matrix is dense).

10.8 Some numerical results

Figure 10.4 shows typical numerical results obtained with three of the methods discussed in the previous sections. The initial data at time $t = 0$, shown in Figure 10.4(a), are smooth and consist of two Gaussian peaks, one sharper than the other:

$$u(x,0) = \eta(x) = \exp(-20(x-2)^2) + \exp(-(x-5)^2). \tag{10.41}$$

The remaining frames in this figure show the results obtained when solving the advection equation $u_t + u_x = 0$ up to time $t = 17$, so the exact solution is simply the initial data shifted by 17 units. Note that only part of the computational domain is shown; the computation was done on the interval $0 \le x \le 25$. The grid spacing $h = 0.05$ was used, with time step $k = 0.8h$ so the Courant number is $ak/h = 0.8$. On this grid one peak is fairly well resolved and the other is poorly resolved.

Figure 10.4(b) shows the result obtained with the upwind method (10.21) and illustrates the extreme numerical dissipation of this method. Figure 10.4(c) shows the result obtained with Lax–Wendroff. The broader peak remains well resolved, while the dispersive nature of Lax–Wendroff is apparent near the sharper peak. Dispersion is even more apparent when the leapfrog method is used, as seen in Figure 10.4(d).

The "modified equation" analysis of the next section sheds more light on these results.

10.9 Modified equations

Our standard tool for estimating the accuracy of a finite difference method has been the "local truncation error." Seeing how well the true solution of the PDE satisfies the difference equation gives an indication of the accuracy of the difference equation. Now we will study a slightly different approach that can be very illuminating since it reveals much more about the structure and behavior of the numerical solution.

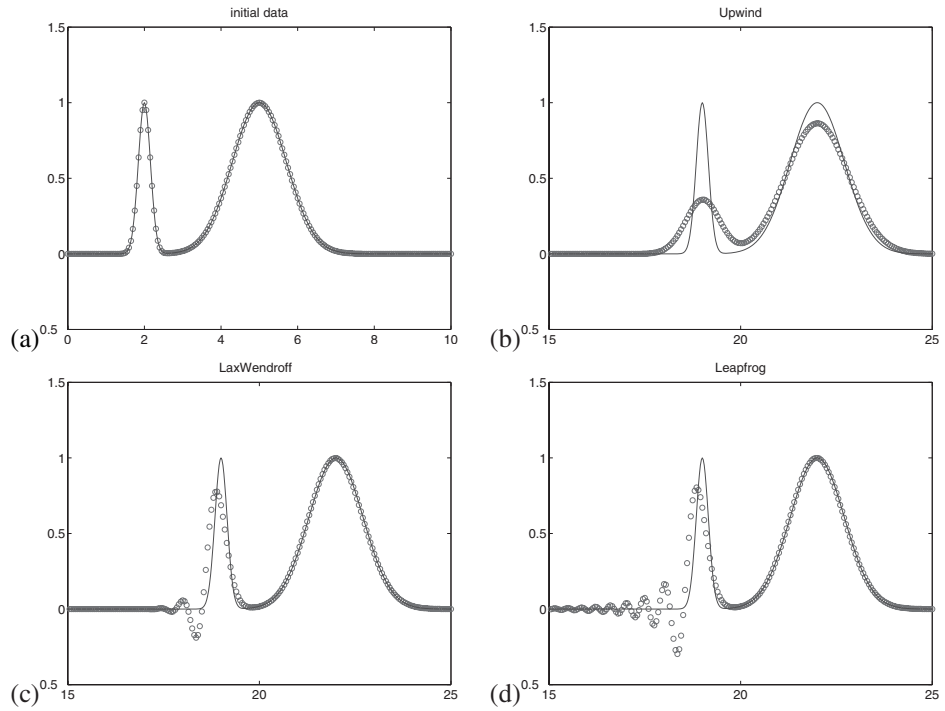

Figure 10.4. *The numerical experiments on the advection equation described in Section* 10.8.

The idea is to ask the following question: is there a PDE $v_t = \cdots$ such that our numerical approximation U_j^n is actually the *exact* solution to this PDE, $U_j^n = v(x_j, t_n)$? Or, less ambitiously, can we at least find a PDE that is better satisfied by U_j^n than the original PDE we were attempting to model? If so, then studying the behavior of solutions to this PDE should tell us much about how the numerical approximation is behaving. This can be advantageous because it is often easier to study the behavior of PDEs than of finite difference formulas.

In fact it is possible to find a PDE that is exactly satisfied by the U_j^n by doing Taylor series expansions as we do to compute the local truncation error. However, this PDE will have an infinite number of terms involving higher and higher powers of k and h. By truncating this series at some point we will obtain a PDE that is simple enough to study and yet gives a good indication of the behavior of the U_j^n.

The procedure of determining a modified equation is best illustrated with an example. See [100] for a more detailed discussion of the derivation of modified equations.

Example 10.9. Consider the upwind method (10.21) for the advection equation $u_t + au_x = 0$ in the case $a > 0$,

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n). \tag{10.42}$$

The process of deriving the modified equation is very similar to computing the local truncation error, only now we insert the formula $v(x,t)$ into the difference equation. This is supposed to be a function that agrees exactly with U_j^n at the grid points and so, unlike $u(x,t)$, the function $v(x,t)$ satisfies (10.42) exactly:

$$v(x,t+k) = v(x,t) - \frac{ak}{h}(v(x,t) - v(x-h,t)).$$

Expanding these terms in Taylor series about (x,t) and simplifying gives

$$\left(v_t + \frac{1}{2}kv_{tt} + \frac{1}{6}k^2 v_{ttt} + \cdots\right) + a\left(v_x - \frac{1}{2}hv_{xx} + \frac{1}{6}h^2 v_{xxx} + \cdots\right) = 0.$$

We can rewrite this as

$$v_t + av_x = \frac{1}{2}(ahv_{xx} - kv_{tt}) + \frac{1}{6}(ah^2 v_{xxx} - k^2 v_{ttt}) + \cdots.$$

This is the PDE that v satisfies. If we take k/h fixed, then the terms on the right-hand side are $O(k)$, $O(k^2)$, etc., so that for small k we can truncate this series to get a PDE that is quite well satisfied by the U_j^n.

If we drop all the terms on the right-hand side, we just recover the original advection equation. Since we have then dropped terms of $O(k)$, we expect that U_j^n satisfies this equation to $O(k)$, as we know to be true since this upwind method is first order accurate.

If we keep the $O(k)$ terms, then we get something more interesting:

$$v_t + av_x = \frac{1}{2}(ahv_{xx} - kv_{tt}). \tag{10.43}$$

This involves second derivatives in both x and t, but we can derive a slightly different modified equation with the same accuracy by differentiating (10.43) with respect to t to obtain

$$v_{tt} = -av_{xt} + \frac{1}{2}(ahv_{xxt} - kv_{ttt})$$

and with respect to x to obtain

$$v_{tx} = -av_{xx} + \frac{1}{2}(ahv_{xxx} - kv_{ttx}).$$

Combining these gives

$$v_{tt} = a^2 v_{xx} + O(k).$$

Inserting this in (10.43) gives

$$v_t + av_x = \frac{1}{2}(ahv_{xx} - a^2 kv_{xx}) + O(k^2).$$

Since we have already decided to drop terms of $O(k^2)$, we can drop these terms here also to obtain

$$v_t + av_x = \frac{1}{2}ah\left(1 - \frac{ak}{h}\right)v_{xx}. \tag{10.44}$$

This is now a familiar advection-diffusion equation. The grid values U_j^n can be viewed as giving a *second order accurate* approximation to the true solution of this equation (whereas they give only first order accurate approximations to the true solution of the advection equation).

The fact that the modified equation is an advection-diffusion equation tells us a great deal about how the numerical solution behaves. Solutions to the advection-diffusion equation translate at the proper speed a but also diffuse and are smeared out. This is clearly visible in Figure 10.4(b).

Note that the diffusion coefficient in (10.44) is $\frac{1}{2}(ah - a^2k)$, which vanishes in the special case $ak = h$. In this case we already know that the exact solution to the advection equation is recovered by the upwind method.

Also note that the diffusion coefficient is positive only if $0 < ak/h < 1$. This is precisely the stability limit of upwind. If this is violated, then the diffusion coefficient in the modified equation is negative, giving an ill-posed problem with exponentially growing solutions. Hence we see that even some information about stability can be extracted from the modified equation.

Example 10.10. If the same procedure is followed for the Lax–Wendroff method, we find that all $O(k)$ terms drop out of the modified equation, as is expected since this method is second order accurate on the advection equation. The modified equation obtained by retaining the $O(k^2)$ term and then replacing time derivatives by spatial derivatives is

$$v_t + av_x + \frac{1}{6}ah^2\left(1 - \left(\frac{ak}{h}\right)^2\right)v_{xxx} = 0. \tag{10.45}$$

The Lax–Wendroff method produces a *third order* accurate solution to this equation. This equation has a very different character from (10.43). The v_{xxx} term leads to *dispersive* behavior rather than diffusion. This is clearly seen in Figure 10.4(c), where the U_j^n computed with Lax–Wendroff are compared to the true solution of the advection equation. The magnitude of the error is smaller than with the upwind method for a given set of k and h, since it is a higher order method, but the dispersive term leads to an oscillating solution and also a shift in the location of the main peak, a *phase error*. This is similar to the dispersive behavior seen in Figure E.1 for an equation very similar to (10.45).

In Section E.3.6 the propagation properties of dispersive waves is analyzed in terms of the dispersion relation of the PDE and the phase and group velocities of different wave numbers. Following the discussion there, we find that for the modified equation (10.45), the group velocity for wave number ξ is

$$c_g = a - \frac{1}{2}ah^2\left(1 - \left(\frac{ak}{h}\right)^2\right)\xi^2,$$

which is less than a for all wave numbers. As a result the numerical result can be expected to develop a train of oscillations behind the peak, with the high wave numbers lagging farthest behind the correct location.

Some care must be used here, however, when looking at highly oscillatory waves (relative to the grid, i.e., waves for which ξh is far from 0). For ξh sufficiently small the modified equation (10.45) is a reasonable model, but for larger ξh the terms we have

neglected in this modified equation may play an equally important role. Rather than determining the dispersion relation for a method from its modified equation, it is more reliable to determine it directly from the numerical method, which is essentially what we have done in von Neumann stability analysis. This is pursued further in Example 10.13 below.

If we retain one more term in the modified equation for Lax–Wendroff, we would find that the U_j^n are fourth order accurate solutions to an equation of the form

$$v_t + av_x + \frac{1}{6}ah^2\left(1 - \left(\frac{ak}{h}\right)^2\right)v_{xxx} = -\epsilon v_{xxxx}, \tag{10.46}$$

where the ϵ in the fourth order dissipative term is $O(k^3+h^3)$ and positive when the stability bound holds. This higher order dissipation causes the highest wave numbers to be damped (see Section E.3.7), so that there is a limit to the oscillations seen in practice.

The fact that this method can produce oscillatory approximations is one of the reasons that the first order upwind method is sometimes preferable in practice. In some situations nonphysical oscillations may be disastrous, for example, if the value of u represents a concentration that cannot become negative or exceed some limit without difficulties arising elsewhere in the modeling process.

Example 10.11. The Beam–Warming method (10.26) has a similar modified equation,

$$v_t + av_x = \frac{1}{6}ah^2\left(2 - \frac{3ak}{h} + \left(\frac{ak}{h}\right)^2\right)v_{xxx}. \tag{10.47}$$

In this case the group velocity is greater than a for all wave numbers in the case $0 < ak/h < 1$, so that the oscillations move ahead of the main hump. If $1 < ak/h < 2$, then the group velocity is less than a and the oscillations fall behind. (Again the dispersion relation for (10.47) gives an accurate idea of the dispersive properties of the numerical method only for ξh sufficiently small.)

Example 10.12. The modified equation for the leapfrog method (10.13) can be derived by writing

$$\left(\frac{v(x,t+k) - v(x,t-k)}{2k}\right) = a\left(\frac{v(x+h,t) - v(x-h,t)}{2h}\right) = 0 \tag{10.48}$$

and expanding in Taylor series. As in Example 10.9 we then further differentiate the resulting equation (which has an infinite number of terms) to express higher spatial derivatives of v in terms of temporal derivatives. The dominant terms look just like Lax–Wendroff, and (10.45) is again obtained.

However, from the symmetric form of (10.48) in both x and t we see that all even-order derivatives drop out. If we derive the next term in the modified equation we will find an equation of the form

$$v_t + av_x + \frac{1}{6}ah^2\left(1 - \left(\frac{ak}{h}\right)^2\right)v_{xxx} = \epsilon v_{xxxxx} + \cdots \tag{10.49}$$

for some $\epsilon = O(h^4 + k^4)$, and higher order modified equations will also involve only odd-order derivatives and even powers of h and k. Hence the numerical solution produced with the leapfrog method is a fourth order accurate solution to the modified equation (10.45).

Moreover, recall from Section E.3.6 that all higher order odd-order derivatives give dispersive terms. We conclude that the leapfrog method is *nondissipative* at all orders. This conclusion is consistent with the observation in Section 10.2.2 that $k\lambda_p$ is on the boundary of the stability region for all eigenvalues of A from (10.10), and so we see neither growth nor decay of any mode. However, we also see from the form of (10.49) that high wave number modes will not propagate with the correct velocity. This was also true of Lax–Wendroff, but there the fourth order dissipation damps out the worst offenders, whereas with leapfrog this dispersion of often much more apparent in computational results, as observed in Figure 10.4(d).

Example 10.13. Since leapfrog is nondissipative it serves as a nice example for calculating the true dispersion relation of the numerical method. The approach is very similar to the von Neumann stability analysis of Section 10.5, only now we use $e^{i(\xi x_j - \omega t_n)}$ as our Ansatz (so that the g from von Neumann analysis is replaced by $e^{-i\omega k}$). Following the same procedure as in Example 10.4, we find that

$$e^{-i\omega k} = e^{i\omega k} - \frac{ak}{h}\left(e^{i\xi h} - e^{-i\xi h}\right), \tag{10.50}$$

which can be simplified to yield

$$\sin(\omega k) = \frac{ak}{h}\sin(\xi h). \tag{10.51}$$

This is the dispersion relation relating ω to ξ. Note that $|\xi h| \le \pi$ for waves that can be resolved on our grid and that for each such ξh there are two corresponding values of ωk. The dispersion relation is multivalued because leapfrog is a three-level method, and different temporal behavior of the same spatial wave can be seen, depending on the relation between the initial data chosen on the two initial levels. For well-resolved waves ($|\xi h|$ small) and reasonable initial data we expect ωk also near zero (not near $\pm\pi$, where the other solution is in this case).

Solving for ω as a function of ξ and expanding in Taylor series for small ξh would show this agrees with the dispersion relation of the infinite modified equation (10.49). We do not need to do this, however, if our goal is to compute the group velocity for the leapfrog method. We can differentiate (10.51) with respect to ξ and solve for

$$\frac{d\omega}{d\xi} = \frac{a\cos(\xi h)}{\cos(\omega k)} = \pm\frac{a\cos(\xi h)}{\sqrt{1 - \nu\sin^2(\xi h)}}, \tag{10.52}$$

where $\nu = ak/h$ is the Courant number and again the $\pm$ arises from the multivalued dispersion relation. The velocity observed would depend on how the initial two levels are set.

Note that the group velocity can be negative and near $-a$ for $|\xi h| \approx \pi$. This is not surprising since the leapfrog method has a 3-point centered stencil, and it is possible for numerical waves to travel from right to left although physically there is advection only to the right. This can be observed in some computations, for example, in Figure 10.5 as discussed in Example 10.14.

10.10 Hyperbolic systems

The advection equation $u_t + au_x = 0$ can be generalized to a first order linear system of equations of the form

$$\begin{aligned} u_t + Au_x &= 0, \\ u(x,0) &= \eta(x), \end{aligned} \tag{10.53}$$

where $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^s$ and $A \in \mathbb{R}^{s \times s}$ is a constant matrix. (Note that this is not the matrix A from earlier in this chapter, e.g., (10.10).)

This is a system of conservation laws (see Section E.2) with the flux function $f(u) = Au$. This system is called *hyperbolic* if A is diagonalizable with real eigenvalues, so that we can decompose

$$A = R\Lambda R^{-1}, \tag{10.54}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_s)$ is a diagonal matrix of eigenvalues and $R = [r_1|r_2|\cdots|r_s]$ is the matrix of right eigenvectors. Note that $AR = R\Lambda$, i.e.,

$$Ar_p = \lambda_p r_p \quad \text{for } p = 1,\ 2,\ \ldots,\ s. \tag{10.55}$$

The system is called *strictly hyperbolic* if the eigenvalues are distinct.

10.10.1 Characteristic variables

We can solve (10.53) by changing to the "characteristic variables"

$$w = R^{-1}u \tag{10.56}$$

in much the same way we solved linear systems of ODEs in Section 7.4.2. Multiplying (10.53) by R^{-1} and using (10.54) gives

$$R^{-1}u_t + \Lambda R^{-1}u_x = 0 \tag{10.57}$$

or, since R^{-1} is constant,

$$w_t + \Lambda w_x = 0. \tag{10.58}$$

Since Λ is diagonal, this decouples into s independent scalar equations

$$(w_p)_t + \lambda_p (w_p)_x = 0, \qquad p = 1,\ 2,\ \ldots,\ s. \tag{10.59}$$

Each of these is a constant coefficient linear advection equation with solution

$$w_p(x,t) = w_p(x - \lambda_p t, 0). \tag{10.60}$$

Since $w = R^{-1}u$, the initial data for w_p is simply the pth component of the vector

$$w(x,0) = R^{-1}\eta(x). \tag{10.61}$$

The solution to the original system is finally recovered via (10.56):

$$u(x,t) = Rw(x,t). \tag{10.62}$$

Note that the value $w_p(x,t)$ is the coefficient of r_p in an eigenvector expansion of the vector $u(x,t)$, i.e., (10.62) can be written out as

$$u(x,t) = \sum_{p=1}^{s} w_p(x,t) r_p. \tag{10.63}$$

Combining this with the solutions (10.60) of the decoupled scalar equations gives

$$u(x,t) = \sum_{p=1}^{s} w_p(x - \lambda_p t, 0) r_p. \tag{10.64}$$

Note that $u(x,t)$ depends only on the initial data at the s points $x - \lambda_p t$. This set of points is the domain of dependent $\mathcal{D}(x,t)$ of (10.39).

The curves $x = x_0 + \lambda_p t$ satisfying $x'(t) = \lambda_p$ are the "characteristics of the pth family," or simply "p-characteristics." These are straight lines in the case of a constant coefficient system. Note that for a strictly hyperbolic system, s distinct characteristic curves pass through each point in the x-t plane. The coefficient $w_p(x,t)$ of the eigenvector r_p in the eigenvector expansion (10.63) of $u(x,t)$ is constant along any p-characteristic.

10.11 Numerical methods for hyperbolic systems

Most of the methods discussed earlier for the advection equation can be extended directly to a general hyperbolic system by replacing a with A in the formulas. For example, the Lax–Wendroff method becomes

$$U_j^{n+1} = U_j^n - \frac{k}{2h} A(U_{j+1}^n - U_{j-1}^n) + \frac{k^2}{2h^2} A^2 (U_{j-1}^n - 2U_j^n + U_{j+1}^n). \tag{10.65}$$

This is second order accurate and is stable provided the Courant number is no larger than 1, where the *Courant number* is defined to be

$$\nu = \max_{1 \le p \le s} |\lambda_p k / h|. \tag{10.66}$$

For the scalar advection equation, there is only one eigenvalue equal to a, and the Courant number is simply $|ak/h|$. The Lax–Friedrichs and leapfrog methods can be generalized in the same way to systems of equations and remain stable for $\nu \le 1$.

The upwind method for the scalar advection equation is based on a one-sided approximation to u_x, using data in the upwind direction. The one-sided formulas (10.21) and (10.22) generalize naturally to

$$U_j^{n+1} = U_j^n - \frac{k}{h} A(U_j^n - U_{j-1}^n) \tag{10.67}$$

and

$$U_j^{n+1} = U_j^n - \frac{k}{h} A(U_{j+1}^n - U_j^n). \tag{10.68}$$

For a system of equations, however, neither of these is useful unless all the eigenvalues of A have the same sign, so that the upwind direction is the same for all characteristic variables. The method (10.67) is stable only if

$$0 \le \frac{k\lambda_p}{h} \le 1 \qquad \text{for all } p = 1,\ 2,\ \ldots,\ s, \tag{10.69}$$

while (10.68) is stable only if

$$-1 \le \frac{k\lambda_p}{h} \le 0 \qquad \text{for all } p = 1,\ 2,\ \ldots,\ s. \tag{10.70}$$

It is possible to generalize the upwind method to more general systems with eigenvalues of both signs, but to do so requires decomposing the system into the characteristic variables and upwinding each of these in the appropriate direction. The resulting method can also be generalized to nonlinear hyperbolic systems and generally goes by the name of *Godunov's method*. These methods are described in much more detail in [66].

10.12 Initial boundary value problems

So far we have studied only numerical methods for hyperbolic problems on a domain with periodic boundary conditions (or the Cauchy problem, if we could use a grid with an infinite number of grid points).

Most practical problems are posed on a bounded domain with nonperiodic boundary conditions, which must be specified in addition to the initial conditions to march forward in time. These problems are called *initial boundary value problems* (IBVPs).

Consider the advection equation $u_t + au_x = 0$ with $a > 0$, corresponding to flow to the right, on the domain $0 \le x \le 1$ where some initial conditions $u(x, 0) = \eta(x)$ are given. This data completely determine the solution via (10.2) in the triangular region $0 \le x - at \le 1$ of the x-t plane. Outside this region, however, the solution is determined only if we also impose boundary conditions at $x = 0$, say, (10.8). Then the solution is

$$u(x,t) = \begin{cases} \eta(x-at) & \text{if } 0 \le x - at \le 1, \\ g_0(t - x/a) & \text{otherwise.} \end{cases} \tag{10.71}$$

Note that boundary data are required only at the *inflow boundary* $x = 0$, not at the *outflow boundary* $x = 1$, where the solution is determined via (10.71). Trying to impose a different value on $u(1, t)$ would lead to a problem with no solution.

If $a < 0$ in the advection equation, then $x = 1$ is the inflow boundary, the solution is transported to the left, and $x = 0$ is the outflow boundary.

10.12.1 Analysis of upwind on the initial boundary value problem

Now suppose we apply the upwind method (10.21) to this IBVP with $a > 0$ on a grid with $h = 1/(m + 1)$ and $x_i = ih$ for $i = 0,\ 1,\ \ldots,\ m + 1$. The formula (10.21) can be applied for $i = 1,\ \ldots,\ m + 1$ in each time step, while $U_0^n = g(t_n)$ is set by the boundary condition. Hence the method is completely specified.

When is this method stable? Intuitively we expect it to be stable if $0 \le ak/h \le 1$. This is the stability condition for the problem with periodic boundary conditions, and here we are using the same method at every point except $i = 0$, where the exact solution is being set in each time step. Our intuition is correct in this case and the method is stable if $0 \le ak/h \le 1$.

Notice, however, that von Neumann analysis cannot be used in this case, as discussed already in Section 10.5: the Fourier modes $e^{i\xi h}$ are no longer eigengridfunctions. But von Neumann analysis is still useful because it generally gives a necessary condition for stability. In most cases a method that is unstable on the periodic domain or Cauchy problem will not be useful on a bounded domain either, since locally on a fine grid, away from the boundaries, any instability indicated by von Neumann analysis is bound to show up.

Instead we can use MOL stability analysis, although it is sometimes subtle to do so correctly. We have a system of ODEs similar to (10.9) for the vector $U(t)$, which again has $m + 1$ components corresponding to $u(x_i, t)$. But now we must incorporate the boundary conditions, and so we have a system of the form

$$U'(t) = AU(t) + g(t), \tag{10.72}$$

where

$$A = -\frac{a}{h}\begin{bmatrix} 1 & & & & \\ -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & & \\ & & & -1 & 1 \end{bmatrix}, \qquad g(t) = \begin{bmatrix} g_0(t)a/h \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{10.73}$$

The upwind method corresponds to using Euler's method on the ODE (10.72).

The change from the matrix of (10.10) to (10.73) may seem trivial but it completely changes the character of the matrix. The matrix (10.10) is circulant and normal and has eigenvalues uniformly distributed about the circle of radius a/h in the complex plane centered at $z = -a/h$. The matrix (10.73) is a defective Jordan block with all its eigenvalues at the point $-a/h$. The eigenvalues have moved distance $a/h \to \infty$ as $h \to 0$.

Suppose we attempt to apply the usual stability analysis of Chapter 7 to this system and require that $k\lambda_p \in \mathcal{S}$ for all eigenvalues of A, where S is the stability region for Euler's method. Since $\mathcal{S}$ contains the interval $[-2, 0]$ on the real axis, this would suggest the stability restriction

$$0 \le ak/h \le 2 \tag{10.74}$$

for the upwind method on the IBVP. This is wrong by a factor of 2. It is a necessary condition but not sufficient.

The problem is that A in (10.73) is highly nonnormal. It is essentially a Jordan block of the sort discussed in Section D.5.1, and on a fine grid its ϵ-pseudospectra roughly fill up the circle of radius a/h about $-a/h$, even for very small ϵ. This is a case where we need to apply a more stringent requirement than simply requiring that $k\lambda$ be inside the stability region for all eigenvalues; we also need to require that

$$\text{dist}(k\lambda_\epsilon, \mathcal{S}) \le C\epsilon \tag{10.75}$$

holds for the ϵ-pseudoeigenvalues (see Section D.5), where C is a modest constant, as suggested in [47], [92]. Requiring (10.75) shows that the expected requirement $0 \le ak/h \le 1$ is needed rather than (10.74).

10.12.2 Outflow boundary conditions

When the upwind method is used for the advection equation, as in the previous section, the outflow boundary poses no problem. The finite difference formula is one sided and the value U^n_{m+1} at the rightmost grid point is computed by the same formula that is used in the interior.

However, if we use a finite difference method whose stencil extends to the right as well as the left, we will need to use a different formula at the rightmost point in the domain. This is called a *numerical boundary condition* or *artificial boundary condition* since it is required by the method, not for the PDE. Numerical boundary conditions may also be required at boundaries where a physical boundary condition is imposed, if the numerical method requires more conditions than the equation. For example, if we use the Beam–Warming method for advection, which has a stencil that extends two grid points in the upwind direction, then we can use this only for updating U^{n+1}_2, U^{n+1}_3, ... The value U^{n+1}_0 will be set by the physical boundary condition but U^{n+1}_1 will have to be set by some other method, which can be viewed as a numerical boundary condition for Beam–Warming.

Naturally some care must be used in choosing numerical boundary conditions, both in terms of the accuracy and stability of the resulting method. This is a difficult topic, particularly the stability analysis of numerical methods for IBVPs, and we will not pursue it here. See, for example, [40], [84], [89].

Example 10.14. We will look at one example simply to give a flavor of the potential difficulties. Suppose we use the leapfrog method to solve the IBVP for the advection equation $u_t + au_x = 0$. At the left (inflow) boundary we can use the given boundary condition, but at the right we will need a numerical boundary condition. Suppose we use the first order upwind method at this point,

$$U^{n+1}_{m+1} = U^n_{m+1} - \frac{ak}{h}(U^n_{m+1} - U^n_m), \tag{10.76}$$

which is also consistent with the advection equation. Figure 10.5 shows four snapshots of the solution when the initial data is $u(x,0) = \eta(x) = \exp(-5(x-2)^2)$ and the first two time levels for leapfrog are initialized based on the exact solution $u(x,t) = \eta(x-at)$. We see that as the wave passes out the right boundary, a reflection is generated that moves to the left, back into the domain. The dispersion relation for the leapfrog method found in Example 10.13 shows that waves with $\xi h \approx \pi$ can move to the left with group velocity approximately equal to $-a$, and this wave number corresponds exactly to the sawtooth wave seen in the figure.

For an interesting discussion of the relation of numerical dispersion relations and group velocity to the stability of numerical boundary conditions, see [89].

Outflow boundaries are often particularly troublesome. Even if a method is formally stable (as the leapfrog method in the previous example is with the upwind boundary condition), it is often hard to avoid spurious reflections. We have seen this even for the advection equation, where in principle flow is entirely to the right, and it can be even more difficult

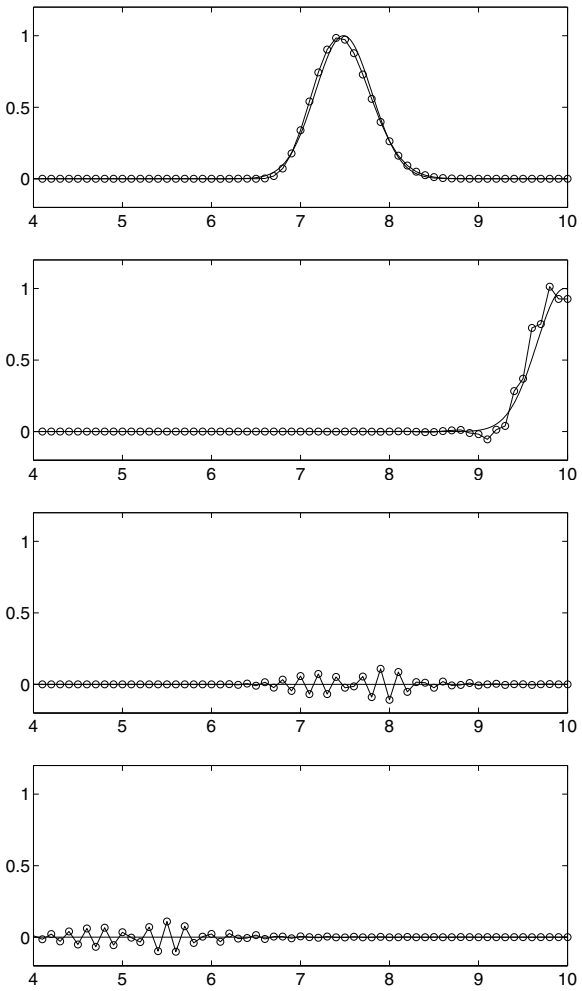

Figure 10.5. *Numerical solution of the advection equation using the leapfrog method in the interior and the upwind method at the right boundary. The solution is shown at four equally spaced times, illustrating the generation and leftward propagation of a sawtooth mode.*

to develop appropriate boundary conditions for wave propagation or fluid dynamics equations that admit wave motion in all directions. Yet in practice we always have to compute over a finite domain and this often requires setting artificial boundaries around the region of interest. The assumption is that the phenomena of interest happen within this region, and the hope is that any waves hitting the artificial boundary will leave the domain with no reflection. Numerical boundary conditions that attempt to achieve this are often called *nonreflecting* or *absorbing* boundary conditions.

10.13 Other discretizations

As in the previous chapter on parabolic equations, we have concentrated on a few basic methods in order to explore some fundamental ideas. We have also considered only the simplest case of constant coefficient linear hyperbolic equations, whereas in practice most hyperbolic problems of interest have variable coefficients (e.g., linear wave propagation in heterogeneous media) or are nonlinear. These problems give rise to a host of new difficulties, not least of which is the fact that the solutions of interest are often discontinuous since nonlinearity can lead to shock formation. There is a well-developed theory of numerical methods for such problems that we will not delve into here; see, for example, [66].

Even in the case of constant coefficient linear problems, there are many other discretizations possible beyond the ones presented here. We end with a brief overview of just a few of these:

- Higher order discretizations of u_x can be used in place of the discretizations considered so far. If we write the MOL system for the advection equation as

$$U_j'(t) = -aW_j(t), \tag{10.77}$$

  where $W_j(t)$ is some approximation to $u_x(x_j, t)$, then there are many ways to approximate $W_j(t)$ from the U_j values beyond the centered approximation (10.3). One-sided approximations are one possibility, as in the upwind method. For sufficiently smooth solutions we might instead use higher order accurate centered approximations, e.g.,

$$W_j = \frac{4}{3}\left(\frac{U_{j+1} - U_{j-1}}{2h}\right) - \frac{1}{3}\left(\frac{U_{j+2} - U_{j-2}}{4h}\right). \tag{10.78}$$

  This and other approximations can be determined using the `fdcoeffV.m` routine discussed in Section 1.5; e.g., `fdcoeffV(1,0,-2:2)` produces (10.78). Centered approximations such as (10.78) generally lead to skew-symmetric matrices with pure imaginary eigenvalues, at least when applied to the problem with periodic boundary conditions. In practice most problems are on a finite domain with nonperiodic boundary conditions, and other issues arise as already seen in Section 10.12. Note that the discretization (10.78) requires more numerical boundary conditions than the upwind or second order centered operator.

- An interesting approach to obtaining better accuracy in W_j is to use a so-called compact method, in which the W_j are determined by solving a linear system rather than explicitly. A simple example is

$$\frac{1}{4}W_{j-1} + W_j + \frac{1}{4}W_{j+1} = \frac{3}{2}\left(\frac{U_{j+1} - U_{j-1}}{2h}\right). \tag{10.79}$$

  This gives a tridiagonal system of equations to solve for the W_j values, and it can be shown that the resulting values will be $O(h^4)$ approximations to $u_x(x_j, t)$. Higher order methods of this form also exist; see Lele [63] for an in-depth discussion. In addition to giving higher order of accuracy with a compact stencil, these approximations also typically have much better dispersion properties than standard finite difference approximations of the same order.

- Spectral approximations to the first derivative can be used, based on the same ideas as in Section 2.21. In this case $W = DU$ is used to obtain the approximations to the first derivative, where D is the dense spectral differentiation matrix. These methods can also be generalized to variable coefficient and even nonlinear problems and often work very well for problems with smooth solutions.

  Stability analysis of these methods can be tricky. To obtain reasonable results a nonuniform distribution of grid points must be used, such as the Chebyshev extreme points as discussed in Section 2.21. In this case the eigenvalues of the matrix D turn out to be $O(1/h^2)$, rather than $O(1/h)$ as is expected for a fixed-stencil discretization of the first derivative. If instead we use the roots of the Legendre polynomial (see Section B.3.1), another popular choice of grid points for spectral methods, it can be shown that the eigenvalues are $O(1/h)$, which appears to be better. In both cases, however, the matrix D is highly nonnormal and the eigenvalues are misleading, and in fact a time step $k = O(h^2)$ is generally required if an explicit method is used for either choice of grid points; see, e.g., [92].

- Other time discretizations can be used in place of the ones discussed in this chapter. In particular, for spectral methods the MOL system is stiff and it may be beneficial to use an implicit method as discussed in Chapters 8 and 9. Another possibility is to use an exponential time differencing method, as discussed in Section 11.6.

- For conservation laws (see Section E.2) numerical methods are often more naturally derived using the integral form (E.9) than by using finite difference approximations to derivatives. Such methods are particularly important for nonlinear hyperbolic problems, where shock waves (discontinuous solutions) can develop spontaneously even from smooth initial data. In this case the discrete value U_i^n is viewed as an approximation to the cell average of $u(x, t_n)$ over the grid cell $[x_{i-1/2},\ x_{i+1/2}]$ of length h centered about x_i,

$$U_i^n \approx \frac{1}{h} \int_{x_{i-1/2}}^{x_{i+1/2}} u(x, t_n)\, dx. \tag{10.80}$$

  According to (E.9) this cell average evolves at a rate given by the difference of fluxes at the cell edges, and a particular numerical method is obtained by approximating these fluxes based on the current cell averages. Methods of this form are often called *finite volume methods* since the spatial domain is partitioned into volumes of finite size. Simple finite volume methods often look identical to finite difference methods, but the change in viewpoint allows the development of more sophisticated methods that are better suited to solving nonlinear conservation laws. These methods also have some advantages for linear problems, particularly if they have variable coefficients with jump discontinuities, as often arises in solving wave propagation problems in heterogeneous media. See [66] for a detailed description of such methods.

Chapter 11
Mixed Equations

We have now studied the solution of various types of time-dependent equations: ordinary differential equations (ODEs), parabolic partial differential equations (PDEs) such as the heat equation, and hyperbolic PDEs such as the advection equation. In practice several processes may be happening simultaneously, and the PDE model will not be a pure equation of any of the types already discussed but rather will be a mixture. In this chapter we discuss several approaches to handling more complicated equations. We restrict our attention to time-dependent PDEs of the form

$$u_t = \mathcal{A}_1(u) + \mathcal{A}_2(u) + \cdots + \mathcal{A}_N(u), \tag{11.1}$$

where each of the $\mathcal{A}_j(u)$ are (possibly nonlinear) functions or differential operators involving only spatial derivatives of u. For simplicity, most of our discussion will be further restricted to only two terms, which we will write as

$$u_t = \mathcal{A}(u) + \mathcal{B}(u), \tag{11.2}$$

but more terms often can be handled by extension or combination of the methods described here.

