Finite Difference Methods for Differential Equations

Randall J. LeVeque

DRAFT VERSION for use in the course
AMath 585–586
University of Washington
Version of September, 2005

WARNING: These notes are incomplete and may contain errors.
They are made available primarily for students in my courses.
Please contact me for other uses.
`rjl@amath.washington.edu`

Contents

Part I

Basic Text

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) which 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 by finite difference approximations. This gives a large algebraic system of equations to be solved in place of the differential equation, something that is easily solved 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

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

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

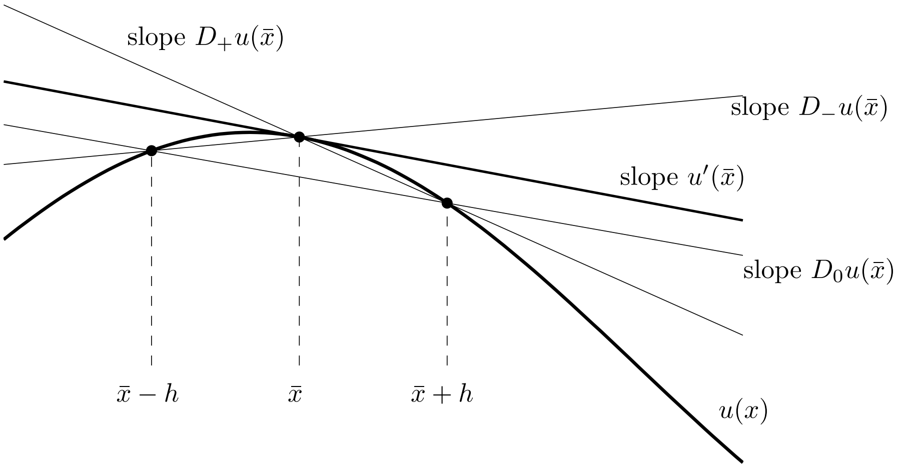


Figure 1.1: Various approximations to $u'(\bar{x})$ interpreted as the slope of secant lines.

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

```
       h           D+            D-            D0            D3

  1.0000e-01  -4.2939e-02   4.1138e-02  -9.0005e-04   6.8207e-05
  5.0000e-02  -2.1257e-02   2.0807e-02  -2.2510e-04   8.6491e-06
  1.0000e-02  -4.2163e-03   4.1983e-03  -9.0050e-06   6.9941e-08
  5.0000e-03  -2.1059e-03   2.1014e-03  -2.2513e-06   8.7540e-09
  1.0000e-03  -4.2083e-04   4.2065e-04  -9.0050e-08   6.9979e-11
```

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 $\bar{x} = 1$, so 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 though 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 either.

We see that

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

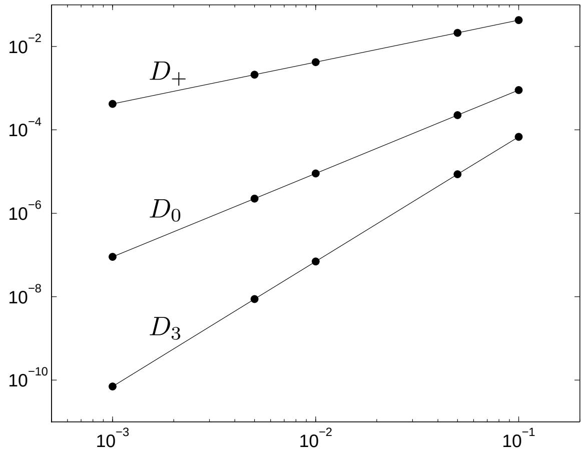


Figure 1.2: The errors in $Du(\bar{x})$ from Table 1.1 plotted against h on a log-log scale.

confirming that these methods are first order, second order, and third order, 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 A1.2 of Appendix A1 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

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

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 of 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.

In order to analyze D_3u we need to also expand $u(\bar{x}-2h)$ as

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

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

$$D_3u(\bar{x}) = u'(\bar{x}) + \frac{1}{12}h^3u''''(\bar{x}) + O(h^4). \tag{1.8}$$

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

$$D_2u(\bar{x}) = au(\bar{x}) + bu(\bar{x}-h) + cu(\bar{x}-2h). \tag{1.9}$$

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

$$\begin{aligned} D_2u(\bar{x}) \quad = \quad & (a+b+c)u(\bar{x}) - (b+2c)hu'(\bar{x}) + \frac{1}{2}(b+4c)h^2u''(\bar{x}) \\ & - \frac{1}{6}(b+8c)h^3u'''(\bar{x}) + \cdots . \end{aligned}$$

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

$$D_2u(\bar{x}) = \frac{1}{2h}[3u(\bar{x}) - 4u(\bar{x}-h) + u(\bar{x}-2h)]. \tag{1.11}$$

The error in this approximation is clearly

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

1.3 Polynomial interpolation

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).

1.4 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

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

Again, since this is a symmetric centered approximation all of 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 $\bar{x}+h$.

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

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

then we find that

$$\hat{D}_0(\hat{D}_0u(\bar{x})) = \frac{1}{h}\left(\left(\frac{u(\bar{x}+h)-u(\bar{x})}{h}\right) - \left(\frac{u(\bar{x})-u(\bar{x}-h)}{h}\right)\right) = D^2u(\bar{x}).$$

1.5 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 way.

Example 1.4. As an example, here are two different approximations to $u'''(\bar{x})$. The first one is uncentered and first order accurate:

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

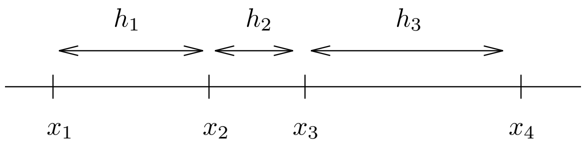
The next approximation is centered and second order accurate:

$$\begin{aligned} D_0D_+D_-u(\bar{x}) &= \frac{1}{2h^3}(u(\bar{x}+2h) - 2u(\bar{x}+h) + 2u(\bar{x}-h) - u(\bar{x}-2h)) \\ &= u'''(\bar{x}) + \frac{1}{4}h^2u'''''(\bar{x}) + O(h^4). \end{aligned}$$

Finite difference approximations of the sort derived above are the basic building blocks of finite difference methods for solving differential equations.

1.6 Exercises

Exercise 1.1 *Consider the nonuniform grid:*



1. *Use polynomial interpolation to derive a finite difference approximation for $u''(x_2)$ that is as accurate as possible for smooth functions u, based on the four values $U_1 = u(x_1),\ \ldots,\ U_4 = u(x_4)$. Give an expression for the dominant term in the error.*

2. *Verify your expression for the error by testing your formula with a specific function and various values of $h_1,\ h_2,\ h_3$.*

3. *Can you define an "order of accuracy" for your method in terms of $h = \max(h_1,\ h_2,\ h_3)$? To get a better feel for how the error behaves as the grid gets finer, do the following. Take a large number (say 500) of different values of H spanning two or three orders of magnitude, choose $h_1,\ h_2$, and h_3 as random numbers in the interval $[0, H]$ and compute the error in the resulting approximation. Plot these values against H on a log-log plot to get a scatter plot of the behavior as $H \to 0$. (Note: in* `matlab` *the command* `h = H * rand(1)` *will produce a single random number uniformly distributed in the range* `[0,H]`*.) Of course these errors will not lie exactly on a straight line since the values of h_k may vary quite a lot even for H's that are nearby, but you might expect the upper limit of the error to behave reasonably.*

4. *Estimate the "order of accuracy" by doing a least squares fit of the form*

$$\log(E(H)) = K + p\log(H)$$

to determine K and p based on the 500 data points. Recall that this can be done by solving the following linear system in the least squares sense:

$$\begin{bmatrix} 1 & \log(H_1) \\ 1 & \log(H_2) \\ \vdots & \vdots \\ 1 & \log(H_{500}) \end{bmatrix} \begin{bmatrix} K \\ p \end{bmatrix} = \begin{bmatrix} \log(E(H_1)) \\ \log(E(H_2)) \\ \vdots \\ \log(E(H_{500})) \end{bmatrix}.$$

In `matlab` *a rectangular system $Ax = b$ can be solved in the least squares sense by* `x = A\b`.

Exercise 1.2 *Use the method of undetermined coefficients to find a fourth-order accurate finite difference approximation to $u''(x)$ based on 5 equally spaced points,*

$$u''(x) = c_{-2}u(x-2h) + c_{-1}u(x-h) + c_0u(x) + c_1u(x+h) + c_2u(x+2h) + O(h^4).$$

Test your formula on some smooth function to verify that it gives the expected accuracy.

Chapter 2

Boundary Value Problems

We will first consider ordinary differential equations 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 study *elliptic partial differential equations* 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 2.15 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 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 $\psi < 0$). 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 [Kev90] 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, or whatever) 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 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 with we will consider the Dirichlet problem for equation (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 $u^0(x)$ approaches $u(x)$ (unless $u^0(x) \equiv u(x)$), but if we are only interested in computing the steady state solution itself, then we can set $u_t = 0$ in (2.2) and obtain an ordinary differential equation 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 in order to specify a unique solution. In our case we have the boundary conditions

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

Remark 2.1 *Having two boundary conditions does not necessarily guarantee there exists a unique solution for a general second order equation — see Section 2.13.*

The problem (2.4), (2.5) is called a *two-point boundary value problem* since one condition is specified at each of the two endpoints of the interval where the solution is desired. If instead we had 2 data values 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* instead. These problems are discussed in Chapter 6.

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 partial differential equation (2.2), as discussed in Chapter 12 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 two-point boundary value problem given by (2.4) and (2.5) directly. This is the first boundary value problem that we will study in detail, starting in the next section. Later in this chapter we will consider some other boundary value problems, 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 ordinary differential equation 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 *two-point boundary value problem* 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

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

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

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

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 A1 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 the boundary value problem 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* of the method and then use some form of *stability* to show that the *global error* can be bounded in terms of the local truncation error.

The global error simply refers to the error $U - \hat{U}$ that we are attempting to bound. The local truncation error (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 by 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 the equation (2.14) from the equation (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, but with the right hand side 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

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

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 $h \to 0$:

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

Then we will have

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

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

Definition 2.7.1 *Suppose a finite difference method for a linear boundary value problem 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*

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

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

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

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

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

Although this has been demonstrated only for the linear boundary value problem, 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 boundary value problem just discussed it is not at all clear how to check the condition (2.19) since these matrices get 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 different definitions of "stability" for different types of problems. The challenge in analyzing finite difference methods for new classes of problems is often 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 A2 contains a discussion of how the error in computed results can be estimated.

2.10 Stability in the 2-norm

Returning to the boundary value problem 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 Appendix A1):

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

(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 analysis is possible. We will also need to know these eigenvalues for other purposes when we discuss parabolic equations later.

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

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

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

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

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

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

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 $\tau^h_j \approx \frac{1}{12}h^2u''''(x_j)$, 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, though this is approximately equal to the function space norm of f'' defined using (A1.11).

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 by 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 (A1.16) we know that

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

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

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

since $\|\tau\|_\infty = O(h^2)$. As in the computation of the eigenvalues in the last section, we can only do this 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 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 boundary value problem. 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 very 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.

Let $e_j \in \mathbb{R}^m$ be the jth coordinate vector or *unit vector* with the value 1 as its jth element and all other elements equal to 0. If B is any matrix then the vector Be_j is simply the jth column of the matrix B. So the jth column of A^{-1} is $A^{-1}e_j$. Let's call this vector v for the time being. Then v is the solution of the linear system

$$Av = e_j. \tag{2.25}$$

This can be viewed as an approximation to a boundary value problem of the form (2.6),(2.7) where $\alpha = \beta = 0$ and $f(x_i) = 0$ unless $i = j$, with $f(x_j) = 1$. This may seem like a rather strange function f, but it corresponds to the delta function that is used in defining Green's functions (or more exactly to a delta function scaled by h). We will come back to the problem of determining the jth column of A^{-1} after a brief review of delta functions and Green's functions for the differential equation.

The delta function, $\delta(x)$, is not an ordinary function but rather the mathematical idealization of a sharply peaked function that is nonzero over an interval $(-\epsilon, \epsilon)$ near the origin and has the property that

$$\int_{-\infty}^{\infty} \phi_\epsilon(x)\,dx = \int_{-\epsilon}^{\epsilon} \phi_\epsilon(x)\,dx = 1. \tag{2.26}$$

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

$$H(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0. \end{cases} \tag{2.27}$$

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 $\phi_\epsilon(x)$.

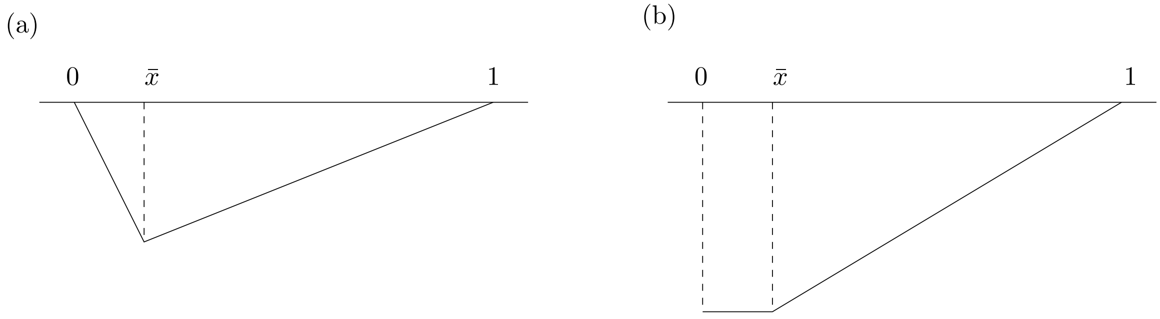


Figure 2.1: (a) The Green's function (2.28) for the Dirichlet problem. (b) The Green's function for the mixed problem with $u'(0) = u(1) = 0$ (see Exercise 2.5).

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 (2.26). 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 can be made rigorous mathematically through the use of "distribution theory", see for example [**?**]. For our purposes it suffices to think of the delta function as being a very sharply peaked function with total integral 1.

Now consider the function $G(x)$ shown in Figure 2.1(a):

$$G(x) = \begin{cases} (\bar{x}-1)x & \text{for } x \le \bar{x} \\ \bar{x}(x-1) & \text{for } x > \bar{x} \end{cases} \tag{2.28}$$

where $\bar{x}$ is some point in the interval $[0, 1]$. The derivative of $G(x)$ is piecewise constant:

$$G'(x) = \begin{cases} \bar{x}-1 & \text{for } x < \bar{x} \\ \bar{x} & \text{for } x > \bar{x}. \end{cases}$$

If we define $G'(\bar{x}) = \bar{x}$, then we can write this as

$$G'(x) = \bar{x} - 1 + H(x - \bar{x}).$$

Differentiating again then gives

$$G''(x) = \delta(x - \bar{x}).$$

It follows that the function $G(x)$ is the solution to the boundary value problem

$$u''(x) = \delta(x - \bar{x}) \quad \text{for } 0 < x < 1 \tag{2.29}$$

with homogeneous boundary conditions

$$u(0) = 0, \qquad u(1) = 0. \tag{2.30}$$

This function is called the *Green's function* for the problem (2.6),(2.7) and is generally written as $G(x; \bar{x})$ to show the dependence of the function $G(x)$ on the parameter $\bar{x}$, the location of the delta function source term.

The solution to the boundary value problem (2.6),(2.7) for more general $f(x)$ and boundary conditions can be written as

$$u(x) = \alpha(1-x) + \beta x + \int_0^1 G(x;\xi) f(\xi)\, d\xi. \tag{2.31}$$

This integral can be viewed as a linear combination of the functions $G(x;\xi)$ at each point ξ, with weights given by the strength of the source term at each such point.

Returning now to the question of determining the columns of the matrix A^{-1} by solving the systems (2.25), we see that the right hand side e_j can be viewed as a discrete version of the delta function, scaled by h. So the system (2.25) is a discrete version of the problem

$$v''(x) = h\delta(x - x_j)$$

with homogeneous boundary conditions, whose solution is $v(x) = hG(x;x_j)$. We therefore expect the vector v to approximate this function. In fact it is easy to confirm that we can obtain the vector v by simply evaluating the function $v(x)$ at each grid point, so $v_i = hG(x_i;x_j)$. This can be easily checked by verifying that multiplication by the matrix A gives the unit vector e_j.

If we now let G be the inverse matrix, $G = A^{-1}$, then the jth column of G is exactly the vector v found above, and so the elements of G are:

$$G_{ij} = hG(x_i;x_j) = \left\{ \begin{array}{ll} h(x_j-1)x_i & i = 1,\ 2,\ \ldots,\ j \\ h(x_i-1)x_j & i = j,\ j+1,\ \ldots,\ m. \end{array} \right. \tag{2.32}$$

Note that each of the elements of G is bounded by h in magnitude. From this we obtain an explicit bound on the max-norm of G:

$$\|A^{-1}\|_\infty = \max_{1\le i\le m} \sum_{j=1}^m |G_{ij}| \le mh < 1.$$

This is uniformly bounded as $h \to 0$ and so the method is stable in the max-norm. Since $\|\tau\|_\infty = O(h^2)$, the method is second order accurate in the max-norm and the pointwise error at each grid point is $O(h^2)$.

Note, by the way, that the representation (2.31) of the true solution $u(x)$ as a superposition of the Green's functions $G(x;\xi)$, weighted by the values of the right hand side $f(\xi)$, has a direct analog for the solution U to the difference equation $AU = F$. The solution is $U = A^{-1}F = Gf$, which can be written as

$$U_i = \sum_{j=1}^m G_{ij} F_j.$$

Using the form of F from (2.10) and the expression (2.32) shows that

$$\begin{array}{rcl} U_i & = & -\dfrac{\alpha}{h^2} G_{i1} - \dfrac{\beta}{h^2} G_{im} + \displaystyle\sum_{j=1}^m G_{ij} f(x_j) \\ & = & \alpha(1-x_i) + \beta x_i + h \displaystyle\sum_{j=1}^m G(x_i;x_j) f(x_j), \end{array}$$

which is simply the discrete form of (2.31).

2.12 Neumann boundary conditions

Suppose now 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

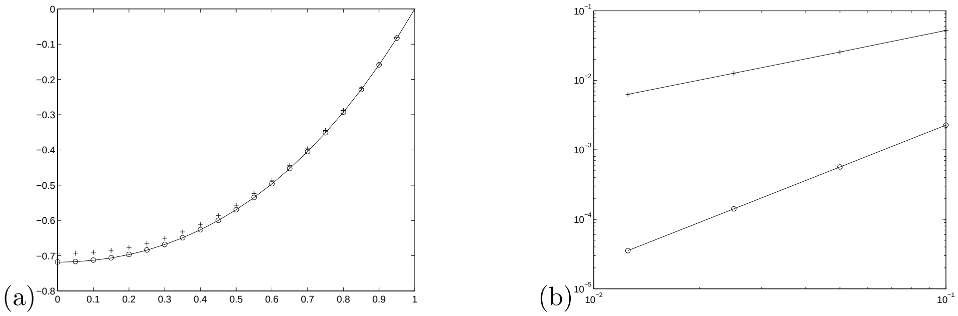


Figure 2.2: (a) Sample solution to the steady-state heat equation with a Neumann boundary condition at the left boundary and Dirichlet at the right. Solid line is the true solution. + shows solution on a grid with 20 points using (2.34). o shows the solution on the same grid using (2.36). (b) A log-log plot of the max-norm error as the grid is refined is also shown for each case.

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 first consider the equation (2.4) with boundary conditions

$$u'(0) = \sigma, \qquad u(1) = \beta. \tag{2.33}$$

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 introduce one more unknown than we previously had: U_0 at the point $x_0 = 0$ since this is now an unknown value. We also need to augment the system (2.9) with one more equation that models the boundary condition (2.33).

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

$$\frac{U_1 - U_0}{h} = \sigma. \tag{2.34}$$

If we append this equation to the system (2.9), we obtain the following system of equations for the unknowns $U_0,\ U_1,\ \ldots, U_m$:

$$\frac{1}{h^2}\begin{bmatrix} -h & h & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{m-1} \\ U_m \end{bmatrix} = \begin{bmatrix} \sigma \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) - \beta/h^2 \end{bmatrix}. \tag{2.35}$$

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 (2.34) 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. But this is not true in the case we are now discussing.

Second attempt. To obtain a second-order accurate method, we should use a centered approximation to $u'(0) = \sigma$ instead of the one-sided approximation (2.34). We can introduce another unknown U_{-1} and instead of the single equation (2.34) use the following two equations:

$$\begin{aligned} \frac{1}{h^2}(U_{-1} - 2U_0 + U_1) = f(x_0) \\ \frac{1}{2h}(U_1 - U_{-1}) = \sigma \end{aligned} \tag{2.36}$$

This results in a system of $m + 2$ equations. (What is the matrix?)

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 (2.36), resulting in a single equation that can be written as

$$\frac{1}{h}(-U_0 + U_1) = \sigma + \frac{h}{2}f(x_0). \tag{2.37}$$

We have now reduced the system to one with only $m+1$ equations for the unknowns $U_0,\ U_1,\ \ldots,\ U_m$. The matrix is exactly the same as the matrix in (2.35), 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 (2.35) 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 (2.37) 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,

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

The method (2.35) is stable, but it is not easy to show this in general in the 2-norm. In the next section we will show that max-norm stability can be proved by directly computing the inverse matrix and examining the size of the elements.

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 boundary value problems may fail to be well posed.

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

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

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

$$\frac{1}{h^2}\begin{bmatrix} -h & h & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & -2 & 1 \\ & & & & & h & -h \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_m \\ U_{m+1} \end{bmatrix} = \begin{bmatrix} \sigma_0 + \frac{h}{2}f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_m) \\ -\sigma_1 + \frac{h}{2}f(x_{m+1}) \end{bmatrix}. \tag{2.38}$$

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 $e = [1,\ 1,\ \dots,\ 1]^T$ is a null vector.

This is not a failure in our numerical model. In fact it reflects the fact 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. This can be easily understood physically by again considering the underlying heat equation discussed in Section 2.1. First consider the case $\sigma_0 = \sigma_1 = 0$ and $f(x) \equiv 0$ so that both ends of the rod are insulated and there is no heat flux through the ends, and no heat source within the rod. Recall that the boundary value problem 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,

$$u(x) = c \tag{2.39}$$

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 (2.39) is a solution of the steady-state boundary value problem, since it satisfies all the conditions $u''(x) \equiv 0$, $u'(0) = u'(1) = 0$. The ordinary differential equation 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, 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 boundary value problem 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 $f = -\psi$). 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 boundary value problem has no solution.

2.14 A general linear second order equation

We now consider the more general linear equation

$$a(x)u''(x) + b(x)u'(x) + c(x)u(x) = f(x) \tag{2.40}$$

together with two boundary conditions, say the Dirichlet conditions

$$u(a) = \alpha, \qquad u(b) = \beta. \tag{2.41}$$

This equation can be discretized to second order by

$$a_i\left(\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2}\right) + b_i\left(\frac{U_{i+1} - U_{i-1}}{2h}\right) + c_iU_i = f_i \tag{2.42}$$

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

$$A = \frac{1}{h^2}\begin{bmatrix} (h^2c_1 - 2a_1) & (a_1 + hb_1/2) & & & \\ (a_2 - hb_2/2) & (h^2c_2 - 2a_2) & (a_2 + hb_2/2) & & \\ \ddots & \ddots & \ddots & & \\ & (a_{m-1} - hb_{m-1}/2) & (h^2c_{m-1} - 2a_{m-1}) & (a_{m-1} + hb_{m-1}/2) \\ & & (a_m - hb_m/2) & (h^2c_m - 2a_m) \end{bmatrix} \tag{2.43}$$

and

$$U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_{m-1} \\ U_m \end{bmatrix}, \qquad F = \begin{bmatrix} f_1 - (a_1/h^2 - b_1/2h)\alpha \\ f_2 \\ \vdots \\ f_{m-1} \\ f_m - (a_m/h^2 + b_m/2h)\beta \end{bmatrix}. \tag{2.44}$$

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, so the parameter $\kappa(x)$ varies with x and is always positive. The steady state heat-conduction problem is then

$$(\kappa(x)u'(x))' = f(x) \tag{2.45}$$

together with some boundary conditions, say the Dirichlet conditions (2.41). To discretize this equation we might be tempted to apply the chain rule to rewrite (2.45) as

$$\kappa(x)u''(x) + \kappa'(x)u'(x) = f(x) \tag{2.46}$$

and then apply the discretization (2.43), yielding the matrix

$$A = \frac{1}{h^2}\begin{bmatrix} -2\kappa_1 & (\kappa_1 + h\kappa_1'/2) & & & \\ (\kappa_2 - h\kappa_2'/2) & -2\kappa_2 & (\kappa_2 + h\kappa_2'/2) & & \\ \ddots & \ddots & \ddots & & \\ & (\kappa_{m-1} - h\kappa_{m-1}'/2) & -2\kappa_{m-1} & (\kappa_{m-1} + h\kappa_{m-1}'/2) \\ & & (\kappa_m - h\kappa_m'/2) & -2\kappa_m \end{bmatrix}. \tag{2.47}$$

However, this is not the best approach. It is better to discretize the physical problem (2.45) 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:

$$\begin{aligned} (\kappa u')'(x_i) &\approx \frac{1}{h}\left[\kappa_{i+1/2}\left(\frac{U_{i+1} - U_i}{h}\right) - \kappa_{i-1/2}\left(\frac{U_i - U_{i-1}}{h}\right)\right] \\ &= \frac{1}{h^2}[\kappa_{i-1/2}U_{i-1} - (\kappa_{i-1/2} + \kappa_{i+1/2})U_i + \kappa_{i+1/2}U_{i+1}]. \end{aligned} \tag{2.48}$$

This leads to the matrix

$$A = \frac{1}{h^2}\begin{bmatrix} -(\kappa_{1/2} + \kappa_{3/2}) & \kappa_{3/2} & & & \\ \kappa_{3/2} & -(\kappa_{3/2} + \kappa_{5/2}) & \kappa_{5/2} & & \\ \ddots & \ddots & \ddots & & \\ & \kappa_{m-3/2} & -(\kappa_{m-3/2} + \kappa_{m-1/2}) & \kappa_{m-1/2} \\ & & \kappa_{m-1/2} & -(\kappa_{m-1/2} + \kappa_{m+1/2}) \end{bmatrix}. \tag{2.49}$$

Comparing (2.47) with (2.49), 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 (2.49) has the advantage of being *symmetric* as we would hope for since the original differential equation is *self adjoint.*Moreover since $\kappa > 0$ the matrix can be shown to be nonsigular and *negative definite*, meaning 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 Exercise **??**). 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.