11.1 Some examples

We begin with some examples of PDEs involving more than one term. See Appendix E for more discussion of some of these equations.

- *Multidimensional problems*, such as the diffusion equation in two dimensions,

$$u_t = \kappa(u_{xx} + u_{yy}), \tag{11.3}$$

  or the three-dimensional version. This problem has already been discussed in Section 9.7, where we saw that efficient methods can be developed by splitting (11.3) into two one-dimensional problems. Hyperbolic equations also arise in multidimensional domains, such as the two-dimensional hyperbolic system

$$u_t + Au_x + Bu_y = 0 \tag{11.4}$$

or nonlinear hyperbolic conservation laws

$$u_t + f(u)_x + g(u)_y = 0, \tag{11.5}$$

where $f(u)$ and $g(u)$ are the flux functions in the two directions.

All the problems discussed below also have multidimensional variants, where even more terms arise. For simplicity we display only the one-dimensional case.

- *Reaction-diffusion equations* of the form

$$u_t = \kappa u_{xx} + R(u), \tag{11.6}$$

where κ is a diffusion coefficient (or diagonal matrix of diffusion coefficients if different components in the system diffuse at different rates) and $R(u)$ represents chemical reactions, and is typically nonlinear. The reaction terms might or might not be stiff. If not, then we typically want to handle these terms explicitly (to avoid solving nonlinear systems of equations in each time step), while the diffusion term is stiff and requires appropriate methods.

Even if the reaction terms are stiff, they apply locally at a point in space, unlike the diffusion term that couples different grid points together. Recognizing this fact can lead to more efficient solution techniques, as discussed further below.

- *Advection-diffusion equations* of the form

$$u_t + au_x = \kappa u_{xx}. \tag{11.7}$$

The diffusion term is stiff and requires an appropriate solver, while the advection term can be handled explicitly.

- *Nonlinear hyperbolic equations* with viscous terms,

$$u_t + f(u)_x = \kappa u_{xx}. \tag{11.8}$$

The advection-diffusion equation (11.7) is one example of this form, but more generally the flux function $f(u)$ can be nonlinear, modeling fluid dynamics, for example, in which case the right-hand side represents viscous terms and perhaps heat conduction. The *Navier–Stokes equations* for compressible gas dynamics have this general form, for example. A simpler example is the *viscous Burgers equation*,

$$u_t + uu_x = \epsilon u_{xx}, \tag{11.9}$$

where the flux function is $f(u) = \frac{1}{2}u^2$. This is a simple scalar model for some of the effects seen in compressible flow, and it has been widely studied.

- *Advection-diffusion-reaction equations* or reacting flow problems,

$$u_t + f(u)_x = \kappa u_{xx} + R(u). \tag{11.10}$$

If chemical reactions are occurring in a fluid flow, then equations of this general form are obtained. Combustion problems are particularly challenging problems of

this type, where exothermic chemical reactions directly influence the fluid dynamics. Chemotaxis problems are also of this type, which arise in biology when substances move in response to concentration gradients of other substances, and often give rise to interesting pattern formation [73].

- The *Korteweg–de Vries (KdV)* equation,

$$u_t + uu_x = \nu u_{xxx}. \tag{11.11}$$

This is similar to the viscous Burgers equation (11.9), but the term on the right-hand side is dispersive rather than dissipative. This leads to very different behavior and is the simplest example of an equation having soliton solutions. It arises as a simple model of certain kinds of wave phenomena in fluid dynamics and elsewhere. The third derivative term is stiffer than a u_{xx} term and would typically require $k = O(h^3)$ for an explicit method. However, similar to the advective terms considered in Chapter 10, the eigenvalues of a discretization of u_{xxx} will typically lie on or near the imaginary axis over an interval stretching distance $O(1/h^3)$ from the origin, rather than along the negative real axis, influencing the type of time discretization appropriate for these equations.

Many other equations that couple nonlinearity with dispersion are of importance in applications, for example, the *nonlinear Schrödinger equation*

$$i\psi_t(x,t) = -\psi_{xx}(x,t) + V(\psi) \tag{11.12}$$

with a nonlinear potential $V(\psi)$. (With $V = 0$ the equation is linear and dispersive, as shown in Section E.3.8.)

- The *Kuramoto–Sivashinsky* equation,

$$u_t + \frac{1}{2}(u_x)^2 = -u_{xx} - u_{xxxx}. \tag{11.13}$$

The right-hand side gives exponential growth of some low wave numbers, as shown in Section E.3.7. The nonlinear term transfers energy from low wave numbers to higher wave numbers, which are damped by the fourth order diffusion. The result is bounded solutions, but ones that can behave quite chaotically. The fourth order diffusion term is even more stiff than second order diffusion. Eigenvalues of a discretization of this term typically lie on the negative real axis over an interval of length $O(1/h^4)$.

Many approaches can be used for problems that involve two or more different terms, and a huge number of specialized methods have been developed for particular equations. The remainder of this chapter contains a brief overview of a few popular approaches, but it is by no means exhaustive.

11.2 Fully coupled method of lines

One simple approach is to discretize the full right-hand side of (11.1) in space using appropriate spatial discretizations of each term to obtain a semidiscrete method of lines (MOL)

system of the form $U'(t) = F(U(t))$, where F represents the full spatial discretization. This system of ODEs can now be solved using an ODE method in MATLAB or with other ODE software. This may work well for equations where the terms all have similar character. The problem in general, however, is that the same ODE method is being applied to all aspects of the spatial discretization, which can be very wasteful for many of the problems listed above.

Consider a reaction-diffusion equation of the form (11.6), for example. If this represents a system of s equations and we discretize in x on a grid with m points, then we obtain a coupled system of ms ODEs. Typically the reaction terms $R(u)$ are nonlinear and so this will be a nonlinear system. Generally an implicit method is used since the diffusion terms are stiff, so in every time step a nonlinear system of dimension ms must be solved. However, if the reaction terms are not stiff, then there is no need to make these terms implicit and it should be possible to solve only linear systems for the diffusion terms, and s decoupled linear systems of size m each (with tridiagonal matrices) rather than a fully coupled nonlinear system of size ms. Even if the reaction terms are stiff, the reaction terms $u_t = R(u)$ are local at a point and by splitting the reaction from the diffusion (using one of the other approaches discussed below), it is possible to solve decoupled nonlinear equations of dimension s at each grid point to advance the reaction terms in a stable manner.

11.3 Fully coupled Taylor series methods

A first order accurate explicit method for the equation (11.2) can be obtained by using the first order term in the Taylor series,

$$u(x, t_n + k) \approx u(x, t_n) + k(\mathcal{A}(u(x, t_n)) + \mathcal{B}(u(x, t_n))), \tag{11.14}$$

and then replacing the spatial operators $\mathcal{A}$ and $\mathcal{B}$ with discretizations. A second order accurate method can sometimes be obtained by adding the next term in the Taylor series, but this requires determining u_{tt} in terms of spatial derivatives of u. We did this for the advection equation $u_t + au_x = 0$ in Section 10.3 to derive the Lax–Wendroff method, in which case $u_{tt} = a^2 u_{xx}$. Whether we can do this in general for (11.2) depends on how complicated the right-hand side is, but in some cases it can be done. For example, for the two-dimensional hyperbolic equation (11.4) we can compute

$$\begin{aligned} u_{tt} &= -Au_{tx} - Bu_{ty} \\ &= A(Au_x + Bu_y)_x + B(Au_x + Bu_y)_y \\ &= A^2 u_{xx} + (AB + BA)u_{xy} + B^2 u_{yy}. \end{aligned} \tag{11.15}$$

A second order accurate Lax–Wendroff method can then be derived from

$$u(x, y, t_n + k) \approx u - k(Au_x + Bu_y) + \frac{1}{2}k^2(A^2 u_{xx} + (AB + BA)u_{xy} + B^2 u_{yy}) \tag{11.16}$$

(where the terms on the right-hand side are all evaluated at (x, y, t_n)) by discretizing in space using second order accurate centered approximations. This gives the two-dimensional Lax–Wendroff method.

For some other problems a similar approach works, e.g., for advection-reaction terms with nonstiff reactions, but this is generally useful only if all terms are nonstiff and can be

advanced with explicit methods. Moreover, it is generally difficult to achieve higher than second order accuracy with this approach.

11.4 Fractional step methods

The idea of a fractional step method (also called a time-split or split-step method, among other things) is to split up the equation into its constituent pieces and alternate between advancing simpler equations in time. The simplest splitting for an equation with two terms of the form (11.2) would be

$$\begin{aligned} U^* &= \mathcal{N}_A(U^n, k), \\ U^{n+1} &= \mathcal{N}_B(U^*, k). \end{aligned} \tag{11.17}$$

Here $\mathcal{N}_A(U^n, k)$ represents some one-step numerical method that solves $u_t = \mathcal{A}(u)$ over a time step of length k starting with data U^n. Similarly, $\mathcal{N}_B(U^*, k)$ solves $u_t = \mathcal{B}(u)$ over a time step of length k starting with the data U^*.

Below we will see that this splitting of the equation does often work—the numerical solution obtained will usually converge to solutions of the original problem as $k \to 0$ provided the numerical methods used in each step are consistent and stable approximations to the separate problems they are designed to solve. The approximation obtained often will be only first order accurate, however, no matter how good each of the constituent numerical methods is. We will see why below and consider some improvements.

First we note that this fractional step approach has several advantages. It allows us to use very different methods for each piece $u_t = \mathcal{A}(u)$ and $u_t = \mathcal{B}(u)$. One can be implicit and the other explicit, for example. For the reaction-diffusion problem (11.6), if $\mathcal{A}(u)$ represents the diffusion terms, then these can be solved with an implicit method, solving tridiagonal linear systems for each component. The reaction terms can be solved with an explicit or implicit method, depending on whether they are stiff. If an implicit method is used, then a nonlinear system is obtained at each grid point, but each is decoupled from the nonlinear system at other grid points, typically leading to a much more efficient solution of these systems.

Another situation in which this type of splitting is often used is in reducing a multidimensional problem to a sequence of one-dimensional problems. In this context the fractional step approach is often called *dimensional splitting*. We saw an example of this in Section 9.8, where the locally one-dimensional (LOD) method for the heat equation was discussed. By decoupling the space dimensions, one obtains a sequence of tridiagonal systems to solve instead of a large sparse matrix with more complicated structure.

Another advantage of the fractional step approach is that existing methods for the simpler subproblems are easily patched together, e.g., ODE methods for the reaction terms can be applied without worrying about the spatial coupling, or a one-dimensional method for a PDE can easily be extended to two or three dimensions by repeatedly applying it on one-dimensional slices.

To see that the fractional step method (11.17) may be only first order accurate, consider a simple linear system of ODEs where the coefficient matrix is split into two matrices as $A + B$, so the system is

$$u_t = Au + Bu. \tag{11.18}$$

Suppose we use the fractional step method (11.17) with the exact solution operator for each step, so

$$\mathcal{N}_A(U,k) = e^{Ak}U, \qquad \mathcal{N}_B(U,k) = e^{Bk}U. \tag{11.19}$$

Then (11.17) gives the numerical method

$$U^{n+1} = e^{Bk}U^* = e^{Bk}e^{Ak}U^n, \tag{11.20}$$

whereas the exact solution satisfies

$$u(t_{n+1}) = e^{(A+B)k}u(t_n). \tag{11.21}$$

By Taylor series expansion of the matrix exponentials (see (D.31)), we find that

$$e^{(A+B)k} = I + k(A+B) + \frac{1}{2}k^2(A+B)^2 + \cdots, \tag{11.22}$$

whereas

$$\begin{aligned} e^{Bk}e^{Ak} &= \left(I + kA + \frac{1}{2}k^2A^2 + \cdots\right)\left(I + kB + \frac{1}{2}k^2B^2 + \cdots\right) \\ &= I + k(A+B) + \frac{1}{2}k^2(A^2 + 2AB + B^2) + \cdots. \end{aligned} \tag{11.23}$$

Note that the quadratic term in (11.22) is

$$(A+B)^2 = A^2 + AB + BA + B^2,$$

which is not the same as the quadratic term in (11.23) if the matrices A and B do not commute.

If they do commute, e.g., in the scalar case, then the splitting is exact and all terms in the Taylor series agree. But in general a one-step error of magnitude $O(k^2)$ is introduced, and so the method is only first order accurate even when the exact solution is used for each piece.

A second order accurate splitting was introduced by Strang [83] in the context of methods for multidimensional hyperbolic equations and is often called the *Strang splitting*:

$$\begin{aligned} U^* &= \mathcal{N}_A(U^n, k/2), \\ U^{**} &= \mathcal{N}_B(U^*, k), \\ U^{n+1} &= \mathcal{N}_A(U^{**}, k/2). \end{aligned} \tag{11.24}$$

Working out the product of the Taylor series expansions in this case for the ODE system (11.18) gives agreement to the quadratic term in (11.22), although there is an error in the $O(k^3)$ term unless A and B commute. A similar result can be shown for general PDEs with smooth solutions split in the form (11.2).

An alternative procedure, which also gives second order accuracy, is to use the splitting (11.17) in every other time step, and in the alternate time steps use a similar splitting but with the order of $\mathcal{N}_A$ and $\mathcal{N}_B$ reversed. Over two time steps this has roughly the same form as the Strang splitting over a time step of length $2k$, although with two applications of $\mathcal{N}_B$ with time step k rather than one application with time step $2k$.

Example 11.1. The LOD method for the heat equation discussed in Section 9.8 uses a splitting of the form (11.17) but is able to achieve second order accuracy because there is no splitting error in this case (except near the boundaries, where appropriate treatment is required). In this case the two-dimensional heat equation is split with $\mathcal{A}(u) = u_{xx}$ and $\mathcal{B}(u) = u_{yy}$, and the operators ∂_x^2 and ∂_y^2 commute. For a more general variable coefficient heat equation with $\mathcal{A}(u) = (\kappa(x, y)u_x)_x$ and $\mathcal{B}(u) = (\kappa(x, y)u_y)_y$, the two operators no longer commute and the LOD method would be only first order accurate.

Another possible way to improve the accuracy of fractional step methods is to combine them with the spectral deferred correction method of [28]. This method improves the accuracy of a time-stepping procedure by a deferred correction process. Application to advection-diffusion-reaction equations in the context of fractional step methods was investigated in [9].

One difficulty with fractional step methods is that boundary conditions may be hard to apply properly when initial boundary value problems are solved. Each application of $\mathcal{N}_A$ or $\mathcal{N}_B$ typically requires boundary conditions, either physical or artificial, and it is not always clear how to properly specify the "intermediate boundary conditions" needed in each stage of the splitting. This has been discussed in relation to the LOD method for the heat equation in Section 9.8. See [65] for a discussion of intermediate boundary conditions for hyperbolic equations.

Another potential difficulty is stability. Even if the methods $\mathcal{N}_A$ and $\mathcal{N}_B$ are each stable methods for the problems they are designed to solve, it is not always clear that alternating between these methods in every time step will lead to a stable procedure. Example D.3 shows the problem that can arise. Suppose $\mathcal{N}_A(U^n, k) = A_0U^n$ and $\mathcal{N}_B(U^*, k) = A_1u^*$, where A_0 and A_1 are given by (D.84). Then each method is stable by itself but the fractional step procedure (11.17) generates exponentially growing solutions. Often stability of fractional step methods can be easily shown, for example, if $\|\mathcal{N}_A(U, k)\| \le \|U\|$ and $\|\mathcal{N}_B(U, k)\| \le \|U\|$ both hold in the same norm, but caution is required.

11.5 Implicit-explicit methods

Suppose we have an equation split as in (11.2), where $\mathcal{A}(u)$ represents stiff terms that we wish to integrate using an implicit method, whereas $\mathcal{B}(u)$ corresponds to nonstiff terms that can be handled explicitly with a reasonable time step. We have seen various examples of this form, such as reaction-diffusion equations with nonstiff reactions, where it may be much more efficient to avoid an implicit solve for the nonlinear reaction terms.

Implicit-explicit (IMEX) methods are fully coupled methods that are designed to handle some terms implicitly and others explicitly. A simple example is obtained by combining forward Euler with backward Euler:

$$U^{n+1} = U^n + k(\mathcal{A}(U^{n+1}) + \mathcal{B}(U^n)). \tag{11.25}$$

Another example is a two-step combination of the second order Adams–Bashforth method for the explicit term with the trapezoidal method for the implicit term:

$$U^{n+1} = U^n + \frac{k}{2}\left(\mathcal{A}(U^n) + \mathcal{A}(U^{n+1}) + 3\mathcal{B}(U^n) - \mathcal{B}(U^{n-1})\right). \tag{11.26}$$

Higher order methods of this type have been derived and widely used. See, for example, [7] for a number of other multistep methods and [6] for some Runge–Kutta methods of this type.

11.6 Exponential time differencing methods

Consider a nonlinear ODE $u' = f(u)$ (possibly an MOL discretization of a PDE) and suppose that over the time interval $[t_n, t_{n+1}]$ we write this as

$$u'(t) = A_n u(t) + \mathcal{B}_n(u(t)), \tag{11.27}$$

where we have split the function $f(u)$ into a linear part and a nonlinear part. The idea of *exponential time differencing* (ETD) methods is to use a form of Duhamel's principle (5.8) to handle the linear part exactly using the matrix exponential and combine this with an appropriate numerical method of the desired order for the $\mathcal{B}_n(u)$ term, typically an explicit method if we assume that the linear term captures the stiff part of the problem.

Two common forms of this type of splitting are as follows:

1. For a general nonlinear function $f(u)$, let $A_n = f'(U^n)$, the Jacobian matrix evaluated at U^n (or perhaps some approximate Jacobian), and then

$$\mathcal{B}_n(u) = f(u) - A_n u. \tag{11.28}$$

2. For problems such as MOL discretizations of reaction-diffusion equations we may take A_n to be the matrix representing the diffusion operator for all n and let $\mathcal{B}_n(u)$ be the reaction terms. In this case A_n is not the full Jacobian of the nonlinear problem, but if the reaction terms are not stiff they might be easily approximated with explicit methods, and there are advantages to having A unchanged from one step to the next—the ETD methods require working with the matrix exponential e^{kA_n}, and if A is constant we may be able to compute this once before beginning the time stepping.

For the system (11.27), Duhamel's principle (5.8) can be generalized to

$$u(t_{n+1}) = e^{A_n k} u(t_n) + \int_{t_n}^{t_{n+1}} e^{A_n(t_{n+1}-\tau)} \mathcal{B}_n(u(\tau))\, d\tau. \tag{11.29}$$

This expression is exact, but the integral must be approximated since we don't know $\mathcal{B}_n(u(\tau))$. Methods of various order can be obtained by different discretizations of this integral. The simplest approximation is obtained by replacing $\mathcal{B}_n(u(\tau))$ with $\mathcal{B}_n(U^n)$. We can then pull this out of the integral and can compute the exact integral of the remaining integrand by integrating the Taylor series for the matrix exponential (D.31) term by term, resulting in (5.12),

$$\begin{aligned} \int_{t_n}^{t_{n+1}} e^{A_n(t_{n+1}-\tau)}\, d\tau &= k + \frac{1}{2}k^2 A_n + \frac{1}{6}k^3 A_n^2 + \cdots \\ &= A_n^{-1}\left(e^{A_n k} - I\right) \quad \text{(if } A_n \text{ is nonsingular).} \end{aligned} \tag{11.30}$$

Using this, we obtain from (11.29) the numerical method

$$U^{n+1} = e^{A_n k} U^n + A_n^{-1} \left(e^{A_n k} - I \right) \mathcal{B}_n(U^n). \tag{11.31}$$

Since $\mathcal{B}_n(U^n) = f(U^n) - A_n U^n$, we can rewrite this as

$$U^{n+1} = U^n + A_n^{-1} \left(e^{A_n k} - I \right) f(U^n). \tag{11.32}$$

Note that if $A_n = 0$, then using the first line of (11.30) we see that (11.32) reduces to Euler's method for $u' = f(u)$ and is only first order accurate. However, we normally assume that A_n is nonsingular and an approximation to the Jacobian matrix. In general we can compute the local truncation error to be

$$\begin{aligned}
\tau^n &= \left(\frac{u(t_{n+1}) - u(t_n)}{k} \right) - k^{-1} A_n^{-1} \left(e^{A_n k} - I \right) u'(t_n) \\
&= \left[u'(t_n) + \frac{1}{2} k u''(t_n) + \frac{1}{6} k^2 u'''(t_n) + \cdots \right] \\
&\quad - \left[I + \frac{1}{2} A_n k + \frac{1}{6} A_n^2 k^2 + \cdots \right] u'(t_n) \\
&= \frac{1}{2} k \left(u''(t_n) - A_n u'(t_n) \right) + \frac{1}{6} k^2 \left(u'''(t_n) - A_n u'(t_n) \right) + \cdots \\
&= \frac{1}{2} k \left(f'(u(t_n)) - A_n \right) u'(t_n) + O(k^2).
\end{aligned} \tag{11.33}$$

We see that the method is second order accurate if $A_n = f'(U^n)$.

Higher order methods can be derived by using better approximations of the integral in (11.29). This can be done either as a multistep method, approximating $\mathcal{B}_n(u)$ by an interpolating polynomial through previous values U^{n-j} as in the derivation of the Adams–Bashforth methods, or as multistage generalizations of the Runge–Kutta methods. See, for example, [8], [19], [48], [53] for more discussion of these methods.

Note that the ETD method is exact on the test problem $u' = \lambda u$ if we take $A_n = \lambda$. So the region of absolute stability for this method is exactly the left half-plane. The method is exact more generally on a linear system of equations, provided of course that we can compute the matrix exponential accurately, as discussed in the next section.

Many mixed equations involve higher order derivative terms that are linear (and often constant coefficient) and ETD methods may be particularly suitable for handling the stiffness of spatial discretizations. Note in particular that for dispersive terms, such as the u_{xxx} term in the KdV equation (11.11), an ETD method that handles this term exactly may be advantageous over an implicit method. This dispersion is nondissipative (eigenvalues are on the imaginary axis), but many implicit methods designed for stiff problems have the imaginary axis in the interior of the stability region, leading to nonphysical dissipation.

11.6.1 Implementing exponential time differencing methods

Computing the matrix exponential is nontrivial—the classic paper [70] presented "19 dubious ways" to do this, and its recent update [71] discusses a 20th way in the appendix,

a more recently developed approach based on Krylov space methods. The latter approach has made the exponential time differencing approach viable for MOL discretizations of parabolic equations and other linear systems of ODEs involving large but sparse coefficient matrices and is discussed further below.

One situation in which ETD methods are relatively easy to implement is when the matrix A_n is diagonal, for then $e^{A_n k}$ is just a diagonal matrix of scalar exponential functions. This arises naturally in some applications, for example, if a problem such as a reaction-diffusion equation is solved with periodic boundary conditions. By Fourier transforming the problem, the diffusion operator is reduced to a diagonal matrix. For this reason ETD methods are often particularly attractive in connection with Fourier spectral methods.

Even in the scalar case, however, the evaluation of the exponential factor

$$\phi(z) = (e^z - 1)/z \tag{11.34}$$

that appears in (11.32) can be susceptible to numerical cancellation effects in floating point arithmetic. For higher order ETD methods such as the fourth order method considered by Cox and Matthews [19], higher order terms of the same nature appear that are even more sensitive to numerical errors. Kassam and Trefethen [53] suggest an approach to evaluating these coefficients in the numerical method using contour integration in the complex plane, numerically approximating the Cauchy integral representation (D.4).

In the nondiagonal case, directly computing the matrix exponential by this sort of approach can still be very effective if the matrix A involved is of modest size, such as may arise from a spectral approximation based on polynomials.

For very large sparse matrices, the Krylov space approach often works best. In this case we do not compute the matrix exponential itself, which is a very large dense matrix, but rather the application of this matrix to a vector. This is all that is needed in (11.32), for example. Actually we need to apply $A^{-1}(e^{Ak} - I) = k\phi(Ak)$ to a vector, which could be done in two steps, first using a Krylov space method for the exponential and then a second Krylov space method to solve the linear system, but the Krylov approach can be applied directly to the function $\phi(Ak)$. This approach has been briefly outlined at the end of Section 4.4. In practice it has been found that in some cases, particularly if a good preconditioner is not available, Krylov space methods may converge faster on the matrix exponential and related functions than it does for a simple linear system with the same coefficient matrix. In such cases the ETD methods may be more efficient than using a traditional implicit method. See, e.g., [32], [48], [77] for more details.

Part III

Appendices

Appendix A
Measuring Errors

To discuss the accuracy of a numerical solution, or the relative virtues of one numerical method, versus another, it is necessary to choose a manner of measuring that error. It may seem obvious what is meant by the error, but as we will see there are often many different ways to measure the error which can sometimes give quite different impressions as to the accuracy of an approximate solution.

A.1 Errors in a scalar value

First consider a problem in which the answer is a single value $\hat{z} \in \mathbb{R}$. Consider, for example, the scalar ordinary differential equation (ODE)

$$u'(t) = f(u(t)), \quad u(0) = \eta,$$

and suppose we are trying to compute the solution at some particular time T, so $\hat{z} = u(T)$. Denote the computed solution by z. Then the error in this computed solution is

$$E = z - \hat{z}.$$

A.1.1 Absolute error

A natural measure of this error would be the absolute value of E,

$$|E| = |z - \hat{z}|.$$

This is called the *absolute error* in the approximation.

As an example, suppose that $\hat{z} = 2.2$, while some numerical method produced a solution $z = 2.20345$. Then the absolute error is

$$|z - \hat{z}| = 0.00345 = 3.45 \times 10^{-3}.$$

This seems quite reasonable—we have a fairly accurate solution with three correct digits and the absolute error is fairly small, on the order of 10^{-3}. We might be very pleased

with an alternative method that produced an error of 10^{-6} and horrified with a method that produced an error of 10^6.

But note that our notion of what is a large error or a small error might be thrown off completely if we were to choose a different set of units for measuring $\hat{z}$. For example, suppose the $\hat{z}$ discussed above were measured in meters, so $\hat{z} = 2.2$ meters is the correct solution. But suppose that instead we expressed the solution (and the approximate solution) in nanometers rather than meters. Then the true solution is $\hat{z} = 2.2 \times 10^9$ and the approximate solution is $z = 2.20345 \times 10^9$, giving an absolute error of

$$|z - \hat{z}| = 3.45 \times 10^6.$$

We have an error that seems huge and yet the solution is just as accurate as before, with three correct digits.

Conversely, if we measured $\hat{z}$ in kilometers, then $\hat{z} = 2.2 \times 10^{-3}$ and $z = 2.20345 \times 10^{-3}$ so

$$|z - \hat{z}| = 3.45 \times 10^{-6}.$$

The error seems much smaller and yet there are still only three correct digits.

A.1.2 Relative error

The above difficulties arise from a poor choice of scaling of the problem. One way to avoid this is to consider the *relative error*, defined by

$$\left| \frac{z - \hat{z}}{\hat{z}} \right|.$$

The size of the error is scaled by the size of the value being computed. For the above examples, the relative error in z is equal to

$$\left| \frac{2.20345 - 2.2}{2.2} \right| = \left| \frac{2.20345 \times 10^9 - 2.2 \times 10^9}{2.2 \times 10^9} \right| = 1.57 \times 10^{-3}.$$

The value of the relative error is the same no matter what units we use to measure $\hat{z}$, a very desirable feature. Also note that in general a relative error that is on the order of 10^{-k} indicates that there are roughly k correct digits in the solution, matching our intuition.

For these reasons the relative error is often a better measure of accuracy than the absolute error. Of course if we know that our problem is "properly" scaled, so that the solution $\hat{z}$ has magnitude order 1, then it is fine to use the absolute error, which is roughly the same as the relative error in this case.

In fact it is generally better to ensure that the problem is properly scaled than to rely on the relative error. Poorly scaled problems can lead to other numerical difficulties, particularly if several different scales arise in the same problem so that some numbers are orders of magnitude larger than others for nonphysical reasons. Unless otherwise noted below, we will assume that the problem is scaled in such a way that the absolute error is meaningful.

A.2 "Big-oh" and "little-oh" notation

In discussing the rate of convergence of a numerical method we use the notation $O(h^p)$, the so-called big-oh notation. In case this is unfamiliar, here is a brief review of the proper use of this notation.

If $f(h)$ and $g(h)$ are two functions of h, then we say that

$$f(h) = O(g(h)) \quad \text{as } h \to 0$$

if there is some constant C such that

$$\left|\frac{f(h)}{g(h)}\right| < C \quad \text{for all } h \text{ sufficiently small}$$

or, equivalently, if we can bound

$$|f(h)| < C|g(h)| \quad \text{for all } h \text{ sufficiently small.}$$

Intuitively, this means that $f(h)$ decays to zero *at least as fast* as the function $g(h)$ does. Usually $g(h)$ is some monomial h^q, but this isn't necessary.

It is also sometimes convenient to use the "little-oh" notation

$$f(h) = o(g(h)) \quad \text{as } h \to 0.$$

This means that

$$\left|\frac{f(h)}{g(h)}\right| \to 0 \quad \text{as } h \to 0.$$

This is slightly stronger than the previous statement and means that $f(h)$ decays to zero *faster* than $g(h)$. If $f(h) = o(g(h))$, then $f(h) = O(g(h))$, although the converse may not be true. Saying that $f(h) = o(1)$ simply means that the $f(h) \to 0$ as $h \to 0$.

Examples:

$2h^3 = O(h^2)$ as $h \to 0$, since $\dfrac{2h^3}{h^2} = 2h < 1$ for all $h < \dfrac{1}{2}$.

$2h^3 = o(h^2)$ as $h \to 0$, since $2h \to 0$ as $h \to 0$.

$\sin(h) = O(h)$ as $h \to 0$, since $\sin h = h - \dfrac{h^3}{3} + \dfrac{h^5}{5} + \cdots < h$ for all $h > 0$.

$\sin(h) = h + o(h)$ as $h \to 0$, since $(\sin h - h)/h = O(h^2)$.

$\sqrt{h} = O(1)$ as $h \to 0$, and also $\sqrt{h} = o(1)$, but $\sqrt{h}$ is not $O(h)$.

$1 - \cos h = o(h)$ and $1 - \cos h = O(h^2)$ as $h \to 0$.

$h^2/\sqrt{h + h^3} = O(h^{1.5})$ and $h^2/\sqrt{h + h^3} = o(h)$ as $h \to 0$.

$e^{-1/h} = o(h^q)$ as $h \to 0$ for every value of q.

To see this, let $x = 1/h$ then $\dfrac{e^{-1/h}}{h^q} = e^{-x}x^q \to 0$ as $x \to \infty$.

Note that saying $f(h) = O(g(h))$ is a statement about how f behaves in the limit as $h \to 0$. This notation is sometimes abused by saying, for example, that if $h = 10^{-3}$, then the number 3×10^{-6} is $O(h^2)$. Although it is clear what is meant, this is really meaningless mathematically and may be misleading when analyzing the accuracy of a numerical method. If the error $E(h)$ on a grid with $h = 10^{-3}$ turns out to be 3×10^{-6}, we cannot conclude that the method is second order accurate. It could be, for example, that the error $E(h)$ has the behavior

$$E(h) = 0.003\, h, \tag{A.1}$$

in which case $E(10^{-3}) = 3 \times 10^{-6}$, but it is not true that $E(h) = O(h^2)$. In fact the method is only first order accurate, which would become apparent as we refined the grid.

Conversely, if

$$E(h) = 10^6\, h^2, \tag{A.2}$$

then $E(10^{-3}) = 1$, which is much larger than h^2, and yet it is still true that

$$E(h) = O(h^2) \quad \text{as } h \to 0.$$

Also note that there is more to the choice of a method than its asymptotic rate of convergence. While in general a second order method outperforms a first order method, if we are planning to compute on a grid with $h = 10^{-3}$, then we would prefer a first order method with error (A.1) over a second order method with error (A.2).

A.3 Errors in vectors

Now suppose $\hat{z} \in \mathbb{R}^s$, i.e., the true solution to some problem is a vector with s components. For example, $\hat{z}$ may be the solution to a system of s ODEs at some particular fixed time T. Then z is a vector of approximate values and the error $e = z - \hat{z}$ is also a vector in $\mathbb{R}^s$. In this case we can use some vector norm to measure the error.

There are many ways to define a vector norm. In general a vector norm is simply a mapping from vectors x in $\mathbb{R}^s$ to nonnegative real numbers, satisfying the following conditions (which generalize important properties of the absolute value for scalars):

1. $\|x\| \geq 0$ for any $x \in \mathbb{R}^s$, and $\|x\| = 0$ if and only if $x = \vec{0}$.
2. If a is any scalar, then $\|ax\| = |a|\, \|x\|$.
3. If $x,\ y \in \mathbb{R}^m$, then $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality).

One common choice is the max-norm (or infinity-norm) denoted by $\|\cdot\|_\infty$:

$$\|e\|_\infty = \max_{1 \leq i \leq s} |e_i|. \tag{A.3}$$

It is easy to verify that $\|\cdot\|_\infty$ satisfies the required properties. A bound on the max-norm of the error is nice because we know that every component of the error can be no greater than the max-norm. For some problems, however, there are other norms which are either more appropriate or easier to bound using our analytical tools.

Two other norms that are frequently used are the 1-norm and 2-norm,

$$\|e\|_1 = \sum_{i=1}^{s} |e_i| \qquad \text{and} \qquad \|e\|_2 = \sqrt{\sum_{i=1}^{s} |e_i|^2}. \tag{A.4}$$

These are special cases of the general family of q-norms, defined by

$$\|e\|_q = \left[\sum_{i=1}^{s} |e_i|^q \right]^{1/q}. \tag{A.5}$$

Note that the max-norm can be obtained as the limit as $q \to \infty$ of the q-norm. (Usually p is used instead of q in defining these norms, but in this book p is often used for the order of accuracy, which might be measured in some q-norm.)

A.3.1 Norm equivalence

With so many different norms to choose from, it is natural to ask whether results on convergence of numerical methods will depend on our choice of norm. Suppose $e(h)$ is the error obtained with some step size h, and that $\|e(h)\| = O(h^p)$ in some norm, so that the method is pth order accurate. Is it possible that the rate will be different in some other norm? The answer is "no," due to the following result on the "equivalence" of all norms on $\mathbb{R}^s$. (Note that this result is valid only as long as the dimension s of the vector is fixed as $h \to 0$. See Section A.5 for an important case where the length of the vector depends on h.)

Let $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ represent two different vector norms on $\mathbb{R}^s$. Then there exist two constants C_1 and C_2 such that

$$C_1\|x\|_\alpha \le \|x\|_\beta \le C_2\|x\|_\alpha \tag{A.6}$$

for all vectors $x \in \mathbb{R}^m$. For example, it is fairly easy to verify that the following relations hold among the norms mentioned above:

$$\|x\|_\infty \le \|x\|_1 \le s\|x\|_\infty, \tag{A.7a}$$
$$\|x\|_\infty \le \|x\|_2 \le \sqrt{s}\|x\|_\infty, \tag{A.7b}$$
$$\|x\|_2 \le \|x\|_1 \le \sqrt{s}\|x\|_2. \tag{A.7c}$$

Now suppose that $\|e(h)\|_\alpha \le Ch^p$ as $h \to 0$ in some norm $\|\cdot\|_\alpha$. Then we have

$$\|e(h)\|_\beta \le C_2\|e(h)\|_\alpha \le C_2 C h^p$$

and so $\|e(h)\|_\beta = O(h^p)$ as well. In particular, if $\|e(h)\| \to 0$ in some norm, then the same is true in any other norm and so the notion of "convergence" is independent of our choice of norm. This will *not* be true in Section A.4, where we consider approximating functions rather than vectors.

A.3.2 Matrix norms

For any vector norm $\|\cdot\|$ we can define a corresponding matrix norm. The norm of a matrix $A \in \mathbb{R}^{s\times s}$ is denoted by $\|A\|$ and has the property that $C = \|A\|$ is the *smallest* value of the constant C for which the bound

$$\|Ax\| \le C\|x\| \tag{A.8}$$

holds for *every* vector $x \in \mathbb{R}^s$. Hence $\|A\|$ is defined by

$$\|A\| = \max_{\substack{x\in\mathbb{R}^s \\ x\neq\vec{0}}} \frac{\|Ax\|}{\|x\|} = \max_{\substack{x\in\mathbb{R}^s \\ \|x\|=1}} \|Ax\|. \tag{A.9}$$

It would be rather difficult to calculate $\|A\|$ from the above definitions, but for the most commonly used norms there are simple formulas for computing $\|A\|$ directly from the matrix:

$$\|A\|_1 = \max_{1\le j\le s} \sum_{i=1}^{s} |a_{ij}| \qquad \text{(maximum column sum)}, \tag{A.10a}$$

$$\|A\|_\infty = \max_{1\le i\le s} \sum_{j=1}^{s} |a_{ij}| \qquad \text{(maximum row sum)}, \tag{A.10b}$$

$$\|A\|_2 = \sqrt{\rho(A^T A)}. \tag{A.10c}$$

In the definition of the 2-norm, $\rho(B)$ denotes the spectral radius of the matrix B (the maximum modulus of an eigenvalue). In particular, if A is a normal matrix, e.g., if $A = A^T$ is symmetric, then $\|A\|_2 = \rho(A)$.

We also mention the *condition number* of a matrix in a given norm, defined by

$$\kappa(A) = \|A\|\,\|A^{-1}\|, \tag{A.11}$$

provided the matrix is nonsingular. If the matrix is normal then the 2-norm condition number is the ratio of largest to smallest eigenvalue (in modulus). The condition number plays a role in the convergence rate of many iterative methods for solving a linear system with the matrix A (see Chapter 4). See, e.g., [35], [91] for more discussion.

A.4 Errors in functions

Now consider a problem in which the solution is a function $u(x)$ over some interval $a \le x \le b$ rather than a single value or vector. Some numerical methods, such as finite element or collocation methods, produce an approximate solution $U(x)$ which is also a function. Then the error is given by a function

$$e(x) = U(x) - u(x).$$

We can measure the magnitude of this error using standard function space norms, which are quite analogous to the vector norms described above. For example, the max-norm is given by

$$\|e\|_\infty = \max_{a\le x\le b} |e(x)|. \tag{A.12}$$

The 1-norm and 2-norm are given by integrals over $[a, b]$ rather than by sums over the vector elements:

$$\|e\|_1 = \int_a^b |e(x)|\, dx, \tag{A.13}$$

$$\|e\|_2 = \left(\int_a^b |e(x)|^2\, dx \right)^{1/2}. \tag{A.14}$$

These are again special cases of the general q-norm, defined by

$$\|e\|_q = \left(\int_a^b |e(x)|^q\, dx \right)^{1/q}. \tag{A.15}$$

A.5 Errors in grid functions

Finite difference methods do not produce a function $U(x)$ as an approximation to $u(x)$. Instead they produce a set of values U_i at grid points x_i. For example, on a uniform grid with N equally spaced in some interval (a, b) and grid spacing h, our approximation to $u(x)$ would consist of the N values $(U_1, U_2, \ldots, U_N)$. (Note that if $h = (b-a)/(m+1)$ as is often assumed in this book, then generally $N = m,\ m+1$, or $m+2$, depending on whether one or both boundary points are included in the set of unknowns. For our discussion here this is immaterial—what is important to note is that $N = O(1/h)$ as $h \to 0$.)

How can we measure the error in this approximation? We want to compare a set of discrete values with a function.

We must first decide what the values U_i are supposed to be approximating. Often the value U_i is meant to be interpreted as an approximation to the pointwise value of the function at x_i, so $U_i \approx u(x_i)$. In this case it is natural to define a vector of errors $e = (e_1, e_2, \ldots, e_N)$ by

$$e_i = U_i - u(x_i).$$

This is not always the proper interpretation of U_i, however. For example, some numerical methods are derived using the assumption that U_i approximates the average value of $u(x)$ over an interval of length h, e.g.,

$$U_i \approx \frac{1}{h} \int_{x_{i-1}}^{x_i} u(x)\, dx.$$

In this case it would be more appropriate to compare U_i to this cell average in defining the error. Clearly the errors will be different depending on what definition we adopt and may even exhibit different convergence rates, so it is important to make the proper choice for the method being studied.

Once we have defined the vector of errors $(e_1, \ldots, e_N)$, we can measure its magnitude using some norm. Since this is simply a vector with N components, it would be tempting to simply use one of the vector norms discussed above, e.g.,

$$\|e\|_1 = \sum_{i=1}^{N} |e_i|. \tag{A.16}$$

However, this choice would give a very misleading idea of the magnitude of the error. The quantity in (A.16) can be expected to be roughly N times as large as the error at any single grid point and here N is not the dimension of some physically relevant space, but rather the number of points on our grid. If we refine the grid and increase N, then the quantity (A.16) might well *increase* even if the error at each grid point decreases, which is clearly not the correct behavior.

Instead we should define the norm of the error by discretizing the integral in (A.13), which is motivated by considering the vector $(e_1, \ \ldots, \ e_N)$ as a discretization of some error function $e(x)$. This suggests defining

$$\|e\|_1 = h \sum_{i=1}^{N} |e_i| \tag{A.17}$$

with the factor of h corresponding to the dx in the integral. Note that since $h \approx (b-a)/N$, this scales the sum by $1/N$ as the number of grid points increases, so that $\|e\|_1$ is the average value of e over the interval (times the length of the interval), just as in (A.13). The norm (A.17) will be called a *grid function norm* and is distinct from the related vector norm. The set of values $(e_1, \ \ldots, \ e_N)$ will sometimes be called a *grid function* to remind us that it is a special kind of vector that represents the discretization of a function.

Similarly, the q-norm should be scaled by $h^{1/q}$, so that the q-norm for grid functions is

$$\|e\|_q = \left(h \sum_{i=1}^{N} |e_i|^q \right)^{1/q}. \tag{A.18}$$

Since $h^{1/q} \to 1$ as $q \to \infty$, the max-norm remains unchanged,

$$\|e\|_\infty = \max_{1 \le i \le N} |e_i|,$$

which makes sense from (A.12).

In two space dimensions we have analogous norms of functions and grid functions, e.g.,

$$\|e\|_q = \left(\iint |e(x, y)|^q \, dx \, dy \right)^{1/q} \qquad \text{for functions,}$$

$$\|e\|_q = \left(\Delta x \, \Delta y \sum_i \sum_j |e_{ij}|^q \right)^{1/q} \qquad \text{for grid functions}$$

with the obvious extension to more dimensions.

A.5.1 Norm equivalence

Note that we still have an equivalence of norms in the sense that, for any *fixed* N (and hence fixed h), there are constants C_1 and C_2 such that

$$C_1 \|x\|_\alpha \le \|x\|_\beta \le C_2 \|x\|_\alpha$$

for any vector $e \in \mathbb{R}^N$. For example, translating (A.7a) to the context of grid function norms gives the bounds

$$h\|e\|_\infty \le \|e\|_1 \le Nh\|e\|_\infty = (b-a)\,\|e\|_\infty, \tag{A.19a}$$

$$\sqrt{h}\,\|e\|_\infty \le \|e\|_2 \le \sqrt{Nh}\,\|e\|_\infty = \sqrt{b-a}\,\|e\|_\infty, \tag{A.19b}$$

$$\sqrt{h}\,\|e\|_2 \le \|e\|_1 \le \sqrt{Nh}\,\|e\|_2 = \sqrt{b-a}\,\|e\|_2. \tag{A.19c}$$

However, since these constants may depend on N and h, this equivalence does not carry over when we consider the behavior of the error as we refine the grid so that $h \to 0$ and $N \to \infty$.

We are particularly interested in the convergence rate of a method. If $e(h)$ is the vector of errors obtained on a grid with spacing h, we would like to show that

$$\|e(h)\| \le O(h^q)$$

for some q. In the last section we saw that the rate is independent of the choice of norm if $e(h)$ is a vector in the space $\mathbb{R}^m$ with fixed dimension m. But now $m = N$ and grows as $h \to 0$, and as a result the rate may be quite different in different norms. This is particularly noticeable if we approximate a discontinuous function, as the following example shows.

Example 3.1. Set

$$u(x) = \begin{cases} 0 & x \le \frac{1}{2}, \\ 1 & x > \frac{1}{2} \end{cases}$$

and define the grid function approximation as indicated in Figure A.1. Then the error $e_i(h)$ is zero at all grid points but the one at the discontinuity, where it has the value $1/2$. Then no matter how fine the grid is, there is always an error of magnitude $1/2$ at one grid point and hence

$$\|e(h)\|_\infty = \frac{1}{2} \quad \text{for all } h.$$

On the other hand, in the 1-norm (A.17) we have

$$\|e(h)\|_1 = h/2 = O(h) \quad \text{as } h \to 0.$$

We see that the 1-norm converges to zero as h goes to zero while the max-norm does not.

How should we interpret this? Should we say that $U(h)$ is a first order accurate approximation to $u(x)$ or should we say that it does not converge? It depends on what we

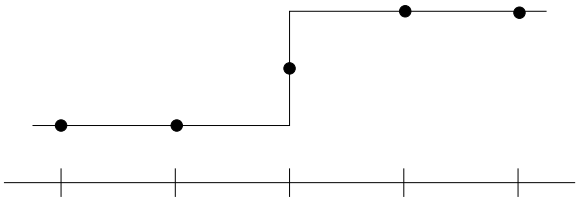

Figure A.1. *The function $u(x)$ and the discrete approximation.*

are looking for. If it is important that the maximum error over all grid points be uniformly small, then the max-norm is the appropriate norm to use and the fact that $\|e(h)\|_\infty$ does not approach zero tells us that we are not achieving our goal. On the other hand, this may not be required, and in fact this example illustrates that it is unrealistic to expect pointwise convergence in problems where the function is discontinuous. For many purposes the approximation shown in Figure A.1 would be perfectly acceptable.

This example also illustrates the effect of choosing a different definition of the "error." If we were to define the error by

$$e_i(h) = U_i - \frac{1}{h}\int_{x_i-h/2}^{x_i+h/2} u(x)\,dx,$$

then we would have $e_i(h) \equiv 0$ for all i and h and $\|e(h)\| = 0$ in every norm, including the max-norm. With this definition of the error our approximation is not only acceptable, it is the best possible approximation.

If the function we are approximating is sufficiently smooth, and if we expect the error to be roughly the same magnitude at all points, then it typically does not matter so much which norm is chosen. The convergence rate for a given method, as observed on sufficiently smooth functions, will often be the same in any q-norm. The norm chosen for analysis is then often determined by the nature of the problem and the availability of mathematical techniques for estimating different error norms. For example, for linear problems where Fourier analysis can be applied, the 2-norm is often a natural choice. For conservation laws where integrals of the solution are studied, the 1-norm is often simplest to use.

A.6 Estimating errors in numerical solutions

When developing a computer program to solve a differential equation, it is generally a good idea to test the code and ensure that it is producing correct results with the expected accuracy. How can we do this?

A first step is often to try the code on a problem for which the exact solution is known, in which case we can compute the error in the numerical solution exactly. Not only can we then check that the error is small on some grid, we can also refine the grid and check how the error is behaving asymptotically, to verify that the expected order of accuracy, and perhaps even error constant, is seen. Of course one must be aware of some of the issues raised earlier, e.g., that the expected order may appear only for h sufficiently small.

It is important to test a computer program by doing grid refinement studies even if the results look quite good on one particular grid. A subtle error in programming (or in deriving the difference equations or numerical boundary conditions) can lead to a program that gives reasonable results and may even converge to the correct solution, but at less than the optimal rate. Consider, for example, the first approach of Section 2.12.

Of course in practice we are usually trying to solve a problem for which we do not know the exact solution, or we wouldn't bother with a numerical method in the first place. However, there are often simplified versions of the problem for which exact solutions are known, and a good place to start is with these special cases. They may reveal errors in the code that will affect the solution of the real problem as well.

This is generally not sufficient, however, even when it is possible, since in going from the easy special case to the real problem new errors may be introduced. How do we estimate the error in a numerical solution if we do not have the exact solution with which to compare it?

The standard approach, when we can afford to use it, is to compute a numerical solution on a very fine grid and use this as a "reference solution" (or "fine-grid solution"). This can be used as a good approximation to the exact solution in estimating the error on other, much coarser, grids. When the fine grid is fine enough, we can obtain good estimates not only for the errors but also for the order of accuracy. See Section A.6.2.

Often we cannot afford to take very fine grids, especially in more than one space dimension. We may then be tempted to use a grid that is only slightly finer than the grid we are testing in order to generate a reference solution. When done properly this approach can also yield accurate estimates of the order of accuracy, but more care is required. See Section A.6.3.

A.6.1 Estimates from the true solution

First suppose we know the true solution. Let $E(h)$ denote the error in the calculation with grid spacing h, as computed using the true solution. In this section we suppose that $E(h)$ is a scalar, typically some norm of the error over the grid, i.e.,

$$E(h) = \|U(h) - \hat{U}(h)\|,$$

where $U(h)$ is the numerical solution vector (grid function) and $\hat{U}(h)$ is the true solution evaluated on the same grid.

If the method is pth order accurate, then we expect

$$E(h) = Ch^p + o(h^p) \quad \text{as } h \to 0,$$

and if h is sufficiently small, then

$$E(h) \approx Ch^p. \tag{A.20}$$

If we refine the grid by a factor of 2, say, then we expect

$$E(h/2) \approx C(h/2)^p.$$

Defining the *error ratio*

$$R(h) = E(h) \,/\, E(h/2), \tag{A.21}$$

we expect

$$R(h) \approx 2^p, \tag{A.22}$$

and hence

$$p \approx \log_2(R(h)). \tag{A.23}$$

Here refinement by a factor of 2 is used only as an example, since this choice is often made in practice. But more generally if h_1 and h_2 are any two grid spacings, then we can estimate p based on calculations on these two grids using

$$p \approx \frac{\log(E(h_1)/E(h_2))}{\log(h_1/h_2)}. \tag{A.24}$$

Hence we can estimate the order p based on any two calculations. (This will be valid only if h is small enough that (A.20) holds, of course.)

Note that we can also estimate the error constant C by

$$C \approx E(h)/h^p$$

once p is known.

A.6.2 Estimates from a fine-grid solution

Now suppose we don't know the exact solution but that we can afford to run the problem on a very fine grid, say, with grid spacing $\bar{h}$, and use this as a reference solution in computing the errors on some sequence of much coarser grids. To compare $U(h)$ on the coarser grid with $U(\bar{h})$ on the fine grid, we need to make sure that these two grids contain coincident grid points where we can directly compare the solutions. Typically we choose the grids in such a way that all grid points on the coarser grid are also fine-grid points. (This is often the hardest part of doing such grid refinement studies—getting the grids and indexing correct.)

Let $\bar{U}(h)$ be the restriction of the fine-grid solution to the h-grid, so that we can define the approximate error $\bar{E}(h) \equiv \|U(h) - \bar{U}(h)\|$, analogous to the true error $E(h) = \|U(h) - \hat{U}(h)\|$. What is the error in this approximate error? We have

$$U(h) - \bar{U}(h) = (U(h) - \hat{U}(h)) + (\hat{U}(h) - \bar{U}(h)).$$

If the method is supposed to be pth order accurate and $\bar{h}^p \ll h^p$, then the second term on the right-hand side (the true error on the $\bar{h}$-grid) should be negligible compared to the first term (the true error on the h-grid) and $\bar{E}(h)$ should give a very accurate estimate of the error.

Warning: Estimating the error and testing the order of accuracy by this approach only confirm that the code is converging to *some* function with the desired rate. It is very possible that the code is converging very nicely to the *wrong* function. Consider a second order accurate method applied to 2-point boundary value problem, for example, and suppose that we code everything properly except that we mistype the value of one of the boundary values. Then a grid-refinement study of this type would show that the method is converging with second order accuracy, as indeed it is. The fact that it is converging to the solution of the wrong problem would not be revealed by this test. One must use other tests as well, not least of which is checking that the computed solutions make sense physically, e.g., that the correct boundary conditions are in fact satisfied.

More generally, a good understanding of the problem being solved, a knowledge of how the solution should behave, good physical intuition, and common sense are all necessary components in successful scientific computing. Don't believe the numbers coming out simply because they are generated by a computer, even if the computer also tells you that they are second order accurate!

A.6.3 Estimates from coarser solutions

Now suppose that our computation is very expensive even on relatively coarse grids, and we cannot afford to run a calculation on a much finer grid to test the order of accuracy.

Suppose, for example, that we are willing to run the calculation only on grids with spacing h, $h/2$, and $h/4$ and we wish to estimate the order of accuracy from these three calculations, without using any finer grids. Since we can estimate the order from any two values of the error, we could define the errors in the two coarser grid calculations by using the $h/4$ calculation as our reference solution. Will we get a good estimate for the order?

In the notation used above, we now have $\bar{h} = h/4$, while $h = 4\bar{h}$ and $h/2 = 2\bar{h}$. Assuming the method is pth order accurate and that h is small enough that (A.20) is valid (a poor assumption, perhaps, if we are using very coarse grids!), we expect

$$\begin{aligned}\bar{E}(h) &= E(h) - E(\bar{h}) \\ &\approx Ch^p - C\bar{h}^p \\ &= (4^p - 1)C\bar{h}^p.\end{aligned}$$

Similarly,

$$\bar{E}(h/2) \approx (2^p - 1)C\bar{h}^p.$$

The ratio of approximate errors is thus

$$\bar{R}(h) = \bar{E}(h)/\bar{E}\left(\frac{h}{2}\right) \approx \frac{4^p - 1}{2^p - 1} = 2^p + 1.$$

For modest p this differs significantly from (A.22). For a first order accurate method with $p = 1$, we now have $\bar{R}(h) \approx 3$ and we should expect the apparent error to decrease by a factor of 3 when we go from h to $h/2$, not by the factor of 2 that we normally expect. For a second order method we expect a factor of 5 improvement rather than a factor of 4. This increase in $\bar{R}(h)$ results from the fact that we are comparing our numerical solutions to another approximate solution that has a similar error.

We can obtain a good estimate of p from such calculations (assuming (A.20) is valid), but to do so we must calculate p by

$$p \approx \log_2(\bar{R}(h) - 1)$$

rather than by (A.23). The approximation (A.23) would overestimate the order of accuracy.

Again we have used refinement by factors of 2 only as an example. If the calculation is very expensive we might want to refine the grid more slowly, using, for example, h, $3h/4$, and $h/2$. One can develop appropriate approximations to p based on any three grids. The tricky part may be to estimate the error at grid points on the coarser grids if these are not also grid points on the $\bar{h}$ grid. Interpolation can be used, but then one must be careful to ensure that sufficiently accurate interpolation formulas are used that the error in interpolation does not contaminate the estimate of the error in the numerical method being studied.

Another approach that is perhaps simpler is to compare the solutions

$$\tilde{E}(h) \equiv U(h) - U(h/2) \quad \text{and} \quad \tilde{E}(h/2) = U(h/2) - U(h/4).$$

In other words, we estimate the error on each grid by using the next finer grid as the reference solution, rather than using the same reference solution for both coarser grids. In this case we have

$$\tilde{E}(h) = E(h) - E\left(\frac{h}{2}\right) \approx C\left(1 - \frac{1}{2^p}\right)h^p$$

and

$$\tilde{E}(h/2) = E\left(\frac{h}{2}\right) - E\left(\frac{h}{4}\right) \approx C\left(1 - \frac{1}{2^p}\right)\frac{h^p}{2^p}$$

and so

$$\tilde{E}(h)/\tilde{E}\left(\frac{h}{2}\right) \approx 2^p.$$

In this case the approximate error decreases by the same factor we would expect if the true solution were used as the reference solution on each grid.

Appendix B

Polynomial Interpolation and Orthogonal Polynomials

B.1 The general interpolation problem

Given a set of discrete points x_i for $i = 0,\ 1,\ \ldots,\ n$ and function values F_i, the interpolation problem is to determine a function $\phi(x)$ of some specified form passing through these points,

$$\phi(x_i) = F_i \quad \text{for } i = 0,\ 1,\ \ldots,\ n. \tag{B.1}$$

We use the notation $\text{Int}(x_0, \ldots, x_n)$ to denote the smallest interval containing all these points (which need not be in increasing order but which are assumed to be distinct).

Interpolation has many uses; for example,

- we may have only discrete data values and want to estimate values in between, $x \in \text{Int}(x_0, \ldots, x_n)$. This is the origin of the term *interpolation*. We might also use this function to *extrapolate* if we evaluate it outside the interval where data are given.
- we may know the true function $F(x)$ but want to approximate it by a function $\phi(x)$ that is cheaper to evaluate, or easier to work with symbolically (to differentiate or integrate, for example).
- we may use it as a starting point for deriving numerical methods for differential equations (or for integral equations or numerical integration).

There are infinitely many possible functions ϕ. Typically ϕ is chosen to be a linear combination of some $n + 1$ given *basis functions* $\phi_0(x),\ \ldots,\ \phi_n(x)$,

$$\phi(x) = c_0\phi_0(x) + \cdots + c_n\phi_n(x). \tag{B.2}$$

Then condition (B.1) gives a linear system of $n + 1$ equations to solve for the coefficients $c_0,\ \ldots,\ c_n$,

$$\begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \vdots & & & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \cdots & \phi_n(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_n \end{bmatrix}. \tag{B.3}$$

This system can we written as $\Phi c = F$. Different choices of basis functions lead to different types of interpolation. Using trigonometric functions gives Fourier series, for example, the basis of Fourier spectral methods.

In this appendix we consider interpolation by polynomials, the basis of many finite difference and spectral methods.

B.2 Polynomial interpolation

Through any $n + 1$ points there is a unique interpolating polynomial $p(x)$ of degree n. There are many ways to represent this function depending on what basis is chosen for $\mathcal{P}_n$, the set of all polynomials of degree n.

B.2.1 Monomial basis

The monomial functions are

$$\phi_0(x) = 1, \quad \phi_1(x) = x, \quad \phi_2(x) = x^2, \quad \ldots, \quad \phi_n(x) = x^n. \tag{B.4}$$

The matrix Φ appearing in (B.3) is then the *Vandermonde matrix*. This matrix may be quite ill-conditioned for larger values of n.

B.2.2 Lagrange basis

The jth Lagrange basis function (based on a given set of interpolation points x_i) is given by

$$\phi_j(x) = \prod_{\substack{i=0 \\ i \neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}. \tag{B.5}$$

This is a polynomial of degree n. Note that

$$\phi_j(x_i) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

Then the matrix in (B.3) is the identity matrix and $c_i = F_i$. The coefficients are easy to determine in this form but the basis functions are a bit cumbersome.

B.2.3 Newton form

The Newton form of the interpolating polynomial is

$$p(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}). \tag{B.6}$$

For these basis functions the matrix Φ is *lower triangular* and the c_i may be found by forward substitution. Alternatively they are most easily computed using *divided differences*, $c_i = F[x_0, \ldots, x_i]$. These can be computed from a tableau of the form

$$\begin{array}{lll} x_0 & F[x_0] & \\ & & F[x_0, x_1] \\ x_1 & F[x_1] & & F[x_0, x_1, x_2] \\ & & F[x_1, x_2] \\ x_2 & F[x_2], & \end{array} \tag{B.7}$$

where

$$F[x_j] = F_j$$

and for $k > 0$,

$$F[x_j, \ldots, x_{j+k}] = \frac{F[x_{j+1}, \ldots, x_{j+k}] - F[x_j, \ldots, x_{j+k-1}]}{x_{j+k} - x_j}. \tag{B.8}$$

Then the Newton form can be built up as follows:

$$\begin{aligned} p_0(x) &= F[x_0] \\ &\quad \text{is the polynomial of degree 0 interpolating at } x_0, \\ p_1(x) &= F[x_0] + F[x_0, x_1](x - x_0) \\ &\quad \text{is the polynomial of degree 1 interpolating at } x_0, x_1, \\ p_2(x) &= F[x_0] + F[x_0, x_1](x - x_0) + F[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad \text{is the polynomial of degree 2 interpolating at } x_0, x_1, x_2, \\ &\text{etc.} \end{aligned}$$

In each step we add a term that vanishes at all the preceding interpolation points and makes the function also interpolate at one new point. Note that the coefficients of previous basis functions do not change.

Relation to Taylor series. Note that

$$F[x_j, x_{j+1}] = \frac{F_{j+1} - F_j}{x_{j+1} - x_j}. \tag{B.9}$$

Suppose the data values F_i come from some underlying smooth function $F(x)$, so $F_i = F(x_i)$. Then (B.9) approximates the derivative $F'(x_j)$. Similarly, if $x_j, \ldots, x_{j+k}$ are close together, then

$$F[x_j, \ldots, x_{j+k}] \approx \frac{1}{k!} F^{(k)}(x_j), \tag{B.10}$$

where $F^{(k)}(x)$ is the kth derivative. In fact, one can show that for sufficiently smooth F,

$$F[x_j, \ldots, x_{j+k}] = \frac{1}{k!} F^{(k)}(\xi) \tag{B.11}$$

for some ξ lying in the interval $\text{Int}(x_j, \ldots, x_{j+k})$. This is true provided that F is k times continuously differentiable on this interval. The Newton form (B.6) thus is similar to the Taylor series

$$F(x) = F(x_0) + F'(x_0)(x - x_0) + \frac{1}{2!} F''(x_0)(x - x_0)^2 + \cdots \tag{B.12}$$

and reduces to this in the limit as $x_j \to x_0$ for all j.

B.2.4 Error in polynomial interpolation

Suppose $F(x)$ is a smooth function, we evaluate $F_i = F(x_i)$ $(i = 0, 1, \ldots, n)$, and we now fit a polynomial $p(x)$ of degree n through these points. How well does $p(\bar{x})$ approximate $F(\bar{x})$ at some other point $\bar{x}$?

Note that we could add $\bar{x}$ as another interpolation point and create an interpolating polynomial $\bar{p}(x)$ of degree $n+1$ that interpolates also at this point,

$$\bar{p}(x) = p(x) + F[x_0, \ldots, x_n, \bar{x}](x - x_0) \cdots (x - x_n).$$

Then $\bar{p}(\bar{x}) = F(\bar{x})$ and so

$$F(\bar{x}) - p(\bar{x}) = F[x_0, \ldots, x_n, \bar{x}](\bar{x} - x_0) \cdots (\bar{x} - x_n).$$

Using (B.11), we obtain an error formula similar to the remainder formula for Taylor series. If $p(x)$ is given by (B.6), then at any point x,

$$F(x) - p(x) = \frac{1}{n!} F^{(n)}(\xi)(x - x_0) \cdots (x - x_n), \tag{B.13}$$

where ξ is some point lying in $\text{Int}(x, x_0, \ldots, x_n)$. How large this is depends on

- how close the point x is to the interpolation points $x_0, \ldots, x_n$, and
- how small the derivative $F^{(n)}(\xi)$ is over this interval, i.e., how smooth the function is.

For a given x we don't know exactly what ξ is in general, but we can often use this expression to obtain an *error bound* of the form

$$|p(x) - F(x)| \le K |(x - x_0) \cdots (x - x_n)|, \tag{B.14}$$

where

$$K = \frac{1}{n!} \max_{\xi \in \text{Int}(x_0, \ldots, x_n)} |F^{(n)}(\xi)|.$$

Note that the bound (B.14) involves values of the polynomial $Q(x) \equiv \prod_{i=1}^{n} (x - x_i)$, the polynomial with roots at the interpolation points x_i and with leading coefficient 1 (i.e., a monic polynomial). If we want to minimize the error over some interval, then we might want to choose the interpolation points to minimize the maximum value that $Q(x)$ takes over that interval. We will return to this in Section B.3.2, where we will see that Chebyshev polynomials satisfy the required optimality condition. These are a particular class of orthogonal polynomials, as described in the next section.

B.3 Orthogonal polynomials

If $w(x)$ is a function on an interval $[a, b]$ that is positive everywhere on the interval, then we can define the *inner product* of two functions $f(x)$ and $g(x)$ on this interval by

$$\langle f, g \rangle = \int_a^b w(x) f(x) g(x)\, dx. \tag{B.15}$$

We say that two functions $f(x)$ and $g(x)$ are *orthogonal* in a given interval $[a, b]$ with respect to the given weight function $w(x)$ if the inner product of f and g is equal to zero.

For a given $[a, b]$ and $w(x)$, one can define a sequence of orthogonal polynomials $P_0(x)$, $P_1(x)$, ... of increasing degree which have the property that

$$\begin{aligned} &P_m(x) \in \mathcal{P}_m \quad \text{(the set of polynomials of degree } m\text{)}, \\ &\langle P_m, P_n \rangle = 0 \quad \text{for } m \neq n. \end{aligned} \tag{B.16}$$

The sequence of polynomials is said to be *orthonormal* if, in addition,

$$\langle P_m, P_m \rangle = 1 \quad \text{for all } m. \tag{B.17}$$

Orthogonal polynomials have many interesting and useful properties and arise in numerous branches of numerical analysis. In particular, the Chebyshev polynomials described in Section B.3.2 are used in several contexts in this book.

The set of polynomials $P_0(x)$, $P_1(x)$, ..., $P_k(x)$ forms an *orthogonal basis* for $\mathcal{P}_k$. Any polynomial $p \in \mathcal{P}_k$ can be uniquely expressed as a linear combination of P_0, ..., P_k. Note that each P_m must have an exact degree of m, meaning the coefficient of x^m is nonzero. Otherwise it would be a linear combination of previous polynomials in the sequence and could not be orthogonal to them all.

Note that if P_m is orthogonal to P_0, P_1, ..., P_{m-1}, then in fact P_m is orthogonal to all polynomials $p \in \mathcal{P}_{m-1}$ of degree less than m, since p can be written as $p(x) = c_0 P_0(x) + \cdots + c_{m-1} P_{m-1}(x)$ and so

$$\langle p, P_m \rangle = c_0 \langle P_0, P_m \rangle + \cdots c_{m-1} \langle P_{m-1}, P_m \rangle = 0.$$

We say that P_m is orthogonal to the space $\mathcal{P}_{m-1}$.

Sequences of orthogonal polynomials can be built up by a Gram–Schmidt process, analogous to the manner in which a sequence of linearly independent vectors is transformed into a sequence of orthogonal vectors. Suppose P_0, P_1, ..., P_m are already mutually orthogonal with P_n having exact degree n. We wish to construct $P_{m+1}(x)$, a polynomial of exact degree $m + 1$ that is orthogonal to all of these. Start with the polynomial

$$Q(x) = \alpha_m x P_m(x)$$

for some $\alpha_m \neq 0$. This polynomial has exact degree $m + 1$ and hence is linearly independent from P_0, P_1, ..., P_m. Moreover, it is already orthogonal to P_0, P_1, ..., P_{m-2}, since

$$\langle Q, P_n \rangle = \langle x P_m, P_n \rangle = \int_a^b w(x) x P_m(x) P_n(x)\, dx = \langle P_m, x P_n \rangle = 0$$

for $n \leq m - 2$, since $x P_n \in \mathcal{P}_{m-1}$ and P_m is orthogonal to this space. We wish to make Q orthogonal to P_{m-1} and P_m and, as in the Gram–Schmidt process for vectors, we do this by subtracting multiples of P_{m-1} and P_m from Q:

$$P_{m+1}(x) = \alpha_m x P_m(x) - \beta_m P_m(x) - \gamma_m P_{m-1}(x). \tag{B.18}$$

Requiring $\langle P_{m+1}, P_m \rangle = 0$ determines

$$\beta_m = \frac{\langle P_m, \alpha_m x P_m \rangle}{\langle P_m, P_m \rangle} \tag{B.19}$$

and then $\langle P_{m+1}, P_{m-1} \rangle = 0$ gives

$$\gamma_m = \frac{\langle P_{m-1}, \alpha_m x P_m \rangle - \beta_m \langle P_{m-1}, P_m \rangle}{\langle P_{m-1}, P_{m-1} \rangle}. \tag{B.20}$$

The relation (B.18) is a *three-term recurrence relation* for the sequence of orthogonal polynomials and can be used to generate the entire sequence once P_0 and P_1 are specified. For many useful sets of orthogonal polynomials the coefficients α_m, β_m, and γ_m take particularly simple forms.

B.3.1 Legendre polynomials

The sequence of polynomials that are orthogonal on $[-1, 1]$ with weight function $w(x) = 1$ are called the *Legendre polynomials*. We must also choose some normalization to uniquely define this sequence (since multiplying two orthogonal polynomials by arbitrary constants leaves them orthogonal). This amounts to choosing the nonzero constants α_m in (B.18). One might choose the polynomials to be orthonormal (i.e., normalize so that (B.17) is satisfied), but this leads to messy coefficients. The traditional choice is to require that $P_m(1) = 1$ for all m. The first few Legendre polynomials are then

$$\begin{aligned} P_0(x) &= 1, \\ P_1(x) &= x, \\ P_2(x) &= \frac{3}{2}x^2 - \frac{1}{2}, \\ P_3(x) &= \frac{5}{2}x^3 - \frac{3}{2}x. \end{aligned} \tag{B.21}$$

These polynomials satisfy a three-term recurrence relation (B.18) with

$$\alpha_m = \frac{2m+1}{m+1}, \quad \beta_m = 0, \quad \gamma_m = \frac{m}{m+1}.$$

The roots of the Legendre polynomials are of importance in various applications. In particular, they are the nodes for *Gaussian quadrature* formulas for approximating the integral of a function; see, e.g., [16], [90]. There is no simple expression for the location of the roots, but they can be found as the eigenvalues of a tridiagonal matrix in MATLAB by the following code (adapted from the program `gauss.m` in [90], which also computes the weights for the associated Gauss quadrature rules):

```
Toff = .5./sqrt(1-(2*(1:m-1)).^(-2));
T = diag(Toff,1) + diag(Toff,-1);
xi = sort(eig(T));
```

These points are also sometimes used as grid points in spectral methods.

B.3.2 Chebyshev polynomials

Several topics discussed in this book involve the Chebyshev polynomials $T_m(x)$. These are a sequence of polynomials that are orthogonal on the interval $[-1, 1]$ with the weight function

$$w(x) = (1 - x^2)^{-1/2}. \tag{B.22}$$

The first few Chebyshev polynomials are

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2x^2 - 1, \\ T_3(x) &= 4x^3 - 3x. \end{aligned} \tag{B.23}$$

Again these are normalized so that $T_m(1) = 1$ for $m = 0,\ 1,\ \ldots$. Chebyshev polynomials satisfy a particularly simple three-term recurrence,

$$T_{m+1}(x) = 2xT_m(x) - T_{m-1}(x). \tag{B.24}$$

While the weight function (B.22) may seem to be a less natural choice than $w(x) = 1$, the Chebyshev polynomials have a number of valuable properties, a few of which are listed below and are used elsewhere in this text.

Property 1. The Chebyshev polynomial $T_m(x)$ *equioscillates* $m + 1$ times in the interval $[-1, 1]$, i.e., $|T_m(x)|$ is maximized at $m + 1$ points $x_0,\ x_1,\ \ldots,\ x_m$ in the interval, points where $T_m(x)$ takes the values

$$T_m(x_j) = (-1)^j.$$

These *Chebyshev extreme points* are given by

$$x_j = \cos(j\pi/m), \quad j = 0,\ 1,\ \ldots,\ m. \tag{B.25}$$

(Note that these are labeled in decreasing order from $x_0 = 1$ to $x_m = -1$.) Figure B.1 shows a plot of $T_7(x)$, for example. This set of extreme points will be useful for spectral methods as discussed in Section 2.21.

Property 2. For x in the interval $[-1, 1]$, the value of $T_m(x)$ is given by

$$T_m(x) = \cos(m \arccos x). \tag{B.26}$$

This does not look much like a polynomial, but it is since $\cos(m\theta)$ can be written as a polynomial in $\cos(\theta)$ using trigonometric identities, and then set as $x = \cos(\theta)$. Note that from this formulation it is easy to check that (B.25) gives the desired extreme points.

Outside this interval there is an analogous formula in terms of the hyperbolic cosine,

$$T_m(x) = \cosh(m \cosh^{-1} x) \qquad \text{for } |x| \geq 1, \tag{B.27}$$

an expression that is used in the analysis of the convergence of conjugate gradients. Note that outside the unit interval the Chebyshev polynomials grow very rapidly.