When solving the resulting linear system by iterative methods (see Chapter 3 and Chapter 5) it is also often desirable that the matrix have properties such as negative definiteness, since some iterative methods (*e.g.*, the conjugate-gradient method, Section 5.3) depend on such properties.

2.15 Nonlinear Equations

We next consider a nonlinear boundary value problem to illustrate the new complications that arise in this case. We will consider a specific example which has a simple physical interpretation that makes it easy to understand and interpret solutions. This example also illustrates that not all 2-point BVP's 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, the differential equation for the pendulum motion can be well approximated by

$$\theta''(t) = -(g/L)\sin(\theta(t)) \tag{2.50}$$

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

$$\theta''(t) = -\sin(\theta(t)) \tag{2.51}$$

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

$$\theta''(t) = -\theta(t) \tag{2.52}$$

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 (2.52) 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 (2.51). For the pendulum problem the *initial value problem* 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

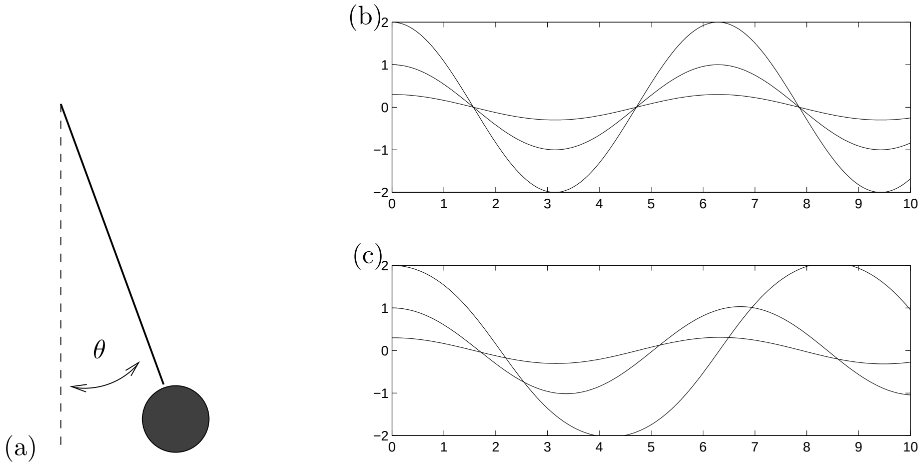


Figure 2.3: (a) Pendulum. (b) Solutions to the linear equation (2.52) for various initial θ and zero initial velocity. (c) Solutions to the nonlinear equation (2.51) for various initial θ and zero initial velocity.

the pendulum will be at a desired location $\theta(T) = \beta$ at some specified later time T. Then we have a 2-point BVP

$$\begin{aligned} &\theta''(t) = -\sin(\theta(t)) \quad \text{for } 0 < t < T, \\ &\theta(0) = \alpha, \qquad \theta(T) = \beta. \end{aligned} \tag{2.53}$$

Similar BVP's do arise in more practical situations, for example trying to shoot a missle 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 BVP's that is discussed in [AMR88], [Kel76], for example.

2.15.1 Discretization of the nonlinear BVP

We can discretize the nonlinear problem (2.51) in the obvious manner, following our approach for linear problems, to obtain the system of equations

$$\frac{1}{h^2}(\theta_{i-1} - 2\theta_i + \theta_{i+1}) + \sin(\theta_i) = 0 \tag{2.54}$$

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

$$G(\theta) = 0 \tag{2.55}$$

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

$$\theta^{[k+1]} = \theta^{[k]} + \delta^{[k]} \tag{2.56}$$

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

$$J(\theta^{[k]})\delta^{[k]} = -G(\theta^{[k]}). \tag{2.57}$$

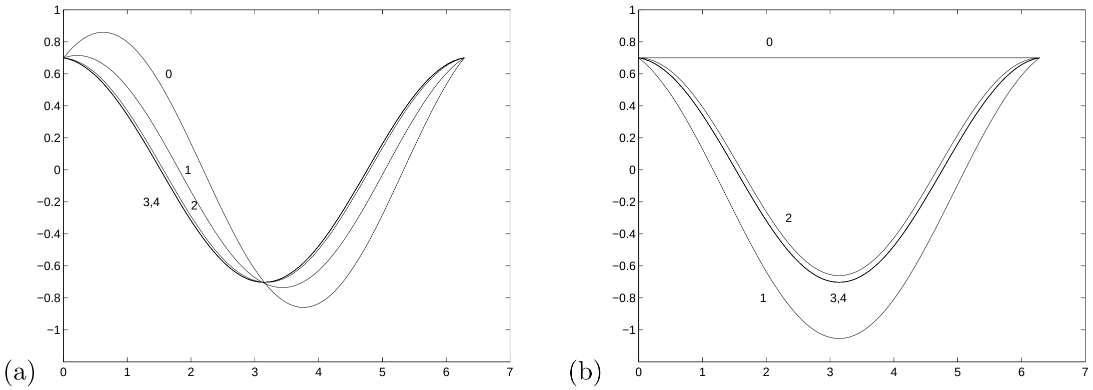


Figure 2.4: Convergence of Newton iterates towards a solution of the pendulum problem. The iterates $\theta^{[k]}$ for $k = 1,\ 2,\ \cdots$ are denoted by the number k in the plots. (a) Starting from $\theta_i^{[0]} = 0.7\cos(t_i) + 0.5\sin(t_i)$ (b) Starting from $\theta_i^{[0]} = 0.7$.

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 i'th component of the vector-valued function G. In our case $G_i(\theta)$ is exactly the left-hand side of (2.54) 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

$$J(\theta) = \frac{1}{h^2}\begin{bmatrix} (-2 + h^2\cos(\theta_1)) & 1 & & \\ 1 & (-2 + h^2\cos(\theta_2)) & 1 & \\ & \ddots & \ddots & \ddots \\ & & & \\ & & 1 & (-2 + h^2\cos(\theta_m)) \end{bmatrix}. \tag{2.58}$$

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 (2.52) 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 and Exercise 2.2.) 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 $k = 0,\ 1,\ 2,\ \cdots$ that are obtained as we iterate with Newton's method. They rapidly converge to a solution to the nonlinear system (2.54). (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 $k = 0,\ 1,\ 2,\ \cdots$ with a different choice $\theta^{[0]} \equiv 0.7$.

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

2.15.2 Nonconvergence

Newton's method can be shown to converge if we start with an initial guess that is sufficiently close to a solution. How close is needed depends on the nature of the problem and is described by the *Newton-Kantorovich Theorem*, see, *e.g.*, [**?**]. For the 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, *e.g.*, [Kel76].

2.15.3 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 (2.53). 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 upwards so that it rises towards 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 has fallen back to $\theta = 0.7$ again at $T = 2\pi$. In fact it is possible to find such a solution for any $T > 0$.

Physically it seems clear that there is a second solution to the BVP. In order 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.

2.15.4 Accuracy on nonlinear equations

The solutions plotted above are not exact solutions to the BVP (2.53). They are only solutions to the discrete system of equations (2.54) 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

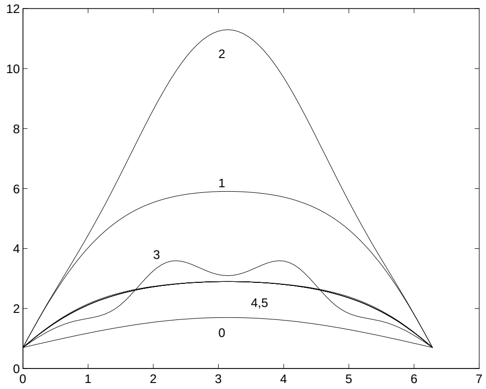


Figure 2.5: Convergence of Newton iterates towards a different solution of the pendulum problem, by starting with initial guess $\theta_i^{[0]} = 0.7 + \sin(t_i/2)$. The iterates k for $k = 1,\ 2,\ \cdots$ are denoted by the number k in the plots.

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 nonlinear 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*:

$$\begin{aligned}\tau_i &= \frac{1}{h^2}(\theta(t_{i-1}) - 2\theta(t_i) + \theta(t_{i+1})) + \sin(\theta(t_i)) \\ &= (\theta''(t_i) + \sin(\theta(t_i))) + \frac{1}{12}h^2\theta''''(t_i) + O(h^4) \\ &= \frac{1}{12}h^2\theta''''(t_i) + O(h^4).\end{aligned} \tag{2.59}$$

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 local truncation error is $O(h^2)$ and has exactly the same form as we found 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

$$G(\theta) - G(\hat{\theta}) = -\tau. \tag{2.60}$$

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 (2.60) 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 2.15.1 *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*

$$\|(\hat{J}^h)^{-1}\| \leq C \quad \textit{for all } h < h_0. \tag{2.61}$$

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 [Kel76] 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: 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. (Though it can be shown that for a stable method it will converge from a sufficiently good initial guess — see [Kel76].) 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.

2.16 Singular perturbations and boundary layers

In this section we consider some singular perturbation problems to illustrate the difficulties that can arise in numerically solving problems with boundary layers or other regions where the solution varies rapidly. See [Kev90], [KC81] for more detailed discussions of singular perturbation problems. In particular the example used here is very similar to one that can be found in [Kev90], 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

$$u_t + au_x = \kappa u_{xx} + \psi \tag{2.62}$$

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 $u(1,t) = \beta(t)$ in order 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

$$\begin{aligned} &au'(x) = \kappa u''(x) + \psi(x) \\ &u(0) = \alpha, \qquad u(1) = \beta. \end{aligned} \tag{2.63}$$

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 2.14 and for small a the solution looks 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 the equation (2.63) in the form

$$\epsilon u''(x) - u'(x) = f(x). \tag{2.64}$$

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 towards 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 (2.64) reduces to a *first order* equation (the steady advection equation)

$$-u'(x) = f(x) \tag{2.65}$$

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 overspecifying the problem.

Figure 2.6(a) shows how solutions to equation (2.64) look for various values of ϵ in the case $\alpha = 1$, $\beta = 3$, and $f(x) = -1$. In this case the exact solution is

$$u(x) = \alpha + x + (\beta - \alpha - 1)\left(\frac{e^{x/\epsilon} - 1}{e^{1/\epsilon} - 1}\right). \tag{2.66}$$

Note that as $\epsilon \to 0$ the solution tends towards 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 (2.63) with $0 < \epsilon \ll 1$ is called a *singularly perturbed equation.* It is a small perturbation of the equation (2.65), but this small perturbation changes the character of the equation completely (from a first order equation 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 $\kappa u_{xx} - au_x = f$ 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 with 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

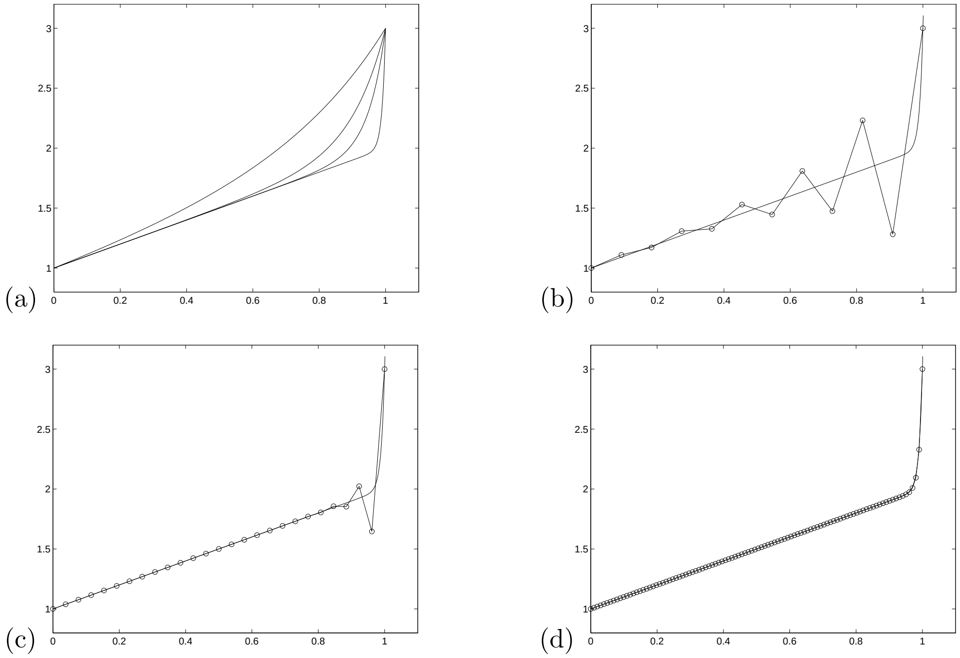


Figure 2.6: (a) Solutions to the steady-state advection-diffusion equation (2.64) for different values of ϵ. The four lines correspond to $\epsilon = 0.3,\ 0.1,\ 0.05$ and 0.01 from top to bottom. (b) Numerical solution with $\epsilon = 0.01$ and $h = 1/10$. (c) $h = 1/25$. (d) $h = 1/100$.

errors in our finite difference approximations. Recall that the error in our approximation to $u''(x)$ is proportional to $h^2u''''(x)$, 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 that the boundary layer is well resolved.

2.16.1 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 to begin with where the interior layer will be. Perturbation theory can often be used to analyse 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.

2.17 Nonuniform grids and adaptive refinement

From Figure 2.6 it is clear that we need to choose our grid fine enough that several points are within the boundary layer in order to 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 developing formulas that are sufficiently accurate on nonuniform grids. For example, a 4-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 4 nonzero diagonals. A little extra care is needed at the boundaries.

Readers who wish to use methods on nonuniform grids are strongly encouraged to investigate some of the software that is freely available. There are good packaged routines which 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. The COLSYS collocation package is one such software package, available from NETLIB. Strategies for *adaptive mesh selection* (*i.e.*, choosing a grid that adapts to the nature of the solution) are discussed, for example, in [AMR88].

2.18 Higher order methods

So far we have only considered second order methods for solving boundary value problems. There are several approaches that can be used to obtain higher order accurate methods.

2.18.1 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 eample, the finite difference approximation

$$\frac{1}{12h^2}[-U_{j-2} + 16U_{j-1} - 30U_j + 16U_{j+1} - U_{j+2}] \tag{2.67}$$

gives a fourth order accurate approximation to $u''(x_j)$. For the Dirichlet boundary value problem considered in Section 2.4, this approximation can be used at grid points $j = 2,\ 3,\ \ldots,\ m-1$ but not for $j = 1$ or $j = m$. In order to maintain fourth order accuracy we need to use formulas that are at least third order accurate at these points. We can get away with one order less at these two points because, as in Section 2.11, the inverse matrix contains elements that are $O(h)$ in magnitude and so the contribution to the global error from each local error τ_j is of magnitude $h\tau_j$.

Third order accurate finite difference methods at $j = 1$ and $j = m-1$ are given by

$$\frac{1}{12h^2}[11U_0 - 20U_1 + 6U_2 + 4U_3 - U_4] \tag{2.68}$$

and

$$\frac{1}{12h^2}[-U_{m-4} + 4U_{m-3} + 6U_{m-2} - 20U_{m-1} + 11U_m] \tag{2.69}$$

respectively. Using these together with (2.67) gives a nearly pentadiagonal matrix problem

$$\frac{1}{12h^2}\begin{bmatrix} -20 & 6 & 4 & -1 & & & \\ 16 & -30 & 16 & -1 & & & \\ -1 & 16 & -30 & 16 & -1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & -1 & 16 & -30 & 16 & -1 \\ & & & -1 & 16 & -30 & 16 \\ & & & -1 & 4 & 6 & -20 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{m-2} \\ U_{m-1} \\ U_m \end{bmatrix} = \begin{bmatrix} f(x_1) - 11\alpha/12h^2 \\ f(x_2) + \alpha/12h^2 \\ f(x_3) \\ \vdots \\ f(x_{m-2}) \\ f(x_{m-1}) + \beta/12h^2 \\ f(x_m) - 11\beta/12h^2 \end{bmatrix}. \tag{2.70}$$

It is also possible to derive fourth order accurate approximations to use at $j = 1$ and $j = m$ by including one more point in the difference formula, e.g.

$$\frac{1}{12h^2}[10U_0 - 15U_1 - 4U_2 + 14U_3 - 6U_4 + U_5] \tag{2.71}$$

and the symmetric formula at $j = m$. Using these rather than (2.68) and (2.69) gives a slightly better error constant though the same order of accuracy.

2.18.2 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 U_j and V_{2j} both 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,

$$U_j - u(jh) = C_2h^2 + C_4h^4 + C_6h^6 + \cdots \tag{2.72}$$

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 exact discretization of the Green's function, (Section 2.11), but we omit the details of justifying this.)

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

$$\begin{aligned} V_{2j} - u(jh) &= C_2(h/2)^2 + C_4(h/2)^4 + C_6(h/2)^6 + \cdots \\ &= \frac{1}{4}C_2h^2 + \frac{1}{16}C_4h^4 + \frac{1}{64}C_6h^6 + \cdots \end{aligned} \tag{2.73}$$

The extrapolated value is given by

$$\bar{U}_j = \frac{1}{3}(4V_{2j} - U_j), \tag{2.74}$$

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

$$\bar{U}_j - u(jh) = \frac{1}{3}\left(\frac{1}{4} - 1\right) C_4 h^4 + O(h^6). \tag{2.75}$$

The result has fourth order accuracy as h is reduced, and a much smaller error than either U_j or V_{2j} (provided $C_4 h^2$ is not larger than C_2, and generally 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 2.18.1, 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.

2.18.3 Deferred corrections

There is another way to combine two different numerical solutions to obtain a higher order accurate approximation, called deferred corrections, that has the advantage that the two problems are both solved on the same grid rather than refining the grid as in 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),

$$AE = -\tau, \tag{2.76}$$

where τ is the local truncation error. Suppose we knew the vector τ. Then we could solve the system (2.76) 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

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

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 $u''(x) = f(x)$ 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

$$F_j = f(x_j) + \frac{1}{12} h^2 f''(x_j), \tag{2.77}$$

with boundary terms added at $j = 1$ and $j = m$. Solving $AU = F$ then gives a fourth order accurate solution directly. An analog of this for the two-dimensional Poisson problem is discussed in Section 3.4.

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 (2.76) to estimate the global error. The global error estimate can then be used to improve the approximate solution. For more details see, *e.g.*, [Kel76], [AMR88].

The table below shows the errors obtained for the problem of Section 2.4 with each of the methods described above, on four different grids. Note that all three of these approaches gives much smaller errors than the original second-order method.

```
     h         second        fourth        extrap.       deferred
               order         order                       corrections

  0.05000    1.7526e-04    1.8055e-06    2.1908e-08    5.8412e-08
  0.02500    4.4120e-05    6.7161e-08    1.3787e-09    3.6765e-09
  0.01250    1.1031e-05    2.2755e-09    8.6173e-11    2.2981e-10
  0.00625    2.7585e-06    7.4911e-11    5.3824e-12    1.4361e-11
```

2.19 Exercises

Exercise 2.1 *Determine the function shown in Figure 2.2 by solving the problem exactly.*

Exercise 2.2 *Consider the following linear boundary value problem with Dirichlet boundary conditions:*

$$\begin{aligned} &u''(x) + u(x) = 0 \quad \textit{for } 0 < x < \pi \\ &u(0) = \alpha, \quad u(\pi) = \beta. \end{aligned} \tag{2.78}$$

For what values of α and β does this have solutions? Sketch a family of solutions in a case where there are infinitely many solutions.

Exercise 2.3 *Show that the local truncation error τ_0 for the discretization of the boundary value problem using the "second attempt" on page 23 is $O(h^2)$. Hint: Note that $f(x_0) = u''(x_0)$.*

Exercise 2.4 *Write out the 3×3 matrix A and inverse matrix A^{-1} explicitly for the problem $u''(x) = f(x)$ with $u(0) = u(1) = 0$ for $h = 0.25$. If $f(x) = x$, determine the discrete approximation to the solution of the boundary value problem on this grid and sketch this solution and the 3 Green's functions whose sum gives this solution.*

Exercise 2.5 *Consider the mixed Dirichlet-Neumann problem with $u'(0) = u(1) = 0$ and determine the Green's function shown in Figure 2.1(b). Using this as guidance, find the inverse of the matrix in (2.35). Write out the 4×4 matrices A and A^{-1} for the case $h = 0.25$.*

Exercise 2.6 *Write a program to solve boundary value problem for the pendulum as discussed in the text. See if you can find yet another solution for the boundary conditions illustrated in Figures 2.4 and 2.5.*

Exercise 2.7 *Find a numerical solution to this BVP with the same general behavior as seen in Figure 2.5 for the case of a longer time interval, say $T = 20$, again with $\alpha = \beta = 0.7$. Try larger values of T. What does $\max_i \theta_i$ approach as T is increased? Note that for large T this solution exhibits "boundary layers" (see Section 2.16).*

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. In two space dimensions a constant-coefficient elliptic equation has the form

$$a_1u_{xx} + a_2u_{xy} + a_3u_{yy} + a_4u_x + a_5u_y + a_6u = f \tag{3.1}$$

where the coefficients $a_1,\ a_2,\ a_3$ satisfy

$$a_2^2 - 4a_1a_3 < 0. \tag{3.2}$$

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 $u(x, y, t)$ will generally 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

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

which is called the *Poisson problem.* In the special case $f \equiv 0$ this reduces to *Laplace's equation*,

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

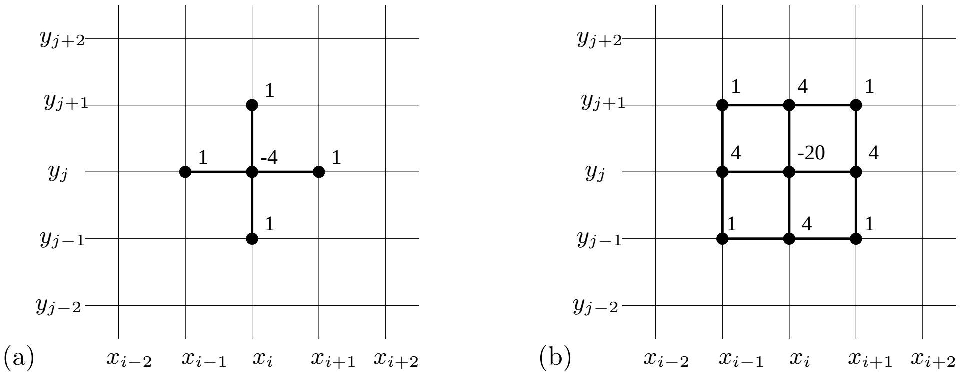


Figure 3.1: Portion of the computational grid for a two-dimensional elliptic equation. (a) The 5-point stencil for the Laplacian about the point (i, j) is also indicated. (b) The 9-point stencil is indicated, which is discussed in Section 3.4.

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 will be used for grid spacing.

3.2 The five-point stencil for the Laplacian

To discuss discretizations, first consider the Poisson problem (3.5) on the unit square $0 \leq x \leq 1$, $0 \leq y \leq 1$ 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.

Let u_{ij} represent an approximation to $u(x_i, y_j)$. In order to discretize (3.5) we replace both 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 $\Delta x = \Delta y \equiv h$, though 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.

If we collect all of these equations together into a matrix equation, we will have an $m^2 \times m^2$ matrix which is very *sparse*, *i.e.*, most of the elements are zero. Since each equation involves at most 5 unknowns (less near the boundary), each row the matrix has at most 5 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 3 nonzeros.

Unfortunately, in two space dimensions the structure of the matrix is not as simple as in one dimension, and in particular the nonzeros will not all be as nicely clustered near the main diagonal. The exact structure of the matrix depends on the order in which we order the unknowns and write down the equations, as we will see below, but no ordering is ideal.

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 one-dimensional 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:

$$\frac{1}{h^2}\left[\begin{array}{ccccc|ccccc}
-2 & & & & & 1 & & & & \\
 & -2 & & & & 1 & 1 & & & \\
 & & -2 & & & & 1 & 1 & & \\
 & & & \ddots & & & & \ddots & \ddots & \\
 & & & & -2 & & & & 1 & 1 \\
\hline
1 & 1 & & & & -2 & & & & \\
 & 1 & 1 & & & & -2 & & & \\
 & & 1 & 1 & & & & -2 & & \\
 & & & \ddots & \ddots & & & & \ddots & \\
 & & & & 1 & & & & & -2
\end{array}\right]
\left[\begin{array}{c}
U_1 \\ U_3 \\ U_5 \\ \vdots \\ U_{m-1} \\ \hline U_2 \\ U_4 \\ U_6 \\ \vdots \\ U_m
\end{array}\right]
=
\left[\begin{array}{c}
f(x_1) - \alpha/h^2 \\ f(x_3) \\ f(x_5) \\ \vdots \\ f(x_{m-1}) \\ \hline f(x_2) \\ f(x_4) \\ f(x_6) \\ \vdots \\ f(x_m) - \beta/h^2
\end{array}\right]. \tag{3.11}$$

This linear system has the same solution as (2.9) modulo the reordering of unknowns, but 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.