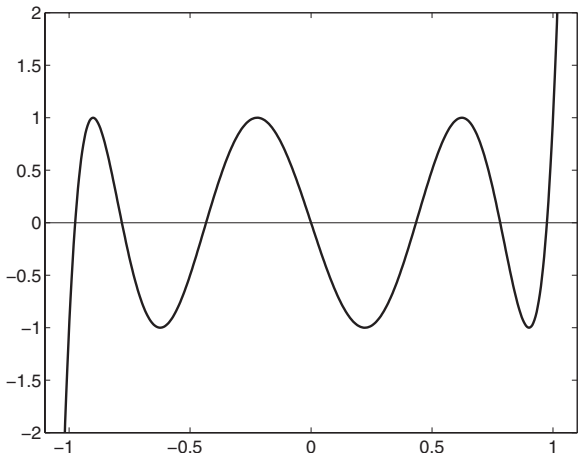

Figure B.1. *The Chebyshev polynomial $T_7(x)$ of degree 7.*

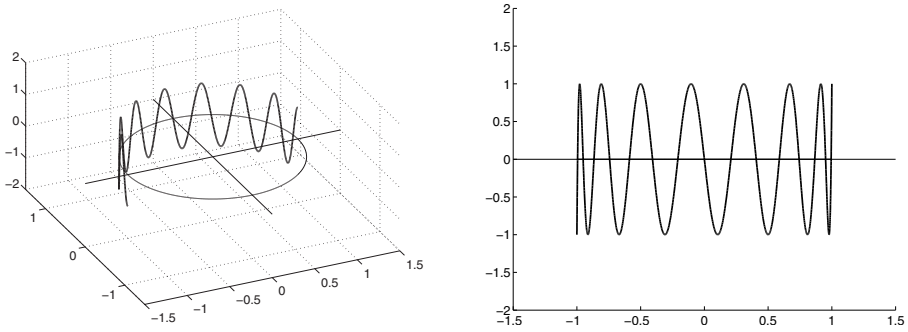

Figure B.2. *The Chebyshev polynomial viewed as a function $C_m(\theta)$ on the unit disk $e^{i\theta}$ and when projected on the x-axis, i.e., as a function of $x = \cos(\theta)$. Shown for $m = 15$.*

Property 3. Consider the function

$$C_m(\theta) = \mathrm{Re}(e^{im\theta}) = \cos(m\theta) \tag{B.28}$$

for $0 \le \theta \le \pi$. We can view this as a function defined on the upper half of the unit circle in the complex plane. If we identify $x = \cos(\theta)$ or $\theta = \arccos x$, then this reduces to (B.26), so we can view the Chebyshev polynomial on the interval $[-1, 1]$ as being the projection of the function (B.28) onto the real axis, as illustrated in Figure B.2. This property is useful in relating polynomial interpolation at Chebyshev points to trigonometric interpolation at equally spaced points on the unit circle and allows the use of the Fast Fourier Transform (FFT) algorithm to efficiently implement Chebyshev spectral methods. Orthogonality of the Chebyshev polynomials with respect to the weight function (B.22) also can be easily interpreted in terms of orthogonality of the trigonometric functions $\cos(m\theta)$ and $\cos(n\theta)$.

Property 4. The m roots of $T_m(x)$ all lie in $[-1, 1]$, at the points

$$\xi_j = \cos\left(\frac{(j - 1/2)\pi}{m}\right) \qquad \text{for } j = 1,\ 2,\ \ldots,\ m. \tag{B.29}$$

This follows directly from the representation (B.26).

Property 5. The Chebyshev polynomial $T_m(x)$ solves the mini-max optimization problem

$$\text{find } p \in \mathcal{P}_m^1 \text{ to minimize } \max_{-1\le x\le 1} |p(x)|, \tag{B.30}$$

where $\mathcal{P}_m^1$ is the set of mth degree polynomials satisfying $p(1) = 1$. Recall that $T_m(x)$ equioscillates with extreme values ± 1 so $\max_{-1\le x\le 1} |T_m(x)| = 1$.

A slightly different formulation of this property is sometimes useful: the scaled Chebyshev polynomial $2^{1-m}T_m(x)$ is the monic polynomial of degree m that minimizes $\max_{-1\le x\le 1} |p(x)|$. A monic polynomial has leading coefficient 1 on the x^m term. The scaled Chebyshev polynomial $2^{1-m}T_m(x) = \prod_{j=1}^m (x - \xi_j)$ has leading coefficient 1 and equioscillates between the values $\pm 2^{1-m}$. If we try to reduce the level of any of these peaks by perturbing the polynomial slightly, at least one of the other peaks will increase in magnitude.

Note that 2^{1-m} decays to zero exponentially fast as we increase the degree. This is responsible for the spectral accuracy of Chebyshev spectral methods and this optimality is also used in proving the rapid convergence of the conjugate gradient algorithm (see Section 4.3.4).

Returning to the formula (B.14) for the error in polynomial interpolation, we see that if we are interested in approximating the function $F(x)$ uniformly well on the interval $[-1, 1]$, then we should use the Chebyshev roots (B.29) as interpolation points. Then (B.14) gives the bound

$$|p(x) - F(x)| \le K\, 2^{1-n}.$$

On a different interval $[a, b]$, we can use the shifted Chebyshev polynomial

$$T_n\left(\frac{2x - (a + b)}{(b - a)}\right). \tag{B.31}$$

The corresponding Chebyshev extreme points and Chebyshev roots are then

$$x_j = \frac{a + b}{2} + \frac{(b - a)}{2}\cos\left(\frac{j\pi}{m}\right) \qquad \text{for } j = 0,\ 1,\ \ldots,\ m \tag{B.32}$$

and

$$\xi_j = \frac{a + b}{2} + \frac{(b - a)}{2}\cos\left(\frac{(j - 1/2)\pi}{m}\right) \qquad \text{for } j = 1,\ 2,\ \ldots,\ m, \tag{B.33}$$

respectively.

Appendix C

Eigenvalues and Inner-Product Norms

The analysis of differential equations and of finite difference methods for their solution relies heavily on "spectral analysis," based on the eigenvalues and eigenfunctions of differential operators or the eigenvalues and eigenvectors of matrices approximating these operators. In particular, knowledge of the spectrum of a matrix (the set of eigenvalues) gives critical information about the behavior of powers or exponentials of the matrix, as reviewed in Appendix D. An understanding of this is crucial in order to analyze the behavior and stability properties of differential or finite difference equations, as discussed in Section D.2.1 and at length in the main text.

This appendix contains a review of basic spectral theory and also some additional results on inner-product norms and the relation between these norms and spectra.

Let $A \in \mathbb{C}^{m\times m}$ be an $m \times m$ matrix with possibly complex components. We will mostly be working with real matrices, but many of the results carry over directly to the complex case or are most easily presented in this generality. Moreover, even real matrices can have complex eigenvalues and eigenvectors, so we must work in the complex plane.

The matrix A has m eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ that are the roots of the *characteristic polynomial*,

$$p_A(z) = \det(A - zI) = (z - \lambda_1)(z - \lambda_2)\cdots(z - \lambda_m).$$

This polynomial of degree m always has m roots, although some may be multiple roots. If no two are equal, then we say the roots are *distinct*. The set of m eigenvalues is called the *spectrum* of the matrix, and the *spectral radius* of A, denoted by $\rho(A)$, is the maximum magnitude of any eigenvalue,

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

If the characteristic polynomial $p_A(z)$ has a factor $(z - \lambda)^s$, then the eigenvalue λ is said to have *algebraic multiplicity* $m_a(\lambda) = s$. If λ is an eigenvalue, then $A - \lambda I$ is a singular matrix and the null space of this matrix is the *eigenspace of* A corresponding to this eigenvalue,

$$\mathcal{N}(A - \lambda I) = \{u \in \mathbb{C}^m : (A - \lambda I)u = 0\} = \{u \in \mathbb{C}^m : Au = \lambda u\}.$$

Any vector u in the eigenspace satisfies $Au = \lambda u$. The dimension of this eigenspace is called the *geometric multiplicity* $m_g(\lambda)$ of the eigenvalue λ. We always have

$$1 \le m_g(\lambda) \le m_a(\lambda). \tag{C.1}$$

If $m_g(\lambda) = m_a(\lambda)$, then A has a *complete set* of eigenvectors for this eigenvalue. Otherwise this eigenvalue is said to be *defective*. If A has one or more defective eigenvalues, then A is a *defective matrix*.

Example C.1. If the eigenvalues of A are all *distinct*, then $m_g = m_a = 1$ for every eigenvalue and the matrix is not defective.

Example C.2. A diagonal matrix cannot be defective. The eigenvalues are simply the diagonal elements, and the unit vectors e_j (the vector with a 1 in the jth element, zeros elsewhere) form a complete set of eigenvectors. For example,

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

has $\lambda_1 = \lambda_2 = 3$ and $\lambda_3 = 5$. The two-dimensional eigenspace for $\lambda = 3$ is spanned by $e_1 = (1, 0, 0)^T$ and $e_2 = (0, 1, 0)^T$. The one-dimensional eigenspace for $\lambda = 5$ is spanned by $e_3 = (0, 0, 1)^T$.

Example C.3. Any upper triangular matrix has eigenvalues equal to its diagonal elements d_i since the characteristic polynomial is simply $p_A(z) = (z - d_1) \cdots (z - d_m)$. The matrix may be defective if there are repeated roots. For example,

$$A = \begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

has $\lambda_1 = \lambda_2 = \lambda_3 = 3$ and $\lambda_4 = 5$. The eigenvalue $\lambda = 3$ has algebraic multiplicity $m_a = 3$ but there is only a two-dimensional space of eigenvectors associated with $\lambda = 3$, spanned by e_1 and e_3, so $m_g = 2$.

C.1 Similarity transformations

Let S be any nonsingular matrix and set

$$B = S^{-1}AS. \tag{C.2}$$

Then B has the same eigenvalues as A. To see this, suppose

$$Ar = \lambda r \tag{C.3}$$

for some vector r and scalar λ. Let $w = S^{-1}r$ and multiply (C.3) by S^{-1} to obtain

$$(S^{-1}AS)(S^{-1}r) = \lambda(S^{-1}r) \quad \Longrightarrow \quad Bw = \lambda w,$$

so λ is also an eigenvalue of B with eigenvector $S^{-1}r$. Conversely, if λ is any eigenvalue of B with eigenvector w, then similar manipulations in reverse show that λ is also an eigenvalue of A with eigenvector Sw.

The transformation (C.2) from A to B is called a *similarity transformation* and we say that the matrices A and B are *similar* if such a relation holds. The fact that similar matrices have the same eigenvalues is exploited in most numerical methods for computing eigenvalues of a matrix—a sequence of similarity transformations is performed to approximately reduce A to a simpler form from which it is easy to determine the eigenvalues, such as a diagonal or upper triangular matrix. See, for example, [35] for introductory discussions of such algorithms.

C.2 Diagonalizable matrices

If A is not defective (i.e., if every eigenvalue has a complete set of eigenvectors), then it is *diagonalizable*. In this case we can choose a set of m linearly independent right eigenvectors r_j spanning all of $\mathbb{C}^m$ such that $Ar_j = \lambda_j r_j$ for $j = 1, 2, \ldots, m$. Let R be the *matrix of right eigenvectors*

$$R = [r_1|r_2|\cdots|r_m]. \tag{C.4}$$

Then

$$AR = R\Lambda, \tag{C.5}$$

where

$$\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m). \tag{C.6}$$

This follows by viewing the matrix multiplication columnwise. Since the vectors r_j are linearly independent, the matrix R is invertible and so from (C.5) we obtain

$$R^{-1}AR = \Lambda, \tag{C.7}$$

and hence we can *diagonalize* A by a similarity transformation. We can also write

$$A = R\Lambda R^{-1}, \tag{C.8}$$

which is sometimes called the *eigendecomposition of* A. This is a special case of the Jordan canonical form discussed in the next section.

Let ℓ_j^T be the jth row of R^{-1}. We can also write the above expressions as

$$R^{-1}A = \Lambda R^{-1}$$

and when these multiplications are viewed rowwise we obtain $\ell_j^T A = \lambda_j \ell_j^T$, which shows that the rows of R^{-1} are the *left eigenvectors* of A.

C.3 The Jordan canonical form

If A is diagonalizable, we have just seen in (C.8) that we can decompose A as $A = R\Lambda R^{-1}$. If A is defective, then it cannot be written in this form; A is not similar to a

diagonal matrix. The closest we can come is to write it in the form $A = RJR^{-1}$, where the matrix J is block diagonal. Each block has nonzeros everywhere except perhaps on its diagonal and superdiagonal, and is a *Jordan block* of some order. The Jordan blocks of orders 1, 2, and 3 are

$$J(\lambda,1) = \lambda, \qquad J(\lambda,2) = \begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix}, \qquad J(\lambda,3) = \begin{bmatrix} \lambda & 1 & 0 \\ 0 & \lambda & 1 \\ 0 & 0 & \lambda \end{bmatrix}.$$

In general a Jordan block of order k has the form

$$J(\lambda,k) = \lambda I_k + S_k, \tag{C.9}$$

where I_k is the $k \times k$ identity matrix and S_k is the $k \times k$ shift matrix

$$S_k = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \qquad \text{for } k > 1 \qquad (\text{with } S_1 = 0), \tag{C.10}$$

so called because $S_k(u_1, u_2, \ldots, u_{k-1}, u_k)^T = (u_2, u_3, \ldots, u_k, 0)^T$. A Jordan block of order k has eigenvalues λ with algebraic multiplicity $m_a = k$ and geometric multiplicity $m_g = 1$. The unit vector $e_1 = (1, 0, \ldots, 0)^T \in \mathbb{C}^k$ is a basis for the one-dimensional eigenspace of this block.

Theorem C.1. *Every $m \times m$ matrix $A \in \mathbb{C}^{m\times m}$ can be transformed into the form*

$$A = RJR^{-1}, \tag{C.11}$$

where J is a block diagonal matrix of the form

$$J = \begin{bmatrix} J(\lambda_1,k_1) & & & \\ & J(\lambda_2,k_2) & & \\ & & \ddots & \\ & & & J(\lambda_s,k_s) \end{bmatrix}. \tag{C.12}$$

Each $J(\lambda_i,k_i)$ is a Jordan block of some order k_i and $\sum_{i=1}^{s} k_i = m$. If λ is an eigenvalue of A with algebraic multiplicity m_a and geometric multiplicity m_g, then λ appears in m_g blocks and the sum of the orders of these blocks is m_a.

The nonsingular matrix R contains eigenvectors of A. In the defective case, R must also contain other vectors since there is not a complete set of eigenvectors in this case. These other vectors are called *principal vectors*.

Example C.4. For illustration, consider a 3×3 matrix A with a single eigenvalue λ with $m_a(\lambda) = 3$ but $m_g(\lambda) = 1$. Then we wish to find a 3×3 invertible matrix R such that

$$AR = RJ = [r_1|r_2|r_3] \begin{bmatrix} \lambda & 1 & 0 \\ 0 & \lambda & 1 \\ 0 & 0 & \lambda \end{bmatrix}.$$

From this we obtain

$$\begin{aligned} Ar_1 = \lambda r_1 \quad &\Longrightarrow \quad (A-\lambda I)r_1 = 0, \\ Ar_2 = r_1 + \lambda r_2 \quad &\Longrightarrow \quad (A-\lambda I)r_2 = r_1 \quad \Longrightarrow \quad (A-\lambda I)^2 r_2 = 0, \\ Ar_3 = r_2 + \lambda r_3 \quad &\Longrightarrow \quad (A-\lambda I)r_3 = r_2 \quad \Longrightarrow \quad (A-\lambda I)^3 r_3 = 0. \end{aligned} \tag{C.13}$$

The vector r_1 forms a basis for the one-dimensional eigenspace. The vectors r_2 and r_3 are principal vectors. They are linearly independent vectors in the null space of $(A-\lambda I)^2$ and the null space of $(A-\lambda I)^3$ that are not in the null space of $A-\lambda I$.

The choice of the value 1 on the superdiagonal of the nontrivial Jordan blocks is the standard convention, but this can be replaced with any nonzero value δ by modifying the matrix R appropriately. This is easy to verify by applying the following similarity transformation to a Jordan block $J(\lambda, k)$. Choose $\delta \neq 0$ and set

$$D = \begin{bmatrix} 1 & & & & \\ & \delta & & & \\ & & \delta^2 & & \\ & & & \ddots & \\ & & & & \delta^{k-1} \end{bmatrix}, \qquad D^{-1} = \begin{bmatrix} 1 & & & & \\ & \delta^{-1} & & & \\ & & \delta^{-2} & & \\ & & & \ddots & \\ & & & & \delta^{-(k-1)} \end{bmatrix}. \tag{C.14}$$

Then

$$D^{-1} J(\lambda, k) D = \lambda I_k + \delta S_k.$$

Note that left multiplying by D^{-1} multiplies the ith row by $\delta^{-(i-1)}$, while right multiplying by D multiplies the jth column by δ^{j-1}. On the diagonal the two effects cancel, while on the superdiagonal the net effect is to multiply each element by δ.

Similarity transformations of this nature are useful in other contexts as well. If this transformation is applied to an arbitrary matrix, then all elements on the pth diagonal will be multiplied by δ^p (with p positive for superdiagonals and negative for subdiagonals).

By applying this idea to each block in the Jordan canonical form with $\delta \ll 1$, we can find a matrix R so that $R^{-1}AR$ is close to diagonal with the 0 or δ at each location on the superdiagonal. This is done, for example, in the proof of Theorem C.4. But note that for $\delta < 1$ the condition number is $\kappa(D) = \delta^{1-k}$ and this blows up as $\delta \to 0$ if $k > 1$, so bringing a defective matrix to nearly diagonal form requires an increasingly ill-conditioned matrix R as the off-diagonals vanish. There is no nonsingular matrix R that will diagonalize A in the defective case.

C.4 Symmetric and Hermitian matrices

If $A \in \mathbb{R}^{m \times m}$ and $A = A^T$, then A is a *symmetric matrix*. Symmetric matrices arise naturally in many applications, in particular when discretizing "self-adjoint" differential equations. The complex analogue of the transpose is the *complex conjugate transpose* or *adjoint* matrix $A^H = \bar{A}^T$, in which the matrix is transposed and then the complex conjugate of each element taken. If A is a real matrix, then $A^H = A^T$. If $A = A^H$, then A is said to be *Hermitian* (so in particular a real symmetric matrix is Hermitian).

Hermitian matrices always have real eigenvalues and are always diagonalizable. Moreover, the eigenvectors $r_1, \ldots, r_m$ can be chosen to be mutually orthogonal, and normalized to have $r_j^H r_j = 1$, so that the eigenvector matrix R is a *unitary matrix*, $R^H R = I$, and hence $R^{-1} = R^H$. (If R is real and Hermitian, then $R^{-1} = R^T$ and R is called an *orthogonal matrix*.)

If $R = R^T$ and the eigenvalues of A are all positive, then A is said to be *symmetric positive definite* (SPD), or Hermitian positive definite in the complex case, or often simply "positive definite." In this case

$$u^H A u > 0 \tag{C.15}$$

for any vector $u \neq 0$.

This concept is generalized to the following: A is

positive definite	$\Longleftrightarrow$	$u^H A u > 0$ for all $u \neq 0$	$\Longleftrightarrow$	$\lambda_p > 0$ for all p,
positive semidefinite	$\Longleftrightarrow$	$u^H A u \geq 0$ for all $u \neq 0$	$\Longleftrightarrow$	$\lambda_p \geq 0$ for all p,
negative definite	$\Longleftrightarrow$	$u^H A u < 0$ for all $u \neq 0$	$\Longleftrightarrow$	$\lambda_p < 0$ for all p,
negative semidefinite	$\Longleftrightarrow$	$u^H A u \leq 0$ for all $u \neq 0$	$\Longleftrightarrow$	$\lambda_p \leq 0$ for all p,
indefinite	$\Longleftrightarrow$	$u^H A u$ indefinite	$\Longleftrightarrow$	$\lambda_p < 0 < \lambda_q$ for some p, q.

The proofs follow directly from the observation that

$$u^H A u = u^H R \Lambda R^H u = w^H \Lambda w = \sum_{i=1}^{m} \lambda^i |w_i|^2,$$

where $w = R^H u$.

C.5 Skew-symmetric and skew-Hermitian matrices

If $A = -A^T$, then A is said to be *skew-symmetric* (or skew-Hermitian in the complex case if $A = -A^H$). Matrices of this form also arise in discretizing certain types of differential equations (e.g., the advection equation as discussed in Chapter 10). Skew-Hermitian matrices are diagonalizable and have eigenvalues that are *pure imaginary*. This is a generalization of the fact that for a scalar λ, if $\bar{\lambda} = -\lambda$, then λ is pure imaginary. As in the Hermitian case, the eigenvectors of a skew-Hermitian matrix can be chosen so that the matrix R is unitary, $R^H R = I$.

C.6 Normal matrices

If A commutes with its adjoint, $AA^H = A^H A$, then A is said to be a *normal matrix*. In particular, Hermitian and skew-Hermitian matrices are normal. Any normal matrix is diagonalizable and R can be chosen to be unitary. Conversely, if A can be decomposed as

$$A = R \Lambda R^H$$

with $R^H = R^{-1}$ and Λ diagonal, then A is normal since $\Lambda \Lambda^H = \Lambda^H \Lambda$ for any diagonal matrix.

Eigenvalue analysis is particularly useful for normal matrices, since they can be diagonalized by a unitary matrix. A unitary matrix R satisfies $\|R\|_2 = \|R^{-1}\|_2 = 1$, and hence the behavior of powers of A is very closely related to the powers of the eigenvalues, for example. Nonnormal matrices can be harder to analyze, and in this case studying only the eigenvalues of A can be misleading. See Section D.4 for more discussion of this.

C.7 Toeplitz and circulant matrices

A matrix is said to be *Toeplitz* if the value along each diagonal is constant, e.g.,

$$A = \begin{bmatrix} d_0 & d_1 & d_2 & d_3 \\ d_{-1} & d_0 & d_1 & d_2 \\ d_{-2} & d_{-1} & d_0 & d_1 \\ d_{-3} & d_{-2} & d_{-1} & d_0 \end{bmatrix}$$

is a 4×4 example. Here we use d_i to denote the constant element along the ith diagonal.

If $d_1 = d_{-3}$, $d_2 = d_{-2}$, and $d_3 = d_{-1}$ in the above example, or more generally if $d_i = d_{i-m}$ for $i = 1, 2, \ldots, m-1$ in the $m \times m$ case, then the matrix is said to be *circulant*.

Toeplitz matrices naturally arise in the study of finite difference methods (see, e.g., Section 2.4) and it is useful to have closed-form expressions for their eigenvalues and eigenvectors. This is often possible because of their simple structure.

First consider a "tridiagonal" circulant matrix (which also has nonzero corner terms) of the form

$$A = \begin{bmatrix} d_0 & d_1 & & & & d_{-1} \\ d_{-1} & d_0 & d_1 & & & \\ & d_{-1} & d_0 & & & \\ & & & \ddots & & \\ & & & & & d_1 \\ d_1 & & & & d_{-1} & d_0 \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}. \tag{C.16}$$

Alternatively we could use the symbol d_m in place of d_{-1}. We take the dimension to be $m+1$ to be consistent with notation used in Chapter 2, since such matrices arise in studying 3-point difference equations on the unit interval with periodic boundary conditions. Then $h = 1/(m+1)$ is the mesh spacing between grid points and the unknowns are $U_1, \ldots, U_{m+1}$.

The pth eigenvalue of the matrix (C.16) is given by

$$\lambda_p = d_{-1}e^{-2\pi iph} + d_0 + d_1 e^{2\pi iph}, \tag{C.17}$$

where $i = \sqrt{-1}$, and the jth element of the corresponding eigenvector r_p is given by

$$r_{jp} = e^{2\pi ipjh}. \tag{C.18}$$

This is the (j, p) element of the matrix R that diagonalizes A. Once the form of the eigenvector has been "guessed," it is easy to compute the corresponding eigenvalue λ_p by computing the jth component of Ar_p and using the fact that

$$e^{2\pi ip(j\pm1)h} = e^{\pm 2\pi iph} e^{2\pi ipjh} \tag{C.19}$$

to obtain

$$(Ar_p)_j = (d_{-1}e^{-2\pi iph} + d_0 + d_1 e^{2\pi iph})r_{jp}.$$

The circulant structure is needed to verify that this formula also holds for $j = 1$ and $j = m + 1$, using $e^{2\pi i(m+1)h} = 1$.

The same vectors r_p with components (C.18) are the eigenvectors of any $(m + 1) \times (m + 1)$ circulant matrix with diagonals $d_0,\ d_1,\ \ldots,\ d_m$. It can be verified, as in the computation above, that the corresponding eigenvalue is

$$\lambda_p = \sum_{k=0}^{m} d_k e^{2\pi ipkh}. \tag{C.20}$$

In the "tridiagonal" example above we used the label d_{-1} instead of d_m, but note that $e^{-2\pi ih} = e^{2\pi imh}$, so the expression (C.20) is invariant under this change of notation.

Any constant coefficient difference equation with periodic boundary conditions gives rise to a circulant matrix of this form and has eigenvectors with components (C.18). Note that the jth component of r_p can be rewritten as

$$r_{jp} = e^{2\pi ipx_j} = \phi_p(x_j),$$

where $x_j = jh$ is the jth grid point and $\phi_p(x) = e^{2\pi ipx}$. The function $\phi_p(x)$ is the pth eigenfunction of the differentiation operator ∂_x on the unit interval with periodic boundary conditions,

$$\partial_x \phi_p(x) = (2\pi ip)\phi_p(x).$$

It is also the eigenfunction of any higher order derivative ∂_x^s, with eigenvalue $(2\pi ip)^s$. This is the basis of Fourier analysis of linear differential equations, and the fact that difference equations have eigenvectors that are discretized versions of $\phi_p(x)$ means that discrete Fourier analysis can be used to analyze finite difference methods for constant coefficient problems, as is done in von Neumann analysis; see Sections 9.6 and 10.5.

Now consider the symmetric tridiagonal Toeplitz matrix (now truly tridiagonal)

$$A = \begin{bmatrix} d_0 & d_1 & & & & & \\ d_1 & d_0 & d_1 & & & & \\ & d_1 & d_0 & d_1 & & & \\ & & & \ddots & & & \\ & & & & & d_1 & \\ & & & & d_1 & d_0 & \end{bmatrix} \in \mathbb{R}^{m\times m}. \tag{C.21}$$

Such matrices arise in 3-point discretizations of u_{xx} with Dirichlet boundary conditions, for example; see Section 2.4. The eigenvalues of A are now

$$\lambda_p = d_0 + 2d_1 \cos(p\pi h), \qquad p = 1,\ 2,\ \ldots,\ m, \tag{C.22}$$

where again $h = 1/(m + 1)$ and now A has dimension m since boundary values are not included in the solution vector. The eigenvector now has components

$$r_{jp} = \sin(p\pi jh), \qquad j = 1,\ 2,\ \ldots,\ m. \tag{C.23}$$

Again it is easy to verify that (C.22) gives the eigenvalue once the form of the eigenvector is known. In this case we use the fact that, for any p,

$$\sin(p\pi jh) = 0 \quad \text{for } j = 0 \text{ and } j = m+1$$

to verify that $(Ar_p)_j = \lambda_p r_{jp}$ for $j = 0$ and $j = m+1$ as well as in the interior.

Now consider a nonsymmetric tridiagonal Toeplitz matrix,

$$A = \begin{bmatrix} d_0 & d_1 & & & & & \\ d_{-1} & d_0 & d_1 & & & & \\ & d_{-1} & d_0 & d_1 & & & \\ & & & \ddots & & & \\ & & & & & d_1 & \\ & & & & d_{-1} & d_0 \end{bmatrix} \in \mathbb{R}^{m\times m}. \tag{C.24}$$

If $d_1 = d_{-1} = 0$, then the matrix is diagonal with all eigenvalues equal to d_0. Otherwise, if one of d_1 or d_{-1} is zero the eigenvalues are all equal to d_0 but the matrix is a single Jordan block, a defective matrix with a one-dimensional eigenspace.

In the general case where both d_1 and d_{-1} are nonzero, the eigenvalues are

$$\lambda_p = d_0 + 2d_1\sqrt{d_{-1}/d_1}\cos(p\pi h), \qquad p = 1,\ 2,\ \ldots,\ m, \tag{C.25}$$

and the corresponding eigenvector r_p has jth component

$$r_{jp} = \left(\sqrt{d_{-1}/d_1}\right)^j \sin(p\pi jh), \qquad j = 1,\ 2,\ \ldots,\ m. \tag{C.26}$$

These formulas hold also if d_{-1}/d_1 is negative, in which case the eigenvalues are complex. For example, the skew-symmetric centered difference matrix with $d_{-1} = -1$, $d_0 = 0$, and $d_1 = 1$ has eigenvalues

$$\lambda_p = 2i\cos(p\pi h). \tag{C.27}$$

C.8 The Gershgorin theorem

If A is diagonal, then its eigenvalues are simply the diagonal elements. If A is "nearly diagonal," in the sense that the off-diagonal elements are small compared to the diagonal, then we might expect the diagonal elements to be good approximations to the eigenvalues. The Gerschgorin theorem quantifies this and also provides bounds on the eigenvalues in terms of the diagonal and off-diagonal elements. These bounds are valid in general and often very useful even when A is far from diagonal.

Theorem C.2. *Let $A \in \mathbb{C}^{m\times m}$ and let D_i be the closed disk in the complex plane centered at a_{ii} with radius $r_i = \sum_{j\neq i} |a_{ij}|$, the sum of the magnitude of all the off-diagonal elements in the ith row of A,*

$$D_i = \{z \in \mathbb{C} :\ |z - a_{ii}| \le r_i\}.$$

Then,

1. *all the eigenvalues of A lie in the union of the disks D_i for $i = 1, 2, \ldots, m$.*
2. *if some set of k overlapping disks is disjoint from all the other disks, then exactly k eigenvalues lie in the union of these k disks.*

Note the following:

- If a disk D_i is disjoint from all other disks, then it contains exactly one eigenvalue of A.
- If a disk D_i overlaps other disks, then it need not contain any eigenvalues (although the union of the overlapping disks contains the appropriate number).
- If A is real, then A^T has the same eigenvalues as A. Then the theorem can also be applied to A^T (or equivalently the disk radii can be defined by summing elements of columns rather than rows).

For a proof of this theorem see Wilkinson [103], for example.

Example C.5. Let

$$A = \begin{bmatrix} 5 & 0.6 & 0.1 \\ -1 & 6 & -0.1 \\ 1 & 0 & 2 \end{bmatrix}.$$

Applying the Gershgorin theorem to A, we have

$$D_1 = \{z : |z - 5| \le 0.7\}, \qquad D_2 = \{z : |z - 6| \le 1.1\}, \qquad D_3 = \{z : |z - 2| \le 1.0\},$$

as shown in Figure C.1(a). From the theorem we can conclude that there is exactly one eigenvalue in D_3 and two eigenvalues in $D_1 \cup D_2$. We can also conclude that all eigenvalues have real parts between 1 and 7.7 (and hence positive real parts, in particular). The eigenvalue in D_3 must be real, since complex eigenvalues must appear in conjugate pairs (since A is real). The eigenvalues in $D_1 \cup D_2$ could be real or imaginary, but the imaginary part must be bounded by 1.1. The actual eigenvalues of A are also shown in Figure C.1(a), and are

$$\lambda \approx 1.9639,\ 5.518 \pm 0.6142i.$$

Applying the theorem to A^T would give

$$D_1 = \{z : |z - 5| \le 2.0\}, \qquad D_2 = \{z : |z - 6| \le 0.6\}, \qquad D_3 = \{z : |z - 2| \le 0.2\},$$

as shown in Figure C.1(b). Note that this gives a tighter bound on the eigenvalue near 2 but a larger region around the complex pair.

A matrix is said to be *reducible* if it is possible to reorder the rows and columns in such a way that the eigenvalue problem is decoupled into simpler problem, specifically if there exists a permutation matrix P so that

$$PAP^{-1} = \begin{bmatrix} A_{11} & 0 \\ A_{12} & A_{22} \end{bmatrix},$$

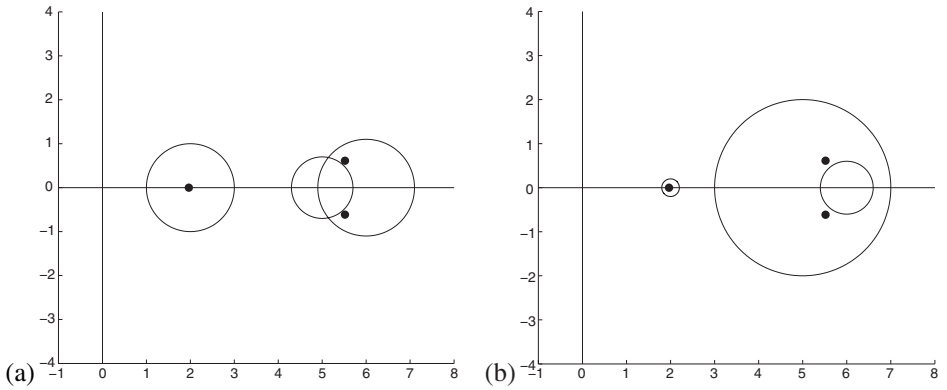

Figure C.1. *Gerschgorin circles containing the eigenvalues of A for Example* C.5.

where A_{11} and A_{22} are square matrices of size at least 1×1. In this case the eigenvalues of A consist of the eigenvalues of A_{11} together with those of A_{22}. If no such P exists, then A is *irreducible*. The matrix of Example C.5 is irreducible, for example.

For irreducible matrices, a more refined version of the Gerschgorin theorem states also that a point on the boundary of a set of Gerschgorin disks can be an eigenvalue only if it is on the boundary of *all* disks.

Example C.6. The tridiagonal matrix A of (2.10) arises from discretizing the second derivative. Since this matrix is symmetric all its eigenvalues are real. By the Gerschgorin theorem they must lie in the circle of radius $2/h^2$ centered at $-2/h^2$. In fact, they must lie in the interior of this disk since the matrix is irreducible and the first and last row of A give disks with radius $1/h^2$. Hence $-4/h^2 < \lambda_p < 0$ for all eigenvalues λ_p. In particular this shows that all the eigenvalues are negative and hence the matrix A is nonsingular and negative definite. Showing nonsingularity is one use of the Gerschgorin theorem.

For the tridiagonal matrix (2.10) the eigenvalues can be explicitly computed and are given by the formula (2.23),

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

where $h = 1/(m+1)$. They are distributed all along the interval $-4/h^2 < \lambda_p < 0$. For related matrices that arise from discretizing variable coefficient elliptic equations the matrices cannot be explicitly computed, but the Gerschgorin theorem can still be used to show nonsingularity.

Example C.7. Consider the matrix (2.73) with all $\kappa > 0$. The Gerschgorin disks all lie in the left half-plane and the disks D_1 and D_m are bounded away from the origin. The matrix is irreducible and hence must be negative definite (and in particular nonsingular).

C.9 Inner-product norms

Some standard vector norms and the corresponding matrix norms were introduced in Section A.3. Here we further investigate the 2-norm and its relation to the spectral radius of

a matrix. We will also see how new inner-product norms can be defined that are closely related to a particular matrix.

Let $A \in \mathbb{C}^{m\times m}$ and $u \in \mathbb{C}^m$. The 2-norm of of u is defined by

$$\|u\|_2^2 = u^H u = \sum_{i=1}^{m} |u_i|^2 = \langle u, u\rangle, \tag{C.28}$$

where $\langle\cdot,\cdot\rangle$ is the standard inner product,

$$\langle u, v\rangle = u^H v = \sum_{i=1}^{m} \bar{u}_i v_i. \tag{C.29}$$

The 2-norm of the matrix A is defined by the formula (A.9) as

$$\|A\|_2 = \sup_{\|u\|_2=1} \|Au\|_2 = \sup_{\|u\|_2=1} \left(u^H A^H A u\right)^{1/2}.$$

Note that if we choose u to be an eigenvector of A, with $Au = \lambda u$, then

$$(u^H A^H A u)^{1/2} = |\lambda|,$$

and so $\|A\|_2 \geq \max_{1\leq p\leq m} |\lambda_p| = \rho(A)$. The 2-norm of A is always at least as large as the spectral radius. Note that the matrix $B = A^H A$ is always Hermitian ($B^H = B$) and so it is diagonalizable with a unitary eigenvector matrix,

$$B = RMR^H \qquad (R^H = R^{-1}),$$

where M is the diagonal matrix of eigenvectors $\mu_j \geq 0$ of B. Any vector u can be written as $u = Rw$ where $w = R^H u$. Note that $\|u\|_2 = \|w\|_2$ since

$$u^H u = w^H R^H R w = w^H w,$$

i.e., multiplication by a unitary matrix preserves the 2-norm. It follows that

$$\begin{aligned} \|A\|_2 &= \sup_{\|u\|_2=1} (u^H B u)^{1/2} \\ &= \sup_{\|w\|_2=1} (w^H R^H B R w)^{1/2} \\ &= \sup_{\|w\|_2=1} (w^H M w)^{1/2} \\ &= \max_{p=1,2,\ldots,m} |\mu_p|^{1/2} = \sqrt{\rho(A^H A)}. \end{aligned} \tag{C.30}$$

If $A^H = A$, then $\rho(A^H A) = (\rho(A))^2$ and $\|A\|_2 = \rho(A)$. More generally this is true for any normal matrix A (as defined in Section C.6). If A is normal, then A and A^H have the same eigenvector matrix R and so

$$A^H A = (R\Lambda^H R)(R\Lambda R^H) = R\Lambda^H \Lambda R^H.$$

It follows that $\rho(A^H A) = \max_{p=1,2,\ldots,m} |\lambda_p|^2$.

If A is not normal, then typically $\|A\|_2 > \rho(A)$. If A is diagonalizable, then an upper bound on $\|A\|_2$ can be obtained from

$$\begin{aligned}\|A\|_2 &= \|R\Lambda R^{-1}\|_2 \\ &\le \|R\|_2\|R^{-1}\|_2 \max_{p=1,2,\ldots,m} |\lambda_p| = \kappa_2(R)\rho(A),\end{aligned} \tag{C.31}$$

where $\kappa_2(R) = \|R\|_2\|R^{-1}\|_2$ is the 2-norm condition number of the eigenvector matrix R. We thus have the general relation

$$\rho(A) \le \|A\|_2 \le \kappa_2(R)\rho(A), \tag{C.32}$$

which holds for any diagonalizable matrix A. If A is normal, then R is unitary and $\kappa_2(R) = 1$.

This relation between the norm and spectral radius is important in studying iterations of the form $U^{n+1} = AU^n$, which leads to $U^n = A^nU^0$ (where the superscript on U is an index and the superscript on A is a power). Such iterations arise both in time-stepping algorithms for solving differential equations and in iterative methods for solving linear systems. We often wish to investigate the behavior of $\|U^n\|$ as $n \to \infty$, or the related question of the behavior of powers of the matrix A. For diagonalizable A we have $A^n = R\Lambda^n R^{-1}$, so that

$$\|A^n\|_2 \le \kappa_2(R)(\rho(A))^n. \tag{C.33}$$

From this we see that $\|A^n\|_2 \to 0$ as $n \to \infty$ if $\rho(A) < 1$. In fact this is true for any A, not just diagonalizable matrices, as can be seen by using the Jordan canonical form. See Appendix D for more about bounding powers of a matrix.

This spectral analysis is particularly useful when A is normal, in which case $\kappa_2(R) = 1$. In this case $\|A^n\|_2 \le (\rho(A))^n$ and if $\rho(A) < 1$, then we have a strictly decreasing upper bound on the norm. The asymptotic behavior is still the same if A is not normal, but convergence is not necessarily monotone and this spectral analysis can be quite misleading if A is far from normal. This topic is discussed in more detail in Appendix D along with a discussion of the nondiagonalizable (defective) case.

C.10 Other inner-product norms

If T is any nonsingular matrix, then we can define an inner product based on T in terms of the standard inner product (C.29) by

$$\langle u, v\rangle_T = \langle T^{-1}u,\ T^{-1}v\rangle = u^H G v, \tag{C.34}$$

where $G = T^{-H}T^{-1}$. The matrix G is always Hermitian positive definite (SPD if T is real). We can define a corresponding norm (the T-norm of u) by

$$\|u\|_T = \langle u, u\rangle_T = \|T^{-1}u\|_2 = (u^H G u)^{1/2} = \langle u, Gu\rangle. \tag{C.35}$$

This satisfies the requirements of a norm summarized in Section A.3.

Inner-product norms of this type naturally arise in the study of conjugate gradient methods for solving linear system $Au = f$ when A is SPD. In this case $G = A$ is used

(see Section 4.3.4) and T could be defined as a "square root" of A, e.g., $T = R\Lambda^{1/2}R^{-1}$ if $A = R\Lambda R^{-1}$.

In studying iterations of the form $U^{n+1} = AU^n$, and variants such as $U^{n+1} = A_nU^n$ (where the matrix A_n changes in each iteration), it is often useful to choose norms that are adapted to the matrix or matrices in question in order to obtain more insight into the asymptotic behavior of U^n. A few results are summarized below that are used elsewhere.

Note that $w = T^{-1}u$ can be viewed as the vector of coefficients obtained if u is written as a linear combination of the columns of T, $u = Tw$. Hence $\|u\|_T = \|w\|_2$ can be viewed as a measure of u based on its representation in the coordinate system defined by T rather than in the standard basis vectors. A particularly useful coordinate system is the coordinates defined by the eigenvectors, as we will see below.

We can compute the matrix T-norm of a matrix A using the standard definition of a matrix norm from (A.9):

$$\begin{aligned}\|A\|_T &= \sup_{u\neq 0} \frac{\|Au\|_T}{\|u\|_T} = \sup_{w\neq 0} \frac{\|ATw\|_T}{\|Tw\|_T} \\ &= \sup_{w\neq 0} \frac{\|T^{-1}ATw\|_2}{\|w\|_2} \\ &= \|T^{-1}AT\|_2.\end{aligned} \tag{C.36}$$

Now suppose A is a diagonalizable matrix with $R^{-1}AR = \Lambda$. Then choosing $T = R$ yields $\|A\|_R = \|R^{-1}AR\|_2 = \rho(A)$. Recall that $\|A\| \geq \rho(A)$ in any matrix norm subordinate to a vector norm. We have just shown that equality can be achieved by an appropriate choice of norm in the case when A is diagonalizable. We have proved the following theorem.

Theorem C.3. *Suppose $A \in \mathbb{C}^{m\times m}$ is diagonalizable. Then there exists a norm $\|\cdot\|$ in which $\|A\| = \rho(A)$. The norm is given by the R-norm based on the eigenvector matrix.*

This theorem will be generalized to the defective case in Theorem C.4 below.

Note that in general the T-norm, for any nonsingular T, is "equivalent" to the 2-norm in the sense of Section A.3.1 with the equivalence inequalities

$$\|T\|_2^{-1}\|u\|_2 \leq \|u\|_T \leq \|T^{-1}\|_2\|u\|_2 \tag{C.37}$$

for the vector norm and

$$\kappa_2(T)^{-1}\|A\|_2 \leq \|A\|_T \leq \kappa_2(T)\|A\|_2 \tag{C.38}$$

for the matrix norm, where $\kappa_2(T)$ is the 2-norm condition number of T. Applying this last inequality in conjunction with Theorem C.3 gives

$$\rho(A) = \|A\|_R \leq \kappa_2(R)\|A\|_2, \tag{C.39}$$

which agrees with the bound (C.31) obtained earlier.

For general matrices $A \in \mathbb{C}^{m\times m}$ that are not necessarily diagonalizable, Theorem C.3 can be generalized to the following.

Theorem C.4. (a) *If $A \in \mathbb{C}^{m\times m}$ has no defective eigenvalues with modulus $\rho(A)$, then there exists a nonsingular matrix T such that*

$$\|A\|_T = \rho(A).$$

(b) *If A has defective eigenvalue(s) of modulus $\rho(A)$, then for every $\epsilon > 0$ there exists a matrix $T(\epsilon)$ such that*

$$\|A\|_{T(\epsilon)} < \rho(A) + \epsilon. \tag{C.40}$$

In the latter case we can find a norm in which $\|A\|$ is arbitrarily close to $\rho(A)$, but $T(\epsilon)$ becomes increasingly ill conditioned as $\epsilon \to 0$. The proof of this theorem is based on a modification of the Jordan canonical form in which the superdiagonal elements are made sufficiently small by the transformation discussed in Section C.3. Let $R^{-1}AR = J$ have the form (C.12), and let $Z = \{i : |\lambda_i| = \rho(A)\}$, the set of indices of the maximal eigenvalues. To prove part (a), if $i \in Z$, then $k_i = 1$ and $J_i = \lambda_i$ with $|\lambda_i| = \rho(A)$. In this case set $D_i = 1$. If $i \notin Z$, let $\delta_i = \rho(A) - |\lambda_i| > 0$ and set

$$D_i = \operatorname{diag}(1,\ \delta_i,\ \ldots,\ \delta_i^{k_i-1}).$$

Let D be the block diagonal matrix formed by these blocks. Then $\tilde{J} = D^{-1}JD$ has Jordan blocks $\lambda_i I + \delta_i S_{k_i}$ and so

$$\|\tilde{J}_i\|_2 \le |\lambda_i| + \delta_i \le \rho(A) \qquad \text{for } i \notin Z.$$

It follows that $\|A\|_T = \|\tilde{J}\|_2 = \rho(A)$, where the matrix A is given by $T = RD$.

To prove part (b), let $\epsilon > 0$ be given and choose

$$\delta_i = \begin{cases} \epsilon & \text{if } i \in Z, \\ \rho(A) - |\lambda_i| > 0 & \text{if } i \notin Z. \end{cases}$$

Define D_i and D as before and we will achieve

$$\begin{aligned} \|\tilde{J}_i\| &\le \rho(A) \ \text{ for } i \notin Z, \\ \|\tilde{J}_i\| &\le \rho(A) + \epsilon \ \text{ for } i \in Z, \end{aligned}$$

and so taking $T(\epsilon) = RD(\epsilon)$ gives $\|A\|_{T(\epsilon)} \le \rho(A) + \epsilon$. Recall that $\kappa_2(D(\epsilon)) \to \infty$ as $\epsilon \to 0$ and so $\kappa_2(T(\epsilon)) \to \infty$ as $\epsilon \to 0$ in case (b).

Appendix D

Matrix Powers and Exponentials

The first order scalar linear differential equation $u'(t) = au(t)$ has the solution $u(t) = e^{at}u(0)$ and so

$$\begin{aligned} |u(t)| \to 0 \text{ as } t \to \infty \text{ for any } u(0) &\iff \operatorname{Re}(a) < 0, \\ |u(t)| \text{ remains bounded as } t \to \infty \text{ for any } u(0) &\iff \operatorname{Re}(a) \le 0, \\ |u(t)| \to \infty \text{ as } t \to \infty \text{ for any } u(0) \neq 0 &\iff \operatorname{Re}(a) > 0. \end{aligned} \tag{D.1}$$

The first order scalar linear difference equation $U^{n+1} = bU^n$ has the solution $U^n = b^n U^0$ and so

$$\begin{aligned} |U^n| \to 0 \text{ as } n \to \infty \text{ for any } U^0 &\iff |b| < 1, \\ |U^n| \text{ remains bounded as } n \to \infty \text{ for any } U^0 &\iff |b| \le 1, \\ |U^n| \to \infty \text{ as } n \to \infty \text{ for any } U^0 \neq 0 &\iff |b| > 1. \end{aligned} \tag{D.2}$$

For these first order scalar equations the behavior is very easy to predict. The purpose of this appendix is to review the extension of these results to the vector case, for linear differential equations $u'(t) = Au(t)$ and difference equations $U^{n+1} = BU^n$, where $u(t),\ U^n \in \mathbb{R}^m$ and $A,\ B \in \mathbb{R}^{m\times m}$, or more generally could be complex valued. This analysis relies on a good understanding of the material in Appendix C on eigendecompositions of matrices, and of the Jordan canonical form for the more interesting defective case.

In the scalar case there is a close relation between the results stated above in (D.1) and (D.2) for $u' = au$ and $U^{n+1} = bU^n$, respectively. If we introduce a time step $\Delta t = 1$ and let $U^n = u(n) = e^{an}u(0)$ be the solution to the ordinary differential equation (ODE) at discrete times, then $U^{n+1} = bU^n$ where $b = e^a$. Since the function e^z maps the left half-plane to the unit circle, we have

$$\begin{aligned} |b| < 1 &\iff \operatorname{Re}(a) < 0, \\ |b| = 1 &\iff \operatorname{Re}(a) = 0, \\ |b| > 1 &\iff \operatorname{Re}(a) > 0. \end{aligned} \tag{D.3}$$

Similarly, for the system cases we will see that boundedness of the solutions depends on eigenvalues of B lying inside the unit circle, or eigenvalues of A lying in the left half-plane,

although the possibility of multiple eigenvalues makes the analysis somewhat more complicated. We will also see in Section D.4 that when the matrix involved is nonnormal the eigenvalues do not tell the full story—rapid transient growth in powers or the exponential can be observed even if there is eventual decay.

In the rest of this chapter we will use the symbol A as our general symbol in discussing both matrix powers and exponentials, but the reader should keep in mind that results on powers typically involve the unit circle, while results on exponentials concern the left half-plane.

D.1 The resolvent

Several of the bounds discussed below involve the resolvent of a matrix A, which is the complex matrix-valued function $(zI - A)^{-1}$. The domain of this function is the complex plane minus the eigenvalues of A, since the matrix $zI - A$ is noninvertible at these points. Bounds on the growth of A^n or e^{At} can be derived based on the size of $\|(zI - A)^{-1}\|$ over suitably chosen regions of the complex plane. These are generally obtained using the following representation of a function $f(A)$, the natural extension of the Cauchy integral formula to matrices.

Definition D.1. *If Γ is any closed contour in the complex plane that encloses the eigenvalues of A and if $f(z)$ is an analytic function within Γ, then the matrix $f(A)$ can be defined by*

$$f(A) = \frac{1}{2\pi i} \int_\Gamma f(z)(zI - A)^{-1}\, dz. \tag{D.4}$$

The value is independent of the choice of Γ.

For some functions $f(z)$, such as e^z, it may not be clear what $f(A)$ means for a matrix, and this can be used as the definition of $f(A)$. (Some other definitions of e^A are given in Section D.3, which are equivalent.) The representation (D.4) can also be useful computationally and is one approach to computing the matrix exponential; see, e.g., [53] and Section 11.6.1.

For the function $f(z) = z^n$ (or any polynomial in z) the meaning of $f(A)$ is clear since we know how to compute powers of a matrix, but (D.4) gives a different representation of the function that is sometimes useful in obtaining upper bounds.

D.2 Powers of matrices

Consider the linear difference equation

$$U^{n+1} = AU^n \tag{D.5}$$

for some iteration matrix A. The study of the asymptotic behavior of $\|U^n\|$ is important both in studying stability of finite difference methods and in studying convergence of iterative method for solving linear systems.

From (D.5) we obtain

$$\|U^{n+1}\| \le \|A\|\, \|U^n\|$$

in any norm and hence

$$\|U^n\| \le \|A\|^n \|U^0\|. \tag{D.6}$$

Alternatively, we can start with $U^{n+1} = A^n U^0$ to obtain

$$\|U^n\| \le \|A^n\| \, \|U^0\|. \tag{D.7}$$

Since $\|A^n\| \le \|A\|^n$ this again leads to (D.6), but in many cases $\|A^n\|$ is much smaller than $\|A\|^n$ and the form (D.7) may be more powerful.

If there exists any norm in which $\|A\| < 1$, then (D.6) shows that $\|U^n\| \to 0$ as $n \to \infty$. This is true not only in this particular norm but also in any other norm. Recall we are only considering matrix norms that are subordinate to some vector norm and that in a finite dimensional space of fixed dimension m all such norms are equivalent in the sense of (A.6). From these inequalities it follows that if $\|U^n\| \to 0$ in any norm, then it goes to zero in every equivalent norm, and similarly if $\|U^n\|$ blows up in some norm, then it blows up in every equivalent norm. Moreover, if $\|A\| = 1$ in some norm, then $\|U^n\|$ remains uniformly bounded as $n \to \infty$ in any equivalent norm.

From Theorem C.4 we thus obtain directly the following results.

Theorem D.1. (a) *Suppose* $\rho(A) < 1$. *Then* $\|U^n\| \to 0$ *as* $n \to \infty$ *in any vector norm.* (b) *Suppose* $\rho(A) = 1$ *and* A *has no defective eigenvalues of modulus* 1. *Then* $\|U^n\|$ *remains bounded as* $n \to \infty$ *in any vector norm, and there exists a norm in which* $\|U^n\| \le \|U^0\|$.

Note that result (a) holds even if A has defective eigenvalues of modulus $\rho(A)$ by choosing $\epsilon < 1 - \rho(A)$ in Theorem C.4(b).

If A is diagonalizable, then the results of Theorem D.1 can also be obtained directly from the eigendecomposition of A. Write $A = R\Lambda R^{-1}$, where

$$\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$$

is the diagonal matrix of eigenvalues of A and R is the matrix of right eigenvectors of A. Then the nth power of A is given by $A^n = R\Lambda^n R^{-1}$ and

$$\begin{aligned} \|U^n\| &\le \|A^n\| \, \|U^0\| \\ &\le \kappa(R)\rho(A)^n \|U^0\|, \end{aligned} \tag{D.8}$$

where $\kappa(R) = \|R\| \, \|R^{-1}\|$ is the condition number of R in whatever norm we are using. Note that if the value of $\kappa(R)$ is large, then $\|U^n\|$ could grow to large values before decaying, even in $\rho(A) < 1$ (see Example D.2).

If A is a normal matrix and we use the 2-norm, then $\kappa_2(R) = 1$ and we have the nice result that

$$\|U^n\|_2 \le \rho(A)^n \|U^0\|_2 \qquad \text{if } A \text{ is normal.} \tag{D.9}$$

For normal matrices, or for those close to normal, $\rho(A)$ gives a good indication of the behavior of $\|U^n\|_2$. For matrices that are far from being normal, the spectral radius may give a poor indication of how $\|U^n\|$ behaves. The nonnormal case is considered further in Section D.4.

More detailed information about the behavior of $\|U^n\|$ can be obtained by decomposing U^0 into eigencomponents. Still assuming A is diagonalizable, we can write

$$U^0 = W_1^0 r_1 + W_2^0 r_2 + \cdots + W_m^0 r_m = RW^0,$$

where $r_1, \ldots, r_m$ are the eigenvectors of A and the vector W^0 is given by $W^0 = R^{-1}U^0$. Multiplying U^0 by A multiplies each r_p by λ_p and so

$$U^n = A^n U^0 = W_1^0 \lambda_1^n r_1 + W_2^0 \lambda_2^n r_2 + \cdots + W_m^0 \lambda_m^n r_m = R\Lambda^n W^0. \tag{D.10}$$

For large n this is dominated by the terms corresponding to the largest eigenvalues, and hence the norm of this vector is proportional to $\rho(A)^n$, at least for generic initial data. This also shows that if $\rho(A) < 1$, then $U^n \to 0$, while if $\rho(A) > 1$ we expect U^n to blow up.

Note that for certain initial data $\|U^n\|$ may behave differently than $\rho(A)^n$, at least in exact arithmetic. For example, if U^0 is void in the dominant eigencomponents, then these terms will be missing from (D.10), and the asymptotic growth or decay rate will be different. In particular, it could happen that $\rho(A) > 1$ and yet $\|U^n\| \to 0$ for special data U^0 if some eigenvalues of A have modulus less than 1 and U^0 contains only these eigencomponents. However, this generally is not relevant for the stability and convergence issues considered in this book, where arbitrary initial data must be considered. Moreover, even if U^0 is void of some eigencomponents, rounding errors introduced computationally in each iteration $U^{n+1} = AU^n$ will typically be random and will contain all eigencomponents. The growing modes may start out at the level of rounding error, but if $\rho(A) > 1$ they will grow exponentially and eventually dominate the solution so that the asymptotic behavior will still be governed by $\rho(A)^n$.

If A is defective, then we cannot express an arbitrary initial vector U^0 as a linear combination of eigenvectors. However, using the Jordan canonical form $A = RJR^{-1}$, we can still write $U^0 = RW^0$, where $W^0 = R^{-1}U^0$. The nonsingular matrix R now contains principal vectors as well as eigenvectors, as discussed in Section C.3.

Example D.1. Consider the iteration $U^{n+1} = AU^n$, where A is the 3×3 matrix from Example C.4, having a single eigenvalue λ with geometric multiplicity 1. If we decompose

$$U^0 = W_1^0 r_1 + W_2^0 r_2 + W_3^0 r_3,$$

then multiplying by A gives

$$\begin{aligned} U^1 &= W_1^0 \lambda r_1 + W_2^0 (r_1 + \lambda r_2) + W_3^0 (r_2 + \lambda r_3) \\ &= (W_1^0 \lambda + W_2^0) r_1 + (W_2^0 \lambda + W_3^0) r_2 + W_3^0 \lambda r_3. \end{aligned} \tag{D.11}$$

Repeating this shows that U^n has the form

$$U^n = A^n U^0 = p_1(\lambda) r_1 + p_2(\lambda) r_2 + W_3^0 \lambda^n r_3,$$

where $p_1(\lambda)$ and $p_2(\lambda)$ are polynomials in λ of degree n. It can be shown that

$$|p_1(\lambda)| \le n^2 \lambda^n \|W^0\|_2, \qquad |p_2(\lambda)| \le n\lambda^n \|W^0\|_2,$$

so that there are now algebraic terms (powers of n) in the asymptotic behavior in addition to the exponential terms (λ^n). More generally, as we will see below, a Jordan block of order k gives rise to terms of the form $n^{k-1}\lambda^n$.

If $|\lambda| > 1$, then the power n^{k-1} is swamped by the exponential growth and the algebraic term is unimportant. If $|\lambda| < 1$, then n^{k-1} grows algebraically but λ^n decays exponentially and the product decays, $n^{k-1}\lambda^n \to 0$ as $n \to \infty$ for any $k > 1$.

The borderline case $|\lambda| = 1$ is where this algebraic term makes a difference. In this case λ^n remains bounded but $n^{k-1}\lambda^n$ does not. Note how this relates to the results of Theorem D.1. If $\rho(A) < 1$, then $\|A^n\| \to 0$ even if A has defective eigenvalues of modulus $\rho(A)$, since the exponential decay overpowers the algebraic growth. However, if $\rho(A) = 1$, then the boundedness of $\|A^n\|$ depends on whether there are defective eigenvalues of modulus 1. If so, then $\|A^n\|$ grows algebraically (but not exponentially).

Recall also from Theorem C.4 that in this latter case we can find, for any $\epsilon > 0$, a norm in which $\|A\| < 1 + \epsilon$. This implies that $\|A^n\| < (1 + \epsilon)^n$. There may be growth, but we can make the exponential growth rate arbitrarily slow. This is consistent with the fact that in this case we have only algebraic growth. Exponential growth at rate $(1 + \epsilon)^n$ eventually dominates algebraic growth n^{k-1} no matter how small ϵ is, for any k.

To determine the algebraic growth factors for the general case of a defective matrix, we can use the fact that if $A = RJR^{-1}$ then $A^n = RJ^nR^{-1}$, and we can compute the nth power of the Jordan matrix J. Recall that J is a block diagonal matrix with Jordan blocks on the diagonal, and powers of J simply consist of powers of these blocks. For a single Jordan block (C.9) of order k,

$$J(\lambda, k) = \lambda I_k + S_k,$$

where S_k is the shift matrix (C.10). Powers of $J(\lambda, k)$ can be found using the binomial expansion and the fact that I_k and S_k commute,

$$\begin{aligned} J(\lambda, k)^n &= (\lambda I_k + S_k)^n \\ &\quad \lambda^n I_k + n\lambda^{n-1} S_k + \binom{n}{2} \lambda^{n-2} S_k^2 + \binom{n}{3} \lambda^{n-3} S_k^3 \\ &\quad + \cdots + \binom{n}{n-1} \lambda S_k^{n-1} + S_k^n. \end{aligned} \tag{D.12}$$

Note that for $j < k$, S_k^j is the matrix with 1's along the jth superdiagonal and zeros everywhere else. For $j \geq k$, S_k^j is the zero matrix. So the series in expression (D.12) always terminates after at most k terms, and when $n > k$ reduces to

$$\begin{aligned} J(\lambda, k)^n = \lambda^n I_k + n\lambda^{n-1} S_k + \binom{n}{2} \lambda^{n-2} S_k^2 + \binom{n}{3} \lambda^{n-3} S_k^3 \\ + \cdots + \binom{n}{k-1} \lambda^{n-k+1} S_k^{k-1}. \end{aligned} \tag{D.13}$$

Since $\binom{n}{j} = O(n^j)$ as $n \to \infty$, we see that $J(\lambda, k)^n = P(n)\lambda^n$, where $P(n)$ is a matrix-valued polynomial of degree $k - 1$.

For example, returning to Example C.4, where $k = 3$, we have

$$J(\lambda, 3)^n = \lambda^n I_3 + n\lambda^{n-1} S_3 + \frac{n(n-1)}{2}\lambda^{n-2} S_3^2$$
$$= \begin{bmatrix} \lambda^n & n\lambda^{n-1} & \frac{n(n-1)}{2}\lambda^{n-2} \\ 0 & \lambda^n & n\lambda^{n-1} \\ 0 & 0 & \lambda^n \end{bmatrix}. \tag{D.14}$$

This shows that $\|J^n\| \approx n^2\lambda^n$, the same result obtained in Example D.1,

If A is not normal, i.e., if $A^H A \neq AA^H$, then $\|A\|_2 > \rho(A)$ and $\rho(A)^n$ may not give a very good indication of the behavior of $\|A^n\|$. If A is diagonalizable, then we have

$$\|A^n\|_2 \leq \|R\|_2 \|\Lambda^n\|_2 \|R^{-1}\|_2 \leq \kappa_2(R)\rho(A)^n, \tag{D.15}$$

but if $\kappa_2(R)$ is large, then this may not be useful, particularly for smaller n. This does give information about the asymptotic behavior as $n \to \infty$, but in practice it may tell us little or nothing about how $\|A^n\|_2$ is behaving for the finite values of n we care about in a particular computation. See Section D.4 for more about the nonnormal case.

D.2.1 Solving linear difference equations

Matrix powers arise naturally when iterating with the first order linear difference equation (D.5). In studying linear multistep methods we need the general solution to an rth order linear difference equation of the form

$$a_0 V^n + a_1 V^{n+1} + a_2 V^{n+2} + \cdots + V^{n+r} = 0 \quad \text{for } n \geq 0. \tag{D.16}$$

We have normalized the equation by assuming $a_r = 1$. One approach to solving this is given in Section 6.4.1, and here we give an alternative based on converting this rth order equation into a first order system of r equations of the form (D.5) and then applying the results just found for matrix powers.

We can rewrite (D.16) as a system of equations by introducing

$$U_1^n = V^n, \quad U_2^n = V^{n+1}, \quad \ldots, \quad U_r^n = V^{n+r-1}. \tag{D.17}$$

Then (D.16) takes the form

$$U^{n+1} = CU^n, \tag{D.18}$$

where

$$C = \begin{bmatrix} 0 & 1 & & & & \\ & 0 & 1 & & & \\ & & 0 & 1 & & \\ & & & \ddots & \ddots & \\ & & & & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & & -a_{r-1} \end{bmatrix}. \tag{D.19}$$

This matrix is called the *companion matrix* for the difference equation. The general solution to (D.18) is $U^n = C^n U^0$, where

$$U^0 = [V^0,\ V^1,\ \ldots,\ V^{r-1}]^T$$

is the vector consisting of the r initial values required by (D.16). The solution can be expressed in terms of the eigenvalues of C.

It can be shown that the characteristic polynomial of C is

$$p(\lambda) = \det(\lambda I - C) = a_0 + a_1\lambda + \cdots + a_{r-1}\lambda^{r-1} + \lambda^r. \tag{D.20}$$

Call the roots of this polynomial $\zeta_1, \ \ldots, \ \zeta_r$, as in Section 6.4.1. Let $C = RJR^{-1}$ be the Jordan canonical form of the matrix C. Then $U^n = RJ^nR^{-1}U^0$ and each element of U^n (and in particular $V^n = U_1^n$) is a linear combination of elements of J^n.

If the roots are distinct, then C is diagonalizable, and so the general solution of (D.16) is

$$V^n = c_1\zeta_1^n + c_2\zeta_2^n + \cdots + c_r\zeta_r^n, \tag{D.21}$$

where the coefficients c_j depend on the initial data.

It can be shown that repeated eigenvalues of a companion matrix always have geometric multiplicity 1, regardless of their algebraic multiplicity. An eigenvalue ζ_j has a one-dimensional space of eigenvectors spanned by $[1, \ \zeta_j, \ \zeta_j^2, \ \ldots, \ \zeta_j^{r-1}]^T$. From the form (D.13) for powers of a Jordan block, we see that if ζ_j is a repeated root of algebraic multiplicity k, then the general solution of the difference equation (D.16) includes terms of the form $\zeta_j^n, \ n\zeta_j^n, \ \ldots, \ n^{k-1}\zeta_j^n$. We can conclude that the general solution to the rth order difference equation (D.16) will be bounded as $n \to \infty$ only if the roots of the characteristic polynomial all lie inside the unit circle, with no repeated roots of magnitude 1. This is known as the *root condition* and is used in the stability analysis of linear multistep methods; see Definition 6.2.

D.2.2 Resolvent estimates

In our analysis of powers of A we have used the eigenstructure of A. An alternative approach is to use the resolvent $(zI - A)^{-1}$ and the expression (D.4) for the function $f(z) = z^n$. For example, we can obtain an alternative proof of part (a) of Theorem D.1 as follows. If $\rho(A) < 1$, then we choose as our contour Γ a circle of radius $1-\epsilon$, where $\epsilon > 0$ is chosen small enough that $\rho(A) < 1 - \epsilon$. The eigenvalues of A then lie inside Γ and $zI - A$ is invertible for z on Γ and so $\|(zI - A)^{-1}\|$ is a bounded periodic function of z and hence attains some maximum value $C(A, \Gamma)$ on Γ. Now consider (D.4) with $f(z) = z^n$,

$$A^n = \frac{1}{2\pi i}\int_\Gamma z^n(zI - A)^{-1}\, dz. \tag{D.22}$$

Taking norms and using the fact that $|z^n| = (1-\epsilon)^n$ and $\|(zI - A)^{-1}\| \le C(A, \Gamma)$ for z on Γ, and that Γ has length $2\pi(1-\epsilon) < 2\pi$, we obtain

$$\begin{aligned} \|A^n\| &\le \frac{1}{2\pi}\int_\Gamma |z^n|\, \|(zI - A)^{-1}\|\, dz \\ &< C(A, \Gamma)(1-\epsilon)^n. \end{aligned} \tag{D.23}$$

Since $(1-\epsilon)^n \to 0$ as $n \to \infty$ this proves Theorem D.1(a).

Note that we also get a uniform bound on $\|A^n\|$ that holds for all n,

$$\|A^n\| < C(A, \Gamma).$$

If A is normal then in the 2-norm we have $\|A^n\|_2 \le 1$ from (D.9), but for nonnormal matrices there can be transient growth in $\|A^n\|$ before it eventually decays, and the resolvent gives one approach to bounding this potential growth

Note that the value $C(A, \Gamma)$ depends on the matrix A and may be very large if $\rho(A)$ is close to 1. The contour Γ must lie between the spectrum of A and the unit circle for the argument above, and $\|(zI - A)^{-1}\|$ is large near the spectrum of A. In the study of numerical methods we are often concerned not just with a single matrix A but with a family of matrices arising from different discretizations, and proving stability results often requires proving *uniform power boundedness* of the family, i.e., that there is a single constant C such that $\|A^n\| \le C$ for all matrices in the family. This is more difficult than proving that a single matrix is power bounded, and it is the subject of the Kreiss matrix theorem discussed in Section D.6.

Note that this resolvent proof does not extend directly to prove part (b) of Theorem D.1, the case where A has nondefective eigenvalues on the unit circle. In this case the contour Γ must lie outside the unit circle, at least near these eigenvalues, for (D.22) to hold. In this case it is necessary to investigate how the product

$$(|z| - 1)\|(zI - A)^{-1}\| \tag{D.24}$$

behaves for $|z| > 1$. Here the resolvent norm is multiplied by a factor that vanishes as $|z| \to 1$ and so there is hope that the product will be bounded even if the resolvent is blowing up.

In fact it can be shown that if $\rho(A) < 1$ or if $\rho(A) = 1$ with only nondefective eigenvalues on the unit circle (i.e., if either of the conditions of Theorem D.1 holds), then the product (D.24) will be uniformly bounded for all z outside the unit circle. The supremum is called the *Kreiss constant* for the matrix A, denoted by $\mathcal{K}(A)$,

$$\mathcal{K}(A) = \sup_{|z|>1} (|z| - 1)\|(zI - A)^{-1}\|. \tag{D.25}$$

An alternative definition based on the ϵ-pseudospectral radius is discussed in Section D.5.

For an idea of how this can be used, again consider the first line of (D.23) but now take Γ to be a circle of radius $1 + \epsilon$ with $\epsilon > 0$. We have $\|(zI - A)^{-1}\| \le \mathcal{K}(A)/\epsilon$ on this circle, and so

$$\begin{aligned} \|A^n\| &\le \frac{1}{2\pi} \int_\Gamma |1 + \epsilon|^n \frac{\mathcal{K}(A)}{\epsilon}\, dz \\ &= \frac{1}{\epsilon}(1 + \epsilon)^{n+1}\mathcal{K}(A), \end{aligned} \tag{D.26}$$

since the circle has radius $2\pi(1 + \epsilon)$. This bound holds for any $\epsilon > 0$. It appears to allows exponential growth for any fixed ϵ, but we are free to choose a different value of ϵ for each n, and taking $\epsilon = 1/(n + 1)$, for example, gives

$$\|A^n\| \le (n + 1)e\mathcal{K}(A) \qquad \text{for all } n \ge 0.$$

Unfortunately, this still allows algebraic growth and so does not prove the theorem.

It fact, it can be shown (by a more complicated argument based on the resolvent that will not be presented here) that a bound of the required form holds,

$$\|A^n\| \le me\mathcal{K}(A) \qquad \text{for all } n \ge 0, \tag{D.27}$$

where m is the dimension of the matrix. This shows that for a fixed matrix A all powers remain bounded provided its Kreiss constant is finite (which in turn is true if and only if $\rho(A) \le 1$ with no defective eigenvalues on the unit circle).

The bound in (D.27) is sharp in a sense made precise in Section 18 of [92] and was the end product of a long sequence of weaker bounds proved over the years (as recounted in [92]). This result was proved by Spijker [81] as a corollary to a more general result on the arclength of the image of the unit circle under a rational function.

Resolvent estimates can also be used to obtain a *lower bound* on the transient growth of $\|A^n\|$; see (D.46).

The bound (D.27) is a major part of the proof of the Kreiss matrix theorem (see Section D.6): a family of matrices is uniformly power bounded if (and only if) there is a uniform bound on the Kreiss constants $\mathcal{K}(A)$ of all matrices in the family.

D.3 Matrix exponentials

Now consider the linear system of m ordinary differential equations $u'(t) = Au(t)$, where $A \in \mathbb{R}^{m\times m}$ (or more generally $A \in \mathbb{C}^{m\times m}$). The nondiagonalizable (defective) case will be considered in Section D.4. When A is diagonalizable we can solve this system by changing variables to $v = R^{-1}u$ and multiplying both sides of the ODE by R^{-1} to obtain

$$R^{-1}u'(t) = R^{-1}AR \cdot R^{-1}u(t)$$

or

$$v'(t) = \Lambda v(t).$$

This is a decoupled set of m scalar equations $v_j'(t) = \lambda_j v_j(t)$ (for $j = 1, 2, \ldots, m$) with solutions $v_j(t) = e^{\lambda_j t}v_j(0)$. Let $e^{\Lambda t}$ denote the matrix

$$e^{\Lambda t} = \operatorname{diag}(e^{\lambda_1 t}, e^{\lambda_2 t}, \ldots, e^{\lambda_m t}). \tag{D.28}$$

Then we have $v(t) = e^{\Lambda t}v(0)$ and hence

$$u(t) = Rv(t) = Re^{\Lambda t}R^{-1}u(0),$$

so

$$u(t) = e^{At}u(0), \tag{D.29}$$

where

$$e^{At} = Re^{\Lambda t}R^{-1}. \tag{D.30}$$

This gives one way to define the matrix exponential e^{At}, at least in the diagonalizable case. The Cauchy integral of Definition D.1 is another. Yet another way to define it is by the Taylor series

$$e^{At} = I + At + \frac{1}{2!}A^2t^2 + \frac{1}{3!}A^3t^3 + \cdots = \sum_{j=0}^{\infty}\frac{1}{j!}A^jt^j. \tag{D.31}$$

This is often useful, particularly when t is small. The definitions (D.30) and (D.31) agree in the diagonalizable case since all powers A^j have the same eigenvector matrix R, and so (D.31) gives

$$\begin{aligned} e^{At} &= R\left[I + \Lambda t + \frac{1}{2!}\Lambda^2 t^2 + \frac{1}{3!}\Lambda^3 t^3 + \cdots\right] R^{-1} \\ &= \mathrm{R}\, e^{\Lambda t} R^{-1}, \end{aligned} \tag{D.32}$$

resulting in (D.30). To go from the first to the second line of (D.32), note that it is easy to verify that the Taylor series applied to the diagonal matrix Λ is a diagonal matrix of Taylor series, each of which converge to the corresponding diagonal element of $e^{\Lambda t}$, the value $e^{\lambda_j t}$.