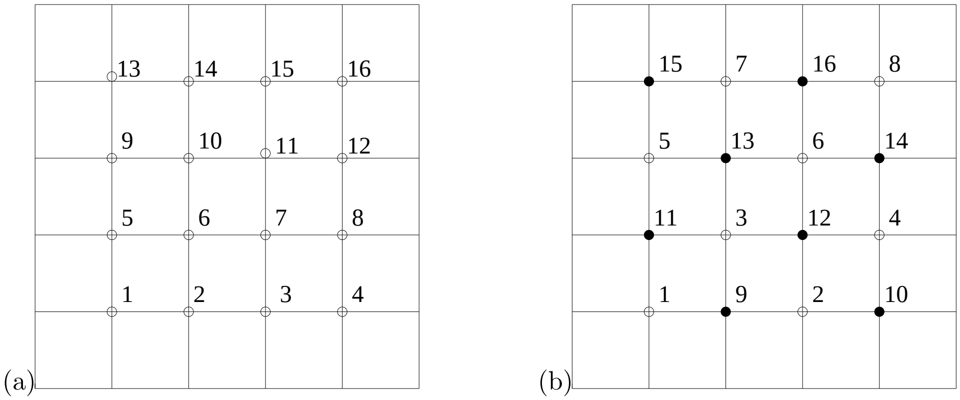


Figure 3.2: (a) The natural rowwise order of unknowns and equations on a 4×4 grid. (b) The red-black ordering.

Returning to the two-dimensional problem, it should be clear that there is no way to order the unknowns so that all nonzeros are clustered adjacent to the diagonal. About the best we can do is to use 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). 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 analog of the odd-even ordering that leads to the matrix (3.11) in one dimension. This ordering is significant because all 4 neighbors of a red grid point are black points, and vice versa, and leads to a matrix equation with the structure

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

where $D = -\frac{4}{h^2}I$ is a diagonal matrix of dimension $m^2/2$ and H is determined in Exercise 3.1.

3.3 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 2 parameters p and k corresponding to wavenumbers in the x and y directions, for $p,\ k = 1,\ 2,\ \ldots,\ m$. The (p,k) eigenvector $u^{p,k}$ has the m^2 elements

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

The corresponding eigenvalue is

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

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 are 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

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

The fact that the matrix becomes more ill-conditioned as we refine the grid is responsible for the slow-down of iterative methods, as discussed in Chapter 5.

3.4 The nine-point Laplacian

Above we have 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

$$\nabla_9^2 u_{ij} = \frac{1}{6h^2}[4u_{i-1,j}+4u_{i+1,j}+4u_{i,j-1}+4u_{i,j+1}+u_{i-1,j-1}+u_{i-1,j+1}+u_{i+1,j-1}+u_{i+1,j+1}-20u_{ij}] \quad (3.17)$$

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

$$\nabla_9^2 u_{ij} = F_{ij} \quad (3.18)$$

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

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

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 analog of the modification (2.77) that gives fourth order accuracy for the boundary value problem $u''(x) = f(x)$.

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

$$F_{ij} = f_{ij} - \frac{h^2}{12}\nabla_5^2 f_{ij}$$

instead of (3.19).

This is a trick that can often 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 in one dimension in Section 2.18.3.

3.5 Solving the linear system

In Chapter 5 we will consider various approaches to solving the linear systems arising from elliptic equations in more detail. Here we will only discuss some of the main issues.

There are two fundamentally different approaches that could be used for solving these linear systems. 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 preferable for various reasons described below.

For certain special problems very fast direct methods can be used, which are much better than standard Gaussian elimination. This is discussed briefly in Section 3.5.2.

3.5.1 Gaussian elimination

Gaussian elimination is the standard direct method for solving a linear system $Au = F$. The matrix A is reduced to upper triangular form by taking linear combinations of the rows to introduce zeros below the diagonal. This can be viewed as giving a factorization $A = LU$ of the matrix into a product of an upper and a lower triangular matrix. Forward and back substitution are then used to solve the triangular systems $Lc = F$ and then $Uu = c$. (See, *e.g.*, [GL96], [TB97]).

It is easy to compute 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 disasterous, or at best extremely wasteful of computer resources. Suppose we are solving the Poisson problem on a 100×100 grid for example. Then $N = m^2 = 10^4$ and $N^3 = 10^{12}$. On a reasonably fast workstation (ca. 2004) which can do on the order of 10^9 floating point operations per second (1 gigaflop), this would take on the order of 10^3 seconds, which is roughly 17 minutes. If we went up to a 1000×1000 grid (a million by million matrix) this would increase by a factor of 10^6 to roughly 31 years. Things are worse in three dimensions. Solving the Poisson problem on a $100 \times 100 \times 100$ gives the same matrix dimension $N = 10^6$ as the 1000×1000 grid in two dimensions. More sophisticated methods can solve even these larger problems in a reasonable amount of time, and problems with several million unknowns are not unusual in applications.

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 $8N^2$ bytes. For the larger problems mentioned above, this would be 8×10^{12} bytes, or 8000 gigabytes. One advangtage 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 eliminataion requires only eliminating 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 valuse, 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, though 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 band 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 $O(m^4)$.

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[GL81]. This algorithm requires $O(m^3)$ operations. It turns out this is the best that can be achieved with a direct method based on Gaussian elimination. George has proved (see [GL81]) that any elimination method for solving this problem requires at least $O(m^3)$ operations.

3.5.2 Fast Poisson solvers

For this very special system of equations there are other techniques that can be used to solve the system quickly. These are generally called *fast Poisson solvers* (recall we are looking at the Poisson problem (3.10) with constant coefficients on a rectangular domain). One standard technique uses fast Fourier transforms to solve the system in $O(m^2 \log m)$ work, which is nearly optimal since there are m^2 grid values to be determined.

To explain the main idea of this approach, we consider only the one-dimensional case, where the Poisson problem reduces to $u''(x) = f(x)$ on the interval $[0, 1]$, and the centered 3-point discretization gives the matrix equation (2.9) (for the case of Dirichlet boundary conditions). In one dimension this tridiagonal system can be solved in $O(m)$ operations and there is no need for a "fast solver", but the idea is easiest to illustrate in this case.

In the one-dimensional case the matrix A is given by (2.10) and in Section 2.10 we determined the eigenvalues and eigenvectors of A. Based on these we can write down the Jordan Canonical Form of A:

$$A = R\Lambda R^{-1} \tag{3.20}$$

where R is the matrix of right eigenvectors (the p'th column of R is the vector u^p from (2.24)) and Λ is a diagonal marix with the eigenvalues on the diagonal. So we have

$$R = \begin{bmatrix} \sin(\pi x_1) & \sin(2\pi x_1) & \cdots & \sin(m\pi x_1) \\ \sin(\pi x_2) & \sin(2\pi x_2) & \cdots & \sin(m\pi x_2) \\ \vdots & \vdots & & \vdots \\ \sin(\pi x_m) & \sin(2\pi x_m) & \cdots & \sin(m\pi x_m) \end{bmatrix} \tag{3.21}$$

and

$$\Lambda = \begin{bmatrix} \frac{2}{h^2}(\cos(\pi h) - 1) & & & \\ & \frac{2}{h^2}(\cos(2\pi h) - 1) & & \\ & & \ddots & \\ & & & \frac{2}{h^2}(\cos(m\pi h) - 1) \end{bmatrix}. \tag{3.22}$$

Since $A^{-1} = R\Lambda^{-1}R^{-1}$, one approach to solving the linear system $Au = F$ is to use this to obtain

$$u = A^{-1}F = R\Lambda^{-1}R^{-1}F.$$

There are at least 4 obvious objections to using this approach to solve a general linear system:

- In general it is not easy to determine the eigenvalues and eigenvectors of a given matrix. Doing so computationally is generally much more expensive than solving the original system by other approaches.

- Even if we know R and Λ, as we do here, we don't in general know R^{-1}. Computing an inverse matrix is much more work than solving a single linear system. Of course all we really need here is $R^{-1}F \equiv \hat{F}$, say, which can be found by solving a single system $R\hat{F} = F$, but this system may be no easier to solve than the original system $Au = F$ (and perhaps much more expensive since R is typically dense even if A is sparse).

- Even if we know R and R^{-1} explicitly, storing them typically requires $O(N^2)$ storage for an $N \times N$ system since these are dense matrices. We probably want to avoid this if A is sparse.

- It also requires $O(N^2)$ work to multiply an $N \times N$ dense matrix by an N-vector. In two dimensions, where $N = m^2$, this is no better than using Gaussian elimination on the banded matrix A. (And in one dimension the $O(m^2)$ work required would be much worse than the $O(m)$ work needed for the tridiagonal system.)

So why would we ever consider such a method? In general we would not, but for the very special case of a Poisson problem on a simple region, these objections all vanish thanks to the Fast Fourier Transform.

We do know Λ and R for our simple problem, and moreover we know R^{-1}, since we can compute using trig identities that

$$R^2 = \left(\frac{m+1}{2}\right) I$$

and so

$$R^{-1} = 2hR.$$

R is symmetric, and, except for the scaling, an orthogonal matrix. If we scaled the eigenvectors differently, using $Q = \sqrt{2h}R$, then the eigenvector matrix Q would be an orthogonal matrix: $Q^2 = I$.

Although R is a dense matrix, it contains only m distinct values (after applying trig identities) and we do not need to compute or store all of the elements. Moreover, and most importantly, multiplying a vector by R or R^{-1} does not require $O(m^2)$ operations, but can be done in $O(m \log m)$ operations if m is a product of small prime factors. Ideally we would choose m to be a factor of 2, say $m = 2^k$. The trick is then to observe (It's not obvious!) that R can be written as a product of k matrices,

$$R = R_1 R_2 \cdots R_k$$

where each R_j is very sparse so that multiplying a vector by R_j requires only $O(m)$ operations. Applying each matrix in turn to a vector gives the desired product in $O(km) = O(m \log m)$ operations.

This algorithm is a special case of the *Fast Fourier Transform*, or FFT, since multiplying a vector by R corresponds to taking a sine transform of the data. See [Loa97] for a brief introduction to the recursive structure of the FFT.

In one space dimension there is still no advantage to this procedure over solving the tridiagonal system directly, which only requires $O(m)$ operations. However, this approach carries over directly to 2 or 3 space dimensions, with operation counts of $O(m^2 \log m)$ and $O(m^3 \log m)$ respectively, which is asymptotically nearly optimal since the grids have m^2 and m^3 points.

To solve the system $Au = F$ using this technique (back in one dimension, now), we would first use the FFT algorithm to compute

$$\hat{F} = R^{-1}F = 2hRF, \tag{3.23}$$

then divide each element of $\hat{F}$ by the corresponding eigenvalue from Λ,

$$\hat{u}_j = \hat{F}_j \Big/ \left(\frac{2}{h^2}(\cos(j\pi h) - 1)\right) \quad \text{for} \quad j = 1,\ 1,\ \ldots,\ m, \tag{3.24}$$

so that $\hat{u} = \Lambda^{-1}R^{-1}F$. Finally we do another FFT to compute

$$u = R\hat{u}. \tag{3.25}$$

Now consider the Poisson problem with the 5-point Laplacian in two dimensions. The difference equations (3.9) with $\Delta x = \Delta y = h$ become

$$\frac{1}{h^2}(u_{i-1,j} - 2u_{ij} + u_{i+1,j}) + \frac{1}{h^2}(u_{i,j-1} - 2u_{ij} + u_{i,j+1}) = f_{ij}. \tag{3.26}$$

For each fixed i we can apply the one-dimensional FFT in the y-direction along a row of data to reduce this to

$$\frac{1}{h^2}(\hat{u}_{i-1,j} - 2\hat{u}_{ij} + \hat{u}_{i+1,j}) + \frac{2}{h^2}(\cos(j\pi h) - 1)\hat{u}_{ij} = \hat{f}_{ij} \tag{3.27}$$

where $\hat{u}_{ij}$ represents the one-dimensional transfrom of u in the y-direction. For each fixed j (3.27) is now a tridiagonal system of equations to solve for $\hat{u}_{ij}$ along this row. So we first do m FFT's in y, then m tridiagonal solves in x for $\hat{u}_{ij}$, and finally m FFT's in y again to recover the solution u_{ij}. Of course the role of x and y could be reversed here, by doing FFT's in x and tridiagonal solves in y.

Alternatively, we could solve the one-dimensional systems (3.27) by applying a second set of FFT's in the x-direction, as in our discussion earlier in this section of the one-dimensional problem. However, solving tridiagonal systems directly is more efficient than using FFT's at this point.

In three space dimensions, we would first apply the FFT in two of the three directions in order to reduce the system to a set of tridiagonal solves in the third direction.

Unfortunately these FFT techniques generally do not extend to other systems of equations, such as those arising in a variable-coefficient problem. The technique depends on knowing the eigendecomposition of the matrix A and on having a fast algorithm for multiplying the dense matrix R times a vector. For a variable-coefficient problem, we won't in general know R and R^{-1} (unless we compute them numerically, which takes much more work than solving the original linear system). Even if we did know them, we wouldn't generally have a fast algorithm to apply them to a vector.

The special case of a constant-coefficient Poisson problem in a rectangular region arises sufficiently often in applications that fast Poisson solvers are often useful, however. Software for this problem is available in FISHPACK, for example. For a review of fast Poisson solvers, see [Swa84].

3.6 Exercises

Exercise 3.1 *What is the matrix H in (3.13) for the grid shown in Figure 3.2?*

Chapter 4

Function Space Methods

Finite difference methods determine an approximate solution only at discrete grid points. A "function space method" on the other hand, determines a function $U(x)$ on the entire domain that approximates the true solution $u(x)$. This book primarily concerns finite difference methods, but a brief introduction to function space methods is given here for comparison purposes, and to show that similar algebraic systems arise from these discretizations.

In order to reduce the original differential equation to a system of algebraic equations that can be solved for a finite number of unknowns, we typically search for the approximation $U(x)$ from a finite-dimensional function space that is spanned by a fixed set of basis functions $\phi_j(x)$, for $j = 1,\ 2,\ \ldots,\ N$ that are specified *a priori*. The function $U(x)$ then has the form

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

If the functions $\phi_j(x)$ are a basis for the space of all such functions, then they must be *linearly independent.* This means that the only way we can get the zero function from such a linear combination is if $c_j = 0$ for all j. It then follows that any function in the span of these functions has a unique set of coefficients c_j.

In order to determine the coefficients c_j that yield a good approximation to $u(x)$, we need to derive a set of N equations to be solved for these coefficients. There are several ways in which this can be done.

4.1 Collocation

One approach to determining the function $U(x)$ is to require that this function satisfy the differential equation at some finite set of *collocation points.* Of course we would really like to have $U(x)$ satisfy the differential equation at all points x, in which case it would be the exact solution, but this won't be possible generally unless the exact solution happens to lie in the finite-dimensional function space spanned by the ϕ_j. Since there are N free parameters and also two boundary conditions that need to be satisfied (for our second order model problem), we can only hope to satisfy the differential equation at some set of $N - 2$ collocation points in general.

Example 4.1. Consider our standard model problem $u''(x) = f(x)$ with Dirichlet boundary conditions. Requiring that the boundary conditions be satisfied gives the two equations

$$\begin{aligned}\sum_{j=1}^{N} c_j\phi_j(0) &= \alpha,\\ \sum_{j=1}^{N} c_j\phi_j(1) &= \beta.\end{aligned} \tag{4.1}$$

We can then require in addition that the differential equation be satisfied at some points $\xi_1, \ \ldots, \ \xi_{N-2}$:

$$\sum_{j=1}^{N} c_j\phi_j''(\xi_i) = f(\xi_i) \tag{4.2}$$

for $i = 1, \ 2, \ \ldots, \ N-2$. The equations (4.1) and (4.2) together give N equations for the N unknowns c_j. Note that this may be a dense system of equations unless we are careful to choose our basis functions so that many of the coefficients $\phi_j''(\xi_i)$ of the linear system are zero. A local basis such as a B-spline basis for spline functions is often used in practice. (An example of a local basis for piecewise linear functions is given in Section 4.3, but this is not a function space that is suitable for collocation on a second order differential equation, since the second derivative is not well behaved.)

Another approach is to choose a set of functions and collocation points in such a way that "fast transform" methods can be used to solve the dense linear system, as in the fast Poisson solvers discussed in Section 3.5.2. This suggests using Fourier series to represent the solution. This turns out to be a very good idea, not only because the fast Fourier transform can be used to solve the resulting system, but also because smooth functions can be represented very accurately with relatively few terms of a Fourier series. Hence the order of accuracy of such a method can be very high, see Section 4.2.2.

4.2 Spectral methods

In Section 3.5.2 we looked at a method based on the FFT for solving the linear system $Au = F$ that arises from the finite difference discretization studied in Chapter 2. Here we will study an entirely different approach to solving the original differential equation $u''(x) = f(x)$, approximating the solution by a Fourier series rather than by a discrete set of points. But the techniques are closely related (at least for this simple problem) as noted at the end of this section.

Spectral methods are exceeding powerful tools for solving a wide class of differential equations, particularly when the solution we seek is a smooth function (so that Fourier series give a good representation) and the domain and boundary conditions are reasonably simple. This is not meant to be a general overview of spectral methods, only a brief introduction to the main idea. Better introductions can be found in many sources, *e.g.*, [CHQZ88], [For96], [GO77], [Tre00].

Here we will consider the simplest possible problem to illustrate the main ideas. The problem we consider is again $u''(x) = f(x)$ on $[0, 1]$, and we will also assume that homogeneous Dirichlet boundary conditions $u(0) = u(1) = 0$ are specified, and also that $f(x)$ is a smooth function satisfying $f(0) = f(1) = 0$. These conditions are not necessary in general, but in this special case we can easily illustrate the so-called *spectral method* using the Fourier sine transform introduced in Section 3.5.2. The name comes from the fact that the set of eigenvalues of a matrix or operator is called its *spectrum*, and these methods are heavily based on the eigenstructure of the problem (at least for this simple problem).

Note that for this problem the functions $\sin(j\pi x)$ satisfy the boundary conditions and differentiating twice gives a scalar multiple $-(j\pi)^2$ times the original function. Hence these are indeed *eigenfunctions* of the operator $\partial^2/\partial x^2$. This is the reason using a sine-series expansion of the solution is valuable. Recall also that discretized versions of these functions are eigenvectors of the tridiagonal matrix arising from the standard finite-difference formulation, which is why the FFT can be used in the fast Poisson solver described in Section 3.5.2.

The function $f(x)$ can be expressed in a Fourier series as

$$f(x)=\sum_{j=1}^{\infty}\hat{f}_j\sin(j\pi x)$$

where

$$\hat{f}_j=2\int_0^1 f(x)\sin(j\pi x)\,dx. \tag{4.3}$$

Similarly, the unknown function $u(x)$ can be expanded as

$$u(x)=\sum_{j=1}^{\infty}\hat{u}_j\sin(j\pi x). \tag{4.4}$$

Differentiating this series term by term (assuming this is valid, which it is for smooth functions) gives

$$u''(x)=\sum_{j=1}^{\infty}-(j\pi)^2\hat{u}_j\sin(j\pi x)$$

and equating this with the series for $f(x)$ shows that

$$\hat{u}_j=-\frac{1}{(j\pi)^2}\hat{f}_j. \tag{4.5}$$

We have just "solved" the differential equation and determined the exact solution, as a Fourier series. Note that this works only because $\sin(j\pi x)$ is an eigenfunction of the differential operator. For a different differential equation, one would have to use the appropriate eigenfunctions in the series representation in place of the sine functions to achieve the same success.

If $f(x)$ is sufficiently simple, we may be able to explicitly calculate the integrals in (4.3) and then the resulting infinite series in (4.4). (Though we are more likely to be able to integrate f twice to compute the exact solution for this trivial equation!) More generally we will not be able to compute the integrals or infinite series exactly, but we can use this approach as a basis for an approximate numerical method. We can approximate the series (4.4) by a truncated finite series,

$$U(x)=\sum_{j=1}^{m}\hat{U}_j\sin(j\pi x) \tag{4.6}$$

and approximate $\hat{U}_j$ using (4.5) where $\hat{f}_j$ is obtained by approximating the integral in (4.3).

For example, we could approximate $\hat{f}_j$ by

$$\hat{F}_j=2h\sum_{i=1}^{m}\sin(j\pi x_i)f(x_i) \tag{4.7}$$

where $x_i=ih$ with $h=1/(m+1)$. If we then calculate

$$\hat{U}_j=-\frac{1}{(j\pi)^2}\hat{F}_j \tag{4.8}$$

we can use (4.6) to compute an approximate value $U(x)\approx u(x)$ at any point x. In particular, we could compute $U(x_i)$ for each of the grid points x_i used in (4.7) to obtain an approximation to $u(x)$ on a discrete grid, similar to what is produced with a finite difference method. Denote the resulting vector of grid values by $U=[U(x_1),\ U(x_2),\ \ldots,\ U(x_m)]^T$. What has just been described is the *spectral method* for computing the approximate solution U.

Note that this is very closely related to what was done in Section 3.5.2 in relation to fast Poisson solvers. In fact, the sum in (4.7) can be written in vector form as

$$\hat{F} = 2hRF$$

exactly as in (3.23), where $F = [f(x_1),\ f(x_2),\ \dots,\ f(x_m)]^T$ (just as it is in (3.23) in the case of homogeneous boundary conditions). Also, computing the vector U by evaluating (4.6) at each grid point is exactly the operation

$$U = R\hat{U}$$

as in (3.25). In practice each of these operations would be done with the FFT.

The only difference between the spectral method and the "fast Poisson solver" (for this trivial equation!) is the manner in which we compute $\hat{U}_j$ from $\hat{F}_j$. In (3.24) we divided by the eigenvalues of the matrix A, since we were solving the linear system $Au = F$, whereas in (4.8) we divide by $-(j\pi)^2$, which is the j'th eigenvalue of the differential operator $-\partial^2/\partial x^2$ from the original differential equation. Intuitively this seems like a better thing to do, and indeed it is. The spectral approximation U converges to the solution of the differential equation much faster than the finite difference approximation u. (Note that in spite of using the exact eigenvalue, U is not the exact solution because it results from a truncated sine series and approximate integrals.)

4.2.1 Matrix interpretation

Note that this spectral method can be interpreted as solving a linear system of equations of the form $BU = F$. To see this, first note that (4.5) can be written in matrix form as

$$\hat{u} = M^{-1}\hat{F},$$

where M is the diagonal matrix $M = \text{diag}(-\pi^2,\ -(2\pi)^2,\ \dots,\ -(m\pi)^2)$. The j'th diagonal element is just μ_j from Section 2.10. Combining the various steps above then gives

$$U = R\hat{u} = RM^{-1}\hat{F} = RM^{-1}R^{-1}F$$

and hence U solves the system $BU = F$ with

$$B = RMR^{-1}. \tag{4.9}$$

This matrix B can be interpreted as an approximation to the second derivative operator, analogous to the tridiagonal matrix A of (2.10). Unlike A, however, the spectral matrix B is a dense matrix. With the finite difference method we approximated each $u''(x_i)$ using only 3 values $u(x_{i-1})$, $u(x_i)$, and $u(x_{i+1})$. The spectral approximation uses *all* the values $u(x_1),\ \dots,\ u(x_m)$ to approximate each $u''(x_i)$. Of course in practice we wouldn't want to form this matrix and solve the system using Gaussian elimination, since this would require $O(m^3)$ operations rather than the $O(m \log m)$ required using FFT's.

4.2.2 Accuracy

We can, however, use this matrix interpretation to help analyze the accuracy and stability of the method. We can define the local truncation error as usual by replacing the approximate solution with the true solution, and for this simple problem where $f(x) = u''(x)$ we see that

$$\tau_i = (Bu)_i - u''(x_i)$$

which is just the error in the spectral approximation to the second derivative.

Based on the convergence properties of Fourier series, it can be shown that if the solution $u(x)$ has p continuous derivatives, then the error in U decays like $1/m^p$ as $m \to \infty$. Since $h = 1/(m+1)$, this means that the method is p'th order accurate. For smooth solutions the method has a very high order of accuracy. If the solution is C^∞ (infinitely differentiable) then this is true for any value of p and it appears to be "infinite-order accurate"! This does not, however, mean that the error is zero. What it means is that it converges to zero faster than any power of $1/m$. Typically it is *exponentially fast* (for example the function $1/2^m$ decays to zero faster than any power of $1/m$ as $m \to \infty$). This is often called *spectral accuracy.*

There is a catch here however. In using a sine series representation of the function f or u we are obtaining an odd periodic function. In order for this to converge rapidly to the true function on the interval $[0,1]$ as we increase the number of terms in the series, we need the odd periodic extension of these functions to be sufficiently smooth, and not just the function specified in this interval. So in deciding whether $u(x)$ is p times continuously differentiable, we need to look at the function defined by setting

$$u(-x) = -u(x)$$

for $-1 \le x \le 0$ and then extended periodically with period 2 from the interval $[-1,1]$ to the whole real line. This requires certain properties in u at the endpoints $x=0$ and $x=1$. In particular, the extended $u(x)$ is C^∞ only if all even derivatives of u vanish at these two points along with u being C^∞ in the interior.

Such difficulties mean that spectral methods based on Fourier series are most suitable in certain special cases (for example if we are solving a problem with periodic boundary conditions, in which case we expect the solution to be periodic and have the required smoothness). Methods based on similar ideas can be developed using other classes of functions rather than trigonometric functions, and are often used in practice. For example, families of orthogonal polynomials such as Chebyshev or Legendre polynomials can be used, and fast algorithms developed that achieve spectral accuracy. This is discussed further in Section 4.2.5.

4.2.3 Stability

To see that the results quoted above for the local error carry over to the global error as we refine the grid, we also need to check that the method is stable. Using the matrix interpretation of the method this is easy to do in the 2-norm. The matrix B in (4.9) is easily seen to be symmetric (recall that $R^{-1} = 2hR = 2hR^T$ and so the 2-norm of B^{-1} is equal to its spectral radius, which is clearly $1/\pi^2$ independent of h. Hence the method is stable in the 2-norm.

4.2.4 Collocation property

Though it may not be obvious, the approximation we derived above for $U(x)$ in fact satisfies $U''(x_i) = f(x_i)$ at each of the points x_1 through x_m. In other words this spectral method is also a special form of a collocation method, as described in Section 4.1.

4.2.5 Pseudospectral methods based on polynomial interpolation

In Chapter 1 we derived finite difference approximations using polynomial interpolation. For example, through any three data points (arbitrarily spaced) there is a unique quadratic polynomial. Computing the derivative of this quadratic at one of the interpolation points yields a second-order accurate finite difference approximation to the derivative at this point based on the three data values. If we instead performed cubic interpolation through four data points we would obtain a third order accurate approximation. Higher degree polynomials based on more data points can be expected to yield higher order accurate approximations.

Suppose we take this idea to its logical conclusion and use the data at *all* the grid points in some interval in order to approximate the derivative at each point. Suppose we have m distinct points $x_1,\ x_2,\ \ldots,\ x_m$ in the interval $[a,b]$ and the values $U_1,\ U_2,\ \ldots,\ U_m$ at these points. Then there is a unique polynomial $p(x)$ of degree $m-1$ that interpolates this data. We could then approximate $u'(x_i)$ by $p'(x_i)$ at each point. Denote this approximation by U_i' and let U' be the vector of these approximations.

Each U_i' will depend on all the data values U_j and since interpolation and differentiation are both linear operators, the vectors U and U' will be related by a linear relation

$$U' = DU \tag{4.10}$$

where the matrix D will be dense.

What sort of accuracy might we hope for? Interpolation through three points gives $O(h^2)$ accuracy, and more generally interpolation through k points gives $O(h^{k-1})$ accuracy, at least if k is fixed as $h \to 0$ and the function we are working with is sufficiently smooth. We are now interpolating through m 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 $h \to 0$, and so this would give "spectral accuracy".

However, it is not at all clear that we really achieve this 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 will obtain disasterous results. High order polynomial interpolation at equally spaced points typically leads to a highly oscillatory polynomial that does not approximate the underlying smooth function well at all, and becomes exponentially *worse* as the grid is refined and the degree increases.

This idea can be saved, however, by choosing the grid points to be clustered near the ends of the interval in a particular manner. This is initially explained most easily on the interval $[-1, 1]$. Later we can translate the results to other intervals. Recall the form of the error in polynomial interpolation. If $p(x)$ is the polynomial of degree $m - 1$ that interpolates $f(x)$ at $x_1,\ x_2,\ \ldots,\ x_m$, then

$$\begin{aligned} f(x) - p(x) &= f[x_1, x_2, \ldots, x_n, x](x - x_1)(x - x_2)\cdots(x - x_m) \\ &= \frac{f^{(m)}(\xi)}{m!}(x - x_1)(x - x_2)\cdots(x - x_m) \end{aligned} \tag{4.11}$$

for some point $\xi \in [-1, 1]$. Here $f[x_1, x_2, \ldots, x_n, x]$ is the divided difference of f based on the points $x_1,\ x_2,\ \ldots,\ x_n,\ x$. Note that if x is near one end of the interval $[-1, 1]$ and x_i is near the other, then $|x - x_i| \approx 2$, so many of the terms in this product $\prod(x - x_i)$ are close to 2 if we are near one end of the interval.

A much better choice of interpolation points (in fact the optimal choice in a sense discussed below) is to use the *Chebyshev points*

$$\xi_j = \cos\left(\frac{(j - 1/2)\pi}{m}\right) \qquad \text{for } j = 1,\ 2,\ \ldots,\ m. \tag{4.12}$$

These points are the projection onto the x-axis of equally spaced points on the unit circle and are clustered near the ends of the interval. If x is near one end of the interval then there are still many factors nearly equal to 2, but there are also many factors that are very small because there are many interpolation points nearby.

With this choice of interpolation points, the polynomial

$$T_m(x) = 2^{(m-1)} \prod_{j=1}^{m} (x - \xi_j) \tag{4.13}$$

is bounded by one in magnitude over the entire interval $[-1, 1]$. The scaled version of this polynomial that comes into the error estimate for interpolation at the Chebyshev points is thus bounded by 2^{1-m}.

The polynomial $T_m(x)$ is the mth degree *Chebyshev polynomial* and the Chebyshev points ξ_j are the roots of this polynomial. This polynomial *equioscillates* between $+1$ and -1, taking these extreme values at $m + 1$ points over the interval as shown in Figure 4.1. Any other polynomial of degree m with leading coefficient 2^{m-1} will go above 1 in magnitude at some point in the interval. The Chebyshev polynomial scaled by 2^{1-m} solves the mini-max optimization problem of choosing the monic polynomial $p(x)$ of degree m that minimizes

$$\max_{-1\le x\le 1} |p(x)|.$$

(A polynomial of the form $\prod_{i=1}^m (x-x_i) = x^m + \cdots$ is called a *monic polynomial* because the coefficient of the highest order term x^m is 1.)

The Chebyshev polynomials have a number of interesting properties and are very important in polynomial approximation theory, with many applications in numerical analysis. They are only briefly discussed here. For more discussion and other applications, see for example [Dav63], [Tre00],

The first few Chebyshev polynomials are

$$\begin{aligned} T_0(x) &= 1\\ T_1(x) &= x\\ T_2(x) &= 2x^2-1\\ T_3(x) &= 4x^3-3x \end{aligned}$$

In general they satisfy a 3-term recurrence relation

$$T_{m+1}(x) = 2xT_m(x) - T_{m-1}(x) \tag{4.14}$$

which allows the computatation of the Chebyshev polynomial of any degree.

For our purposes we don't need the polynomials themselves, only the expression (4.12) for the roots, which we use as the interpolation points to define the differentiation matrix D of (4.10). If U is the vector of function values at these points then $U' = DU$ will be the vector of approximate derivative values, and will exhibit spectral accuracy if the underlying function is sufficiently smooth. We can approximate the second derivative by again applying D,

$$U'' = DU' = D^2U, \tag{4.15}$$

so the matrix D^2 approximates the second derivative operator, again with spectral accuracy for smooth functions.

So far we have only talked about approximating derivatives, but we can now use this approximation to solve differential equations. In order to solve the boundary value problem $u''(x) = f(x)$, for example, we can solve a discrete system of the form

$$D^2U = F.$$

However, we also need to impose boundary conditions, and for this reason it is actually best to choose a slightly different set of grid points. Rather than the Chebyshev points (4.12), which are the roots of the Chebyshev polynomial, we instead use the points

$$x_j = \cos(j\pi/(m+1)), \qquad j = 0,\ 1,\ 2,\ \ldots,\ m+1, \tag{4.16}$$

which are the extreme points of the Chebyshev polynomial, the points where this polynomial attains the values ±1. These points are distributed in essentially the same way as the roots, and work nearly as well for polynomial interpolation, but include the two endpoints of the interval $x_{m+1} = -1$ and $x_0 = 1$. (Note that these points, and also the ξ_j of (4.12), are labeled in decreasing order.)

Explicit formulas for the differentiation matrix D corresponding to these grid points can be worked out, along with D^2, see [Tre00]. These are dense matrices and so solving the discrete system $D^2U = F$ by Gaussian elimination requires $O(m^3)$ operations. This is considerably more expensive than using the second-order accurate finite difference method, which gives a tridiagonal matrix. It might still be worthwhile because of the greatly increased accuracy, but in fact it turns out that the FFT can also be used for solving this dense problem and the cost can be reduced to $O(m \log m)$.

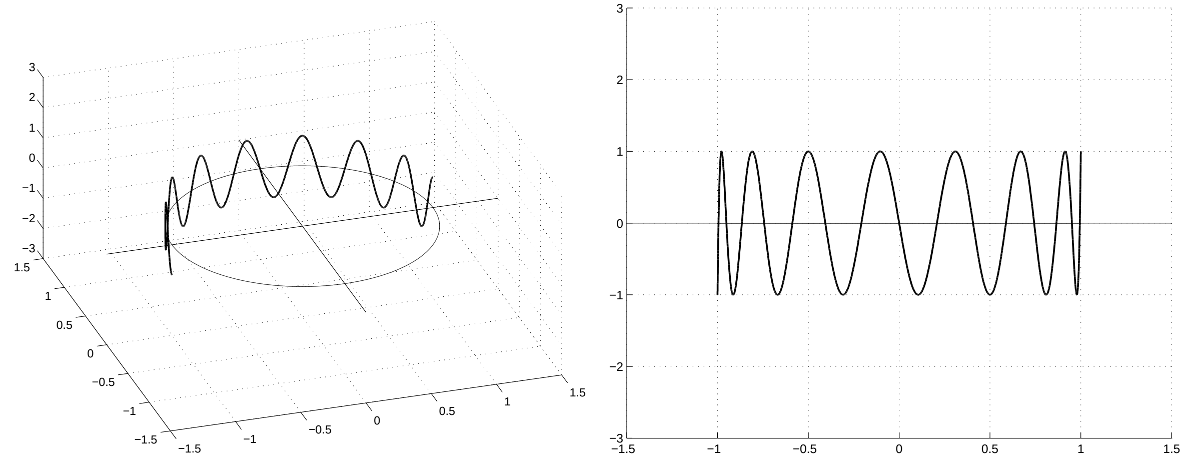


Figure 4.1: The Chebyshev polynomial viewed as a function $C_m(\theta)$ on the unit disk $e^{i\theta}$ and when projected on the x axis, *i.e.*, as a function of $x = \cos(\theta)$. Shown for $m = 15$.

In general there is no way to use the FFT to solve polynomial interpolation problems, since the Fourier transform is based on "trigonometric polynomials" rather than algebraic polynomials. However, the Chebyshev polynomials are very special and have the property that for x in the range $-1 \le x \le 1$,

$$T_m(x) = \cos(m \arccos x). \tag{4.17}$$

This does not look much like a polynomial, but it is because $\cos(m\theta)$ can be written as a polynomial in $\cos(\theta)$ using trig identities, and then set $x = \cos(\theta)$. Now consider the function

$$C_m(\theta) = \cos(m\theta) = \mathrm{Re}(e^{im\theta}) \tag{4.18}$$

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. It is a function that passes through zero at the points

$$\tilde{\theta}_j = \frac{(j - 1/2)\pi}{m}. \tag{4.19}$$

Note that the roots (4.12) of the Chebyshev polynomial T_m are just $\xi_j = \cos(\tilde{\theta}_j)$ and in fact the Chebyshev polynomial is just the projection onto the x-axis of the function $C_m(\theta)$; see Figure 4.1. Since $x = \cos(\theta)$, this correspondence gives $\theta = \arccos(x)$, resulting in (4.17).

The function $C_m(\theta)$ equioscillates between the values ± 1 at the points $\theta_j = j\pi/(m+1)$ and the projection of these points onto the x-axis gives the interpolation points (4.16). In order to use the FFT, we interpret the data values $U_j = u(x_j)$ as being function values at the points θ_j on the unit circle. Our polynomial approximation on the real line can be reinterpreted as a trigonometric polynomial on the unit circle, and so we want to compute a Fourier transform of this data on the unit circle. Since the points θ_j are equally spaced on the unit circle, the FFT can be applied to accomplish this. More details may be found in [Tre00], for example.

It is quite remarkable that interpolating at the Chebyshev points, which is optimal from the approximation standpoint, also allows the use of Fourier transform techniques to solve this polynomial problem. Of course this is not just coincidence, but a full understanding of the underlying mathematics goes beyond the scope of these notes.

4.3 The finite element method

The finite element method determines an approximate soution that is a linear combination of some specified basis functions in a very different way from collocation or expansion in eigenfunctions. This method is typically based on some "weak form" of the differential equation, which roughly speaking means that we have integrated the equation. Here we give only a very superficial introduction to the finite element method.

Consider, for example, the heat conduction problem in one dimension with a variable conductivity $\kappa(x)$ so the steady-state equation is

$$(\kappa u')' = f. \tag{4.20}$$

Again for simplicity assume that the boundary conditions are $u(0) = u(1) = 0$. If we multiply both sides of the equation (4.20) by an arbitrary smooth function $v(x)$ and integrate the resulting product over the domain $[0, 1]$, we obtain

$$\int_0^1 (\kappa(x)u'(x))'v(x)\,dx = \int_0^1 f(x)v(x)\,dx. \tag{4.21}$$

On the left-hand side we can integrate by parts. Since v is arbitrary, let's restrict our attention to v that satisfy $v(0) = v(1) = 0$ so that the boundary terms drop out, yielding

$$-\int_0^1 \kappa(x)u'(x)v'(x)\,dx = \int_0^1 f(x)v(x)\,dx. \tag{4.22}$$

It can be shown that if $u(x)$ satisfies this equation for all v in some suitable class of functions, then $u(x)$ is in fact the solution to the original differential equation.

Now suppose we replace $u(x)$ by an approximation $U(x)$ in this expression, where $U(x)$ is a linear combination of specified basis functions,

$$U(x) = \sum_{j=1}^m c_j\phi_j(x). \tag{4.23}$$

Let's suppose that our basis functions are chosen to satisfy $\phi_j(0) = \phi_j(1) = 0$, so that $U(x)$ automatically satisfies the boundary conditions regardless of how we choose the c_j. Then we could try to choose the coefficients c_j in $U(x)$ so that the equality (4.22) is satisfied for a large class of functions $v(x)$. Since we only have m free parameters, we can't require that (4.22) be satisfied for all smooth functions $v(x)$, but we can require that it be satisfied for all functions in some m-dimensional function space. Such a space is determined by a set of m basis functions $\psi_i(x)$ (which might or might not be the same as the functions $\phi_j(x)$). If we require that (4.22) be satisfied for the special case where v is chosen to be any one of these functions, then by linearity (4.22) will also be satisfied for any v that is an arbitrary linear combination of these functions, and hence for all v in this m-dimensional linear space.

Hence we are going to require that

$$-\int_0^1 \kappa(x)\left(\sum_{j=1}^m c_j\phi_j'(x)\right)\psi_i'(x)\,dx = \int_0^1 f(x)\psi_i(x)\,dx \tag{4.24}$$

for $i = 1,\ 2,\ \ldots,\ m$. We can rearrange this to give

$$\sum_{j=1}^m K_{ij}c_j = \int_0^1 f(x)\psi_i(x)\,dx \tag{4.25}$$

where

$$K_{ij} = -\int_0^1 \kappa(x)\phi_j'(x)\psi_i'(x)\,dx. \tag{4.26}$$

The equations (4.25) for $i = 1,\ 2,\ \ldots,\ m$ give an $m \times m$ linear system of equations to solve for the c_j, which we could write as

$$Kc = F$$

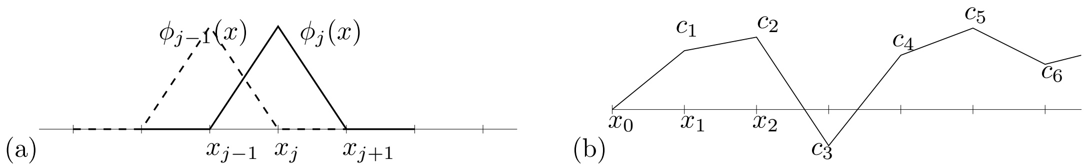


Figure 4.2: (a) Two typical basis functions $\phi_{j-1}(x)$ and $\phi_j(x)$ with continuous piecewise linear elements. (b) $U(x)$, a typical linear combination such basis functions.

with

$$F_i = \int_0^1 f(x)\psi_i(x)\,dx. \tag{4.27}$$

The functions ψ_i are generally called "test functions" while the basis functions ϕ_i for our approximate solution are called "trial functions". Frequently the same basis functions are used for both spaces. The resulting method is known as the *Galerkin method.* If the trial space is different from the test space we have a *Petrov-Galerkin method.*

Example 4.2. As a specific example, consider the Galerkin method for the above problem with basis functions defined as follows on a uniform grid with $x_i = ih$, and $h = 1/(m+1)$. The j'th basis function $\phi_j(x)$ is

$$\phi_j(x) = \begin{cases} (x - x_{j-1})/h & \text{if } x_{j-1} \le x \le x_j \\ (x_{j+1} - x)/h & \text{if } x_j \le x \le x_{j+1} \\ 0 & \text{otherwise} \end{cases} \tag{4.28}$$

Each of these functions is continuous and piecewise linear, and $\phi_j(x)$ takes the value 1 at x_j and the value 0 at all other nodes x_i for $i \neq j$. (See Figure 4.2(a).) Note that any linear combination (4.23) of these functions will still be continuous and piecewise linear, and will take the value x_i at the point x_i since $U(x_i) = \sum_j c_j\phi_j(x_i) = c_i$ since all other terms in the sum are zero. Hence the function $U(x)$ has the form shown in Figure 4.2(b).

The set of functions $\{\phi_j(x)\}$ form a basis for the space of all continuous piecewise linear functions defined on $[0,1]$ with $u(0) = u(1) = 0$ and with kinks at the points $x_1,\ x_2,\ \ldots,\ x_m$, which are called the *nodes*. Note that the coefficient c_j can be interpreted as the value of the approximate solution at the point x_j.

To use these basis functions in the Galerkin equations (4.24) (with $\psi_j = \phi_j$), we need to compute the derivatives of these basis functions and then the elements of the matrix K and right-hand side F. We have

$$\phi_j'(x) = \begin{cases} 1/h & \text{if } x_{j-1} < x < x_j \\ -1/h & \text{if } x_j < x < x_{j+1} \\ 0 & \text{otherwise.} \end{cases}$$

For general functions $\kappa(x)$ we might have to compute an approximation to the integral in (4.26), but as a simple example consider the case $\kappa(x) \equiv 1$ (so the equation is just $u''(x) = f(x)$). Then we can compute that

$$K_{ij} = -\int_0^1 \phi_j'(x)\phi_i'(x)\,dx = \begin{cases} 1/h & \text{if } j = i-1 \text{ or } j = i+1, \\ -2/h & \text{if } j = i, \\ 0 & \text{otherwise.} \end{cases}$$

The matrix K is quite familiar (except for the different power of h):

$$K = \frac{1}{h}\begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix}. \tag{4.29}$$

In some cases we may be able to evaluate the integral in (4.27) for F_i explicitly. More generally we might use a discrete approximation. Note that since $\phi_i(x)$ is nonzero only near x_i, and $\int_0^1 \phi_i(x)\,dx = h$, this is roughly

$$F_i \approx hf(x_i). \tag{4.30}$$

In fact the trapezoidal method applied to this integral on the same grid would give exactly this result. Using (4.30) in the system $Kc = F$, and dividing both sides by h, gives exactly the same linear system of equations that we obtained in Section 2.4 from the finite difference method (for the case $\alpha = \beta = 0$ we are considering here).

Some comments on this method:

- The matrix K above is tridiagonal because each $\phi_j(x)$ is nonzero on only two elements, for $x_{j-1} < x < x_{j+1}$. The function $\phi_i'(x)\phi_j'(x)$ is identically zero unless $j = i-1,\ i$ or $i+1$. More generally, if we choose basis functions that are nonzero only on some region $x_{j-b} < x < x_{j+a}$, then the resulting matrix would be banded with b diagonals of nonzeros below the diagonal and a bands above. In the finite element method one almost always chooses local basis functions of this sort, that are each nonzero over only a few elements.
- Why did we integrate by parts to obtain equation (4.22), rather than working directly with (4.21)? One could go through the same process based on (4.21), but then we would need an approximate $U(x)$ with meaningful second derivatives. This would rule out the use of the simple piecewise linear basis functions used above. (Note that the piecewise linear functions don't have meaningful first derivatives at the nodes, but since only integrals of these functions are used in defining the matrix this is not a problem.)
- This is one advantage of the finite element method over collocation, for example. One can often use functions $U(x)$ for which the original differential equation does not even make sense because U is not sufficiently smooth.
- There are other good reasons for integrating by parts. The resulting equation (4.22) can also be derived from a variational principle and has physical meaning in terms of minimizing the "energy" in the system. (See, *e.g.*, [SF73].)

4.3.1 Two space dimensions

In the last example we saw that the one-dimensional finite element method based on piecewise linear elements is equivalent to the finite difference method derived in Section 2.4. Since it is considerably more complicated to derive via the finite element approach, this may not seem like a useful technique. However, in more than one dimension this method can be extended to irregular grids on complicated regions for which it would not be so easy to derive a finite difference method.

Consider, for example, the Poisson problem with homogeneous Dirichlet boundary conditions on the region shown in Figure 4.3, which also shows a fairly coarse "triangulation" of the region. The points (x_k, y_k) at the corners of the triangles are called *nodes*. The Galerkin form of the Poisson problem is

$$-\int\int_\Omega \nabla u \cdot \nabla v\,dx\,dy = \int\int_\Omega fv\,dx\,dy. \tag{4.31}$$

This should hold for all test functions $v(x,y)$ in some class. Again we can approximate $u(x,y)$ by some linear combination of specified basis functions:

$$U(x,y) = \sum_{j=1}^N c_j\phi_j(x,y). \tag{4.32}$$

Taking an approach analogous to the one-dimensional case above, we can define a basis function $\phi_j(x,y)$ associated with each node (x_j, y_j) to be the unique function that is linear on each triangle, and which

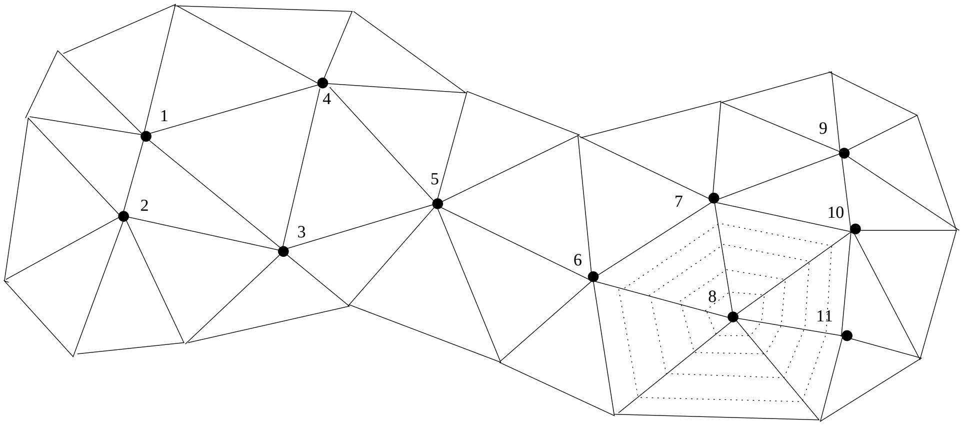


Figure 4.3: Triangulation of a two-dimensional region with 11 nodes. Contourlines for the basis function $\phi_8(x, y)$ are also shown as dashed lines.

takes the value 1 at the node (x_j, y_j) and 0 at all other nodes. This function is continuous across the boundaries between triangles and nonzero only for the triangles that have Node j as a corner. For example, Figure 4.3 indicates contour lines for the basis function $\phi_8(x, y)$ as dashed lines.

Using (4.32) in (4.31) gives an $N \times N$ linear system of the form $Kc = F$ where

$$K_{ij} = -\int\int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, dx \, dy. \tag{4.33}$$

These gradients are easy to compute and in fact are constant within each triangle since the basis function is linear there. Since $\nabla\phi_i$ is identically zero in any triangle for which Node i is not a corner, we see that $K_{ij} = 0$ unless Nodes i and j are two corners of a common triangle. For example, in Figure 4.3 the eighth row of the matrix K will have have nonzeros only in columns 6, 7, 8, 10, and 11.

Note also that K will be a symmetric matrix, since the expression (4.33) is symmetric in i and j. It can also be shown to be positive definite.

For a typical triangulation on a much finer grid, we would have a large but very sparse matrix K. The structure of the matrix, however, will not generally be as simple as what we would obtain with a finite difference method on a rectangular grid. The pattern of nonzeros will depend greatly on how we order the unknowns and equations. Direct methods for solving such systems efficiently often rely on sophisticated graph theoretical algorithms for ordering the unknowns to minimize the bandwidth; see for example [DER86]. In practice iterative methods are often used to solve the sparse linear system.

4.4 Exercises

Exercise 4.1 *Consider the problem from Example 4.2 but suppose we have more general Dirichlet boundary conditions $u(0) = \alpha$ and $u(1) = \beta$. Introduce two additional basis functions $\phi_0(x)$ and $\phi_{m+1}(x)$ which are also defined by (4.28). Then we know $c_0 = \alpha$ and $c_{m+1} = \beta$ and these terms in the extended sum appearing in (4.22) can be moved to the right hand side. Carry this through to see that we get essentially the system (2.9) in this more general case as well.*

Exercise 4.2 *Consider the problem from Example 4.2 but suppose $\kappa(x)$ is a general smooth function. Use the midpoint rule to approximate the quadrature to compute*

$$K_{ij} = -\int_0^1 \kappa(x)\phi_j'(x)\phi_i'(x)\,dx \approx h\sum_{i=0}^{m} \kappa(x_{i+1/2})\phi_j'(x_{i+1/2})\phi_i'(x_{i+1/2})$$

and confirm that this gives a matrix system with the same structure as (2.49).