Note that the matrix e^{At} has the same eigenvectors as A and its eigenvalues are $e^{\lambda_j t}$. To investigate the behavior of $u(t) = e^{At}u(0)$ as $t \to \infty$, we need only look at the real part of each eigenvalue λ_j. If none of these are greater than 0, then the solution will remain bounded as $t \to \infty$ (assuming still that A is diagonalizable) since $|e^{\lambda_j t}| \le 1$ for all j.

It is useful to introduce the *spectral abscissa* $\alpha(A)$, defined by

$$\alpha(A) = \max_{1 \le j \le m} \mathrm{Re}(\lambda_j). \tag{D.33}$$

Then $u(t)$ remains bounded provided $\alpha(A) \le 0$ and $u(t) \to 0$ as $t \to \infty$ if $\alpha(A) < 0$.

Note that for integer values of $t = n$, we have $e^{An} = (e^A)^n$ and $\rho(e^A) = e^{\alpha(A)}$, so this result is consistent with what was found in the last section for matrix powers.

If A is not diagonalizable, then the case $\alpha(A) = 0$ is more subtle, as is the case $\rho(A) = 1$ for matrix powers. If A has a defective eigenvalue λ with $\mathrm{Re}(\lambda) = 0$, then the solution may still grow, although with polynomial growth in t rather than exponential growth.

When A is not diagonalizable, we can still write the solution to $u' = Au$ as $u(t) = e^{At}u(0)$, but we must reconsider the definition of e^{At}. In this case the Jordan canonical form $A = RJR^{-1}$ can be used, yielding

$$e^{At} = \mathrm{R} e^{Jt} R^{-1}.$$

If J has the block structure (C.12) then e^{Jt} is also block diagonal,

$$e^{Jt} = \begin{bmatrix} e^{J(\lambda_1,k_1)t} & & & \\ & e^{J(\lambda_2,k_2)t} & & \\ & & \ddots & \\ & & & e^{J(\lambda_s,k_s)t} \end{bmatrix}. \tag{D.34}$$

For a single Jordan block the Taylor series expansion (D.31) can be used in conjunction with the expansion (D.12) for powers of the Jordan block. We find that

$$
\begin{aligned}
e^{J(\lambda,k)t} &= \sum_{j=0}^{\infty} \frac{t^j}{j!}\left[\lambda^j I + j\lambda^{j-1}S_k + \binom{j}{2}\lambda^{j-2}S_k^2 + \cdots + \binom{j}{j-1}\lambda S_k^{j-1} + S_k^j\right] \\
&= \sum_{j=0}^{\infty} \frac{t^j}{j!}\lambda^j I + t\sum_{j=1}^{\infty} \frac{t^{j-1}}{(j-1)!}\lambda^{j-1}S_k + \frac{t^2}{2!}\sum_{j=2}^{\infty} \frac{t^{j-2}}{(j-2)!}\lambda^{j-2}S_k^2 + \cdots \\
&= e^{\lambda t}I + te^{\lambda t}S_k + \frac{t^2}{2!}e^{\lambda t}S_k^2 + \cdots + \frac{t^{(k-1)}}{(k-1)!}e^{\lambda t}S_k^{k-1} \\
&= \begin{bmatrix}
e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \cdots & & \frac{t^{(k-1)}}{(k-1)!}e^{\lambda t} \\
 & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \cdots & \\
 & & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \\
 & & & \ddots & \ddots & \\
 & & & & & \\
 & & & & & e^{\lambda t}
\end{bmatrix}.
\end{aligned}
\tag{D.35}
$$

Here we have used the fact that

$$
\frac{1}{j!}\binom{j}{p} = \frac{1}{p!}\frac{1}{(j-p)!}
$$

and the fact that $S_k^q = 0$ for $q \geq k$. The $k \times k$ matrix $e^{J(\lambda,k)t}$ is an upper triangular Toeplitz matrix with elements $d_0 = e^{\lambda t}$ on the diagonal and $d_j = \frac{t^j}{j!}e^{\lambda t}$ on the jth superdiagonal for $j = 1, 2, \ldots, k-1$.

We see that the situation regarding boundedness of e^{At} is exactly analogous to what we found for matrix powers A^n. If $\mathrm{Re}(\lambda) < 0$, then $t^j e^{\lambda t} \to 0$ despite the t^j factor, but if $\mathrm{Re}(\lambda) = 0$, then $t^j e^{\lambda t}$ grows algebraically. We obtain the following theorem, analogous to Theorem D.1.

Theorem D.2. *Let $A \in \mathbb{C}^{m\times m}$ be an arbitrary square matrix, and let $\alpha(A) = \max \mathrm{Re}(\lambda)$ be the spectral abscissa of A. Let $u(t) = e^{At}u(0)$ solve $u'(t) = Au(t)$. Then*

(a) *if $\alpha(A) < 0$, then $\|u(t)\| \to 0$ as $t \to \infty$ in any vector norm.*

(b) *if $\alpha(A) = 0$ and A has no defective eigenvalues with $\mathrm{Re}(\lambda) = 0$, then $\|u(t)\|$ remains bounded in any norm, and there exists a vector norm in which $\|u(t)\| \leq \|u(0)\|$ for all $t \geq 0$.*

If A is normal, then

$$
\|e^{At}\|_2 = \|e^{\Lambda t}\|_2 = e^{\alpha(A)t} \tag{D.36}
$$

since $\|R\|_2 = 1$. In this case the spectral abscissa gives precise information on the behavior of e^{At}. If A is nonnormal, then the behavior of $e^{\alpha(A)t}$ may not give a good indication of the behavior of e^{At} (except asymptotically for t sufficiently large), just as $\rho(A)^n$ may not give a good indication of how powers $\|A^n\|$ behave if A is not normal. See Section D.4 for some discussion of this case.

D.3.1 Solving linear differential equations

In Section D.2.1 we saw that the rth order linear difference equation (D.16) can be rewritten as a first order system (D.18) and solved using matrix powers. A similar approach can be used to convert the homogeneous rth order constant coefficient linear differential equation

$$a_0 v(t) + a_1 v'(t) + a_2 v''(t) + \cdots + a_{r-1} v^{(r-1)}(t) + v^{(r)}(t) = 0 \tag{D.37}$$

to a first order system of r equations and solve this using the matrix exponential. We follow the approach of Example 5.1 and introduce

$$u_1(t) = v(t), \quad u_2(t) = v'(t), \ldots, u_r(t) = v^{(r-1)}(t). \tag{D.38}$$

The equation (D.37) becomes $u'(t) = Cu(t)$, where C is the companion matrix (D.19). The solution is

$$u(t) = e^{Ct}u(0) = \mathrm{R}e^{Jt}R^{-1}u(0),$$

where $u(0)$ is the initial data (consisting of v and its first $r-1$ derivatives at time $t = 0$), and $C = RJR^{-1}$ is the Jordan canonical form of C. If the roots of the characteristic polynomial (D.20) are distinct, then $v(t)$ is a linear combination of the exponentials $e^{\zeta_j t}$. If there are repeated roots, then they are defective, and examining the expression (D.35) we see that a root of algebraic multiplicity k leads to terms of the form $e^{\zeta_j t}$, $te^{\zeta_j t}$, $\ldots$, $t^{k-1}e^{\zeta_j t}$. We can thus conclude that in general solutions to the linear differential equations (D.37) are bounded for all time only if the roots of the characteristic polynomial are in the left half-plane, with no repeated roots on the imaginary axis.

D.4 Nonnormal matrices

We have seen that if a matrix A has the Jordan form $A = RJR^{-1}$ (where J may be diagonal), then we can bound powers and the matrix exponential by the following expressions:

$$\begin{aligned} &\|A^n\| \le \kappa(R)\|J^n\| \text{ in general and} \\ &\|A^n\| \le \kappa(R)\rho(A)^n \text{ if } A \text{ is diagonalizable,} \end{aligned} \tag{D.39}$$

$$\begin{aligned} &\|e^{At}\| \le \kappa(R)\|e^{Jt}\| \text{ in general and} \\ &\|e^{At}\| \le \kappa(R)e^{\alpha(A)t} \text{ if } A \text{ is diagonalizable.} \end{aligned} \tag{D.40}$$

Here $\rho(A)$ and $\alpha(A)$ are the spectral radius and spectral abscissa, respectively. If A is defective, then J is not diagonal and algebraic growth terms can arise from the $\|J^n\|$ or $\|e^{Jt}\|$ factors.

If A is not normal, then even in the nondefective case the above bounds may not be very useful if $\kappa(R)$ is large. In particular, with nonnormal matrices matrix powers or exponentials can exhibit exponential *growth* during an initial transient phase (i.e., for n or t small enough), even if the bounds guarantee eventual exponential decay. Moreover, in these cases a small perturbation of the matrix may result in a matrix whose powers or exponential is not bounded.

This growth can be disastrous in terms of stability, particularly since in practice most interesting problems are nonlinear and often the matrix problems we consider are obtained

from a local linearization of the problem. Transient growth or instability of perturbed problems can easily lead to nonlinear instabilities in the original problem.

A related problem is that in practice we often are dealing with coefficient variable problems, where the matrix changes in each iteration. This issue is discussed more in Section D.7. In this section we continue to consider a fixed matrix A and explore some upper and lower bounds on powers and exponentials in the nonnormal case.

D.4.1 Matrix powers

If A is a normal matrix, $A^H A = AA^H$, then A is diagonalizable and the eigenvector matrix R can be chosen as an unitary matrix, for which $R^H R = I$ and $\kappa(R) = 1$. (We assume the 2-norm is always used in this section.) In this case $\|A\| = \rho(A)$ and $\|A^n\| = (\rho(A))^n$, so the eigenvalues of A give precise information about the rate of growth or decay of $\|A^n\|$, and similarly for the matrix exponential.

If A is not normal, then $\|A\| > \rho(A)$ and $(\rho(A))^n$ may not give a very good indication of the behavior of $\|A^n\|$ even in the diagonalizable case. From (D.39) we know that $\|A^n\|$ eventually decays at worst like $(\rho(A))^n$ for large enough n, but if $\kappa(R)$ is huge, then there can be enormous growth of $\|A^n\|$ before decay sets in. This is easily demonstrated with a simple example.

Example D.2. Consider the nonnormal matrix

$$A = \begin{bmatrix} 0.8 & 100 \\ 0 & 0.9 \end{bmatrix}. \tag{D.41}$$

This matrix is diagonalizable and the spectral radius is $\rho(A) = 0.9$. We expect $\|A^n\| \sim C(0.9)^n$ for large n, but for smaller n we observe considerable growth before the norm begins to decay. For example, starting with $U^0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ and computing $U^n = A^n U^0$ for $n = 1, 2, \ldots$ we find

$$U^0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad U^1 = \begin{bmatrix} 100 \\ 0.9 \end{bmatrix}, \quad U^2 = \begin{bmatrix} 170 \\ 0.81 \end{bmatrix}, \quad U^3 = \begin{bmatrix} 217 \\ 0.729 \end{bmatrix}, \quad \ldots.$$

We have the bound $\|U^n\|_2 \le \kappa_2(R)(0.9)^n \|U^0\|_2$ but in this case

$$R = \begin{bmatrix} 1 & 1 \\ 0 & 0.001 \end{bmatrix}, \qquad R^{-1} = \begin{bmatrix} 1 & -1000 \\ 0 & 1000 \end{bmatrix},$$

so $\kappa_2(R) = 2000$. Figure D.1 shows $\|U^n\|_2$ for $n = 1, \ldots, 30$ along with the bound.

Clearly this example could be made much more extreme by replacing $a_{22} = 100$ by a larger value. Larger matrices can exhibit similar growth before decay even if all the elements of the matrix are modest.

To gain some insight into the behavior of $\|A^n\|$ for a general matrix A, recall that the 2-norm of A is

$$\|A\| = \sqrt{\rho(A^H A)},$$

Hence

$$\begin{aligned} \|A^n\| &= [\rho((A^n)^H A^n)]^{1/2} \\ &= [\rho(A^H A^H \cdots A^H AA \cdots A)]^{1/2}. \end{aligned} \tag{D.42}$$

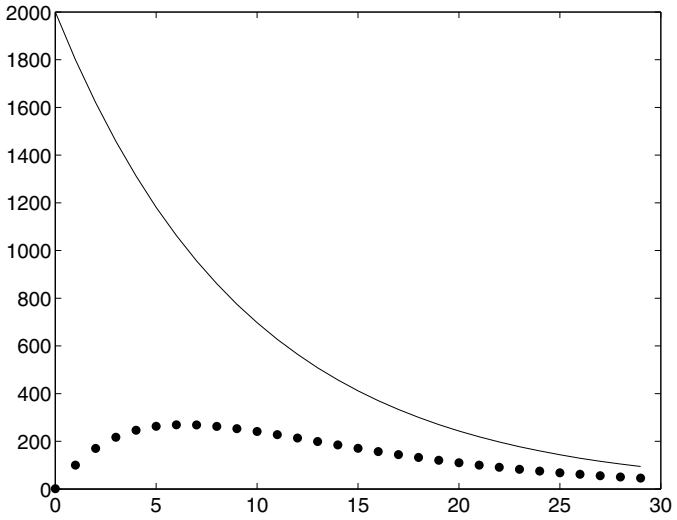

Figure D.1. *The points show* $\|U^n\|_2$ *for Example* D.2 *and the line shows the upper bound* $2000(0.9)^n$.

If A is normal, then $A^H A = AA^H$ and the terms in this product of $2n$ matrices can be rearranged to give

$$\|A^n\| = [\rho((A^H A)^n)]^{1/2} = (\rho(A^H A))^{n/2} = (\rho(A))^n = \|A\|^n. \tag{D.43}$$

If A is not normal, then we cannot rearrange the product, and in general we have

$$(\rho(A))^n \le \|A^n\| \le \|A\|^n. \tag{D.44}$$

If $\rho(A) < 1 < \|A\|$, as is often the case for nonnormal matrices of interest, then the lower bound is decaying exponentially to zero while the upper bound is growing exponentially. If we expect to see transient growth followed by decay, as illustrated, for example, in Figure D.1, then neither of these bounds tells us anything about how much growth is expected before the decay sets in. We would like to have lower and upper bounds on

$$\mathcal{P}(A) = \sup_{n \ge 0} \|A^n\|. \tag{D.45}$$

The matrix A is said to be *power bounded* if $\mathcal{P}(A) < \infty$, and of course a necessary condition for this is that the eigenvalues of A lie in the unit disk, with no defective eigenvalues on the disk.

Lower and upper bounds on $\mathcal{P}(A)$ can be written very concisely in terms of the Kreiss constant (D.25):

$$\mathcal{K}(A) \le \mathcal{P}(A) \le em\mathcal{K}(A) \tag{D.46}$$

for any $A \in \mathbb{C}^{m\times m}$. The upper bound has already been discussed in Section D.1. The lower bound is easier to obtain; it says that if $\|A^n\| \le C$ for all n, then

$$(|z| - 1)\|(zI - A)^{-1}\| \le C.$$

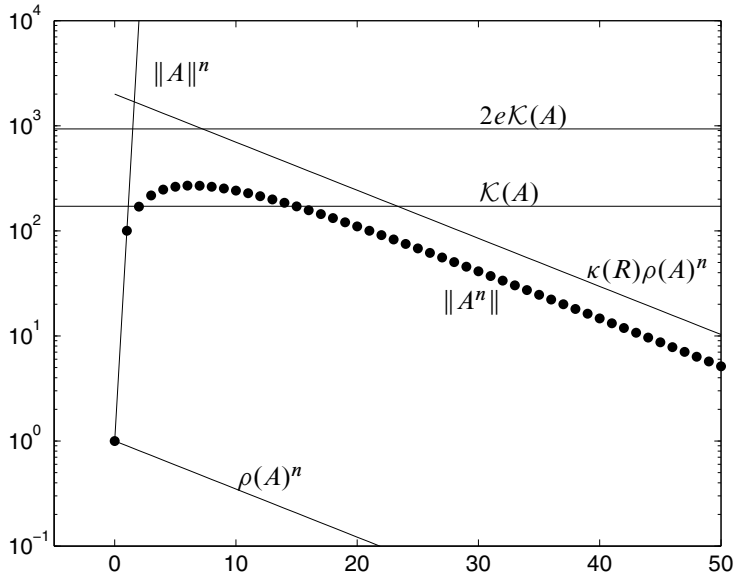

Figure D.2. *The norm of the matrix power, $\|A^n\|$ plotted on a logarithmic scale for the nonnormal matrix* (D.41)*. Also shown are the lower bound $\rho(A)^n$ and the upper bounds $\|A\|^n$ and $\kappa(R)\rho(A)^n$, as well as the value of the Kreiss constant $\mathcal{K}$ and $2e\mathcal{K}$ that give lower and upper bounds on $\sup_{t\geq 0}\|A^n\|$. Note that $\|A^n\|$ initially grows like $\|A\|^n$ and asymptotically decays like $\rho(A)^n$.*

To prove this note that $(zI - A)^{-1}$ has the series expansion

$$(zI - A)^{-1} = z^{-1}(I - z^{-1}A)^{-1} = z^{-1}\left(I + (z^{-1}A) + (z^{-1}A)^2 + (z^{-1}A)^3 + \cdots\right).$$

Taking norms and using $\|A^n\| \leq C$ gives

$$\|(zI - A)^{-1}\| \leq |z^{-1}|\left(1 + |z^{-1}| + |z^{-1}|^2 + |z^{-1}|^3 + \cdots\right)C = \frac{C}{|z| - 1}.$$

Figure D.2 shows the various bounds discussed above along with $\|A^n\|$ for the nonnormal matrix (D.41), this time on a logarithmic scale. For this matrix $\rho(A) = 0.9$, $\|A\| \approx 100$, and the Kreiss constant is $\mathcal{K}(A) = 171.5$.

D.4.2 Matrix exponentials

For the matrix exponential there are similar bounds (Theorem 18.5 in [92]):

$$\mathcal{K}_e(A) \leq \sup_{t\geq 0}\|e^{At}\| \leq em\mathcal{K}_e(A), \tag{D.47}$$

where the Kreiss constant with respect to the matrix exponential $\mathcal{K}_e(A)$ is defined by

$$\mathcal{K}_e(A) = \sup_{\mathrm{Re}(z)>0} \mathrm{Re}(z)\|(zI - A)^{-1}\|. \tag{D.48}$$

This measures how the resolvent of A behaves near the imaginary axis, which is the stability boundary for the matrix exponential.

It is also possible to derive upper and lower bounds on the norm of the matrix exponential as functions of t, analogous to (D.44) for powers of the matrix. For an arbitrary matrix A these take the form

$$e^{\alpha(A)t} \le \|e^{At}\| \le e^{\omega(A)t} \qquad \text{for all } t \ge 0, \tag{D.49}$$

where $\alpha(t)$ is the spectral abscissa (D.33) and $\omega(A)$ is the *numerical abscissa* defined by

$$\omega(A) = \frac{1}{2}\rho(A^H + A). \tag{D.50}$$

Note that $A^H + A$ is always hermitian and has real eigenvalues, but they may be positive even if $\alpha(A) < 0$. In general,

$$\alpha(A) \le \omega(A). \tag{D.51}$$

Also note that for any vector u,

$$\begin{aligned} \mathrm{Re}(u^H Au) &= \frac{1}{2}(u^H Au + u^H A^H u) \\ &= u^H \left(\frac{1}{2}(A^H + A)\right) u \\ &\le \omega(A) u^H u, \end{aligned} \tag{D.52}$$

since the Rayleigh quotient $u^H Bu/u^H u$ is always bounded by $\rho(B)$ for any hermitian matrix B. Another way to characterize $\omega(A)$ is as the maximum value that the real part of $u^H Au$ can take over any unit vector u. The set

$$W(A) = \{z \in \mathbb{C} : \ z = u^H Au \text{ for some } u \text{ with } \|u\|_2 = 1\} \tag{D.53}$$

is called the *numerical range* or *field of values* of the matrix A, and

$$\omega(A) = \max_{z \in W(A)} \mathrm{Re}(z). \tag{D.54}$$

For a normal matrix $W(A)$ is the convex hull of the eigenvalues, but for a nonnormal matrix it may be larger.

The bound (D.52) is of direct interest in studying the matrix exponential and can be used to prove the upper bound in (D.49) as follows. Suppose $u'(t) = Au(t)$ and consider

$$\begin{aligned} \frac{d}{dt}(u^H u) &= (u')^H u + u^H u' \\ &= u^H A^H u + u^H Au \\ &= 2\mathrm{Re}(u^H Au) \\ &\le 2\omega(A) u^H u. \end{aligned} \tag{D.55}$$

In other words, using the 2-norm,

$$\frac{d}{dt}\left(\|u(t)\|^2\right) \le 2\omega(A)\|u(t)\|^2, \tag{D.56}$$

which in turn implies

$$\frac{d}{dt}\|u(t)\| \le \omega(A)\|u(t)\|, \tag{D.57}$$

or, dividing by $\|u(t)\|$,

$$\frac{d}{dt}\log(\|u(t)\|) \le \omega(A). \tag{D.58}$$

Integrating gives

$$\|u(t)\| \le e^{\omega(A)t}\|u(0)\|. \tag{D.59}$$

Since $u(t) = e^{At}u(0)$, we have

$$\|e^{At}u(0)\| \le e^{\omega(A)t}\|u(0)\| \tag{D.60}$$

for any vector $u(0)$ and hence the matrix norm of e^{At} is bounded as in (D.49).

For a normal matrix A, $\alpha(A) = \omega(A)$ and (D.49) reduces to

$$\|e^{At}\| = e^{\alpha(A)t} = e^{\omega(A)t} \qquad \text{for all } t \ge 0. \tag{D.61}$$

In the scalar case, for a complex number A, the spectral abscissa and numerical abscissa are both equal to the real part of A, so each can be viewed as a generalization of the real part.

Finally, it is sometimes useful to investigate the initial transient growth of $\|e^{At}\|$ at $t = 0$. This can be determined by differentiating $\|e^{At}\|$ with respect to t and evaluating at $t = 0$. The result is

$$\begin{aligned}\frac{d}{dt}\|e^{At}\|\bigg|_{t=0} &= \lim_{k\to 0}\frac{\|e^{Ak}\| - 1}{k} \\ &= \lim_{k\to 0}\frac{\|I + Ak\| - 1}{k}.\end{aligned} \tag{D.62}$$

We can compute

$$\begin{aligned}\|I + kA\| &= \left[\rho\left((I + A^H k)(I + Ak)\right)\right]^{1/2} \\ &= \left[\rho(I + (A + A^H)k + A^H A k^2)\right]^{1/2} \\ &= \left[1 + \rho(A + A^H)k + O(k^2)\right]^{1/2} \\ &= 1 + \frac{1}{2}\rho(A + A^H)k + O(k^2),\end{aligned} \tag{D.63}$$

and so

$$\frac{d}{dt}\|e^{At}\|\bigg|_{t=0} = \lim_{k\to 0}\frac{\|I + Ak\| - 1}{k} = \frac{1}{2}\rho(A + A^H) = \omega(A). \tag{D.64}$$

Hence we expect

$$\|e^{At}\| = e^{\omega(A)t} + o(t) \quad \text{as } t \to 0. \tag{D.65}$$

From (D.49) we know that $e^{\omega(A)t}$ is an upper bound on the norm, but this shows that for small t we will observe transient growth at this rate.

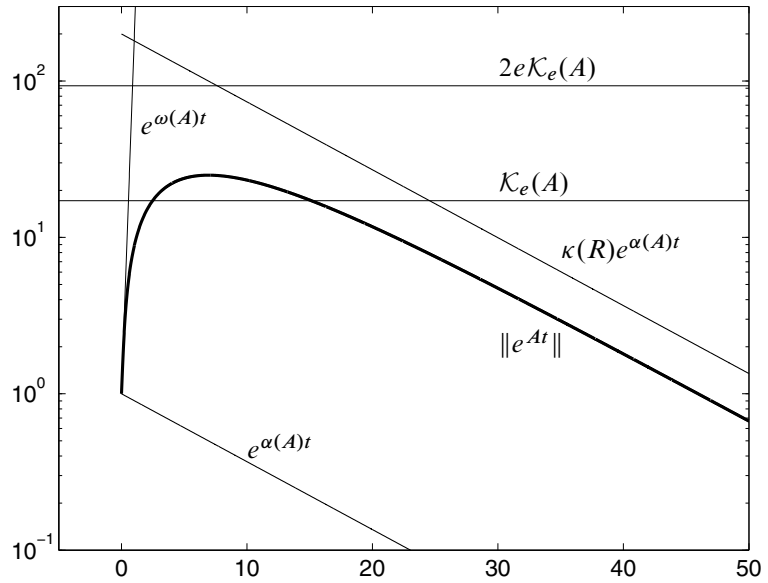

Figure D.3. *The norm of the matrix exponential $\|e^{At}\|$ plotted on a logarithmic scale for the nonnormal matrix* (D.66)*. Also shown are the lower bound $e^{\alpha(A)t}$ and the upper bounds $e^{\omega(A)t}$ and $\kappa(R)e^{\alpha(A)t}$, as well as the value of the Kreiss constant $\mathcal{K}_e$ and $2e\mathcal{K}_e$ that give lower and upper bounds on $\sup_{t\ge 0}\|e^{At}\|$. Note that $\|e^{At}\|$ initially grows like $e^{\omega(A)t}$ and asymptotically decays like $e^{\alpha(A)t}$.*

For example, consider the nonnormal matrix

$$A = \begin{bmatrix} -0.2 & 10 \\ 0 & -0.1 \end{bmatrix}. \tag{D.66}$$

Figure D.3 shows $\|e^{At}\|$ as a function of t on a logarithmic scale. The initial growth has slope $\omega(A) = 5.15$ and the eventual decay has slope $\alpha(A) = -0.1$, which follows from (D.40). The Kreiss constant for this matrix is $\mathcal{K}_e(A) = 17.17$.

The quantity $\omega(A)$ is sometimes defined as

$$\omega(A) = \lim_{k\to 0} \frac{\|I + Ak\| - 1}{k} \tag{D.67}$$

and in the ODE literature often goes by the name of the *logarithmic norm of A*. Of course it is not really a norm since it can be negative, but this terminology is motivated by inequalities such as (D.58).

D.5 Pseudospectra

Various tools have been developed to better understand the behavior of matrix powers or exponentials in the case of nonnormal matrices. One powerful approach is to investigate the *pseudospectra* of the matrix. Roughly speaking, this is the set of eigenvalues of all

"nearby" matrices. For a highly nonnormal matrix a small perturbation to the matrix can give a very large change in the eigenvalues of the matrix. For example, perturbing the matrix (D.41) to

$$\tilde{A} = \begin{bmatrix} 0.8 & 100 \\ 0.001 & 0.9 \end{bmatrix} \tag{D.68}$$

changes the eigenvalues from $\{0.8,\ 0.9\}$ to $\{0.53,\ 1.17\}$. A perturbation to A of magnitude 10^{-3} leads to an eigenvalue that is greater than 1. Since A is so close to a matrix $\tilde{A}$ for which $\tilde{A}^n$ blows up as $n \to \infty$, it is perhaps not so surprising that A^n exhibits initial growth before decaying. We say that the value $z = 1.17$ lies in the ϵ-pseudospectrum σ_ϵ of A for $\epsilon = 10^{-3}$.

The eigenvalues of A are isolated points in the complex plane where $(zI - A)$ is singular. We know that if any of these points lies outside the unit circle, then A^n blows up. The idea of pseudospectral analysis is to expand these isolated points to larger regions, the pseudospectra σ_ϵ, for some small ϵ, and see whether these pseudospectra extend beyond the unit circle.

There are various equivalent ways to define the ϵ-pseudospectrum of a matrix. Here are three.

Definition D.3. *For each $\epsilon \geq 0$, the ϵ-pseudospectrum $\sigma_\epsilon(A)$ of A is the set of numbers $z \in \mathbb{C}$ satisfying any one of the following equivalent conditions:*

(a) *z is an eigenvalue of $A + E$ for some matrix E with $\|E\| < \epsilon$,*
(b) *$\|Au - zu\| \leq \epsilon$ for some vector u with $\|u\| = 1$, or*
(c) *$\|(zI - A)^{-1}\| \geq \epsilon^{-1}$.*

In all these conditions the 2-norm is used (although the ideas can be extended to a general Banach space). Condition (a) is the easiest to understand and the one already illustrated above by example: z is an ϵ-pseudoeigenvalue of A if it is a genuine eigenvalue of some ϵ-sized perturbation of A. Condition (b) says that z is an ϵ-pseudoeigenvalue if there is a unit vector that is almost an eigenvector for this z. Condition (c) relates pseudoeigenvalues to the resolvent $(zI - A)^{-1}$, which we have already seen plays a role in obtaining bounds on the behavior of matrix powers and exponentials. The value z is an ϵ-pseudoeigenvalue of A if the resolvent is sufficiently large at z. This fits with the notion of expanding the singular points λ_i into regions σ_ϵ where $zI - A$ is nearly singular. (Note that by convention we set $\|(zI - A)^{-1}\| = \infty$ if z is an eigenvalue of A.)

The ϵ-pseudospectral radius $\rho_\epsilon(A)$ and ϵ-pseudospectral abscissa $\alpha_\epsilon(A)$ can be defined in the natural way as the maximum absolute value and maximum real part of any ϵ-pseudoeigenvalue of A, respectively. The Kreiss constants (D.25) and (D.48) can then be expressed in terms of pseudospectra as

$$\mathcal{K}(A) = \sup_{\epsilon > 0} \frac{\rho_\epsilon(A) - 1}{\epsilon}, \qquad \mathcal{K}_e(A) = \sup_{\epsilon > 0} \frac{\alpha_\epsilon(A)}{\epsilon}. \tag{D.69}$$

Hence the Kreiss constants can be viewed as a measure of how far the pseudospectra of A extend outside the unit circle or into the right half-plane.

The MATLAB package `eigtool` developed by Wright [104] provides tools for computing and plotting the pseudospectra of matrices and also quantities such as the ϵ-pseudospectral radius and ϵ-pseudospectral abscissa. See the book by Trefethen and Embree [92] for an in-depth discussion of pseudospectra with many examples of their use.

D.5.1 Nonnormality of a Jordan block

In Section D.2 we found an explicit expression for powers of a Jordan block, and we see that in addition to terms of the form λ^n, a block of order k has terms of order $n^k\lambda^n$ in its nth power. This clearly exhibits transient growth even in $\lambda < 1$. It is interesting to further investigate the Jordan block as an example of a highly nonnormal matrix.

For this discussion, let

$$J_\epsilon = \begin{bmatrix} c & 1 & & & & \\ & c & 1 & & & \\ & & & \ddots & & \\ & & & & c & 1 \\ \epsilon & & & & & c \end{bmatrix} \in \mathbb{R}^{k\times k} \tag{D.70}$$

with the entries not shown all equal to 0, so that for $\epsilon = 0$, J_0 is a Jordan block of the form (C.9) with all k of its eigenvalues at c.

If $\epsilon > 0$ on the other hand, the characteristic equation is

$$(\lambda - c)^k - \epsilon = 0$$

and the eigenvalues are

$$\lambda_p = c + \epsilon^{1/k} e^{2\pi i p/k}, \qquad p = 1,\ 2,\ \ldots,\ k. \tag{D.71}$$

The eigenvalues are now equally spaced around a circle of radius $\epsilon^{1/k}$ centered at $z = c$ in the complex plane.

Note that if k is large, $\epsilon^{1/k}$ will be close to 1 even for very small ϵ. For example, if $k = 1000$ and $\epsilon = 10^{-16}$, then $\epsilon^{1/k} \approx 0.96$. So although the eigenvalues of J_0 are all at c, a perturbation on the order of the machine round-off will blast them apart to a circle of radius nearly 1 about this point. For large k the ϵ-pseudospectrum of J_0 tends to fill up this circle, even for very small ϵ.

A similar matrix arises when studying the upwind method for advection, in which case k corresponds to the number of grid points and can easily be large. An implication of this nonnormality in stability analysis is explored in Section 10.12.1.

D.6 Stable families of matrices and the Kreiss matrix theorem

So far we have studied the behavior of powers of a single matrix A. We have seen that if the eigenvalues of A are inside the unit circle, then the powers of A are uniformly bounded,

$$\|A^n\| \le C \quad \text{for all } n, \tag{D.72}$$

for some constant C. If A is normal, then it fact $\|A^n\| \le \|A\|$. Otherwise $\|A^n\|$ may initially grow, perhaps to a very large value if the deviation from normality is large, but will eventually decay and hence some bound of the form (D.72) holds.

In studying the stability of discretizations of differential equations, we often need to consider not just a single matrix but an entire family of matrices. A particular discretization with mesh width h and/or time step k leads to a particular matrix A, but to study stability and prove convergence we need to let $h, k \to 0$ and study the whole family of resulting matrices. This is quite difficult to study in general because typically the dimensions of the matrices involved is growing as we refine the grid. However, at least in simple cases we can use von Neumann analysis to decouple the system into Fourier modes, each of which leads to a system of fixed dimension (the number of equations in the original differential equation). As we refine the grid we obtain more modes and the matrices involved may depend explicitly on h and k as well as on the wave number, but the matrices all have fixed dimension and it is this case that we consider here. (In Sections 9.6 and 10.5 we consider von Neumann analysis applied to scalar problems, in which case proving stability only requires studying powers of the scalar amplification factor g for each wave number, and powers of a scalar are uniformly bounded if and only if $|g| \le 1$. The considerations of this section come into play if von Neumann analysis is applied to a system of differential equations.)

Let $\mathcal{F}$ represent a family of matrices, say all the amplification matrices for different wave numbers that arise from discretizing a particular differential equation with different mesh widths. We say that $\mathcal{F}$ is *uniformly power bounded* if there is a constant $C > 0$ such that (D.72) holds for all matrices $A \in \mathcal{F}$. The bound must be uniform in both A and n, i.e., a single constant for all matrices in the family and all powers.

If $\mathcal{F}$ consists of only normal matrices and if $\rho(A) \le 1$ for all $A \in \mathcal{F}$, then the family is uniformly power bounded and (D.72) holds in general with $C = 1$.

When the matrices are not normal it can be more difficult to establish such a bound. Obviously a necessary condition is that $\rho(A) \le 1$ for all $A \in \mathcal{F}$ and that any eigenvalues of modulus 1 must be nondefective. If this condition fails for any $A \in \mathcal{F}$, then that particular matrix will fail to be power bounded and so the family cannot be. However, this condition is not sufficient—even if each matrix is power bounded they may not be uniformly so. For example, the infinite family of matrices

$$A_\epsilon = \begin{bmatrix} 1-\epsilon & 1 \\ 0 & 1-\epsilon \end{bmatrix} \tag{D.73}$$

for $\epsilon > 0$ are all individually power bounded but not uniformly power bounded. We have

$$A_\epsilon^n = \begin{bmatrix} (1-\epsilon)^n & n(1-\epsilon)^{n-1} \\ 0 & (1-\epsilon)^n \end{bmatrix},$$

and the off-diagonal term can be made arbitrarily large for large n by choosing ϵ small enough.

One fundamental result on power boundedness of matrix families is the Kreiss matrix theorem.

Theorem D.4. *The following conditions on a family $\mathcal{F}$ of matrices are equivalent:*

(a) *There exists a constant C such that $\|A^n\| \le C$ for all n and for all $A \in \mathcal{F}$. (The family is power bounded.)*

(b) *There exists a constant C_1 such that, for all $A \in \mathcal{F}$ and all $z \in \mathbb{C}$ with $|z| > 1$, the resolvent $(zI - A)^{-1}$ exists and is bounded by*

$$\|(zI - A)^{-1}\| \le \frac{C_1}{|z| - 1}. \tag{D.74}$$

In other words, the $\mathcal{K}(A) \le C_1$ for all $A \in \mathcal{F}$, where $\mathcal{K}(A)$ is the Kreiss constant (D.25)*.*

(c) *There exist constants C_2 and C_3 such that for each $A \in \mathcal{F}$ a nonsingular matrix S exists such that*

(i) $\|S\| \le C_2, \quad \|S^{-1}\| \le C_2$,
(ii) $B = S^{-1}AS$ *is upper triangular with off-diagonal elements bounded by*

$$|b_{ij}| \le C_3 \min(1 - |b_{ii}|,\ 1 - |b_{jj}|). \tag{D.75}$$

(Note that the diagonal elements of b are the eigenvalues of A.)