Chapter 5

Iterative Methods for Sparse Linear Systems

This chapter contains a brief overview of several iterative methods for solving the large sparse linear systems that arise from elliptic equations, either from finite-difference approximations (Chapter 3) or from finite element approximations (Section 4.3). Large sparse linear systems arise from many other practical problems too, of course, and the methods discussed here are important more generally.

The classical Jacobi and Gauss-Seidel methods will first be introduced for the 5-point Laplacian. Next we will analyze the convergence properties of these methods based on viewing them in terms of matrix splittings.

The SOR and conjugate gradient methods will also be briefly introduced. The reader can find considerably more theoretical analysis of these methods in the literature. See for example [GO89], [GO92], [Gre97], [HY81], [She94], [TB97], [Var62], [You71].

5.1 Jacobi and Gauss-Seidel

Except when the matrix has very special structure and fast direct methods of the type discussed in Section 3.5 apply, iterative methods are usually the method of choice for large sparse linear systems. 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, but they have the virtue of being simple to explain. More efficient methods are discussed later in this chapter.

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

$$u_{ij} = \frac{1}{4}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) - \frac{h^2}{4} f_{ij}. \tag{5.1}$$

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 analog 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 towards or away from this point.

The equation (5.1) suggests the following iterative method to produce a new estimate $u^{[k+1]}$ from a current guess $u^{[k]}$:

$$u_{ij}^{[k+1]} = \frac{1}{4}(u_{i-1,j}^{[k]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k]} + u_{i,j+1}^{[k]}) - \frac{h^2}{4} f_{ij}. \tag{5.2}$$

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 $u^{[0]}$ (though 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
```

where it is assumed that `u` initially contains the guess $u^{[0]}$ and that boundary data is stored into `u(1,:), u(m+2,:), u(:,1), and u(:,m+2).` The indexing is off by one from what one might expect 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 by

```
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 (5.2). In the second code we have already updated `u(i-1,j)` and `u(i,j-1)` before we update `u(i,j)`, and these new values will be used instead of the old ones. The latter code thus corresponds to the method

$$u_{ij}^{[k+1]} = \frac{1}{4}(u_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k]} + u_{i,j-1}^{[k+1]} + u_{i,j+1}^{[k]}) - \frac{h^2}{4} f_{ij}. \tag{5.3}$$

This is in fact 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's method.

Convergence of these methods will be discussed in Section 5.2. 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 them all.)

- 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. In Chapter 5 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, though 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 can achieve this, not only for Poisson's problem but also for many other elliptic equations.

5.2 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-dimenisional analog 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 of the analysis done here carries over almost unchanged to the 2-dimensional (or even 3-dimensional) case.

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

$$\text{Jacobi:} \qquad u_i^{[k+1]} = \frac{1}{2}(u_{i-1}^{[k]} + u_{i+1}^{[k]} - h^2 f_i), \tag{5.4}$$

$$\text{Gauss-Seidel:} \qquad u_i^{[k+1]} = \frac{1}{2}(u_{i-1}^{[k+1]} + u_{i+1}^{[k]} - h^2 f_i), \tag{5.5}$$

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

$$A = M - N \tag{5.6}$$

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

$$Mu^{[k+1]} = Nu^{[k]} + f. \tag{5.7}$$

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 (5.7) 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. In order to discuss these in a unified framework, write

$$A = D - L - U \tag{5.8}$$

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

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

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 (5.7) is then diagonal and extremely easy to solve:

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

which agrees with (5.4).

In Gauss-Seidel, we take M to be the full lower triangular portion of A, so $M = D - L$ and $N = U$. The system (5.7) is then solved using forward substitution, which results in (5.5).

To analyze these methods, we derive from (5.7) the update formula

$$\begin{aligned} u^{[k+1]} &= M^{-1}Nu^{[k]} + M^{-1}f \\ &\equiv Gu^{[k]} + b, \end{aligned} \tag{5.9}$$

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

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

$$u^* = Gu^* + b. \tag{5.10}$$

This shows that the true solution is a fixed point, or equilibrium, of the iteration (5.9), *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 towards u^* if we start from some incorrect initial guess.

If $e^{[k]} = u^{[k]} - u^*$ represents the error, then subtracting (5.10) from (5.9) gives

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

and so after k steps we have

$$e^{[k]} = G^k e^{[0]}. \tag{5.11}$$

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

$$G^k = R\Gamma^k R^{-1}, \tag{5.12}$$

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.

5.2.1 Rate of convergence

From (5.11) we can also determine how rapidly the method can be expected to converge in cases where it is convergent. Using (5.12) in (5.11) and using the 2-norm, we obtain

$$\|e^{[k]}\|_2 \le \|\Gamma^k\|_2\|R\|_2\|R^{-1}\|_2\|e^{[0]}\|_2 = \rho^k\kappa_2(R)\|e^{[0]}\|_2 \tag{5.13}$$

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 (meaning it commutes with its transpose, in particular if it is symmetric as when Jacobi is applied to the Poisson problem), then the eigenvectors are orthogonal and $\kappa_2(R) = 1$. In this case we have

$$\|e^{[k]}\|_2 \le \rho^k\|e^{[0]}\|_2. \tag{5.14}$$

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.

The *average rate of convergence* for a method after k iterations is sometimes defined by

$$R_k(G) = -\frac{1}{k}\log\|G^k\|,$$

while the *asymptotic rate of convergence* is

$$R_\infty(G) = -\log(\rho(G)).$$

Example 5.1. 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

$$\rho(G) = |\gamma_1| = \cos(\pi h) \approx 1 - \frac{1}{2}\pi^2h^2 + O(h^4). \tag{5.15}$$

The spectral radius is less than one for any $h > 0$ and the Jacobi method converges, but we see that $\rho(G) \to 1$ as $h \to 0$ and for small h is very close to one, resulting in very slow convergence.

How many iterations are required to obtain a good solution? Since G is symmetric the eigenvector matrix R has $\kappa_2(R) = 1$ and so (5.13) gives $\|e^{[k]}\|_2 \le \rho^k \|e^{[0]}\|_2$. 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

$$k \approx \log(\epsilon)/\log(\rho). \tag{5.16}$$

How small should we choose ϵ? To get full machine precision we might choose ϵ to be close to the machine roundoff level. However, this would typically be very wasteful. For one thing, we rarely need this many correct digits. More importantly, 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

$$k = (\log(C) + 2\log(h))/\log(\rho). \tag{5.17}$$

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 (5.17) gives

$$k = O(m^2 \log m) \quad \text{as } m \to \infty. \tag{5.18}$$

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

$$\text{total work} = O(m^3 \log m).$$

Of course this tridiagonal problem can be solved exactly in $O(m)$ work, so we would be very foolish to use an iterative method at all here!

For a Poisson problem in 2 or 3 dimensions it can be verified that (5.18) still holds, though 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 2 dimensions we would thus find that

$$\text{total work} = O(m^4 \log m). \tag{5.19}$$

Recall from Section 3.5 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.

For the Gauss-Seidel method applied to the Poisson problem, it can be shown that

$$\rho(G) = 1 - \pi^2 h^2 + O(h^4) \quad \text{as } h \to 0. \tag{5.20}$$

This still approaches 1 as $h \to 0$, but is better than (5.15) by a factor of 2 and the number of iterations required to reach a given tolerance will typically be half the number required with Jacobi. The order of magnitude figure (5.19) still holds, however, and this method is also not widely used.

5.2.2 SOR

If we look at how iterates $u^{[k]}$ behave when Gauss-Seidel is applied to a typical problem, we will generally 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

[1] Assuming we are using some grid function norm, as discussed in Appendix A1. Note that for the 2-norm in one dimension this requires introducing a factor of $\sqrt{h}$ in the definition of both $\|e^{[k]}\|$ and $\|e^{[0]}\|$, but these factors cancel out in choosing an appropriate ϵ.

u_i in the right direction, but is far too conservative in the amount it allows u_i to move. This suggests that we use the following two-stage update, illustrated again for the problem $u'' = f$:

$$\begin{aligned} u_i^{\text{GS}} &= \frac{1}{2}(u_{i-1}^{[k+1]} + u_{i+1}^{[k]} - h^2 f_i) \\ u_i^{[k+1]} &= u_i^{[k]} + \omega(u_i^{\text{GS}} - u_i^{[k]}) \end{aligned} \tag{5.21}$$

where ω is some scalar parameter. If $\omega = 1$ then $u_i^{[k+1]} = u_i^{\text{GS}}$ is the Gauss-Seidel update. If $\omega > 1$ then we move farther than Gauss-Seidel suggests. In this case the method is known as *successive overrelaxation* (SOR).

If $\omega < 1$ then we would be underrelaxing, rather than overrelaxing. This would be even less effective than Gauss-Seidel as a stand-alone iterative method for most problems, though underrelaxation is sometimes used in connection with multigrid methods [BEM01].

The formulas in (5.21) can be combined to yield

$$u_i^{[k+1]} = \frac{\omega}{2}(u_{i-1}^{[k+1]} + u_{i+1}^{[k]} - h^2 f_i) + (1-\omega)u_i^{[k]}, \tag{5.22}$$

and it can be verified that this corresponds to a matrix splitting method with

$$M = \frac{1}{\omega}(D - \omega L), \qquad N = \frac{1}{\omega}((1-\omega)D + \omega U).$$

Analyzing this method is considerably trickier than 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 and $D - \omega L$ is nonsingular, then the SOR method converges for all $0 < \omega < 2$. Young [You50] 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 [GO92], [HY81], [Var62], [You71]. See also [LT88] for a different introductory treatment based on Fourier series and modified equations in the sense of Section 13.6, and [ALY88] for applications of this approach to the 9-point Laplacian.)

For the Poisson problem 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 (5.18), is now

$$k_{opt} = O(m \log m).$$

Figure 5.1 shows some computational results for the methods described above on the 2-point boundary value problem.

5.3 Descent methods and conjugate gradients

The conjugate gradient (CG) method is a powerful technique for solving linear systems $Au = f$ when the matrix A is symmetric positive definite (SPD), or negative definite since negating the system then

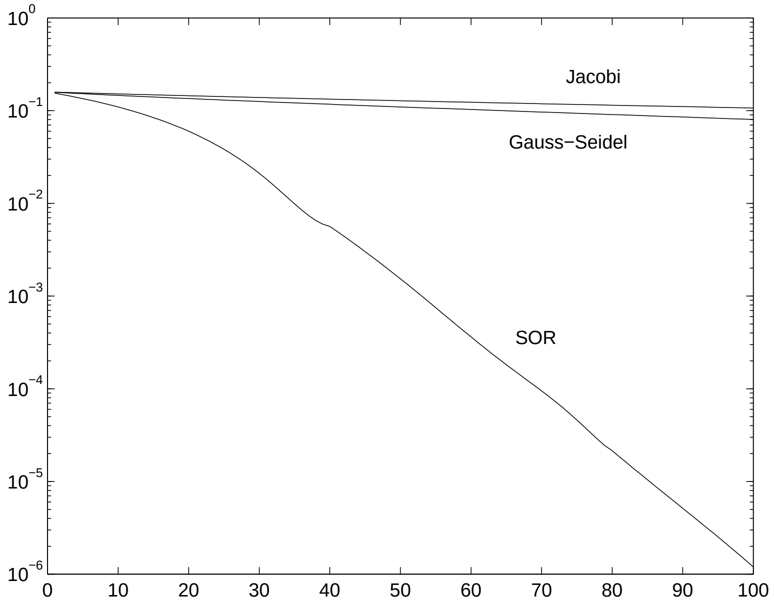


Figure 5.1: Errors vs. k for three methods.

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 [Gre97], [She94], [TB97] 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

$$\phi(u) = \frac{1}{2}u^T Au - u^T f. \tag{5.23}$$

This is a quadratic function of the variables $u_1, \ \ldots, \ u_m$. For example, if $m = 2$ then

$$\phi(u) = \phi(u_1, u_2) = \frac{1}{2}(a_{11}u_1^2 + 2a_{12}u_1u_2 + a_{22}u_2^2) - u_1f_1 - u_2f_2.$$

Note that since A is symmetric, $a_{21} = a_{12}$. If A is positive definite then plotting $\phi(u)$ as a function of u_1 and u_2 gives a parabolic bowl as shown in Figure 5.2(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

$$\begin{aligned} \frac{\partial \phi}{\partial u_1} &= a_{11}u_1 + a_{12}u_2 - f_1 = 0 \\ \frac{\partial \phi}{\partial u_2} &= a_{21}u_1 + a_{22}u_2 - f_2 = 0. \end{aligned} \tag{5.24}$$

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 (5.23) has a unique minimum at the point u^* where $\nabla\phi(u^*) = 0$, and

$$\nabla\phi(u) = Au - f \tag{5.25}$$

so the minimizer solves the linear system $Au = f$.

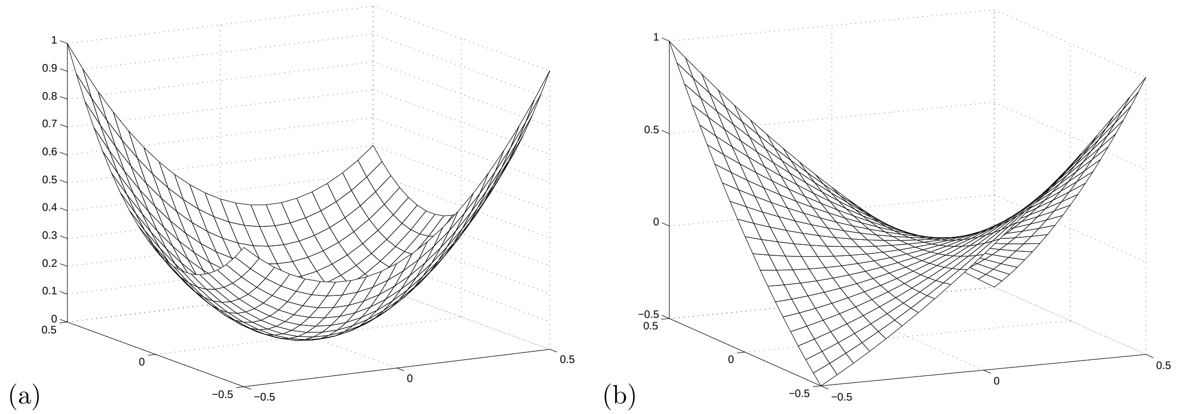


Figure 5.2: (a) The function $\phi(u)$ for $m = 2$ in a case where A is symmetric and positive definite. (b) The function $\phi(u)$ for $m = 2$ in a case where A is symmetric but indefinite.

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 or 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 5.2(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 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.

5.3.1 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 generally 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

$$u_k = u_{k-1} - \alpha_{k-1}\nabla\phi(u_{k-1}) \tag{5.26}$$

for some scalar α_{k-1}, chosen to solve the minimization problem

$$\min_{\alpha\in\mathbb{R}} \phi\left(u_{k-1} - \alpha_{k-1}\nabla\phi(u_{k-1})\right). \tag{5.27}$$

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 (5.23), the gradient is given by (5.25) and so

$$\nabla\phi(u_{k-1}) = Au_{k-1} - f \equiv -r_{k-1} \tag{5.28}$$

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 (5.27), we compute the derivative with respect to α and set this to zero. Note that

$$\phi(u+\alpha r) = \left(\frac{1}{2}u^TAu - u^Tf\right) + \alpha(r^TAu - r^Tf) + \frac{1}{2}\alpha^2 r^TAr \tag{5.29}$$

and so

$$\frac{d\phi(u+\alpha r)}{d\alpha} = r^TAu - r^Tf + \alpha r^TAr.$$

Setting this to zero and solving for α gives

$$\alpha = \frac{r^Tr}{r^TAr}. \tag{5.30}$$

The steepest descent algorithm thus takes the form:

```
choose a guess $u_0$
for $k = 1,\ 2,\ \dots$
    $r_{k-1} = f - Au_{k-1}$
    if $\|r_{k-1}\|$ is less than some tolerance then stop
    $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(r_{k-1}^T A r_{k-1})$
    $u_k = u_{k-1} + \alpha_{k-1} r_{k-1}$
    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

$$\begin{aligned} r_k &= f - Au_k \\ &= f - A(u_{k-1} + \alpha_{k-1}r_{k-1}) \\ &= r_{k-1} - \alpha_{k-1}Ar_{k-1}. \end{aligned} \tag{5.31}$$

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 $k = 1,\ 2,\ \dots$
    $w_{k-1} = Ar_{k-1}$
    $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(r_{k-1}^T w_{k-1})$
    $u_k = u_{k-1} + \alpha_{k-1} r_{k-1}$
    $r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
    if $\|r_k\|$ is less than some tolerance then stop
    end
```

Figure 5.3 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 of the steepest descent algorithm $u_0,\ u_1, \ldots$. 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 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 the particular starting vector u_0 chosen.

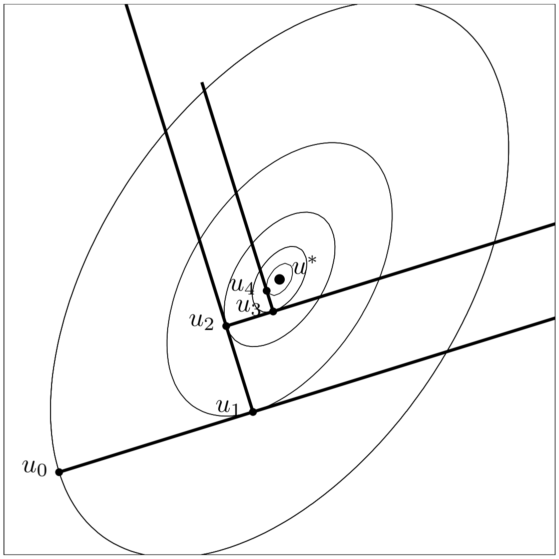


Figure 5.3: Several iterates of the method of steepest descent in the case $m = 2$. The concentric ellipses are level sets of $\phi(u)$.

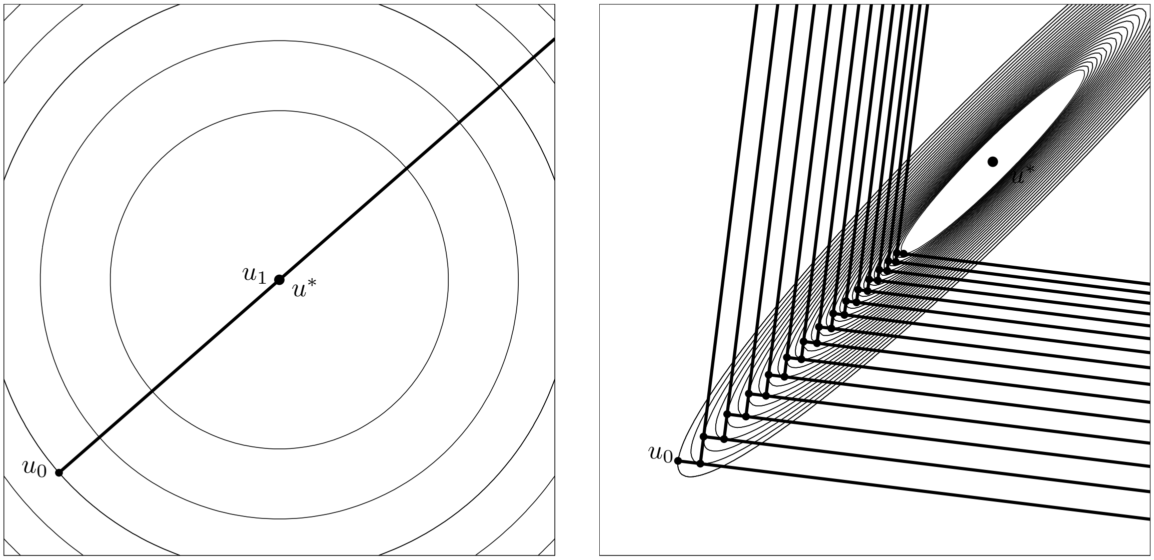


Figure 5.4: (a) If A is a scalar multiple of the identity then the level sets of $\phi(u)$ are circular and steepest descent converges in one iteration from any initial guess u_0. (b) If the level sets of $\phi(u)$ are far from circular then steepest descent may converge slowly.

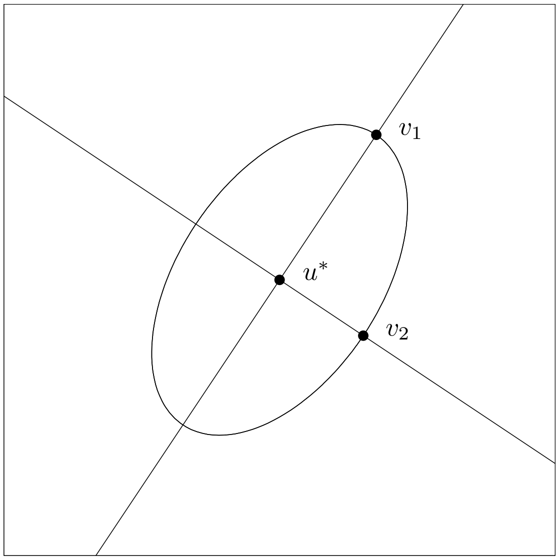


Figure 5.5: The major and minor axes of the elliptical level set of $\phi(u)$ point in the directions of the eigenvectors of A.

Figure 5.4(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 5.4(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 5.3 and 5.4, each ellipse can be characterized by a major and minor axis, as shown in Figure 5.5 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.*,

$$Av_j - f = \lambda_j(v_j - u^*) \tag{5.32}$$

for some scalar λ_j. Since $f = Au^*$, this gives

$$A(v_j - u^*) = \lambda_j(v_j - u^*) \tag{5.33}$$

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 non-circular and there are two unique directions for which the relation (5.32) holds, since there are two 1-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 5.3(a).

The length of the major and minor axes are 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

$$\frac{1}{2}v_j^T Av_j - v_j^T Au^* = 1. \tag{5.34}$$

Taking the inner product of (5.33) with $(v_j - u^*)$ and combining with (5.34) yields

$$\|v_j - u^*\|_2^2 = \frac{2 + u^{*T}Au^*}{\lambda_j}. \tag{5.35}$$

Hence the ratio of the length of the major axis to the length of the minor axis is

$$\frac{\|v_1 - u^*\|_2}{\|v_2 - u^*\|_2} = \sqrt{\frac{\lambda_2}{\lambda_1}} = \sqrt{\kappa_2(A)}, \tag{5.36}$$

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 5.4(b) has $\kappa_2 = 50$, not particularly ill-conditioned compared to the matrices that often arise in solving differential equations.

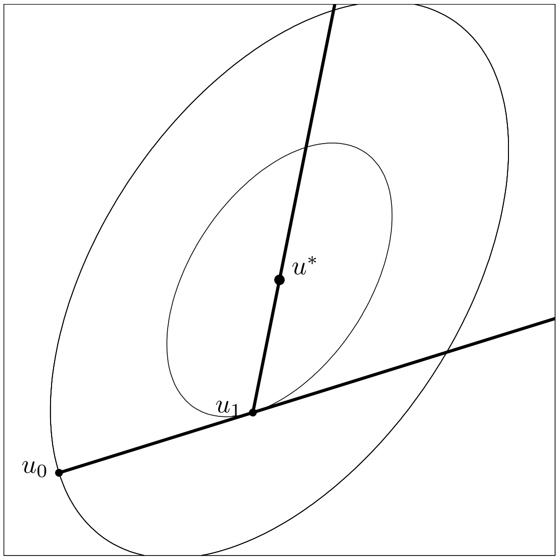


Figure 5.6: The conjugate gradient algoritm converges in 2 iterations from any initial guess u_0 in the case $m = 2$. The two search directions used are A-conjugate.

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 principle axes and the spread in the size of the eigenvalues determines how stretched the ellipse is in each direction.

5.3.2 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

$$u_k = u_{k-1} + \alpha_{k-1} p_{k-1} \tag{5.37}$$

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

$$\alpha_{k-1} = \frac{p_{k-1}^T r_{k-1}}{p_{k-1}^T A p_{k-1}}. \tag{5.38}$$

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} and $\phi(u)$ could only increase along 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 5.4(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 towards the solution u^* as shown in Figure 5.6. Then minimizing ϕ along this line would give $u_k = u^*$ and 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 5.6. Once we have obtained u_1 by the formulas (5.37) and (5.38), we choose the next search direction p_1 to be a vector satisfying

$$p_1^T A p_0 = 0. \tag{5.39}$$

Below we will show that this is the optimal search direction, leading directly to $u_2 = u^*$. When $m > 2$ we cannot generally converge in 2 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 (5.39) 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 (5.23). Consider an arbitrary point on an ellipse. The direction tangent to the ellipse at this point and the direction that points towards the center of the ellipse are always A-conjugate. This is the fact that allows us to determine the direction towards 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 towards the center is simply the radial direction, which is orthogonal to the tangent direction.

To prove that the two directions shown in Figure 5.6 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

$$p_0^T A(u^* - u_1) = 0. \tag{5.40}$$

On the other hand $u^* - u_1 = \alpha p_1$ for some scalar $\alpha \neq 0$ and using this in (5.40) gives (5.39).

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 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 (5.38) 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 (5.38) 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 5.6.

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 occuring 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 (5.38)). 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 towards 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 minimum u^*.

For $m = 3$ this procedure always converges in *at most* three iterations (in exact arithmetic at least; see Section 5.3.4). 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 towards u^* and so $u_1 = u^*$. This is rather unlikely, though.

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 towards 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 look circular when cut by certain planes but 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 $A \in \mathbb{R}^{m\times m}\quad (n \leq m)$.

5.3.3 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 [HS52] but it took some time for this and related methods to be fully understood and widely used. See Golub and O'Leary [GO89] 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.

- On the other hand, in practice CG frequently "converges" to a sufficiently accurate approximation to u^* in *far less* than n iterations. For example, in Section **??** we 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 only 144 iterations, however.

The fact that effective convergence is often 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 5.3.5, but first we present the basic CG algorithm and explore its convergence properties more fully.

The CG algorithm takes the form:

```
$u_0 = 0$
$r_0 = f$
$p_0 = r_0$
for $k = 1,\ 2,\ \dots$
    $w_{k-1} = Ap_{k-1}$
    $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(p_{k-1}^T w_{k-1})$
    $u_k = u_{k-1} + \alpha_{k-1} p_{k-1}$
    $r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
    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 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

$$p_{k-1}^T r_{k-1} = r_{k-1}^T r_{k-1}$$

in order to rewrite the expression (5.38).

Compare this algorithm to the steepest descent algorithm presented on page 70. 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. One slight change is that we have taken u_0 to be the zero vector as the initial guess, which is assumed in the theorem below, and results in $r_0 = f - Au_0 = f$. In practice a better initial guess can be used if available.

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 $j = 0,\ 1,\ ,\dots,\ k-1$. This is part of the following theorem, which also summarizes some other nice properties that this algorithm has. These properties are interrelated and used in the proof, which can be found in Trefethen and Bau [TB97], for example.

Theorem 5.3.1 *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 p_j for $j = 0,\ 1,\ ,\dots,\ k-1$.*
2. *The residual r_k is orthogonal to all previous residuals, $r_k^T r_j = 0$ for $j = 0,\ 1,\ ,\dots,\ k-1$.*
3. *The following four subspaces of $\mathbb{R}^m$ are identical:*

$$\begin{aligned}&span(p_0, p_1, \dots, p_{k-1}), \quad span(r_0, r_1, \dots, r_{k-1}),\\ &span(u_1, u_2, \dots, u_k), \quad span(f, Af, A^2 f, \dots, A^{k-1} f).\end{aligned}$$

The subspace $\mathcal{K}_k = \text{span}(f, Af, A^2f, \dots, A^{k-1}f)$ spanned by the vector f and the first $k-1$ powers of A applied to this vector is called a *Krylov space*. We have seen that the CG algorithm can be interpreted as minimizing the function $\phi(u)$ over the space $\text{span}(p_0, p_1, \dots, p_{k-1})$ in the kth iteration, and by the theorem above this is equivalent to minimizing $\phi(u)$ over the Krylov space $\mathcal{K}_k$. Many other iterative methods in linear algebra are also based on the idea of solving problems on an expanding sequence of Krylov spaces, for example the Arnoldi and Lanczos algorithms for finding eigenvalues of general and symmetric matrices, respectively; see [TB97].

Nonsymmetric linear systems $Au = f$ can also be solved by Krylov space methods such as GMRES (generalized minimum residual), which effectively minimizes the residual $\|f - Au\|_2$ over all choices of u in $\mathcal{K}_k$ in the kth iteration in order to define u_k. The GMRES algorithm is more complicated than CG because more past information must be kept and a $(k+1) \times k$ least squares problems (with Hessenberg structure) solved in each iteration. When A is symmetric GMRES can be simplified by using certain three-term recurrence relations to reduce the work, yielding the MINRES (minimum residuals) algorithm. If the matrix is also positive definite then this simplifies further and the CG algorithm presented above results. For an introduction to these more general algorithms, see for example [Gre97], [TB97].

In MATLAB the built-in functions `pcg` (for preconditioned conjugate gradients), `gmres`, and `minres` can be used to solve systems by these methods. The matrix A can be specified either as a matrix or as a function that computes the product Ap for any vector p. The second option is more commonly used for the large sparse matrices arising in differential equations, since the formula for computing the product is generally easier to work with than the matrix itself.

5.3.4 Convergence of CG

The convergence theory for CG is related to the fact that u_k minimizes $\phi(u)$ over the Krylov space $\mathcal{K}_k$ defined in the previous section. We now show that the A-norm of the error e_k is also minimized over all possible choices of vectors u in $\mathcal{K}_k$. The A-norm is defined by

$$\|e\|_A = \sqrt{e^T A e}. \tag{5.41}$$

This defines a norm that satisfies the requirements of a vector norm provided that A is SPD, which we are assuming in studying the CG method. This is a natural norm to use because

$$\begin{aligned} \|e\|_A^2 &= (u - u^*)^T A (u - u^*) \\ &= u^T A u - 2u^T A u^* + u^{*T} A u^* \\ &= 2\phi(u) + u^{*T} A u^*. \end{aligned} \tag{5.42}$$

Since $u^{*T} A u^*$ 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 5.3.1 lies in $\text{span}(f, Af, \dots, A^{k-1}f)$. Since $u_0 = 0$ we have $f = Au^* = -Ae_0$ and so $e_k - e_0$ also lies in $\text{span}(Ae_0, A^2e_0, \dots, A^ke_0)$. So $e_k = e_0 + c_1 A e_0 + c_2 A^2 e_0 + \cdots + c_k A^k e_0$ for some coefficients $c_1, \ \dots, \ c_k$. In other words,

$$e_k = P_k(A) e_0 \tag{5.43}$$

where

$$P_k(A) = I + c_1 A + c_2 A^2 + \cdots + c_k A^k \tag{5.44}$$

is a polynomial in A. For a scalar value x we have

$$P_k(x) = 1 + c_1x + c_2x^2 + \cdots + c_kx^k \tag{5.45}$$

and $P_k \in \mathcal{P}_k$ where

$$\mathcal{P}_k = \{\text{polynomials } P(x) \text{ of degree at most } k \text{ satisfying } P(0) = 1\}. \tag{5.46}$$

The polynomial P_k constructed implicitly by the CG algorithm solves the minimization problem

$$\min_{P\in\mathcal{P}_k} \|P(A)e_0\|_A. \tag{5.47}$$

In order to understand how a polynomial function of a matrix behaves, recall that

$$A = R\Lambda R^{-1} \quad\Longrightarrow\quad A^j = R\Lambda^j R^{-1}$$

and so

$$P_k(A) = RP_k(\Lambda)R^{-1},$$

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 n 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. The polynomial that CG automatically constructs is simply $P_n(x) = (1-x/\lambda_1)\cdots(1-x/\lambda_n)$.

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

$$\frac{\|P(A)e_0\|_A}{\|e_0\|_A} \le \max_{1\le j\le m} |P(\lambda_j)| \tag{5.48}$$

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, A has an orthonormal set of eigenvectors r_j, $j = 1,\ 2,\ \ldots,\ m$, and any vector e_0 can be written as

$$e_0 = \sum_{j=1}^m a_jr_j$$

for some coefficients $a_1,\ \ldots,\ a_m$. Since the r_j are orthonormal, we find that

$$\|e_0\|_A^2 = \sum_{j=1}^m a_j^2\lambda_j. \tag{5.49}$$

This may be easiest to see in matrix form: $e_0 = Ra$ where R is the matrix of eigenvectors and a the vector of coefficients. Since $R^T = R^{-1}$, we have

$$\|e_0\|_A^2 = e_0^TAe_0 = a^TR^TARe_0 = a^T\Lambda a,$$

which gives (5.49).

If $P(A)$ is any polynomial in A then

$$P(A)e_0 = \sum_{j=1}^m a_jP(\lambda_j)r_j$$

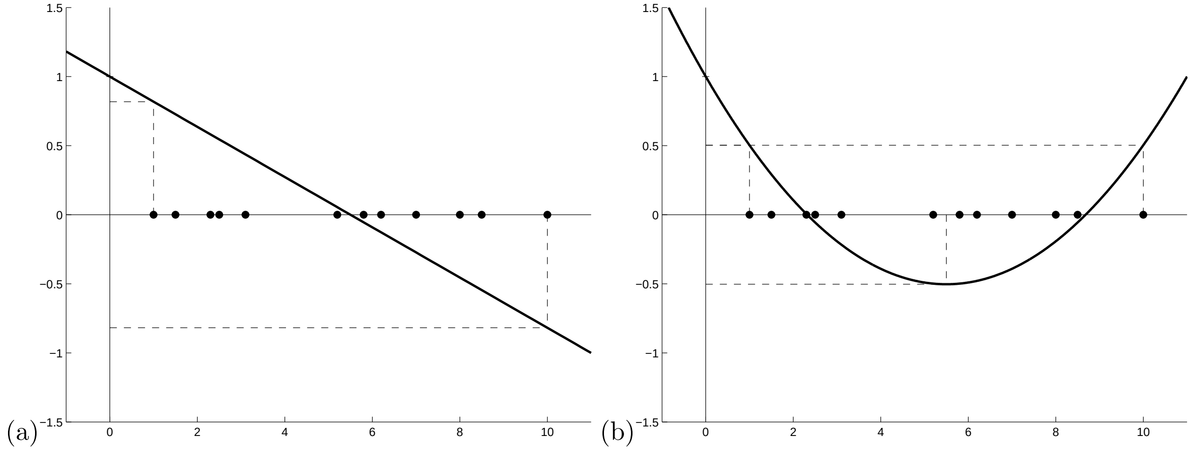


Figure 5.7: (a) The polynomial $\tilde{P}_1(x)$ based on a sample set of eigenvalues marked by dots on the x-axis. (b) The polynomial $\tilde{P}_2(x)$ for the same set of eigenvalues. See Figure 5.8(a) for the polynomial $\tilde{P}_5(x)$.

and so

$$\begin{aligned} \|P(A)e_0\|_A^2 &= \sum_{j=1}^m a_j^2 P(\lambda_j)^2 \lambda_j \\ &\le \left[\max_{1\le j\le m} (P(\lambda_j))^2\right] \sum_{j=1}^m a_j^2 \lambda_j. \end{aligned} \tag{5.50}$$

Combining this with (5.49) gives (5.48).

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 (5.48) and obtain a bound that decreases with increasing k. Since the polynomial P_k constructed by CG solves the problem (5.47), 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

$$\tilde{P}_1(x) = 1 - \frac{2x}{\lambda_m + \lambda_1}, \tag{5.51}$$

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 5.7(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 5.7(a). For this polynomial we have

$$\begin{aligned} \max_{1\le j\le m} |\tilde{P}_1(\lambda_j)| = \tilde{P}_1(\lambda_1) = 1 - \frac{2\lambda_1}{\lambda_m + \lambda_1} &= \frac{\lambda_m/\lambda_1 - 1}{\lambda_m/\lambda_1 + 1} \\ = \frac{\kappa - 1}{\kappa + 1} \end{aligned} \tag{5.52}$$

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 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 $e_0 = a_j r_j$ 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 U^*. 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 5.7(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 can also 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 is often quite realistic.

The polynomials we want are simply shifted and scaled versions of the Chebyshev polynomials discussed in Section 4.2.5. 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 $[\lambda_1, \lambda_m]$ and scale it so that the value at $x = 0$ is 1, and obtain

$$\tilde{P}_k(x) = \frac{T_k\left(\frac{\lambda_m + \lambda_1 - 2x}{\lambda_m - \lambda_1}\right)}{T_k\left(\frac{\lambda_m + \lambda_1}{\lambda_m - \lambda_1}\right)}. \tag{5.53}$$

For $k = 1$ this gives (5.51) since $T_1(x) = x$. We now only need to compute

$$\max_{1 \le j \le m} |\tilde{P}_k(\lambda_j)| = \tilde{P}_k(\lambda_1)$$

in order to obtain the desired bound on $\|e_k\|_A$. We have

$$\tilde{P}_k(\lambda_1) = \frac{T_k(1)}{T_k\left(\frac{\lambda_m + \lambda_1}{\lambda_m - \lambda_1}\right)} = \frac{1}{T_k\left(\frac{\lambda_m + \lambda_1}{\lambda_m - \lambda_1}\right)}. \tag{5.54}$$

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 $[-1, 1]$. Recall the formula (4.17) for the Chebyshev polynomial that is valid for x in the interval $[-1, 1]$. Outside this interval there is an analogous formula in terms of the hyperbolic cosine,

$$T_k(x) = \cosh(k \cosh^{-1} x).$$

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 (5.54) we need to evaluate T_k at $x = (\kappa + 1)/(\kappa - 1)$, where we obtain

$$\begin{aligned} y &= \frac{\kappa+1}{\kappa-1} \pm \sqrt{\left(\frac{\kappa+1}{\kappa-1}\right)^2 - 1} \\ &= \frac{\kappa + 1 \pm \sqrt{4\kappa}}{\kappa - 1} \\ &= \frac{(\sqrt{\kappa} \pm 1)^2}{(\sqrt{\kappa}+1)(\sqrt{\kappa}-1)} \\ &= \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \quad \text{or} \quad \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}. \end{aligned} \tag{5.55}$$

Either choice of y gives the same value for

$$T_k\left(\frac{\kappa+1}{\kappa-1}\right) = \frac{1}{2}\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right]. \tag{5.56}$$

Using this in (5.54) and combining with (5.48) gives

$$\frac{\|P(A)e_0\|_A}{\|e_0\|_A} \le 2\left[\left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k + \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k\right]^{-1} \le 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k. \tag{5.57}$$

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.

Figure 5.8 shows some examples for the case $k = 5$. In Figure 5.8(a) the same eigenvalue distribution as in Figure 5.7 is assumed, and the shifted Chebyshev polynomial $\tilde{P}_5(x)$ is plotted. This gives an upper bound $\|e_5\|_A/\|e_0\|_A \le 0.0756$ for a matrix A with these eigenvalues, which has condition number $\kappa = 10$.

Figure 5.8(b) shows a different eigenvalue distribution, for a matrix A that is better conditioned, with $\lambda_1 = 2$ and $\lambda_m = 10$ so $\kappa = 5$. In this case $\|e_5\|_A/\|e_0\|_A \le 0.0163$.

Figure 5.8(c) shows the situation for a matrix that is more poorly conditioned, with $\lambda_1 = 0.2$ and $\lambda_m = 10$ so $\kappa = 50$. Using the Chebyshev polynomial $\tilde{P}_5(x)$ shown in this figure gives an upper bound of $\|e_5\|_A/\|e_0\|_A \le 0.4553$. For a matrix A with this condition number but with many eigenvalues scattered more or less uniformly throughout the interval $[0.2, 10]$, this would be a realistic estimate of the reduction in error after 5 steps of CG. For the eigenvalue distribution shown in the figure, however, CG is in fact able to do much better and constructs a polynomial $P_5(x)$ that might look more like the

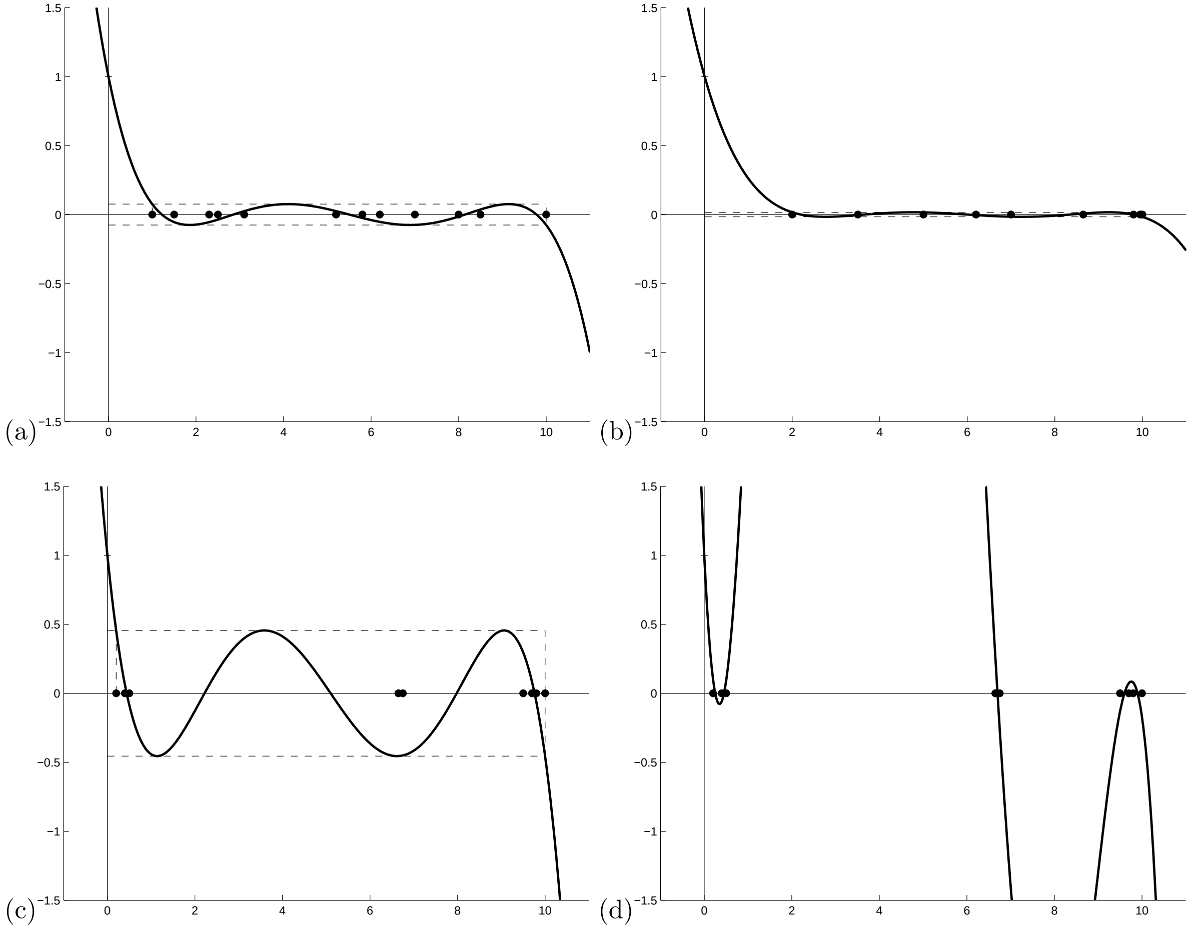


Figure 5.8: (a) The polynomial $\tilde{P}_5(x)$ based on a sample set of eigenvalues marked by dots on the x-axis, the same set as in Figure 5.7. (b) The polynomial $\tilde{P}_5(x)$ for a matrix with smaller κ. (c) The polynomial $\tilde{P}_5(x)$ for a matrix with larger κ. (d) A better polynomial $P(x)$ of degree 5 for the same eigenvalue distribution as in figure (c).

one shown in Figure 5.8(d), which is small near each of the three clusters of eigenvalues but huge in between.

The bound (5.57) is only an upper bound and may be pessimistic when the eigenvalues are clustered, which sometimes happens in practice. 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 (5.57) 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

$$2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx 2\left(1-\frac{2}{\sqrt{\kappa}}\right)^k \approx 2e^{-2k/\sqrt{\kappa}}, \tag{5.58}$$

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 (5.58) 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 to just 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, CG may still work very well while SOR only works well if the optimal ω is found, which may be impossible, and FFT methods are inapplicable. Similar comments apply in three dimensions.

5.3.5 Preconditioners

We saw in the last section that the convergence rate of CG often 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.

If M is any nonsingular matrix then

$$Au = f \quad \Longleftrightarrow \quad M^{-1}Au = M^{-1}f. \tag{5.59}$$

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 $M^{-1}f = A^{-1}f$. 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 (*i.e.*, applying M^{-1} is cheap).

A problem with this approach to preconditioning 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 (5.59). Instead we can consider solving a different system, again equivalent to the original:

$$(C^{-1}AC^{-1})(Cu) = C^{-1}f, \tag{5.60}$$

where C is a nonsingular symmetric matrix. Write this system as

$$\tilde{A}\tilde{u} = \tilde{f}. \tag{5.61}$$

If C is symmetric then so is C^{-1} and hence so is $\tilde{A}$. Moreover $\tilde{A}$ is positive definite (provided A is) since

$$u^T\tilde{A}u = u^TC^{-1}AC^{-1}u = (C^{-1}u)^TA(C^{-1}u) > 0$$

for any vector u.

How should we choose C? Since A is multiplied twice by C^{-1} we want C^2 to be some approximation to A. This seems potentially harder to accomplish than choosing M to approximate A, and also the system (5.60) seems more cumbersome to work with than the system on the right in (5.59). Luckily it turns out that if we have a reasonable preconditioner M that is SPD then a minor variation of the CG algorithm can be used that in essence solves (5.60) for a matrix C with $C^2 = M$, but that only requires solving systems using M and never needs C itself.

To see this, suppose we apply CG to (5.61) and generate vectors $\tilde{u}_k$, $\tilde{p}_k$, $\tilde{w}_k$, and $\tilde{r}_k$. Now define

$$u_k = C^{-1}\tilde{u}_k, \quad p_k = C^{-1}\tilde{p}_k, \quad w_k = C^{-1}\tilde{w}_k,$$

and

$$r_k = C\tilde{r}_k.$$

Note that $\tilde{r}_k$ is multiplied by C, not C^{-1}. Here $\tilde{r}_k$ is the residual when $\tilde{u}_k$ is used in the system (5.61). Note that if $\tilde{u}_k$ approximates the solution to (5.60) 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:

```
$u_0 = 0$
$r_0 = f$
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 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 a couple of places in the last two lines.

A very simple preconditioner that is effective for some problems is simply to use $M = \text{diag}(A)$, a diagonal approximation. This doesn't help at all 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. More sophisticated preconditioners are discussed in many places, for example there is a list of possible approaches in Trefethen and Bau [TB97].

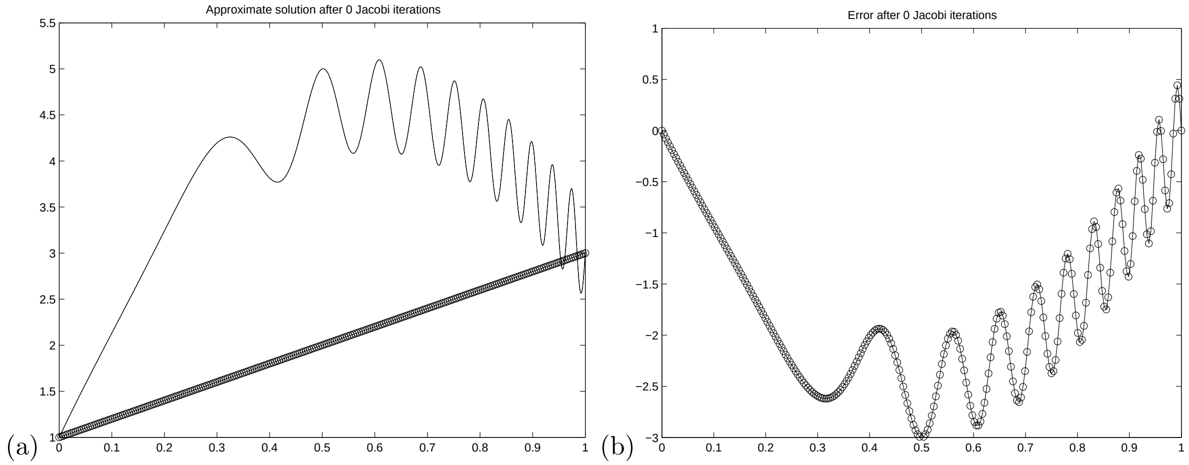


Figure 5.9: (a) The solution $u(x)$ (solid line) and initial guess u_0 (circles). (b) The error e_0 in the initial guess.

5.4 Multigrid methods

The main idea of the multigrid method will be briefly presented in the context of the one-dimensional model problem $u''(x) = f(x)$. For more discussion, see for example [BEM01], [Jes84].

Let

$$f(x) = -20 + a\phi''(x)\cos(\phi(x)) - a(\phi'(x))^2\sin(\phi(x)) \tag{5.62}$$

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

$$u(x) = 1 + 12x - 10x^2 + a\sin(\phi(x)), \tag{5.63}$$

which is plotted in Figure 5.9(a). This function has been chosen because it clearly contains variations on many different spatial scales, *i.e.*, large components of many different frequencies.

Discretize this problem with the standard tridiagonal systems (2.10) and apply the Jacobi iterative method of Section 5.1 with the linear initial guess u_0 with components $1 + 2x_i$, which is also shown in Figure 5.9(a). Figure 5.9(b) shows the error e_0 in this initial guess on a grid with $m = 255$ grid points.