(d) *There exists a constant C_4 such that for each $A \in \mathcal{F}$ a positive definite matrix G exists with*

$$\begin{aligned} &C_4^{-1} I \le G \le C_4 I, \\ &A^H G A \le G. \end{aligned} \tag{D.76}$$

In condition (d) we say that two Hermitian matrices A and B satisfy $A \le B$ if $u^H A u \le u^H B u$ for any vector u. This condition can be rewritten in a more familiar form as follows:

(d′) *There exists a constant C_5 so that for each $A \in \mathcal{F}$ there is a nonsingular matrix T such that*

$$\|A\|_T \le 1 \ \text{ and } \ \kappa(T) \le C_5. \tag{D.77}$$

Here the T-norm of A is defined as in (C.35) in terms of the 2-norm,

$$\|A\|_T = \|T^{-1} A T\|.$$

Condition (d′) is related to (d) by setting $G = T^{-H} T^{-1}$, and (d′) states that we can define a set of norms, one for each $A \in \mathcal{F}$, for which the norm of A is less than 1 and therefore

$$\|A^n\|_T \le 1 \quad \text{for all } n \ge 0.$$

From this we can obtain uniform power boundedness by noting that

$$\|A^n\| \le \kappa(T) \|A^n\|_T \le C_5.$$

To make sense of condition (c), consider the case where all matrices $A \in \mathcal{F}$ are normal. Then each A can be diagonalized by a unitary similarity transformation and so

(c) holds with $\|S\|_2 = \|S^{-1}\|_2 = 1$ and $b_{ij} = 0$ for $i \neq j$. More generally, condition (c) requires that we can bring all $A \in \mathcal{F}$ to upper triangular form by uniformly well-conditioned similarity transformations, and with a uniform bound on the off-diagonals that is related to how close the diagonal elements (which are the eigenvalues of A) are to the unit circle.

Several other equivalent conditions have been identified and are sometimes more useful in practice; see Richtmyer and Morton [75] or the more recent paper of Strikwerda and Wade [85].

The equivalence of the conditions in Theorem D.4 can be proved by showing that (a) $\Longrightarrow$ (b) $\Longrightarrow$ (c) $\Longrightarrow$ (d) $\Longrightarrow$ (d′) $\Longrightarrow$ (a). For a more complete discussion and proofs see [75] or [85].

The equivalence of (a) and (b) also follows directly from (D.46) and a proof of this can be found in [92]. It is this equivalence that is the most interesting part of the theorem and that has received the most attention in subsequent work, to the point where the term "Kreiss matrix theorem" is often applied to inequalities of the form (D.46).

D.7 Variable coefficient problems

So far we have only considered solving equations of the form $U^{n+1} = AU^n$ or $u'(t) = Au(t)$, where the matrix A is constant (independent of n or t), and the solution can be written in terms of powers or matrix exponentials. In most applications, however, the matrix changes with time. Often A represents the Jacobian matrix for a nonlinear system and so it certainly varies with time as the solution changes. Adding this complication makes it considerably more difficult to analyze the behavior of solutions. Often a study of the "frozen coefficient" problem where A is frozen at a particular value as we solve forward in time is valuable, however, to gain some information about issues such as boundedness of the solution, and the theory presented earlier in this appendix will be useful in many contexts. However, new issues can come into play when the matrices vary, particularly if they vary rapidly, or more accurately, particularly if the *eigenvectors* of the matrix vary rapidly in time. We will not discuss this in detail—just give a brief introduction to this topic.

We first consider a discrete time iteration of the form

$$U^{n+1} = A_n U^n, \tag{D.78}$$

where A_n may vary with n. The solution is

$$U^j = A_{j-1} A_{j-2} \cdots A_1 A_0 U^0. \tag{D.79}$$

If $A_n \equiv A$ for all n, then this reduces to $U^j = A^j U^0$, but more generally the matrix product is harder to analyze than powers of a single matrix.

One case is relatively simple: suppose all the matrices A_n have the same eigenvectors, although possibly different eigenvalues, so

$$A_n = R \Lambda_n R^{-1} \tag{D.80}$$

for some fixed matrix R. In this case we say the A_n are *simultaneously diagonalizable*. Then the product in (D.79) reduces to

$$U^j = R\Lambda_{j-1}\Lambda_{j-2}\cdots\Lambda_1\Lambda_0 R^{-1}U^0$$

and $\Lambda_{j-1}\Lambda_{j-2}\cdots\Lambda_1\Lambda_0$ is a diagonal matrix whose ith diagonal element is the product of the ith eigenvalue of each of the matrices $A_0,\ A_1,\ \ldots,\ A_{j-1}$. Then we clearly have, for example, that if $\rho(A_n) \le 1$ for all n, then $\|U^n\|$ is uniformly bounded as $n \to \infty$. In fact we don't need $\rho(A_n) \le 1$ for all n; it is sufficient to have

$$\rho(A_n) \le 1 + \gamma_n \tag{D.81}$$

for some sequence of values γ_n satisfying

$$\sum_{j=0}^{\infty} \gamma_j < \infty. \tag{D.82}$$

From (D.81) it follows that $\rho(A_n) \le e^{\gamma_n}$ and so

$$\begin{aligned} \rho(R\Lambda_{j-1}\Lambda_{j-2}\cdots\Lambda_1\Lambda_0 R^{-1}) &\le \prod_{n=0}^{j-1} \rho(A_n) \\ &\le \prod_{n=0}^{j-1} e^{\gamma_n} \\ &\le \exp\left(\sum_{n=0}^{j-1} \gamma_n\right), \end{aligned} \tag{D.83}$$

and hence $\|U^n\|$ is uniformly bounded.

If the eigenvectors vary with n, however, then it happens that $\|U^n\|$ will grow without bound even if $\rho(A_n) < 1$ for all n.

Example D.3. As a simple example, consider

$$A_0 = \begin{bmatrix} 0 & 0 \\ 2 & 0.1 \end{bmatrix}, \qquad A_1 = \begin{bmatrix} 0.1 & 2 \\ 0 & 0 \end{bmatrix} \tag{D.84}$$

and then let A_n alternate between these two matrices for larger n, so $A_{2i} = A_0$ and $A_{2i+1} = A_1$. Then $\rho(A_n) = 0.1$ for all n. However, we see that after an even number of steps

$$U^{2i} = (A_1A_0)^iU^0$$

and

$$A_1A_0 = \begin{bmatrix} 4 & 0.2 \\ 0 & 0 \end{bmatrix},$$

so $\|U^j\|$ grows like 2^j.

If we iterated with either A_0 or A_1 alone, then U^n would go rapidly to zero. But note that these matrices are nonnormal and in either case there can be transient growth before decay sets in. For example,

$$U^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \Longrightarrow \quad A_0 U^0 = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \quad \Longrightarrow \quad A_0^2 U^0 = \begin{bmatrix} 0 \\ 0.2 \end{bmatrix}$$
$$\Longrightarrow \quad A_0^3 U^0 = \begin{bmatrix} 0 \\ 0.02 \end{bmatrix} \quad \Longrightarrow \quad \text{etc.}$$

Beyond the first iteration there is exponential decay by a factor 0.1 each iteration since $A_0 U^0$ is in the eigenspace of A_0 corresponding to $\lambda = 0.1$. But if we apply A_0 only once to U^0 and then apply A_1, we instead obtain

$$U^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \Longrightarrow \quad A_0 U^0 = \begin{bmatrix} 0 \\ 2 \end{bmatrix} \quad \Longrightarrow \quad A_1 A_0 U^0 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}.$$

Instead of decay we see amplification by another factor of 2. Moreover, the vector has been moved by A_1 out of the eigenspace it was in and into a vector that again suffers transient growth when A_0 is next applied. This couldn't happen if A_0 and A_1 shared the same eigenspaces.

One way to try to guarantee that the vectors U^n generated by the process (D.78) are uniformly bounded is to look for a single norm $\|\cdot\|$ in which

$$\|A_n\| \le 1 + \gamma_n \tag{D.85}$$

with (D.82) holding. In the simultaneously diagonalizable case considered above we can base the norm on the joint eigenvector matrix R using Theorem C.4. Another case in which we have stability is if the matrices A_n are all normal and satisfy (D.81), for then we can use the 2-norm.

But even if the matrices are not normal and the eigenvectors vary, if they do so slowly enough we may be able to prove boundedness using the following theorem. Here $\|\cdot\|_{T_n}$ is the T_n-norm defined by (C.35) in terms of some fixed norm $\|\cdot\|$, and we may be able to use the eigenvector matrix R_n of each A_n for these norms, although the theorem allows more flexibility.

Theorem D.5. *Suppose that for the difference equation* (D.78) *we can find a sequence of nonsingular matrices T_n such that*

1. $\|A_n\|_{T_n} \le 1 + \gamma_n$ *with* $\sum_{j=0}^{\infty} \gamma_j < \infty$,
2. $\|T_n\| \le C$, *a constant independent of n, and*
3. $\|T_n^{-1} T_{n-1}\| \le 1 + \beta_n$ *with* $\sum_{j=0}^{\infty} \beta_j < \infty$.

Then $\|U^n\|$ is uniformly bounded for all n.

Note that the third condition is the requirement that the norm vary sufficiently slowly. The proof can be found in [23].

Similar considerations apply to solutions to the variable coefficient ODE

$$u'(t) = A(t)u(t). \tag{D.86}$$

If the $A(t)$ are simultaneously diagonalizable for all t, then this reduces to $v'(t) = \Lambda(t)v(t)$, where $v(t) = R^{-1}u(t)$. Then

$$v_i(t) = \exp\left(\int_0^t \lambda_i(\tau)\,d\tau\right)$$

and $\|u(t)\|$ is uniformly bounded in t if $\int_0^t \alpha(A(\tau))\,d\tau$ is uniformly bounded, where $\alpha(A(t))$ is the spectral abscissa (D.33). (For the constant case $A(t) \equiv A$, this requires $\alpha(A) \le 0$.)

If the $A(t)$ are not simultaneously diagonalizable, then there are examples, similar to the Example D.3, where $\|u(t)\|$ may grow without bound even if all the matrices $A(t)$ have eigenvalues only in the left half-plane.

For a general function $A(t)$ we have (D.57),

$$\frac{d}{dt}\|u(t)\| \le \omega(A(t))\|u(t)\|,$$

where $\omega(A)$ is the numerical abscissa (D.50). Dividing by $\|u(t)\|$ gives

$$\frac{d}{dt}\log(\|u(t)\|) \le \omega(A(t))$$

and integrating yields

$$\|u(t)\| \le \exp\left(\int_0^t \omega(A(t))\,d\tau\right)\,\|u(0)\|. \tag{D.87}$$

This shows that the solution $u(t)$ is bounded in norm for all t provided that $\int_0^t \omega(A(t))\,d\tau$ is bounded above uniformly in t.

Recall, however, that $\omega(A)$ may be positive even when the eigenvalues of A all have negative real part (in the nonnormal case). So requiring, for example, $\omega(A(t)) \le 0$ for all t is akin to requiring $\|A_n\| \le 1$ in the same norm for all matrices A_n in the difference equation (D.78). As in the case of the difference equation this requirement can be relaxed by introducing the notion of a logarithmic T-norm that varies with time, and a theorem similar to Theorem D.5 obtained for differential equations if the eigenvector matrix of $A(t)$ is not varying too rapidly; see [23].

For a hint of what's involved, suppose the matrices are all diagonalizable, $A(t) = R(t)\Lambda(t)R^{-1}(t)$, and that $R(t)$ is differentiable. Note that $(R^{-1})'(t) = -R^{-1}(t)R'(t)R^{-1}(t)$, obtained by differentiating $RR^{-1} = I$. If we set $v(t) = R^{-1}(t)u(t)$ we find that

$$\begin{aligned} v' &= R^{-1}u' + (R^{-1})'u \\ &= R^{-1}ARv + (R^{-1})'Rv \\ &= (\Lambda - R^{-1}R')v. \end{aligned} \tag{D.88}$$

So the boundedness of $v(t)$ depends on the matrices $\Lambda(t) - R^{-1}(t)R'(t)$, and if the eigenvectors vary rapidly, then the latter term can lead to unbounded growth even if the eigenvalues are all in the left half-plane.

Appendix E
Partial Differential Equations

In this appendix we briefly discuss some of the basic partial differential equations (PDEs) that are used in this book to illustrate the development of numerical methods, and we review the manner in which Fourier analysis can be used to gain insight into these problems.

E.1 Classification of differential equations

First we review the classification of differential equations into elliptic, parabolic, and hyperbolic equations. Not all PDEs fall into one of these classes, by any means, but many important equations that arise in practice do. These classes of equations model different sorts of phenomena, display different behavior, and require different numerical techniques for their solution. Standard texts on partial differential equations such as Kevorkian [55] give further discussion.

E.1.1 Second order equations

In most elementary texts the classification is given for a linear second-order differential equation in two independent variables of the form

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu = g.$$

The classification depends on the sign of the discriminant,

$$b^2 - 4ac \begin{cases} < 0 & \Longrightarrow \quad \text{elliptic,} \\ = 0 & \Longrightarrow \quad \text{parabolic,} \\ > 0 & \Longrightarrow \quad \text{hyperbolic,} \end{cases}$$

and the names arise by analogy with conic sections. The canonical examples are the Poisson problem $u_{xx} + u_{yy} = g$ for an elliptic problem, the heat equation $u_t = \kappa u_{xx}$ (with $\kappa > 0$) for a parabolic problem, and the wave equation $u_{tt} = c^2 u_{xx}$ for a hyperbolic problem. In the parabolic and hyperbolic case t is used instead of y since these are typically time-dependent problems. These can all be extended to more space dimensions. These

equations describe different types of phenomena and require different techniques for their solution (both analytically and numerically), and so it is convenient to have names for classes of equations exhibiting the same general features. Other equations have some of the same features, and the classification scheme can be extended beyond the second order linear form given above. Some hint of this is given in the next few sections.

E.1.2 Elliptic equations

The classic example of an elliptic equation is the *Poisson problem*

$$\nabla^2 u = f, \tag{E.1}$$

where ∇^2 is the Laplacian operator and f is a given function of $\vec{x} = (x, y)$ in some spatial domain Ω. We seek a function $u(\vec{x})$ in Ω satisfying (E.1) together with some *boundary conditions* all along the boundary of Ω. Elliptic equations typically model steady-state or equilibrium phenomena, and so there is no temporal dependence (however, see Section 2.16 for a counterexample). Elliptic equations may also arise in solving time-dependent problems if we are modeling some phenomena that are always in local equilibrium and equilibrate on time scales that are much faster than the time scale being modeled. For example, in "incompressible" flow the fast acoustic waves are not modeled and instead the pressure is computed by solving a Poisson problem at each time step which models the global effect of these waves.

Elliptic equations give boundary value problems where the solution at all points must be simultaneously determined based on the boundary conditions all around the domain. This typically leads to a very large sparse system of linear equations to be solved for the values of U at each grid point. If an elliptic equation must be solved in every time step of a time-dependent calculation, as in the examples above, then it is crucial that these systems be solved as efficiently as possible.

More generally, a linear elliptic equation has the form

$$Lu = f, \tag{E.2}$$

where L is some *elliptic operator*. For our purposes we will consider only constant coefficient second order operators, which in N space dimensions have the form

$$L = \sum_{j,k=1}^{N} A_{jk} \frac{\partial^2}{\partial x_j \partial x_k} + \sum_{j=1}^{N} B_j \frac{\partial}{\partial x_j} + C, \tag{E.3}$$

where the A_{jk}, B_j, C are real numbers. Note that since $\partial^2 u/\partial x_j \partial x_k = \partial^2 u/\partial x_k \partial x_j$, we can always choose the $N \times N$ matrix A defined by the second order term to be symmetric. This operator is said to be elliptic if A is positive definite or negative definite, as defined in Section C.4. This means that $v^T A v$ has the same sign for all nonzero vectors $v \in \mathbb{R}^N$ and cannot pass through zero. This can be shown to ensure that the boundary value problem (E.2) has a unique solution. For an indication of why this is true, see Section E.3.5.

In two space dimensions writing the matrix as

$$A = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}$$

and considering when this matrix is definite, we find that the operator is elliptic if $b^2 - 4ac < 0$, as in the classification of the previous section.

For the Laplacian operator $\nabla^2 u$, A is the $N \times N$ identity matrix and so this is an elliptic operator. Note that in one space dimension $\nabla^2 u$ reduces to $u''(x)$ and the problem (E.1) is the 2-point boundary value problem considered in Chapter 2.

E.1.3 Parabolic equations

If L is an elliptic operator with a positive definite A, then the time-dependent equation

$$u_t = Lu - f \tag{E.4}$$

is well posed (see Section E.3.5) and is called *parabolic*. If $L = \nabla^2$ is the Laplacian, then (E.4) is known as the *heat equation* or *diffusion equation* and models the diffusion of heat in a material, for example.

Now $u(\vec{x}, t)$ varies with time and we require *initial data* $u(\vec{x}, 0)$ for every $\vec{x} \in \Omega$ as well as boundary conditions around the boundary at each time $t > 0$. If the boundary conditions are independent of time, then we might expect the heat distribution to reach a steady state in which u is independent of t. We could then solve for the steady state directly by setting $u_t = 0$ in (E.4), which results in the elliptic equation (E.2).

Marching to steady state by solving the time-dependent equation (E.4) numerically would be one approach to solving the elliptic equation (E.2), but this is typically not the fastest method if all we require is the steady state.

E.1.4 Hyperbolic equations

Rather than discretizing second order hyperbolic equations such as the wave equation $u_{tt} = c^2 u_{xx}$, we will consider a related form of hyperbolic equations known as *first order hyperbolic systems*. The linear problem in one space dimension has the form

$$u_t + Au_x = 0, \tag{E.5}$$

where $u(x, t) \in \mathbb{R}^s$ and A is an $s \times s$ matrix. The problem is called *hyperbolic* if A has *real* eigenvalues and is *diagonalizable*, i.e., has a complete set of linearly independent eigenvectors. These conditions allow us to view the solution in terms of propagating waves, and indeed hyperbolic systems typically arise from physical processes that give wave motion or advective transport. This is explored more in Section 10.10.

The simplest example of a hyperbolic equation is the constant-coefficient *advection equation*

$$u_t + au_x = 0, \tag{E.6}$$

where u is the advection velocity. The solution is simply $u(x, t) = u(x - at, 0)$, so any u profile simply advects with the flow at velocity a.

As a simple example of a linear hyperbolic system, the equations of linearized acoustics arising from elasticity or gas dynamics can be written as a first order system of two equations in one space dimension as

$$\begin{bmatrix} p \\ u \end{bmatrix}_t + \begin{bmatrix} 0 & \kappa_0 \\ 1/\rho_0 & 0 \end{bmatrix} \begin{bmatrix} p \\ u \end{bmatrix}_x = 0 \tag{E.7}$$

in terms of pressure and velocity perturbations, where ρ_0 is the background density and κ_0 is the "bulk modulus" of the material. Note that if we differentiate the first equation with respect to t, the second with respect to x, and then eliminate $u_{xt} = u_{tx}$, we obtain the second order wave equation for the pressure:

$$p_{tt} = c^2 p_{xx},$$

where

$$c = \sqrt{\kappa_0/\rho_0}$$

is the speed of sound in the material.

Often hyperbolic equations arise most naturally as first order systems, as motivated in the next section, and we consider only this formulation.

E.2 Derivation of partial differential equations from conservation principles

Many physically relevant partial differential equations can be derived based on the principle of conservation. We can view $u(x,t)$ as a *concentration* or *density* function for some substance or chemical that is in dilute suspension in a liquid, for example. Basic equations of the same form arise in many other applications, however. The material presented here is meant to be a brief review, and much more complete discussions are available in many sources. See, for example, [55], [61], [66], [102].

A reasonable model to consider in one space dimension is the concentration or density of a contaminant in a stream or pipe, where the variable x represents distance along the pipe. The concentration is assumed to be constant across any cross section, so that its value varies only with x. The density function $u(x,t)$ is defined in such a way that integrating the function $u(x,t)$ between any two points x_1 and x_2 gives the total mass of the substance in this section of the pipe at time t:

$$\text{Total mass between } x_1 \text{ and } x_2 \text{ at time } t \;\; = \int_{x_1}^{x_2} u(x,t)\,dx.$$

The density function in measured in units such as grams/meter. (Note that this u really represents the integral over the cross section of the pipe of a density function that is properly measured in grams/meter^3.)

The basic form of differential equation that models many physical processes can be derived in the following way. Consider a section $x_1 < x < x_2$ and the manner in which $\int_{x_1}^{x_2} u(x,t)\,dx$ changes with time. This integral represents the total mass of the substance in this section, so if we are studying a substance that is neither created nor destroyed within this section, then the total mass within this section can change only due to the *flux* or flow of particles through the endpoints of the section at x_1 and x_2. This flux is given by some function f which, in the simplest case, depends only on the value of u at the corresponding point.

E.2.1 Advection

If the substance is simply carried along (advected) in a flow at some constant velocity a, then the flux function is

$$f(u) = au. \tag{E.8}$$

The local density $u(x,t)$ (in grams/meter, say) multiplied by the velocity (in meters/sec, say) gives the flux of material past the point x (in grams/sec).

Since the total mass in $[x_1, x_2]$ changes only due to the flux at the endpoints, we have

$$\frac{d}{dt}\int_{x_1}^{x_2} u(x,t)\,dx = f(u(x_1,t)) - f(u(x_2,t)). \tag{E.9}$$

The minus sign on the last term comes from the fact that f is, by definition, the flux to the right.

If we assume that u and f are smooth functions, then this equation can be rewritten as

$$\frac{d}{dt}\int_{x_1}^{x_2} u(x,t)\,dx = \int_{x_1}^{x_2} \frac{\partial}{\partial x} f(u(x,t))\,dx$$

or, with some further modification, as

$$\int_{x_1}^{x_2} \left[\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x} f(u(x,t))\right] dx = 0.$$

Since this integral must be zero for all values of x_1 and x_2, it follows that the integrand must be identically zero. This gives, finally, the differential equation

$$\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x} f(u(x,t)) = 0. \tag{E.10}$$

This form of equation is called a *conservation law*.

For the case considered in Section E.2.1, $f(u) = au$ with a constant and this equation becomes the advection equation (E.6). This equation requires initial conditions and possibly also boundary conditions in order to determine a unique solution. The simplest case is the *Cauchy problem* on $-\infty < x < \infty$ (with no boundary), also called the pure initial value problem. Then we need only to specify initial data

$$u(x,0) = \eta(x). \tag{E.11}$$

Physically, we would expect the initial profile of η to simply be carried along with the flow at speed a, so we should find

$$u(x,t) = \eta(x - at). \tag{E.12}$$

It is easy to verify that this function satisfies the advection equation (E.6) and is the solution of the PDE.

The curves

$$x = x_0 + at$$

through each point x_0 at time 0 are called the *characteristics* of the equation. If we set

$$U(t) = u(x_0 + at, t)$$

then

$$\begin{aligned} U'(t) &= au_x(x_0 + at, t) + u_t(x_0 + at, t) \\ &= 0 \end{aligned}$$

using (E.6). Along these curves the PDE reduces to a simple ordinary differential equation (ODE) $U' = 0$ and the solution must be constant along each such curve, as is also seen from the solution (E.12).

E.2.2 Diffusion

Now suppose that the fluid in the pipe is not flowing and has zero velocity. Then according to the above equation, $u_t = 0$ and the initial profile $\eta(x)$ does not change with time. However, if η is not constant in space then in fact it will tend to slowly change due to molecular diffusion. The velocity a should really be thought of as a *mean velocity*, the average velocity that the roughly 10^{23} molecules in a given drop of fluid have. But individual molecules are bouncing around in different directions and so molecules of the substance we are tracking will tend to get spread around in the ambient fluid, just as a drop of ink spreads in water. There will tend to be a net motion from regions where the density is large to regions where it is smaller, and in fact it can be shown that the flux (in one dimension) is proportional to $-u_x$. The flux at a point x now depends on the value of u_x at this point, rather than on the value of u, so we write

$$f(u_x) = -\kappa u_x, \tag{E.13}$$

where κ is the *diffusion coefficient*. The relation (E.13) is known as *Fick's law*. Using this flux in (E.10) gives

$$u_t = \kappa u_{xx}, \tag{E.14}$$

which is known as the *diffusion equation*.

This equation is also called the *heat equation* since heat diffuses in much the same way. In this case $u(x, t)$ represents the density of thermal energy, which is proportional to the temperature. The proportionality factor is the *heat capacity* of the material, which we'll take to be the value 1 (with suitable units) so that u can also be viewed as the temperature. The one-dimensional equation models the conduction of heat in a rod. The heat conduction coefficient κ depends on the material and how well it conducts heat. The relation (E.13) is known as *Fourier's law of heat conduction*, which states more generally that the flux of thermal energy is proportional to the temperature gradient.

In some problems the diffusion coefficient may vary with x, for example, in a rod made of a composite of different materials. Then $f = -\kappa(x)u_x$ and the equation becomes

$$u_t = (\kappa(x)u_x)_x.$$

Returning to the example of fluid flow, more generally there would be both advection and diffusion occurring simultaneously. Then the flux is $f(u, u_x) = au - \kappa u_x$, giving the *advection-diffusion* equation

$$u_t + au_x = \kappa u_{xx}. \tag{E.15}$$

The diffusion and advection-diffusion equations are examples of the general class of PDEs called *parabolic*.

E.2.3 Source terms

In some situations $\int_{x_1}^{x_2} u(x,t)\,dx$ changes due to effects other than flux through the endpoints of the section, if there is some source or sink of the substance within the section. Denote the density function for such a source by $\psi(x,t)$. (Negative values of ψ correspond to a sink rather than a source.) Then the equation becomes

$$\frac{d}{dt}\int_{x_1}^{x_2} u(x,t)\,dx = -\int_{x_1}^{x_2} \frac{\partial}{\partial x} f(u(x,t))\,dx + \int_{x_1}^{x_2} \psi(x,t)\,dx.$$

This leads to the PDE

$$u_t(x,t) + f(u(x,t))_x = \psi(x,t). \tag{E.16}$$

For example, if we have heat conduction in a rod together with an external source of heat energy distributed along the rod with density ψ, then we have

$$u_t = \kappa u_{xx} + \psi.$$

In some cases the strength of the source may depend on the value of u. For example, if the rod is immersed in a liquid that is held at constant temperature u_0, then the flux of heat into the rod at the point (x,t) is proportional to $u_0 - u(x,t)$ and the equation becomes

$$u_t(x,t) = \kappa u_{xx}(x,t) + \alpha(u_0 - u(x,t)).$$

E.2.4 Reaction-diffusion equations

One common form of source term arises from chemical kinetics. If the components of $u \in \mathbb{R}^s$ represent concentrations of s different species reacting with one another, then the kinetics equations have the form $u_t = \mathbb{R}(u)$, as described in Section 7.4.1. This assumes the different species are well mixed at all times and so the concentrations vary only with time. If there are spatial variations in concentrations, then these equations may be combined with diffusion of each species. This would lead to a system of *reaction-diffusion equations* of the form

$$u_t = \kappa u_{xx} + \mathbb{R}(u). \tag{E.17}$$

The diffusion coefficient could be different for each species, in which case κ would be a diagonal matrix instead of a scalar. This generalizes to more space dimensions by replacing u_{xx} by $\nabla^2 u$, the Laplacian of u.

Advection terms might also be present if the reactions are taking place in a flowing fluid. More generally the reaction-diffusion equations may be coupled with nonlinear equations of fluid dynamics, which may themselves contain both hyperbolic terms and parabolic viscous terms.

E.3 Fourier analysis of linear partial differential equations

For *linear* PDEs, Fourier analysis is often used to obtain solutions or perform theoretical analysis. This is because the functions $e^{i\xi x} = \cos(\xi x) + i\sin(\xi x)$ are essentially[1]

[1] On a periodic domain. For the Cauchy problem these functions are not L^2 functions and so strictly speaking are not called eigenfunctions, but this is unimportant for our purposes.

eigenfunctions of the differentiation operator $\partial_x = \frac{\partial}{\partial x}$. Differentiating this function gives a scalar multiple of the function, and hence simple differential equations (linear constant coefficient ones, at least) are simplified and can be reduced to algebraic equations.

Fourier analysis is equally important in the study of finite difference methods for *linear* PDEs for the same reason: these same functions are eigenfunctions of translation invariant finite difference operators. This is exploited in Sections 9.6 and 10.5, where von Neumann stability analysis of finite difference methods is discussed. An understanding of Fourier analysis of PDEs is also required in Section 10.9, where finite difference methods are analyzed by studying "modified equations."

E.3.1 Fourier transforms

Recall that a function $v(x)$ is in the space L^2 if it has a finite 2-norm, defined by

$$\|v\|_2 = \left(\int_{-\infty}^{\infty} |v(x)|^2 \, dx \right)^{1/2}.$$

If $v \in L^2$, then we can define its Fourier transform $\hat{v}(\xi)$ by

$$\hat{v}(\xi) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} v(x) e^{-i\xi x} \, dx. \tag{E.18}$$

The function $\hat{v}(\xi)$ is also in L^2 and in fact it has exactly the same 2-norm as v,

$$\|\hat{v}\|_2 = \|v\|_2. \tag{E.19}$$

This is known as *Parseval's relation*.

We can express the original function $v(x)$ as a linear combination of the set of functions $e^{i\xi x}$ for different values of ξ, which together form a basis for the infinite dimensional function space L^2. The Fourier transform $\hat{v}(\xi)$ gives the coefficients in the expression

$$v(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{v}(\xi) e^{i\xi x} \, d\xi, \tag{E.20}$$

which is known as the *inverse Fourier transform*. This is analogous to writing a vector as a linear combination of basis vectors.

E.3.2 The advection equation

We already know the solution (E.12) to the advection equation (E.6), but to illustrate the role of Fourier analysis we will solve the advection equation $u_t + au_x = 0$ using Fourier transforms. We will transform in x only and denote the transform of $u(x,t)$ (a function of x at each fixed t) by $\hat{u}(\xi, t)$:

$$\hat{u}(\xi, t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} u(x,t) e^{-i\xi x} \, dx. \tag{E.21}$$

Then

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{u}(\xi, t) e^{i\xi x} \, d\xi \tag{E.22}$$

and differentiating this with respect to t and x gives

$$u_t(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{u}_t(\xi,t) e^{i\xi x}\, dx,$$
$$u_x(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{u}(\xi,t) i\xi e^{i\xi x}\, dx.$$

From this we see that the Fourier transform of $u_t(x,t)$ is $\hat{u}_t(\xi,t)$ and the Fourier transform of $u_x(x,t)$ is $i\xi\hat{u}(\xi,t)$. Fourier transforming the advection equation by computing

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (u_t + a u_x) e^{-i\xi x}\, dx = 0$$

thus gives

$$\hat{u}_t(\xi,t) + a i\xi \hat{u}(\xi,t) = 0$$

or

$$\hat{u}_t = -i\xi a \hat{u}.$$

This is a time-dependent ODE for the evolution of $\hat{u}(\xi,t)$ in time. There are two important points to notice:

- Since differentiation with respect to x has become multiplication by $i\xi$ after Fourier transforming, the original PDE involving derivatives with respect to x and t has become an ODE in t alone.
- The ODEs for different values of ξ are decoupled from one another. We have to solve an infinite number of ODEs, one for each value of ξ, but they are decoupled scalar equations rather than a coupled system.

It is easy to solve these ODEs. We need initial data $\hat{u}(\xi,0)$ at time $t = 0$ for each value of ξ, but this comes from Fourier transforming the initial data $u(x,0) = \eta(x)$,

$$\hat{u}(\xi,0) = \hat{\eta}(\xi) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \eta(x) e^{-i\xi x}\, dx.$$

Solving the ODEs then gives

$$\hat{u}(\xi,t) = e^{-i\xi a t} \hat{\eta}(\xi). \tag{E.23}$$

We can now Fourier transform back using (E.22) to get the desired solution $u(x,t)$:

$$\begin{aligned} u(x,t) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-i\xi a t} \hat{\eta}(\xi) e^{i\xi x}\, d\xi \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi) e^{i\xi(x-at)}\, d\xi \\ &= \eta(x-at). \end{aligned}$$

This last equality comes from noting that we are simply evaluating the inverse Fourier transform of $\hat{\eta}$ at the point $x - at$. We see that we have recovered the standard solution (E.12) of the advection equation in this manner.

We can also calculate the "Green's function" for the advection equation, the solution to $u_t + a u_x = 0$ with special initial data $\eta(x) = \delta(x - \bar{x})$. The solution is clearly

$$G(x,t;\bar{x}) = \delta(x - \bar{x} - at). \tag{E.24}$$

The general solution for arbitrary $\eta(x)$ can be written as a linear combination of these Green's functions, weighted by the data:

$$\begin{aligned} u(x,t) &= \int_{-\infty}^{\infty} \eta(\bar{x}) G(x,t;\bar{x})\, d\bar{x} \\ &= \int_{-\infty}^{\infty} \eta(\bar{x}) \delta(x - \bar{x} - at)\, d\bar{x} \\ &= \eta(x - at). \end{aligned} \tag{E.25}$$

E.3.3 The heat equation

Now consider the heat equation,

$$u_t = \kappa u_{xx}. \tag{E.26}$$

Since the Fourier transform of $u_{xx}(x,t)$ is $(i\xi)^2 \hat{u}(\xi,t) = -\xi^2 \hat{u}(\xi,t)$, Fourier transforming (E.26) gives the ODE

$$\hat{u}_t(\xi,t) = -\kappa \xi^2 \hat{u}(\xi,t). \tag{E.27}$$

Again we have initial data $\hat{u}(\xi,0) = \hat{\eta}(\xi)$ from the given initial data on u. Now solving the ODE gives

$$\hat{u}(\xi,t) = e^{-\kappa \xi^2 t} \hat{\eta}(\xi). \tag{E.28}$$

Note that this has a very different character than (E.23), the Fourier transform obtained from the advection equation. For the advection equation, $\hat{u}(\xi,t) = e^{ia\xi t}\hat{\eta}(\xi)$ and $|\hat{u}(\xi,t)| = |\hat{\eta}(\xi)|$ for all t. Each Fourier component maintains its original amplitude and is modified only in phase, leading to a traveling wave behavior in the solution.

For the heat equation, however, $|\hat{u}(\xi,t)|$ decays in time exponentially fast. The decay rate depends on κ, the diffusion coefficient, and also on ξ, the wave number. Highly oscillatory components (with ξ^2 large) decay much faster than those with low wave numbers. This results in a *smoothing* of the solution as time evolves. (See Figure 9.3.)

The fact that the solution contains components that decay at very different rates leads us to expect numerical difficulties with stiffness, similar to those discussed for ODEs in Chapter 8. In Section 9.4 we will see that this is indeed the case and that implicit methods must generally be used to efficiently solve the heat equation.

A single Fourier mode decaying exponentially in time is one special solution to the heat equation. Another class of special solutions that is useful to know about arises from Gaussian initial data. The Fourier transform of a Gaussian is another Gaussian. Take

$$\eta(x) = e^{-\beta x^2} \tag{E.29}$$

for some β. Then

$$\begin{aligned} \hat{\eta}(\xi) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\beta x^2} e^{i\xi x}\, dx \\ &= \frac{1}{\sqrt{2\beta}} e^{-\xi^2/4\beta}. \end{aligned} \tag{E.30}$$

Then (E.22) combined with (E.28) gives the solution

$$\begin{aligned} u(x,t) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\kappa\xi^2 t} \hat{\eta}(\xi) e^{i\xi x}\, d\xi \\ &= \frac{1}{2\sqrt{\pi\beta}} \int_{-\infty}^{\infty} e^{-\xi^2(\kappa t + 1/4\beta)} e^{i\xi x}\, d\xi. \end{aligned} \tag{E.31}$$

This is just the inverse Fourier transform of another Gaussian, with $e^{-\xi^2/4C}$ in place of $e^{-\xi^2/4\beta}$, where $C = 1/(4\kappa t + 1/\beta)$, and so

$$\begin{aligned} u(x,t) &= \sqrt{\frac{C}{\beta}} e^{-Cx^2} \\ &= \frac{1}{\sqrt{4\beta\kappa t + 1}} e^{-x^2/(4\kappa t + 1/\beta)}. \end{aligned} \tag{E.32}$$

As t increases this Gaussian becomes more spread out and the magnitude decreases, as we expect from diffusion. You can check that (E.32) solves the heat equation directly by differentiating.