The left column of Figure 5.10 shows the approximations obtained after $k = 20,\ 100$, and 1000 iterations of Jacobi. This method converges very slowly and it would take tens of thousands of iterations to obtain a useful approximation to the solution. However, notice something very interesting in Figure 5.10. 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 counter-intuitive since we might expect the small-sclae features to be harder to capture. 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 quadratic component of the error remains. 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 5.1 that the error $e_k = u_k = u^*$ satisifies

$$e_k = Ge_{k-1},$$

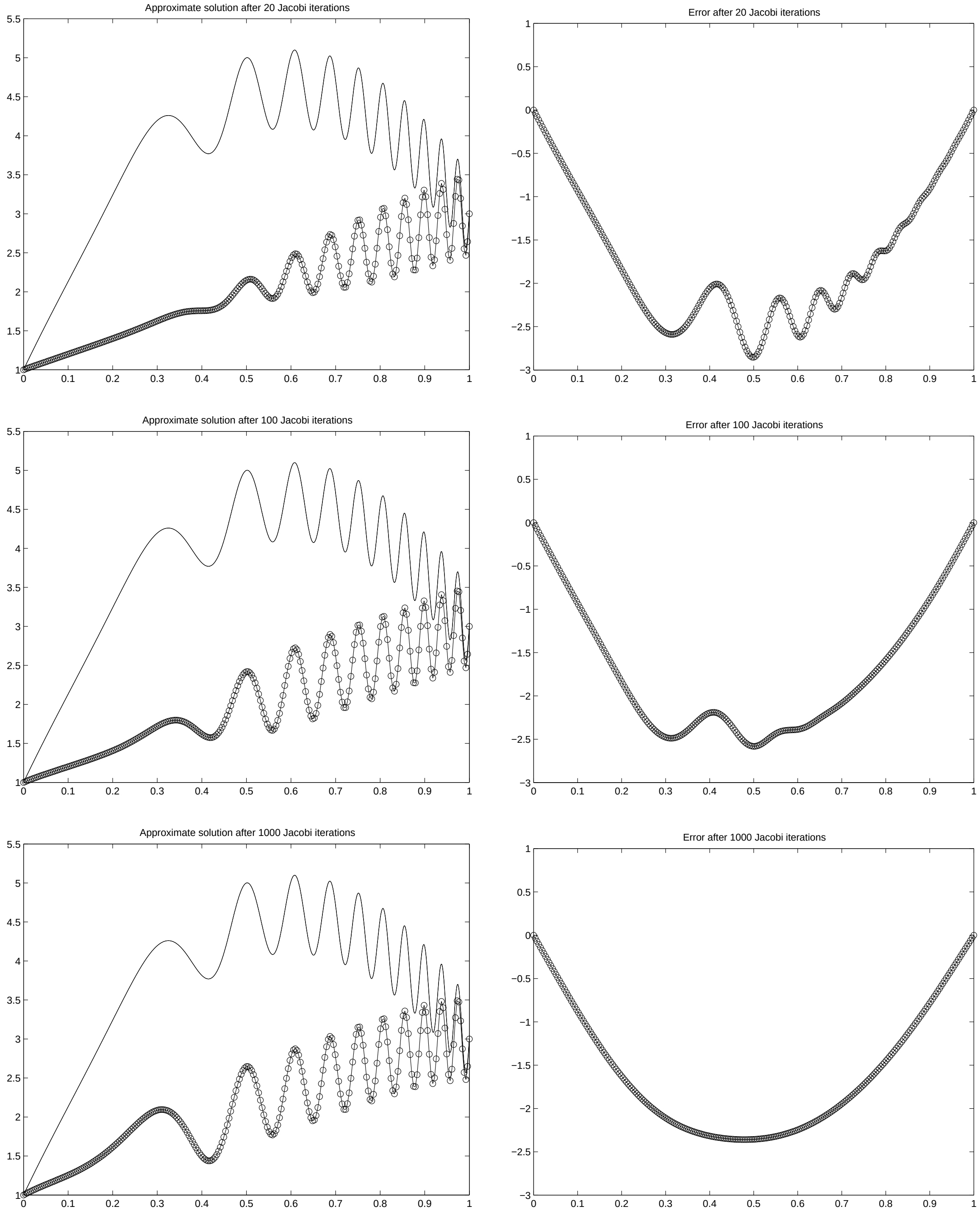


Figure 5.10: On the left: The solution $u(x)$ (solid line) and Jacobi iterate u_k. On the right: The error e_k. Shown for $k = 20$ (top), $k = 100$ (middle), and $k = 1000$ (bottom).

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 5.1 that the eigenvectors of G are the same as the eigenvectors of A. The eigenvector u^p has components

$$u_j^p = \sin(\pi p x_j), \qquad (x_j = jh, \quad j = 1,\ 2,\ \ldots,\ m), \tag{5.64}$$

while the corresponding eigenvalue is

$$\gamma_p = \cos(p\pi h), \tag{5.65}$$

for $p = 1,\ 2,\ \ldots,\ m$. If we decompose the initial error e_0 into eigencomponents,

$$e_0 = c_1 u^1 + c_2 u^2 + \cdots + c_m u^m, \tag{5.66}$$

then we have

$$e_k = c_1 \gamma_1^k u^1 + c_2 \gamma_2^k u^2 + \cdots + c_m \gamma_m^k u^m. \tag{5.67}$$

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 5.1.

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. Recall from Chapter 4 that decomposing the error e_0 as in (5.66) gives a Fourier sine series representation of the error, since u^p in (5.64) 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 5.10 has negligible component of these highest frequencies, however, and we are observing the rapid decay of the intermediate frequencies in this figure.

We are finally ready to introduce the multigrid algorithm. In Figure 5.10 we see that after 20 iterations the higher frequency components of the error have already decayed significantly but that convergence then slows 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 in order 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 importantly, 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 $u^{m/8}$, which has a convergence factor $\gamma_{m/8} \approx \cos(\pi/8) \approx 0.92$. If we take this grid function $\sin(\pi(m/8)x)$ and represent it on a coarser grid with $m_c = (m-1)/2$ points, then it becomes approximately $\sin(\pi(m_c/4)x)$, an eigenvector of the $m_c \times m_c$ version of the G matrix that has eigenvalue $\gamma_{m_c/4} = \cos((m_c/4)\pi h_c) \approx \cos(\pi/4) \approx 0.7$, since $h_c = 1/(m_c+1)$. In 20 iterations on the original grid this component of the error decayed by only a factor of $(0.92)^{20} \approx 0.2$, *i.e.*, a factor of 5 improvement. On the other hand 20 iterations on the coarsened grid would reduce this same component of the error by a factor of about 1000 (since $(0.7)^{20} \approx 10^{-3}$). 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 we 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

$$Ae_\nu = -r_\nu. \tag{5.68}$$

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 (5.68) 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.*, $\nu = 20$ or less) of a simple iterative method (*e.g.*, 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. Subract 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.*, $\nu = 20$ or less) of a simple iterative method (*e.g.*, 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 above, 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 in order to speed up the solution to this problem. In other words the entire algorithm given above is applied within Step 4 in order 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 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.

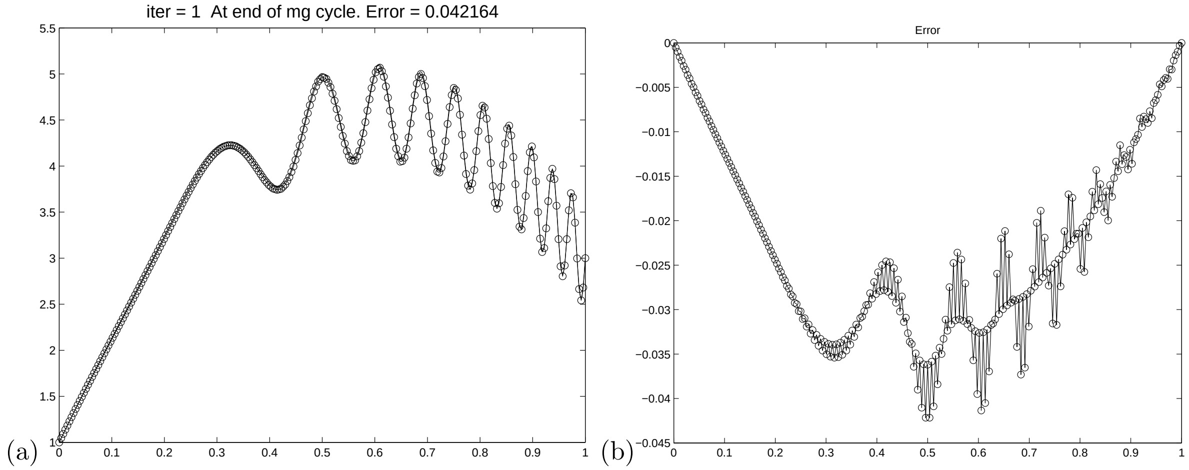


Figure 5.11: (a) The solution $u(x)$ (solid line) and approximate solution (circles) obtained after one V-cycle of the multigrid algorithm with $\nu = 10$. (b) The error in this approximation. Note the change in scale from Figure 5.10.

Figure 5.11 shows the results obtained when the above algorithm is used starting with $m = 2^8 - 1 = 255$, using $\nu = 10$, and recursing down to a grid with 3 grid points, *i.e.*, 7 levels of grids. On each level we apply 10 iterations of Jacobi, do a coarse grid correction, and then apply 10 more iterations of Jacobi. Hence a total of 20 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 140 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

$$20 \sum_{j=2}^{8} 2^j \approx 20 \cdot 2^9 \approx 10,000,$$

roughly the same amount of work as 40 iterations on the original grid would require. But the improvement in accuracy is dramatic — compare Figure 5.11 to the results in Figure 5.10 obtained by simply iterating on the original grid.

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

$$2\nu \sum_{j=2}^{J} 2^j \approx 4\nu 2^J \approx 4\nu m = O(m). \tag{5.69}$$

Note that this is *linear* in the number of grid points m, even though as m increases we are using an increasing number of coarser grids. The number of grids grows like $\log_2(m)$ 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 5.12(a) and (b). Taking a single V-cycle often results in a significant reduction in the error, as illustrated in Figure 5.11, 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-cyles would be needed to reach a given level of error, so that the total work would in fact grow like $O(m \log m)$.

We might also consider taking more that one iteration of the cycle on each of the coarser grids in order to solve the coarse grid problems within each cycle on the finest grid. Suppose, for example,

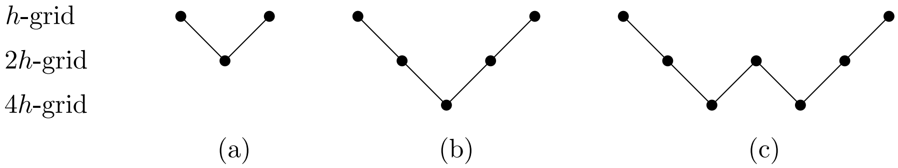


Figure 5.12: (a) One V-cycle with 2 levels. (b) One V-cycle with 3 levels. (c) One W-cycle with 3 levels.

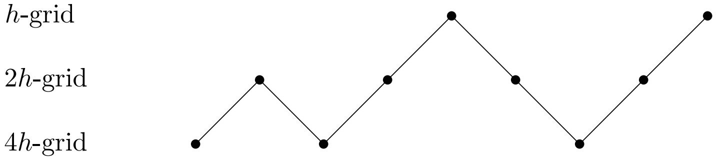


Figure 5.13: Full multigrid (FMG) with one V-cycle on 3 levels.

that we take 2 cycles at each stage on each of the finer grids. This gives the W-cycle illustrated in Figure 5.12(c).

Even better results are typically obtained by using the "full multigrid" or 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 2-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 5.13.

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 this case the total work is $O(m)$, which is optimal.

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 full multigrid 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. See the literature for more details.

Chapter 6

The Initial Value Problem for ODE's

In this chapter we begin a study of time-dependent differential equations, beginning with the initial value problem (IVP) for a time-dependent ODE. Standard introductory texts are Lambert[Lam73] and Gear[Gea71]. Henrici[Hen62] gives a more complete description of some theoretical issues, although stiff equations are not discussed. Hairer, Norsett, and Wanner[HNW87, HNW93] is a more recent and complete survey of the field.

The initial value problem takes the form

$$u'(t) = f(u(t), t) \quad \text{for } t > t_0 \tag{6.1}$$

with some initial data

$$u(t_0) = \eta. \tag{6.2}$$

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

In general (6.1) 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. The problem is *linear* if

$$f(u,t) = A(t)u + b(t) \tag{6.3}$$

where $A(t) \in \mathbb{R}^{s\times s}$ and $b(t) \in \mathbb{R}^s$.

We will consider only the first order equation (6.1) 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 6.1. Consider the initial value problem for the ODE

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

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

$$\begin{aligned} u(0) &= \eta_1 \\ u'(0) &= \eta_2 \\ u''(0) &= \eta_3 \end{aligned} \tag{6.4}$$

We can rewrite this as a system of the form (6.1) and (6.2) by introducing the variables

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

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 (6.2) where the three components of η come from (6.4). More generally, any single equation of order m can be reduced to m first order equations by defining $u_j(t) = u^{(j-1)}(t)$, and an mth order system of s equations can be reduced to a system of ms first order equations.

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

$$u'(t) = f(u(t)) \tag{6.5}$$

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 (6.5) 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.

We will always assume that f is Lipschitz continuous in u as described in the next section, which implies that the initial value problem has a unique solution over some time interval.

6.1 Lipschitz continuity

The standard theory for the existence of a solution to the initial value problem

$$u'(t) = f(u,t), \quad u(0) = \eta \tag{6.6}$$

is discussed in many texts, *e.g.*, [CL55]. 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 (6.6). We say that the function $f(u,t)$ is *Lipschitz continuous* in u over some range of t and u, if there exist some constant $L > 0$ so that

$$|f(u,t) - f(u^*,t)| \le L|u - u^*| \tag{6.7}$$

for all u and u^* in this range. 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 $u \to u^*$. The function is *uniformly Lipschitz continuous* over some time period if there is a single constant L that works for all u and u^* and all t in this period.

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

$$L = \max_u |f_u(u,t)|,$$

since

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

for some value v between u and u^*.

Example 6.2. 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

$$u(t) = e^{\lambda(t-t_0)}\eta + \int_{t_0}^{t} e^{\lambda(t-\tau)} g(\tau)\, d\tau. \tag{6.8}$$

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 $g(t)$,

$$u(t) = \eta + \int_{t_0}^{t} g(\tau)\, d\tau. \tag{6.9}$$

6.1.1 Existence and uniqueness of solutions

The basic existence and uniqueness theorem states that if f is uniformly Lipschitz continuous over some time period $0 \le t \le T$ then there is a unique solution to the initial value problem (6.6) from any initial value η. If f is Lipschitz but not uniformly so, then there will be a unique solution through any value η over some finite time interval, but this solution may not exist past some time, as the next example shows.

Example 6.3. Consider the initial value problem

$$u'(t) = (u(t))^2$$

with initial conditions

$$u(0) = \eta > 0.$$

The function $f(u) = u^2$ is Lipschitz continuous over any finite interval $[\eta - a,\ \eta + a]$ with $L = 2(\eta + a)$. From this it can be shown that the initial value problem has a unique solution over *some* time interval $0 \le t < \bar{T}$ with $\bar{T} > 0$. However, since f is not uniformly Lipschitz for all u (*i.e.*, there is not a single value of L that works for all u), we cannot prove that a solution exists for all time, and in fact it does not. The solution to the initial value problem is

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

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

If the function f is not Lipschitz continuous at some point then the initial value problem may fail to have a unique solution over any time interval.

Example 6.4. Consider the initial value problem

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

with initial conditions

$$u(0) = 0.$$

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 (6.7) 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

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

6.1.2 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 some norm $\|\cdot\|$ if there is a constant L such that

$$\|f(u,t) - f(u^*,t)\| \leq L\|u - u^*\| \tag{6.10}$$

for all u in a neighborhood of u^*. By the equivalence of finite-dimensional norms (Appendix A1), if f is Lipschitz continuous in one norm then it is Lipschiz continuous in any other norm, though the Lipschitz constant may depend on the norm chosen.

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

Example 6.5. Consider the pendulum problem from Section 2.15,

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

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

$$u = \left[\begin{array}{c}\theta\\ v\end{array}\right], \qquad \frac{d}{dt}\left[\begin{array}{c}\theta\\ v\end{array}\right] = \left[\begin{array}{c}v\\ -\sin(\theta)\end{array}\right].$$

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 $|v - v^*| \leq \|u - u^*\|_\infty$. We also have

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

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

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

6.1.3 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 6.6. Consider the trivial equation $u'(t) = g(t)$, which has Lipschitz constant $L = 0$ and solutions given by (6.8). Several solution curves are sketched in Figure 6.1. Note that all of 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 6.7. 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 6.2 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 towards one another (in the case $\lambda < 0$) or diverging away from one another (in the case $\lambda > 0$). If the magnitude of λ is 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 to diverge rapidly.

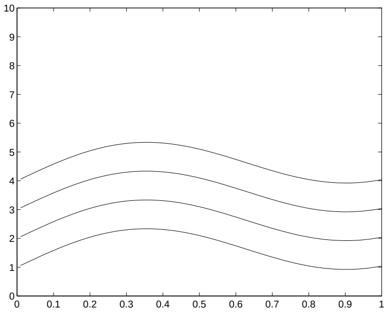


Figure 6.1: Solution curves for Example 6.6, where $L = 0$.

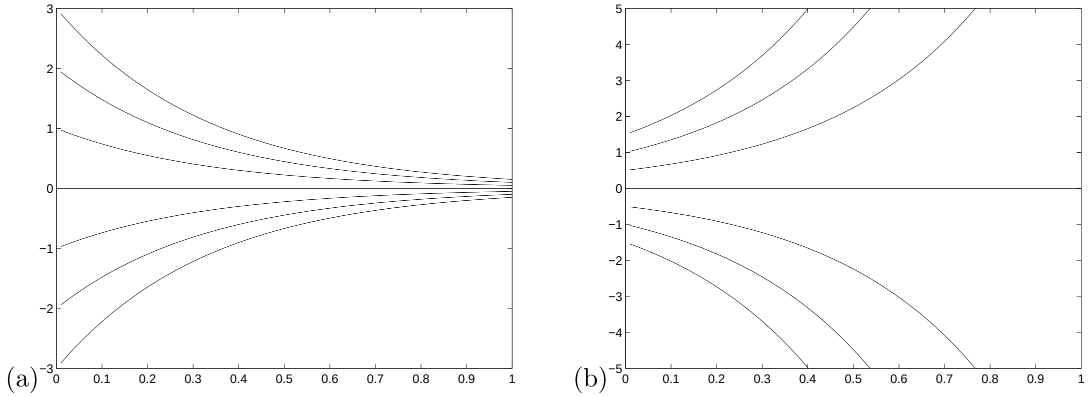


Figure 6.2: Solution curves for Example 6.7 with (a) $\lambda = -3$ and (b) $\lambda = 3$.

6.1.4 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 6.2(a) and Figure 6.2(b) the Lipschitz constant has the same value $L = |\lambda| = 3$. But we would expect that rapidly convergent solution curves as in Figure 6.2(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 9 on "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 6.8. Consider two solutions to the pendulum problem from Example 6.5, 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 differs by 2ϵ, which can be arbitarily small, and yet the solutions soon diverge dramatically, as Solution 1 falls towards $\theta = 0$ while in Solution 2 the penudulum falls the other way, towards $\theta = 2\pi$.

6.2 Some basic numerical methods

We begin by listing a few standard approaches to discretizing the equation (6.1). 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 that 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

$$U^0 = \eta \tag{6.11}$$

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 $u'(t_n)$ by $D_+U^n = (U^{n+1} - U^n)/k$ from (1.1). This gives the method

$$\frac{U^{n+1} - U^n}{k} = f(U^n), \qquad n = 0,\ 1,\ \ldots \tag{6.12}$$

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

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

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 $u'(t_{n+1})$ by D_-U^{n+1}:

$$\frac{U^{n+1} - U^n}{k} = f(U^{n+1}) \tag{6.14}$$

or

$$U^{n+1} = U^n + kf(U^{n+1}). \tag{6.15}$$

Again we can march forward in time since computing U^{n+1} only requires that we know the previous value U^n. In the Backward Euler method, however, (6.15) 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 (6.13) is an *explicit* method.

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

$$\frac{U^{n+1} - U^n}{k} = \frac{1}{2}(f(U^n) + f(U^{n+1})). \tag{6.16}$$

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 *multi-step* 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*),

$$\frac{U^{n+1} - U^{n-1}}{2k} = f(U^n) \tag{6.17}$$

or

$$U^{n+1} = U^{n-1} + 2kf(U^n) \tag{6.18}$$

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

$$\frac{3U^{n+1} - 4U^n + U^{n-1}}{2k} = f(U^{n+1}). \tag{6.19}$$

This is one of the *BDF methods* that will be discussed further in Chapter 9.

6.3 Truncation errors

The truncation error for these methods is defined in the same way as in Chapter 2. We write the difference euqation in the form that directly models the derivatives (*e.g.*, in the form (6.17) rather than (6.18)) 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 6.9. The local truncation error of the midpoint method (6.17) 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.

6.4 One-step errors

In much of the literature concerning numerical methods for ordinary differential equations, a slightly different definition of the local truncation error is used that is based on the form (6.18), for example, rather than (6.17). Denoting this value by $\mathcal{L}^n$, we have

$$\begin{aligned} \mathcal{L}^n &= u(t_{n+1}) - u(t_{n-1}) - 2kf(u(t_n)) \\ &= \frac{1}{3}k^3u'''(t_n) + O(k^5). \end{aligned} \tag{6.20}$$

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 a local truncation error that is $O(k^{p+1})$. With the notation we are using, a pth order accurate method has a 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.

I prefer to call $\mathcal{L}^n$ the *one-step error*, since this can be viewed as the error that would be introduced in one time step if the past values $U^n,\ U^{n-1},\ \ldots$ were all taken to be the exact values from $u(t)$. For example, in the midpoint method (6.18) 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 (6.20) 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 $T > 0$. In order 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

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

in which f is independent of u and the solution is simply the integral of f, 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 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 essentially correct, by insuring 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.

6.5 Taylor series methods

The Forward Euler method (6.13) can be derived using a Taylor series expansion of $u(t_{n+1})$ about $u(t_n)$:

$$u(t_{n+1}) = u(t_n) + ku'(t_n) + \frac{1}{2}k^2u''(t_n) + \cdots. \tag{6.21}$$

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

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

This suggests the method (6.13). 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 p'th-order accurate method. The problem is that we are only given

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

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

$$\begin{aligned} u''(t) &= f_u(u(t), t)u'(t) + f_t(u(t), t) \\ &= f_u(u(t), t)f(u(t), t) + f_t(u(t), t) \end{aligned} \tag{6.22}$$

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 6.10. Suppose we want to solve the equation

$$u'(t) = t^2 \sin(u(t)). \tag{6.23}$$

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

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

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.

6.6 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 6.8 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 *multi-stage* method, where intermediate values of the solution and its derivative are generated and used within a single time step.

Example 6.11. A 2-stage explicit Runge-Kutta method is given by

$$\begin{aligned} U^* &= U^n + \frac{1}{2}kf(U^n) \\ U^{n+1} &= U^n + kf(U^*). \end{aligned} \tag{6.24}$$

In the first stage an intermediate value is generated which 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 local truncation error.

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

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

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

$$\tau^n = \frac{1}{k}(u(t_{n+1}) - u(t_n)) - f\left(u(t_n) + \frac{1}{2}kf(u(t_n))\right). \tag{6.25}$$

Note that

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

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 (6.25) 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. An easier way to determine the order of accuracy is to apply the method 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 local truncation error is $O(k^2)$. Of course we have only checked that the local truncation error is $O(k^2)$ on one particular function $u(t) = e^{\lambda t}$, not on all smooth solutions, but it can be shown that this is a reliable indication of the order more generally. Applying a method to this special equation is also a fundamental tool in stability analysis — see Chapter 8.

Example 6.12. The Runge-Kutta method (6.24) can also be applied to nonautonomous equations of the form $u'(t) = f(u(t), t)$:

$$\begin{aligned} U^* &= U^n + \frac{1}{2}kf(U^n, t_n) \\ U^{n+1} &= U^n + kf(U^*, t_n + k/2). \end{aligned} \tag{6.26}$$

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

Example 6.13. One simple higher-order Runge-Kutta method is the fourth-order 4-stage method given by

$$
\begin{aligned}
F_0 &= f(U^n,\ t_n)\\
F_1 &= f\left(U^n + \frac{1}{2}kF_0,\ t_n + \frac{1}{2}k\right)\\
F_2 &= f\left(U^n + \frac{1}{2}kF_1,\ t_n + \frac{1}{2}k\right)\\
F_3 &= f\left(U^n + kF_2,\ t_{n+1}\right)\\
U^{n+1} &= U^n + \frac{k}{6}(F_0 + 2F_1 + 2F_2 + F_3).
\end{aligned}
\tag{6.27}
$$

This method was particularly popular in the pre-computer 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 explicit r-stage Runge-Kutta method has the form

$$
\begin{aligned}
F_0 &= f(U^n,\ t_n)\\
F_1 &= f\left(U^n + kb_{10}F_0,\ t_n + kb_{10}\right)\\
F_2 &= f\left(U^n + k(b_{20}F_0 + b_{21}F_1),\ t_n + k(b_{20} + b_{21})\right)\\
F_3 &= f\left(U^n + k(b_{30}F_0 + b_{31}F_1 + b_{32}F_2),\ t_n + k(b_{30} + b_{31} + b_{32})\right)\\
&\ \vdots\\
F_r &= f\left(U^n + k\sum_{s=0}^{r-1} b_{rs}F_s,\ t_n + k\sum_{s=0}^{r-1} b_{rs}\right)\\
U^{n+1} &= U^n + k(c_0F_0 + c_1F_1 + \cdots + c_rF_r).
\end{aligned}
\tag{6.28}
$$

Consistency requires $\sum_{s=0}^{r} c_s = 1$, and there are typically many ways that the coefficients b_{js} and c_s can be chosen to achieve a given accuracy.

Implicit Runge-Kutta methods can also be defined, in which each intermediate F_j depends on all the intermediate values $F_0,\ F_1,\ \ldots,\ F_r$ rather than only on $F_0,\ \ldots,\ F_{j-1}$. These are typically expensive to implement, however, and not so often used in practice.

One subclass of implicit methods that are simpler to implement are the *diagonally implicit* Runge-Kutta methods (DIRK methods) in which F_j depends on $F_0,\ F_1,\ \ldots,\ F_j$ but not on $F_{j+1},\ \ldots,\ F_r$. For a system of m equations, DIRK methods require solving a sequence of r implicit systems, each of size m, rather than a coupled set of mr equations as would be required in a fully implicit Runge-Kutta method.