Note what happens if we shift the initial data to a different location,

$$\eta(x) = e^{-\beta(x-\bar{x})^2}. \tag{E.33}$$

Then the solution simply shifts too,

$$u(x,t) = \frac{1}{\sqrt{4\beta\kappa t + 1}} e^{-(x-\bar{x})^2/(4\kappa t + 1/\beta)}. \tag{E.34}$$

As a special case we can find the Green's function for the heat equation. Scale the data (E.33) by $\sqrt{\beta/\pi}$ so that it has integral equal to 1 and represents a smeared out version of the delta function, setting

$$v_\beta(x, 0; \bar{x}) = \sqrt{\frac{\beta}{\pi}} e^{-\beta(x-\bar{x})^2}. \tag{E.35}$$

The solution to (E.26) with this data is then

$$v_\beta(x, t; \bar{t}) = \frac{1}{\sqrt{4\pi\kappa t + \pi/\beta}} e^{-(x-\bar{x})^2/(4\kappa t + 1/\beta)}. \tag{E.36}$$

Now let $\beta \to 0$ so the initial data approaches a delta function. The solution $v_\beta(x, t; \bar{t})$ then approaches the Green's function for (E.26),

$$G(x, t; \bar{x}) = \frac{1}{\sqrt{4\pi\kappa t}} e^{-(x-\bar{x})^2/(4\kappa t)}. \tag{E.37}$$

Delta function initial data spreads out into a decaying Gaussian. Note that initial data concentrated at a single point (an idealization of a very tiny drop of ink in water, say)

spreads out immediately to have a nonzero value for all x. More generally if we look at the solution for general initial data by integrating $\eta(\bar{x})$ against the Green's function, we see that the data at each $\bar{x}$ immediately have an effect everywhere. Thus information propagates infinitely quickly in the heat equation. This is quite different from the advection equation, where the data at $\bar{x}$ affect the solution at only one point $\bar{x} + at$ at a later time t.

Of course physically information cannot propagate at infinite speed, and the discrepancy with the behavior of the heat equation simply shows that the heat equation is only a model of reality, and one that is not exactly correct. But note that away from the point $\bar{x}$ the effect decays very rapidly and so this is often a very accurate model of reality.

E.3.4 The backward heat equation

Note that the diffusion coefficient κ is required to be positive (because heat flows from warm to cool regions, not the other way around). Mathematically we could consider trying to solve the equation (E.26) with $\kappa < 0$ but this equation turns out to be *ill-posed*[2] One way to interpret this physically is to view it as solving the heat equation with coefficient $-\kappa > 0$ backward in time, starting at some final heat distribution and working backward to the heat distribution at earlier times. Intuitively this can be seen to be ill-posed because many different sets of initial data can give rise to very similar solutions at later times since any high-frequency components in initial data for the heat equation are very rapidly smoothed out. We can formally solve the backward heat equation in Fourier space with the expression (E.28), but for $\kappa < 0$ each Fourier mode is growing exponentially in time instead of decaying. Exponential growth in itself doesn't make the problem ill posed—many well-posed equations have exponentially growing solutions—but the problem with (E.28) is that the growth rate depends on the wave number ξ and increases without bound with ξ. We can make an infinitesimal high-frequency perturbation to the initial data that will make an order 1 change in the solution at some fixed time t. Hence the solution to the backward heat equation does not depend continuously on the data.

E.3.5 More general parabolic equations

Consider a second order parabolic equation $u_t = Lu$ in N space dimensions as defined in Section E.1.3. For simplicity, just consider the second order part of the system, so

$$L = \sum_{j,k=1}^{N} A_{jk} \frac{\partial^2}{\partial x_j \partial x_k}, \tag{E.38}$$

where the $N \times N$ coefficient matrix A is symmetric positive definite. Let $\xi = (\xi_1, \ \ldots, \ \xi_N)$ be a wave number vector, one for each space dimension, so that a general Fourier mode has the form $e^{i\xi \cdot x}$, where $x = (x_1, \ \ldots, \ x_N)$. Let $\hat{u}(\xi, t)$ be the Fourier transform of $u(x, t)$ in all space dimensions, defined by

$$\hat{u}(\xi, t) = \frac{1}{\sqrt{2\pi}} \int u(x, t) e^{-i\xi \cdot x} \, dx, \tag{E.39}$$

[2] A problem is said to be *well posed* (in the sense of Hadamard) if it has a unique solution for every valid set of data and if the solution depends continuously on the data.

where the integral is now over all of N-dimensional space. Then it can be verified that the parabolic equation $u_t = Lu$ transforms to

$$\hat{u}_t(\xi, t) = -\xi^T A \xi \, \hat{u}(\xi, t). \tag{E.40}$$

The requirement that A be positive definite is just what is needed to ensure that all Fourier modes decay, giving a well-posed problem. If $\xi^T A \xi < 0$ for some vector ξ, then it is also negative for any scalar multiple $\alpha\xi$ of this wave vector, and there would be exponential growth of some Fourier modes with arbitrarily large growth rate $\alpha^2 \xi^T A \xi$. As observed for the backward heat equation, this would give an ill-posed problem. (For the heat equation with $N = 1$, the matrix A is just the scalar coefficient κ.)

E.3.6 Dispersive waves

Now consider the equation

$$u_t = u_{xxx}. \tag{E.41}$$

Fourier transforming now leads to the ODE

$$\hat{u}_t(\xi, t) = -i\xi^3 \hat{u}(\xi, t),$$

so

$$\hat{u}(\xi, t) = e^{-i\xi^3 t} \hat{\eta}(\xi).$$

This has a character similar to advection problems in that $|\hat{u}(\xi, t)| = |\hat{\eta}(\xi)|$ for all time and each Fourier component maintains its original amplitude. However, when we recombine with the inverse Fourier transform we obtain

$$u(x, t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi) e^{i\xi(x - \xi^2 t)} \, d\xi, \tag{E.42}$$

which shows that the Fourier component with wave number ξ is propagating with velocity ξ^2. In the advection equation all Fourier components propagate with the same speed a, and hence the shape of the initial data is preserved with time. The solution is the initial data shifted over a distance at.

With (E.41), the shape of the initial data in general will not be preserved, unless the data is simply a single Fourier mode. This behavior is called *dispersive* since the Fourier components disperse relative to one another. Smooth data typically lead to oscillatory solutions since the cancellation of high wave number modes that smoothness depends on will be lost as these modes shift relative to one another. See, for example, Whitham [102] for an extensive discussion of dispersive waves.

Extending this analysis to an equation of the form

$$u_t + au_x + bu_{xxx} = 0, \tag{E.43}$$

we find that the solution can be written as

$$u(x, t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi) e^{i\xi(x - (a - b\xi^2)t)} \, d\xi,$$

where $\hat{\eta}(\xi)$ is the Fourier transform of the initial data $\eta(x)$. Each Fourier mode $e^{i\xi x}$ propagates at velocity $a - b\xi^2$, called the phase velocity of this wave number. In general the initial data $\eta(x)$ is a linear combination of infinitely many different Fourier modes. For $b \neq 0$ these modes propagate at different speeds relative to one another. Their peaks and troughs will be shifted relative to other modes and they will no longer add up to a shifted version of the original data. The waves are called dispersive since the different modes do not move in tandem. Moreover, we will see below that the "energy" associated with different wave numbers also disperses.

E.3.7 Even- versus odd-order derivatives

Note that odd-order derivatives $\partial_x,\ \partial_x^3,\ \ldots$ (as in the advection equation or the dispersive equation (E.41)) have pure imaginary eigenvalues $i\xi,\ -i\xi^3,\ \ldots$, which results in Fourier components that propagate with their magnitude preserved. Even-order derivatives, such as the ∂_x^2 in the heat equation, have real eigenvalues ($-\xi^2$ for the heat equation), which results in exponential decay of the eigencomponents. Another such equation is

$$u_t = -u_{xxxx},$$

in which case $\hat{u}(\xi, t) = e^{-\xi^4 t}\hat{\eta}(\xi)$. Solutions to this equation behave much like solutions to the heat equation but with even more rapid damping of oscillatory data.

Another interesting example is

$$u_t = -u_{xx} - u_{xxxx}, \tag{E.44}$$

for which

$$\hat{u}(\xi, t) = e^{(\xi^2 - \xi^4)t}\hat{\eta}(\xi). \tag{E.45}$$

Note that the u_{xx} term has the "wrong" sign—it looks like a backward heat equation and there is exponential growth of some wave numbers. But for $|\xi| > 1$ the fourth order diffusion dominates and $\hat{u}(\xi, t) \to 0$ exponentially fast. For all ξ we have $|\hat{u}(\xi, t)| \leq e^{t/4}|\hat{\eta}(\xi)|$ (since $\xi^2 - \xi^4 \leq 1/4$ for all ξ) and the equation is well posed.

The Kuramoto–Sivashinsky equation (11.13) involves terms of this form, and the exponential growth of some wave numbers leads to chaotic behavior and interesting pattern formation.

E.3.8 The Schrödinger equation

The discussion of the previous section supposed that $u(x, t)$ is a real-valued function. The vacuum Schrödinger equation for a complex wave function $\psi(x, t)$ has the form (dropping some physical constants)

$$i\psi_t(x, t) = -\psi_{xx}(x, t). \tag{E.46}$$

This involves a second derivative, but note the crucial fact that ψ_t is multiplied by i. Fourier transforming thus gives

$$i\hat{\psi}_t(\xi, t) = \xi^2\hat{\psi}(\xi, t),$$

so

$$\hat{\psi}(\xi, t) = e^{-i\xi^2 t}\hat{\psi}(\xi, 0)$$

and

$$\psi(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\psi}(\xi,0) e^{i\xi(x-(a-\xi)t)}\, d\xi.$$

Hence the Schrödinger equation has dispersive wavelike solutions in spite of the even-order derivative.

E.3.9 The dispersion relation

Consider a general real-valued PDE of the form

$$u_t + a_1 u_x + a_3 u_{xxx} + a_5 u_{xxxxx} + \cdots = 0 \tag{E.47}$$

that contains only odd-order derivative in x. The Fourier transform $\hat{u}(\xi,t)$ satisfies

$$\hat{u}_t(\xi,t) + a_1 i\xi \hat{u}(\xi,t) - a_3 i\xi^3 \hat{u}(\xi,t) + a_5 i\xi^5 \hat{u}(\xi,t) + \cdots = 0,$$

and hence

$$\hat{u}(\xi,t) = e^{-i\omega t}\hat{\eta}(\xi),$$

where

$$\omega = \omega(\xi) = a_1\xi - a_3\xi^3 + a_5\xi^5 - \cdots. \tag{E.48}$$

The solution can thus be written as

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi) e^{i(\xi x - \omega(\xi)t)}\, d\xi. \tag{E.49}$$

The relation (E.48) between ξ and ω is called the *dispersion relation* for the PDE. Once we've gone through this full Fourier analysis a couple times we realize that since the different wave numbers ξ decouple, the dispersion relation for a linear PDE can be found simply by substituting a single Fourier mode of the form

$$u(x,t) = e^{-i\omega t} e^{i\xi x} \tag{E.50}$$

into the PDE and canceling the common terms to find the relation between ω and ξ. This is similar to what is done when applying von Neumann analysis for analyzing finite difference methods (see Section 9.6). In fact, there is a close relation between determining the dispersion relation and doing von Neumann analysis, and the dispersion relation for a finite difference method can be defined by an approach similar to von Neumann analysis by setting $U_j^n = e^{-i\omega nk} e^{i\xi jh}$, i.e., using $e^{-i\omega k}$ in place of g.

Note that this same analysis can be done for equations that involve even-order derivatives, such as

$$u_t + a_1 u_x + a_2 u_{xx} + a_3 u_{xxx} + a_4 u_{xxxx} + \cdots = 0,$$

but then we find that

$$\omega(\xi) = a_1\xi + ia_2\xi^2 - a_3\xi^3 - ia_4\xi^4 - \cdots.$$

The even-order derivatives give imaginary terms in $\omega(\xi)$ so that

$$e^{-i\omega t} = e^{(a_2\xi^2 - a_4\xi^4 + \cdots)t}\, e^{i(a_1\xi - a_3\xi^3 + \cdots)t}.$$

The first term gives exponential growth or decay, as we expect from Section E.3.3, rather than dispersive behavior. For this reason we call the PDE (purely) dispersive only if $\omega(\xi)$ is real for all $\xi \in \mathbb{R}$. Informally we also speak of an equation like $u_t = u_{xx} + u_{xxx}$ as having both a diffusive and a dispersive term.

In the purely dispersive case (E.47) the single Fourier mode (E.50) can be written as

$$u(x,t) = e^{i\xi(x-(\omega/\xi)t)}$$

and so a pure mode of this form propagates at velocity ω/ξ. This is called the *phase velocity* for this wave number,

$$c_p(\xi) = \frac{\omega(\xi)}{\xi}. \tag{E.51}$$

Most physical problems have data $\eta(x)$ that is not simply sinusoidal for all $x \in (-\infty, \infty)$ but instead is concentrated in some restricted region, e.g., a Gaussian pulse as in (E.29),

$$\eta(x) = e^{-\beta x^2}. \tag{E.52}$$

The Fourier transform of this function is a Gaussian in ξ, (E.30),

$$\hat{\eta}(\xi) = \frac{1}{\sqrt{2\beta}} e^{-\xi^2/4\beta}. \tag{E.53}$$

Note that for β small, $\eta(x)$ is a broad and smooth Gaussian with a Fourier transform that is sharply peaked near $\xi = 0$. In this case $\eta(x)$ consists primarily of low wave number smooth components. For β large $\eta(x)$ is sharply peaked while the transform is broad. More high wave number components are needed to represent the rapid spatial variation of $\eta(x)$ in this case.

If we solve the dispersive equation with data of this form, then the different modes propagate at different phase velocities and will no longer sum to a Gaussian, and the solution evolves as shown in Figure E.1, forming "dispersive ripples." Note that for large times it is apparent that the wave length of the ripples is changing through this wave and that the energy associated with the low wave numbers is apparently moving faster than the energy associated with larger wave numbers. The propagation velocity of this energy is not, however, the phase velocity $c_p(\xi)$. Instead it is given by the *group velocity*

$$c_g(\xi) = \frac{d\omega(\xi)}{d\xi}. \tag{E.54}$$

For the advection equation $u_t + au_x = 0$ the dispersion relation is $\omega(\xi) = a\xi$ and the group velocity agrees with the phase velocity (since all waves propagate at the same velocity a), but more generally the two do not agree. For the dispersive equation (E.43), $\omega(\xi) = a\xi - b\xi^3$ and we find that

$$c_g(\xi) = a - 3b\xi^2,$$

whereas

$$c_p(\xi) = a - b\xi^2.$$

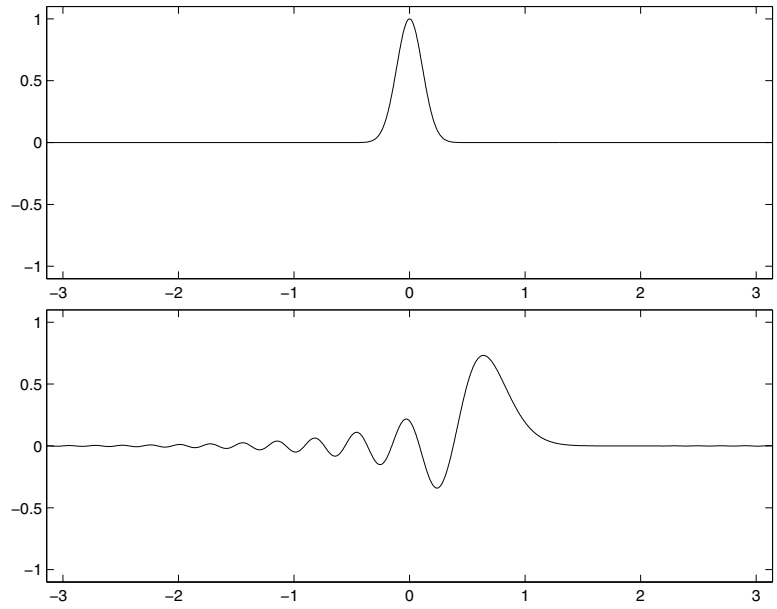

Figure E.1. *Gaussian initial data propagating with dispersion.*

E.3.10 Wave packets

The notion and importance of group velocity is easiest to appreciate by considering a "wave packet" with data of the form

$$\eta(x) = e^{i\xi_0 x} e^{-\beta x^2} \tag{E.55}$$

or the real part of such a wave,

$$\eta(x) = \cos(\xi_0 x) e^{-\beta x^2}. \tag{E.56}$$

This is a single Fourier mode modulated by a Gaussian, as shown in Figure E.2.

The Fourier transform of (E.55) is

$$\hat{\eta}(\xi) = \frac{1}{\sqrt{2\beta}} e^{-(\xi-\xi_0)^2/4\beta}, \tag{E.57}$$

a Gaussian centered about $\xi - \xi_0$. If the packet is fairly broad (β small), then the Fourier transform is concentrated near $\xi = \xi_0$ and hence the propagation properties of the wave packet are well approximated in terms of the phase velocity $c_p(\xi)$ and the group velocity $c_g(\xi)$. The wave crests propagate at the speed $c_p(\xi_0)$, while the envelope of the packet propagates at the group velocity $c_g(\xi_0)$.

To get some idea of why the packet propagates at the group velocity, consider the expression (E.49),

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi) e^{i(\xi x - \omega(\xi)t)}\, d\xi.$$

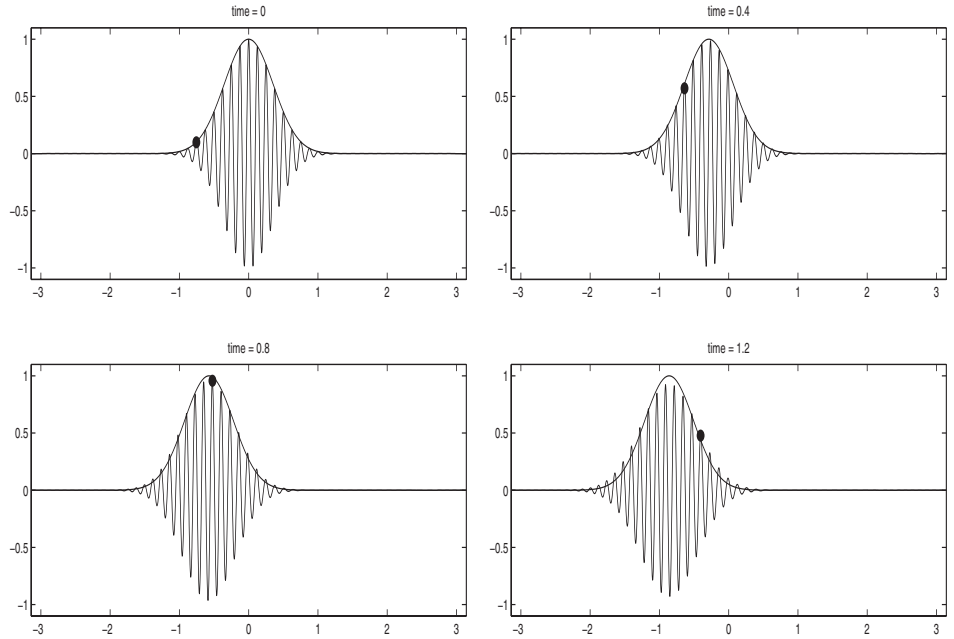

Figure E.2. *The oscillatory wave packet satisfies the dispersive equation* $u_t + au_x + bu_{xxx} = 0$. *Also shown is a black dot attached to one wave crest, translating at the phase velocity* $c_p(\xi_0)$, *and a Gaussian that is translating at the group velocity* $c_g(\xi_0)$. *Shown for a case in which* $c_g(\xi_0) < 0 < c_p(\xi_0)$.

For a concentrated packet, we expect $u(x,t)$ to be very close to zero for most x, except near some point ct, where c is the propagation velocity of the packet. To estimate c we will ask where this integral could give something nonzero. At each fixed x the integral is a Gaussian in ξ (the function $\hat{\eta}(\xi)$) multiplied by an oscillatory function of ξ (the exponential factor). Integrating this product will given essentially zero at a particular x provided the oscillatory part is oscillating rapidly enough in ξ that it averages out to zero, although it is modulated by the Gaussian $\hat{\eta}(\xi)$. This happens provided the function $\xi x - \omega(\xi)t$ appearing as the phase in the exponential is rapidly varying as a function of ξ at this x. Conversely, we expect the integral to be significantly different from zero only near points x where this phase function is stationary, i.e., where

$$\frac{d}{d\xi}(\xi x - \omega(\xi)t) = 0.$$

This occurs at

$$x = \omega'(\xi)t,$$

showing that the wave packet propagates at the group velocity $c_g = \omega'(\xi)$. This approach to studying oscillatory integrals is called the "method of stationary phase" and is useful in other applications as well. See, for example, [55], [102] for more on dispersive waves.

Bibliography

[1] A. Abdulle. Fourth order Chebyshev methods with recurrence relation. *SIAM J. Sci. Comput.*, 23:2041–2054, 2002. (cited on *176*)

[2] A. Abdulle and A. A. Medovikov. Second order Chebyshev methods based on orthogonal polynomials. *Numer. Math.*, 90:1–18, 2001. (cited on *176*)

[3] L. M. Adams, R. J. LeVeque, and D. M. Young. Analysis of the SOR iteration for the 9-point Laplacian. *SIAM J. Numer. Anal.*, 25:1156–1180, 1988. (cited on 77)

[4] U. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Prentice–Hall, Englewood Cliffs, NJ, 1988. (cited on *38, 52, 55*)

[5] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, 1998. (cited on *113, 129*)

[6] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri. Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations. *Appl. Numer. Math.*, 25:151–167, 1997. (cited on *240*)

[7] U. M. Ascher, S. J. Ruuth, and B. T. R. Wetton. Implicit-explicit methods for time-dependent partial differential equations. *SIAM J. Numer. Anal.*, 32:797–823, 1995. (cited on *240*)

[8] G. Beylkin, J. M. Keiser, and L. Vozovoi. A new class of time discretization schemes for the solution of nonlinear pdes. *J. Comput. Phys.*, 147:362–387, 1998. (cited on *241*)

[9] A. Bourlioux, A. T. Layton, and M. L. Minion. High-order multi-implicit spectral deferred correction methods for problems of reactive flow. *J. Comput. Phys.*, 189:651–675, 2003. (cited on *239*)

[10] J. P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover, New York, 2001. (cited on *58*)

[11] W. L. Briggs, V. Emden Henson, and S. F. McCormick. *A Multigrid Tutorial, 2nd ed.* SIAM, Philadelphia, 2000. (cited on *103*)

[12] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations,* John Wiley, Chichester, UK, 2003. (cited on *113*)

[13] J. C. Butcher. *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*. John Wiley, Chichester, UK, 1987. (cited on *128, 171*)

[14] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral Methods in Fluid Dynamics*. Springer, New York, 1988. (cited on *58*)

[15] E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. McGraw–Hill, New York, 1955. (cited on *116*)

[16] S. D. Conte and C. de Boor. *Elementary Numerical Analysis*. McGraw–Hill, New York, 1980. (cited on *264*)

[17] R. Courant, K. O. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Ann.*, 100:32–74, 1928. (cited on *216*)

[18] R. Courant, K. O. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM Journal*, 11:215–234, 1967. (cited on *216*)

[19] S. M. Cox and P. C. Matthews. Exponential time differencing for stiff systems. *J. Comput. Phys.*, 176:430–455, 2002. (cited on *241, 242*)

[20] C. F. Curtiss and J. O. Hirschfelder. Integration of stiff equations. *Proc. Nat. Acad. Sci. USA*, 38:235–243, 1952. (cited on *173*)

[21] G. Dahlquist. A special stability problem for linear multistep methods. *BIT*, 3:27–43, 1963. (cited on *171*)

[22] G. Dahlquist, Convergence and stability in the numerical integration of ordinary differential equations, *Math. Scand.,* 4:33–53, 1956. (cited on *147*)

[23] G. Dahlquist and R. LeVeque. Linear difference equations and matrix theorems. Lecture Notes, Royal Institute of Technology (KTH), Stockholm, `http://www.amath.washington.edu/~rjl/pubs/kth81`, (1981). (cited on *309, 310*)

[24] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006. (cited on *68*)

[25] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulas. *J. Comput. Appl. Math.*, 6:19–26, 1980. (cited on *130*)

[26] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Trans. AMS*, 82:421–439, 1956. (cited on *199*)

[27] D. R. Durran. *Numerical Methods for Wave Equations in Geophysical Fluid Dynamics*. Springer, New York, 1999. (cited on *xv*)

[28] A. Dutt, L. Greengard, and V. Rokhlin. Spectral deferred correction methods for ordinary differential equations. *BIT*, 40:241–266, 2000. (cited on *58*, *239*)

[29] B. Fornberg. *A Practical Guide to Pseudospectral Methods*. Cambridge University Press, London, 1996. (cited on *58*)

[30] B. Fornberg. Calculation of weights in finite difference formulas. *SIAM Rev.*, 40:685–691, 1998. (cited on *11*)

[31] F. G. Friedlander and M. Joshi. *Introduction to the Theory of Distributions*. Cambridge University Press, London, 1998. (cited on *24*)

[32] E. Gallopoulos and Y. Saad. Efficient solution of parabolic equations by Krylov approximation methods. *SIAM J. Sci. Statist. Comput.*, 13:1236–1264, 1992. (cited on *242*)

[33] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice–Hall, Englewood Cliffs, NJ, 1971. (cited on *113*, *173*)

[34] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive-Definite Systems*. Prentice–Hall, Englewood Cliffs, NJ, 1981. (cited on *68*)

[35] G. H. Golub and C. F. Van Loan. *Matrix Computations*, 3rd ed. Johns Hopkins University Press, Baltimore, 1996. (cited on *67*, *250*, *271*)

[36] G. H. Golub and D. P. O'Leary. Some history of the conjugate gradient and Lanczos algorithms: 1948–1976. *SIAM Rev.*, 31:50–102, 1989. (cited on *86*)

[37] G. H. Golub and J. M. Ortega. *Scientific Computing and Differential Equations: An Introduction to Numerical Methods*. Academic Press, New York, 1992. (cited on *76*)

[38] D. Gottlieb and S. A. Orszag. *Numerical Analysis of Spectral Methods*. CBMS-NSF Regional Conference Series in Applied Mathematics, 26, SIAM, Philadelphia, 1977. (cited on *58*)

[39] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia, 1997. (cited on *78*, *87*, *88*, *99*, *100*)

[40] B. Gustafsson, H.-O. Kreiss, and J. Oliger. *Time Dependent Problems and Difference Methods*. John Wiley, New York, 1995. (cited on *xv*, *191*, *214*, *228*)

[41] W. Hackbusch. *Multigrid Methods and Applications*. Springer-Verlag, Berlin, 1985. (cited on *103*)

[42] L. A. Hageman and D. M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981. (cited on *76*)

[43] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems*. Springer-Verlag, Berlin, Heidelberg, 1987. (cited on *113*, *128*, *129*, *148*)

[44] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*. Springer-Verlag, New York, 1993. (cited on *113, 128, 166, 171, 173*)

[45] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations*. John Wiley, New York, 1962. (cited on *113*)

[46] M. R. Hestenes and E. Stiefel. Method of conjugate gradients for solving linear equations. *J. Res. Nat. Bureau Standards*, 49:409–436, 1952. (cited on *86*)

[47] D. J. Higham and L. N. Trefethen. Siffness of ODEs. *BIT*, 33:285–303, 1993. (cited on *156, 228*)

[48] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput.*, 19:1552–1574, 1998. (cited on *241, 242*)

[49] A. Iserles. *Numerical Analysis of Differential Equations*. Cambridge University Press, Cambridge, UK, 1996. (cited on *xv*)

[50] A. Iserles. Think globally, act locally: Solving highly-oscillatory ordinary differential equations. *Appl. Numer. Math.*, 43:145–160, 2002. (cited on *169*)

[51] A. Iserles and S. P. Norsett. *Order Stars*. Chapman and Hall, London, 1991. (cited on *166, 171*)

[52] D. C. Jesperson. Multigrid methods for partial differential equations. In *Studies in Numerical Analysis*, G. H. Golub, ed. MAA Studies in Mathematics, Vol. 24, 1984, pages 270–317. (cited on *103*)

[53] A.-K. Kassam and L. N. Trefethen. Fourth-order time-stepping for stiff PDEs. *SIAM J. Sci. Comput.*, 26:1214–1233, 2005. (cited on *241, 242, 286*)

[54] H. B. Keller. *Numerical Solution of Two Point Boundary Value Problems*. SIAM, Philadelphia, 1976. (cited on *38, 43, 55*)

[55] J. Kevorkian. *Partial Differential Equations*. Wadsworth & Brooks/Cole, Pacific Corove, CA, 1990. (cited on *14, 43, 216, 311, 314, 328*)

[56] J. Kevorkian and J. D. Cole. *Perturbation Methods in Applied Mathematics*. Springer, New York, 1981. (cited on *43*)

[57] D. A. Knoll and D. E. Keyes. Jacobian-free Newton-Krylov methods: A survey of approaches and applications. *J. Comput. Phys.*, 193:357–397, 2004. (cited on *102*)

[58] D. Kröner. *Numerical Schemes for Conservation Laws*. Wiley-Teubner, New York, 1997. (cited on *201*)

[59] J. D. Lambert. *Computational Methods in Ordinary Differential Equations*. John Wiley, New York, 1973. (cited on *113, 173*)

[60] J. D. Lambert. *Numerical Methods for Ordinary Differential Systems: The initial value Problem*. John Wiley, Chichester, 1991. (cited on *113*)

[61] P. D. Lax. *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*. CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, 11, 1973. (cited on *314*)

[62] R. Lehoucq, K. Maschhoff, D. Sorensen, and C. Yang. ARPACK software. `http://www.caam.rice.edu/software/ARPACK/(1997)`. (cited on *100*)

[63] S. K. Lele. Compact difference schemes with spectral-like resolution. *J. Comput. Phys.*, 103:16–42, 1992. (cited on *230*)

[64] R. J. LeVeque. CLAWPACK software, 2006. `http://www.amath.washington.edu/~claw`. (cited on *201*)

[65] R. J. LeVeque. Intermediate boundary conditions for time-split methods applied to hyperbolic partial differential equations. *Math. Comput.*, 47:37–54, 1986. (cited on *239*)

[66] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, London, 2002. (cited on *201*, *214*, *216*, *226*, *230*, *231*, *314*)

[67] R. J. LeVeque and L. N. Trefethen. Fourier analysis of the SOR iteration. *IMA J. Numer. Anal.*, 8:273–279, 1988. (cited on *76*)

[68] D. Levy and E. Tadmor. From semidiscrete to fully discrete: Stability of Runge–Kutta schemes by the energy method. *SIAM Rev.*, 40:40–73, 1998. (cited on *191*)

[69] A. A. Medovikov. High order explicit methods for parabolic equations. *BIT*, 38:372–390, 1998. (cited on *176*)

[70] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Rev.*, 20:801–836, 1978. (cited on *241*)

[71] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45:3–49, 2003. (cited on *241*)

[72] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, Cambridge, UK, 1994. (cited on *xv*)

[73] J. D. Murray. *Mathematical Biology*. Springer-Verlag, Berlin, Heidelberg, 1989. (cited on *235*)

[74] L. R. Petzold, L. O. Jay, and J. Yen. Numerical solution of highly oscillatory ordinary differential equations. *Acta Numer.*, 6:437–484, 1997. (cited on *169*)

[75] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-Value Problems*. Wiley-Interscience, New York, 1967. (cited on *190, 191, 192, 214, 307*)

[76] J. W. Ruge and K. Stüben. *Algebraic multigrid*, in multigrid methods, S.F. McCormick, ed., SIAM, Philadelphia, 1987, pages 73–103. (cited on *110*)

[77] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 29:209–228, 1992. (cited on *242*)

[78] L. F. Shampine and M. W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comput.*, 18:1–22, 1997. (cited on *129, 130, 135*)

[79] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* Technical report, available from `http://www.cs.cmu.edu/~jrs/jrspapers.html` (1994). (cited on *78*)

[80] B. P. Sommeijer, L. F. Shampine, and J. G. Verwer. RKC: An explicit solver for parabolic PDEs. *J. Comput. Appl. Math.*, 88:315–326, 1997. (cited on *176, 178*)

[81] M. N. Spijker. On a conjecture by LeVeque and Trefethen related to the Kreiss matrix theorem. *BIT*, 31:551–555, 1991. (cited on *293*)

[82] G. W. Stewart. *Introduction to Matrix Computations*. Academic Press, New York, 1973. (cited on *67*)

[83] G. Strang. On the construction and comparison of difference schemes. *SIAM J. Numer. Anal.*, 5:506–517, 1968. (cited on *238*)

[84] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*, 2nd ed. SIAM, Philadelphia, 2004. (cited on *xv, 191, 192, 228*)

[85] J. C. Strikwerda and B. A. Wade. A survey of the Kreiss matrix theorem for power bounded families of matrices and its extensions. In *Linear Operators*, Banach Center Publ., 38, Polish Acad. Sci., Warsaw, 1997, pages 329–360. (cited on *307*)

[86] K. Stüben. A review of algebraic multigrid. *J. Comput. Appl. Math.*, 128:281–309, 2001. (cited on *110*)

[87] Paul N. Swarztrauber. Fast Poisson Solvers. In *Studies in Numerical Analysis*, volume 24, G. H. Golub, ed., Mathematical Association of America, Washington, D.C., 1984, pp. 319–370. (cited on *68*)

[88] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer-Verlag, Berlin, Heidelberg, 1997. (cited on *201*)

[89] L. N. Trefethen. Group velocity in finite difference schemes. *SIAM Rev.*, 24:113–136, 1982. (cited on *228*)

[90] L. N. Trefethen. *Spectral Methods in MATLAB*. SIAM, Philadelphia, 2000. (cited on *58, 264*)

[91] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*. SIAM, Philadelphia, 1997. (cited on *xiv, 67, 78, 88, 94, 100, 250*)

[92] L. N. Trefethen and M. Embree. *Spectra and Pseudospectra*. Princeton University Press, Princeton, NJ, 2005. (cited on *74, 228, 231, 293, 299, 304, 307*)

[93] A. Tveito and R. Winther. *Introduction to Partial Differential Equations: A Computational Approach*. Springer, New York, 1998. (cited on *xv*)

[94] P. J. van der Houwen and B. P. Sommeijer. On the internal stability of explicit m-stage Runge–Kutta methods for large values of m. *Z. Angew. Math. Mech.*, 60:479–485, 1980 (in German). (cited on *178*)

[95] H. A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992. (cited on *100*)

[96] R. S. Varga. *Matrix Iterative Analysis*. Prentice–Hall, Englewood Cliffs, NJ, 1962. (cited on *76*)

[97] J. G. Verwer. Explicit Runge-Kutta methods for parabolic differential equations. *Appl. Numer. Math.*, 22:359, 1996. (cited on *176, 177, 178*)

[98] G. Wanner. Order stars and stability. In *The State of the Art in Numerical Analysis*, A. Iserles and M. J. D. Powell, eds., Clarendon Press Oxford, UK, 1987, pp. 451–471. (cited on *166*)

[99] G. Wanner, E. Hairer, and S. P. Nørsett. Order stars and stability theorems. *BIT*, 18:475–489, 1978. (cited on *165*)

[100] R. Warming and B. Hyett. The modified equation approach to the stability and accuracy analysis of finite-difference methods. *J. Comput. Phys.*, 14:159–179, 1974. (cited on *219*)

[101] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley, New York, 1992. (cited on *103*)

[102] G. Whitham. *Linear and Nonlinear Waves*. Wiley-Interscience, New York, 1974. (cited on *314, 323, 328*)

[103] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, London, 1965. (cited on *278*)

[104] T. Wright. Eigtool, 2002. `http://web.comlab.ox.ac.uk/projects/pseudospectra/eigtool/`. (cited on *304*)

[105] D. M. Young. *Iterative Methods for Solving Partial Differential Equations of Elliptic Type*. Ph.D. thesis, Harvard University, Cambridge, 1950. (cited on *76*)

[106] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, New York, 1971. (cited on *76*)

Index