Example 6.14. A second-order DIRK method is given by

$$
\begin{aligned}
F_0 &= f(U^n,\ t_n)\\
F_1 &= f\left(U^n + \frac{k}{4}(F_0 + F_1),\ t_n + \frac{k}{2}\right)\\
F_2 &= f\left(U^n + \frac{k}{3}(F_0 + F_1 + F_2),\ t_n + k\right)\\
U^{n+1} &= U^n + \frac{k}{3}(F_0 + F_1 + F_2).
\end{aligned}
\tag{6.29}
$$

This method is known as the TR-BDF2 method and is derived in a different form in Section 9.5.

6.7 1-step vs. multistep methods

Taylor series and Runge-Kutta methods are *1-step methods*; the approximation U^{n+1} depends on U^n but not on previous values $U^{n-1},\ U^{n-2},\ \ldots$. In the next section we will consider a class of multistep methods where previous values are also used (one example is the midpoint method (6.18)).

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 6.8.3.
- 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.
- 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 updating the solution nearby, 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.

An alternative is to use a multistep method in which values of f already computed in previous time steps are re-used 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.

6.8 Linear Multistep Methods

All of the methods introduced in Section 6.2 are members of a class of methods called Linear Multistep Methods (LMMs). In general, an r-step LMM has the form

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = k \sum_{j=0}^{r} \beta_j f(U^{n+j}, t_{n+j}). \tag{6.30}$$

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 re-used if f is expensive to evaluate).

If $\beta_r = 0$ then the method (6.30) is explicit, otherwise it is implicit. Note that we can multiply both sides by any constant and have essentially the same method, though 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 $f(U^{n+j})$ by $f(U^{n+j}, t_{n+j})$ in any of the formulas.

Example 6.15. The *Adams methods* have the form

$$U^{n+r} = U^{n+r-1} + k \sum_{j=0}^{r} \beta_j f(U^{n+j}). \tag{6.31}$$

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 $\beta_0,\ \beta_1,\ \ldots,\ \beta_{r-1}$ can be chosen so that the method has order r. This gives the r-step *Adams-Bashforth* method. The first few are given below.

Explicit Adams-Bashforth methods

$$\begin{aligned}
&\text{1-step:} && U^{n+1} = U^n + kf(U^n) \qquad \text{(Forward Euler)}\\
&\text{2-step:} && U^{n+2} = U^{n+1} + \frac{k}{2}(-f(U^n) + 3f(U^{n+1}))\\
&\text{3-step:} && U^{n+3} = U^{n+2} + \frac{k}{12}(5f(U^n) - 16f(U^{n+1}) + 23f(U^{n+2}))\\
&\text{4-step:} && U^{n+4} = U^{n+3} + \frac{k}{24}(-9f(U^n) + 37f(U^{n+1}) - 59f(U^{n+2}) + 55f(U^{n+3}))
\end{aligned}$$

These can be derived by various approaches, see Exercise 6.3, for example.

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 local truncation error. This gives an implicit method of order $r+1$ called the r-step *Adams-Moulton* method. The first few are given below.

Implicit Adams-Moulton methods

$$\begin{aligned}
&\text{1-step:} && U^{n+1} = U^n + \frac{k}{2}(f(U^n) + f(U^{n+1})) \qquad \text{(Trapezoidal method)}\\
&\text{2-step:} && U^{n+2} = U^{n+1} + \frac{k}{12}(-f(U^n) + 8f(U^{n+1}) + 5f(u^{n+2}))\\
&\text{3-step:} && U^{n+3} = U^{n+2} + \frac{k}{24}(f(U^n) - 5f(U^{n+1}) + 19f(U^{n+2}) + 9f(U^{n+3}))\\
&\text{4-step:} && U^{n+4} = U^{n+3} + \frac{k}{720}(-19f(U^n) + 106f(U^{n+1}) - 264f(U^{n+2})\\
& && \qquad + 646f(U^{n+3}) + 251f(U^{n+4}))
\end{aligned}$$

Example 6.16. The explict *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 (6.17) is a 2-step explicit Nyström method. A 2-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 $u'(t) = f(t)$.

6.8.1 Local truncation error

For LMMs it is easy to derive a general formula for the local truncation error. 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

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

and so

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

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:

$$\sum_{j=0}^{r}\alpha_j = 0 \qquad \text{and} \qquad \sum_{j=0}^{r} j\alpha_j = \sum_{j=0}^{r}\beta_j. \tag{6.32}$$

If the first $p+1$ terms vanish then the method will be pth order accuate. 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.

6.8.2 Characteristic polynomials

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

$$\rho(\zeta)=\sum_{j=0}^{r}\alpha_j\zeta^j \qquad \text{and} \qquad \sigma(\zeta)=\sum_{j=0}^{r}\beta_j\zeta^j. \tag{6.33}$$

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 (6.32) can be written quite concisely as conditions on these two polynomials:

$$\rho(1)=0 \qquad \text{and} \qquad \rho'(1)=\sigma(1). \tag{6.34}$$

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 chapter.

Example 6.17. The 2-step Adams-Moulton method

$$U^{n+2}=U^{n+1}+\frac{k}{12}(-f(U^n)+8f(U^{n+1})+5f(U^{n+2})) \tag{6.35}$$

has characteristic polynomials

$$\rho(\zeta)=\zeta^2-\zeta, \qquad \sigma(\zeta)=\frac{1}{12}(-1+8\zeta+5\zeta^2). \tag{6.36}$$

6.8.3 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 must typically be generated by some other numerical method or methods.

Example 6.18. If we want to use the midpoint method (6.17) then we need to generate U^1 by some other method before we begin to apply (6.17) with $n=1$. We can obtain U^1 from U^0 using any

Table 6.1: Error in the solution of Example 6.19 with $k = 2^{-j}$ when Euler and Trapezoidal are used to generate U^1.

j	Error with Euler	Ratio	Error with Trapezoidal	Ratio
5	1.3127e−01	3.90	8.5485e−02	3.80
6	3.3008e−02	3.97	2.1763e−02	3.92
7	8.2641e−03	3.99	5.4816e−03	3.97
8	2.0668e−03	3.99	1.3750e−03	3.98
9	5.1674e−04	3.99	3.4428e−04	3.99
10	1.2919e−04	3.99	8.6134e−05	3.99

1-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 that this second order accuarcy 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 in fact 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 for the boundary conditions than what we have elsewhere.

In the present context this is easiest to explain in terms of the 1-step error. The midpoint method has a 1-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.

Example 6.19. Suppose we solve

$$u'(t) = 3u(t), \quad u(0) = 1$$

with solution $u(t) = e^{3t}$. We take $U^0 = 1$. Generating U^1 using Euler's method gives $U^1 = (1+3k)U^0 = 1 + 3k$ which agrees with $u(k) = e^{3k}$ to $O(k^2)$. Table 6.1 shows the resulting errors at time $T = 1$ if we now proceed with the midpoint method. The error with values of $k = 2^{-j}, \ j = 5, \ 6, \ \ldots$ along with the ratio of errors for successive values of k. This ratio approaches 4, confirming second order accuracy. Table 6.1 also shows the results obtained if the Trapezoidal method is used to generate U^1. Although the error is slightly smaller, the order of accuracy is the same. (Of course it may still be worthwhile to do a bit of extra work in the first step to obtain a smaller error at all later times, even though formally the order of accuracy does not improve.)

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 a local truncation error 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. As in the case $p = 2$, somewhat better accuracy may be achieved by using a pth order accurate method for the starting values.

6.9 Exercises

Exercise 6.1 *Compute the local truncation error of the trapezoidal method (6.16) and the BDF method (6.19) to show that both are second order accurate.*

Exercise 6.2 *Derive the third order Taylor series method for the differential equation (6.23).*

Exercise 6.3 *One way to derive the Adams-Bashforth methods is by writing*

$$\begin{aligned} u(t_{n+r}) &= u(t_{n+r-1}) + \int_{t_{n+r-1}}^{t_{n+r}} u'(t)\,dt \\ &= u(t_{n+r-1}) + \int_{t_{n+r-1}}^{t_{n+r}} f(u(t))\,dt \end{aligned} \tag{6.37}$$

and then applying a quadrature rule to this integral to approximate

$$\int_{t_{n+r-1}}^{t_{n+r}} f(u(t))\,dt \approx k\sum_{j=1}^{r-1} \beta_j f(u(t_{n+j})) \tag{6.38}$$

This quadrature rule can be derived by interpolating $f(u(t))$ by a polynomial $p(t)$ at the points $t_n,\ t_{n+1},\ \ldots t_{n+r-1}$ and then integrating the interpolating polynomial. Carry this out for $r = 2$.

Chapter 7

Zero-Stability and Convergence for Initial Value Problems

7.1 Convergence

In order 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

$$\lim_{\substack{k\to 0\\ Nk=T}} U^N = u(T). \tag{7.1}$$

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. In order 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 $k \to 0$:

$$\lim_{k\to 0} U^\nu(k) = \eta \quad \text{for } \nu = 0,\ 1,\ \ldots,\ r-1. \tag{7.2}$$

We can now state the definition of convergence.

Definition 7.1.1 *An r-step method is said to be* convergent *if applying the method to any ODE (6.1) with $f(u,t)$ Lipschitz continuous in u, and with any set of starting values satisfying (7.2), we obtain convergence in the sense of (7.1) for every fixed time $T > 0$.*

In order to be convergent, a method must be *consistent*, meaning as before that the 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 1-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 1-step methods.

7.2 Linear equations and Duhamel's principle

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

$$u'(t) = \lambda u(t) + g(t) \tag{7.3}$$

with initial data

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

Here λ is a constant and $g(t)$ is a given function. In the special case $g(t) \equiv 0$ (the *homogeneous equation*), the solution is simply

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

If we let $\mathcal{S}(t, t_0) = e^{\lambda(t-t_0)}$ be this solution operator, which maps data at time t_0 to the solution at time t, then we can write the solution of the more general linear problem (7.3) using *Duhamel's principle*, which states that the solution to the nonhomogeneous problem is obtained by adding to the solution of the homogeneous problem a superposition of $\mathcal{S}(t,\tau)g(\tau)$ over all times τ between t_0 and t. In this sense $\mathcal{S}(t,\tau)$ acts like a Green's function (compare to (2.31)). So we have

$$\begin{aligned} u(t) &= \mathcal{S}(t,t_0)\eta + \int_{t_0}^{t} \mathcal{S}(t,\tau)g(\tau)\,d\tau \qquad (7.4)\\ &= e^{\lambda(t-t_0)}\eta + \int_{t_0}^{t} e^{\lambda(t-\tau)}g(\tau)\,d\tau. \end{aligned}$$

7.3 One-step methods

7.3.1 Euler's method on linear problems

If we apply Euler's method to the equation (7.3), we obtain

$$\begin{aligned} U^{n+1} &= U^n + k(\lambda U^n + g(t_n)) \\ &= (1 + k\lambda)U^n + kg(t_n). \end{aligned} \tag{7.5}$$

The local truncation error for Euler's method is given by

$$\begin{aligned} \tau^n &= \left(\frac{u(t_{n+1}) - u(t_n)}{k}\right) - (\lambda u(t_n) + g(t_n)) \\ &= \left(u'(t_n) + \frac{1}{2}ku''(t_n) + O(k^2)\right) - u'(t_n) \qquad (7.6)\\ &= \frac{1}{2}ku''(t_n) + O(k^2). \end{aligned}$$

Rewriting this equation as

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

and subtracting this from (7.5) gives a difference equation for the global error E^n:

$$E^{n+1} = (1 + k\lambda)E^n - k\tau^n. \tag{7.7}$$

Note that this has exactly the same form as (7.5) but with a different nonhomogeneous term: τ^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 (7.7) that we can explicitly solve for the global error E^n. Applying the recursion (7.7) 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

$$E^n = (1+k\lambda)^n E^0 - k\sum_{m=1}^{n}(1+k\lambda)^{n-m}\tau^{m-1}. \tag{7.8}$$

(Note that some of the superscripts are powers while others are indices!) This has a form that is very analogous to the solution (7.4) of the corresponding ordinary differential equation, 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 (7.8) is sometimes called the discrete form of Duhamel's principle.

We are now ready to prove that Euler's method converges on the equation (7.3). We need only observe that

$$|1+k\lambda| \le e^{k|\lambda|} \tag{7.9}$$

and so

$$(1+k\lambda)^{n-m} \le e^{(n-m)k|\lambda|} \le e^{nk|\lambda|} \le e^{|\lambda|T} \tag{7.10}$$

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 (7.8) that

$$\begin{aligned} |E^n| &\le e^{|\lambda|T}\left(|E^0| + k\sum_{m=1}^{n}|\tau^{m-1}|\right) \\ &\le e^{|\lambda|T}\left(|E^0| + nk\max_{1\le m\le n}|\tau^{m-1}|\right). \end{aligned} \tag{7.11}$$

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 (7.6) 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 (7.11) that

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

If (7.2) 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

$$|E^n| \le e^{|\lambda|T}T\|\tau\|_\infty = O(k) \quad \text{as } k \to 0 \tag{7.12}$$

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 $(1+k\lambda)^{n-1}\mathcal{L}^{m-1}$ to the global error. The fact that $|(1+k\lambda)^{n-1}| < e^{|\lambda|T}$ 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 6.4 is in fact valid, and the global error has the same order of magnitude as the local truncation error.

7.3.2 Relation to stability for BVP's

In order to see how this ties in with the definition of stability used in Chapter 2 for the boundary value problem, it may be useful to view Euler's method as giving a linear system in matrix form, even though this is not the way it is used computationally. If we view the equations (7.5) 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 the equation (7.5) 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 (7.5).

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 the equation (2.15) (the matrix form of (7.7)) with solution (7.8). 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 2.7.1, *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 (7.8) 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 (A1.8a) 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 (7.12).

7.3.3 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 6.1.

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

$$U^{n+1} = U^n + kf(U^n) \tag{7.13}$$

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 (7.13) gives

$$E^{n+1} = E^n + k(f(U^n) - f(u(t_n))) - k\tau^n. \tag{7.14}$$

In the linear case $f(U^n) - f(u(t_n)) = \lambda E^n$ and we get the relation (7.7) 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 (7.14) gives

$$|E^{n+1}| \le |E^n| + kL|E^n| + k|\tau^n| = (1 + kL)|E^n| + k|\tau^n| \tag{7.15}$$

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 (7.12) (and again assuming $E^0 = 0$), we obtain

$$|E^n| \le e^{LT}T\|\tau\|_\infty = O(k) \quad \text{as } k \to 0 \tag{7.16}$$

for all n with $nk \le T$, proving that the method converges. In the linear case $L = |\lambda|$ and this reduces to exactly (7.12).

7.3.4 General 1-step methods

A general explicit one-step method takes the form

$$U^{n+1} = U^n + k\Psi(U^n, t_n, k) \tag{7.17}$$

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 7.1. For the 2-stage Runge-Kutta method of Example 6.11, we have

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

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

The 1-step method (7.17) 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 methods 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 (7.17) 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

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

This has exactly the same form as (7.15) and the proof of convergence proceeds exactly as from there.

7.4 Zero-stability of linear multistep methods

The convergence proof of the previous section shows that for 1-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 local truncation error as $k \to 0$. This form of stability is often called *zero-stabilty* 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 linear multistep methods (LMM's), 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 7.2. The LMM

$$U^{n+2} - 3U^{n+1} + 2U^n = -kf(U^n) \tag{7.18}$$

has a local truncation error 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

$$u'(t) = 0, \quad u(0) = 0 \tag{7.19}$$

Table 7.1: Solution U^N to (7.20) with $U^0 = 0$, $U^1 = k$ and various values of $k = 1/N$.

N	U^N
5	4.2
10	258.4
20	1954408

with solution $u(t) \equiv 0$. On this equation (7.18) takes the form

$$U^{n+2} - 3U^{n+1} + 2U^n = 0. \tag{7.20}$$

We need two starting values U^0 and U^1. If we take $U^0 = U^1 = 0$ then (7.20) 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 7.1 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 7.1.1 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 (7.20) can be solved explicitly for U^n in terms of the starting values U^0 and U^1 We obtain

$$U^n = 2U^0 - U^1 + 2^n(U^1 - U^0). \tag{7.21}$$

It is easy to verify that this satisfies (7.20) 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. In order to rule out this sort of growth of errors, we need to be able to solve a general linear difference equation.

7.4.1 Solving linear difference equations

We briefly review the solution technique for linear difference equations. See Appendix A5 for more details. Consider the general homogeneous linear difference equation

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = 0. \tag{7.22}$$

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

$$U^n = \zeta^n \tag{7.23}$$

for some value of ζ (here ζ^n is the nth power!). Plugging this into (7.22) gives

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

and dividing by ζ^n yields

$$\sum_{j=0}^{r} \alpha_j \zeta^j = 0. \tag{7.24}$$

We see that (7.23) is a solution of the difference equation if ζ satisfies the equation (7.24), *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 (6.33). 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 (7.22) has the form

$$U^n = c_1\zeta_1^n + c_2\zeta_2^n + \cdots + c_r\zeta_r^n \tag{7.25}$$

where $c_1,\ \ldots,\ c_r$ are arbitrary constants. In this case, every solution of the difference equation (7.22) 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

$$\begin{array}{rcl} c_1 + c_2 + \cdots + c_r & = & U^0 \\ c_1\zeta_1 + c_2\zeta_2 + \cdots + c_r\zeta_r & = & U^1 \\ \vdots & & \vdots \\ c_1\zeta_1^{r-1} + c_2\zeta_2^{r-1} + \cdots + c_r\zeta_r^{r-1} & = & U^{r-1} \end{array} \tag{7.26}$$

Example 7.3. The characteristic polynomial for the difference equation (7.20) is

$$\rho(\zeta) = 2 - 3\zeta + \zeta^2 = (\zeta - 1)(\zeta - 2) \tag{7.27}$$

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 (7.21).

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 of 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 (7.25), while still a solution, is not the most general solution. The system (7.26) 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 r'th order linear ordinary differential equation. See Appendix A5 for more details.

Example 7.4. Applying the consistent LMM

$$U^{n+2} - 2U^{n+1} + U^n = \frac{1}{2}k(f(U^{n+2}) - f(U^n)) \tag{7.28}$$

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

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

The characteristic polynomial is

$$\rho(\zeta) = \zeta^2 - 2\zeta + 1 = (\zeta - 1)^2 \tag{7.29}$$

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, though not as dramatically as in Example 7.2 (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 (7.28) is not convergent.

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

Example 7.5. Now consider the consistent LMM

$$U^{n+3} - 2U^{n+2} + \frac{5}{4}U^{n+1} - \frac{1}{4}U^n = \frac{1}{4}hf(U^n). \tag{7.30}$$

Applying this to (7.19) gives

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

and the characteristic polynomial is

$$\rho(\zeta) = \zeta^3 - 2\zeta^2 + \frac{5}{4}\zeta - \frac{1}{4} = (\zeta - 1)(\zeta - 0.5)^2. \tag{7.31}$$

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 should then be overwhelmed by the decay of $(1/2)^n$.

For this 3-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 (7.26). 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

$$U^N = c_1(k) + c_2(k)\left(\frac{1}{2}\right)^N + c_3(k)N\left(\frac{1}{2}\right)^N. \tag{7.32}$$

Now we see that

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

and so the method (7.30) 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 r times, then U^N will involve terms of the form $N^s\zeta_j^N$ for $s = 1,\ 2,\ ,\dots,\ r$. 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 7.4.1 *An r-step Linear Multistep Method is said to be* **zero-stable** *if the roots of the characteristic polynomial $\rho(\zeta)$ defined by (6.33) satisfy the following conditions:*

$$\begin{aligned} &|\zeta_j| \le 1 \quad \textit{for } j = 1,\ 2,\ ,\dots,\ r \\ &\textit{If } \zeta_j \textit{ is a repeated root, then } \ |\zeta_j| < 1. \end{aligned} \tag{7.33}$$

If the conditions (7.33) are satisfied for all roots of ρ, then the polynomial is said to satisfy the *root condition.*

Example 7.6. 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 of the Adams-Bashforth and Adams-Moulton methods are zero stable.

The examples given above certainly do not prove that zero-stability as defined above is a sufficient condition for convergence. We only looked at 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 7.1.1). For the initial value problem we have the general result that

$$\text{consistency} \ + \ \text{zero-stabilty} \iff \text{convergence}. \tag{7.34}$$

This is the analog of the statement (2.21) for the boundary value problem. A proof of this result can be found in [Hen62]

Note: A consistent linear multistep method always has one root equal to 1, say $\zeta_1 = 1$, called the *principal root.* This follows from (6.34). Hence a consistent 1-step LMM (such as Euler, backward Euler, trapezoidal) is certainly zero-stable. More generally we have proved in Section 7.3.4 that any consistent 1-step method (that is 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 the time step k is chosen too large in an actual computation. Additional stability considerations are the subject of the next chapter.

7.5 Exercises

Exercise 7.1 *Use (7.4) to solve the equation $u'(t) = 3u(t) + 2t$ with $u(0) = 1$.*

Exercise 7.2 *Verify the Lipschitz constant L' for Example 7.1.*

Chapter 8

Absolute Stability for ODEs

8.1 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 actually 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 8.1. Consider the 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 is

$$\begin{aligned} \tau(t) &= \frac{1}{2}ku''(t) + O(k^2) \\ &= -\frac{1}{2}k\cos(t) + O(k^2) \end{aligned} \tag{8.1}$$

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 (7.12) 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 8.2. Now suppose we modify the above equation to

$$u'(t) = \lambda(u - \cos t) - \sin t \tag{8.2}$$

Table 8.1: Errors in the computed solution using Euler's method in Example 8.3, 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

where λ is some constant. If we take the same initial data as before, $u(0) = 0$, then the solution is also the same as before, $u(t) = \sin t$. As a concrete example, let's take $\lambda = -10$. Now how small do we need to take k in order to get an error that is 10^{-3}? Since the LTE (8.1) depends only on the true solution $u(t)$, which is unchanged from Example 8.1, 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 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 8.3. Now consider the problem (8.2) 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 8.1 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 (8.2) is a linear equation of the form (7.3) and so the analysis of Section 7.3.1 applies directly to this problem. From (7.7) we see that the global error E^n satisfies the recursion relation

$$E^{n+1} = (1 + k\lambda)E^n - k\tau^n \tag{8.3}$$

where the local error $\tau^n = \tau(t_n)$ from (8.1). The expression (8.3) reveals the source of the exponential growth in the error — in each time step the previous error is multiplied by a factor $(1 + k\lambda)$. 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 (7.8)). 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 8.2 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 8.2 and in fact a better result than in Example 8.1, where $1 + k\lambda = 1$.

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 $k = 0.00095$ in Table 8.1.

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 (7.12),

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

that we used to prove convergence, allows the possibility of exponential growth with time. The bound is valid for all values of 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.

8.2 Absolute stability

In order 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-stabilty. There are a wide variety of other forms of "stability" that have been studied in various contexts. The one which is most basic and suggests itself from the above examples is *absolute stability.* This notion is based on the linear test equation (7.3) 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.

In fact 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 is is only their product $z \equiv k\lambda$ that 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 8.1a). Allowing λ to be complex comes from the fact that in practice we are typically solving a nonlinear system of ODEs. After linearization we obtain a linear system of equations and it is the eigenvalues of the resulting Jacobian matrix that are important in determining stability. (This is discussed in Section 8.6.3.) Hence λ represents a typical eigenvalue and these may be complex even if the matrix is real.

8.3 Stability regions for LMMs

For a general LMM of the form (6.30), 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

$$\sum_{j=0}^{r} (\alpha_j - z\beta_j) U^{n+j} = 0. \tag{8.4}$$

Note again that it is only the product $z = k\lambda$ that is important, not the values of k or λ separately. Note also that this is a dimensionless quantity since the decay rate λ has dimensions time^{-1} while the time step has dimensions of time.

The recurrence (8.4) is a homogeneous linear difference equation of the same form considered in Section 7.4.1. The solution has the general form (7.25) 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

$$\pi(\zeta; z) = \rho(\zeta) - z\sigma(\zeta).$$

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 7.4.1, this requires that the polynomial $\pi(\zeta; z)$ satisfy the root condition (7.33).

Definition 8.3.1 *The* **region of absolute stability** *for the LMM (6.30) is the set of points z in the complex plane for which the polynomial $\pi(\zeta; z)$ satisfies the root condition (7.33).*

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

Example 8.4. 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 circle in Figure 8.1a.

Example 8.5. For the Backward Euler method (6.15),

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

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

$$|(1 - z)^{-1}| \leq 1 \iff |1 - z| \geq 1$$

so the stability region is the *exterior* of the disk of radius 1 centered at $z = 1$, as shown in Figure 8.1b.

Example 8.6. For the Trapezoidal method (6.16),

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

with root

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

This is known as a *linear fractional transformation* in complex analysis and it can be shown that

$$|\zeta_1| \leq 1 \iff \text{Re}(z) \leq 0$$

where Re(z) is the real part. So the stability region is the left half plane as shown in Figure 8.1c.

Example 8.7. For the Midpoint method (6.17),

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

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 $\zeta_1 \neq \zeta_2$, 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 8.1d.

Since k is always real, this means the Midpoint method is only useful 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, as discussed later.

Example 8.8. Figures 8.2 and 8.3 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 an elegant technique called the *boundary locus method* that makes it simple to determine the regions shown in the figures. This is briefly described in the next section (see also Lambert[Lam73], for example).

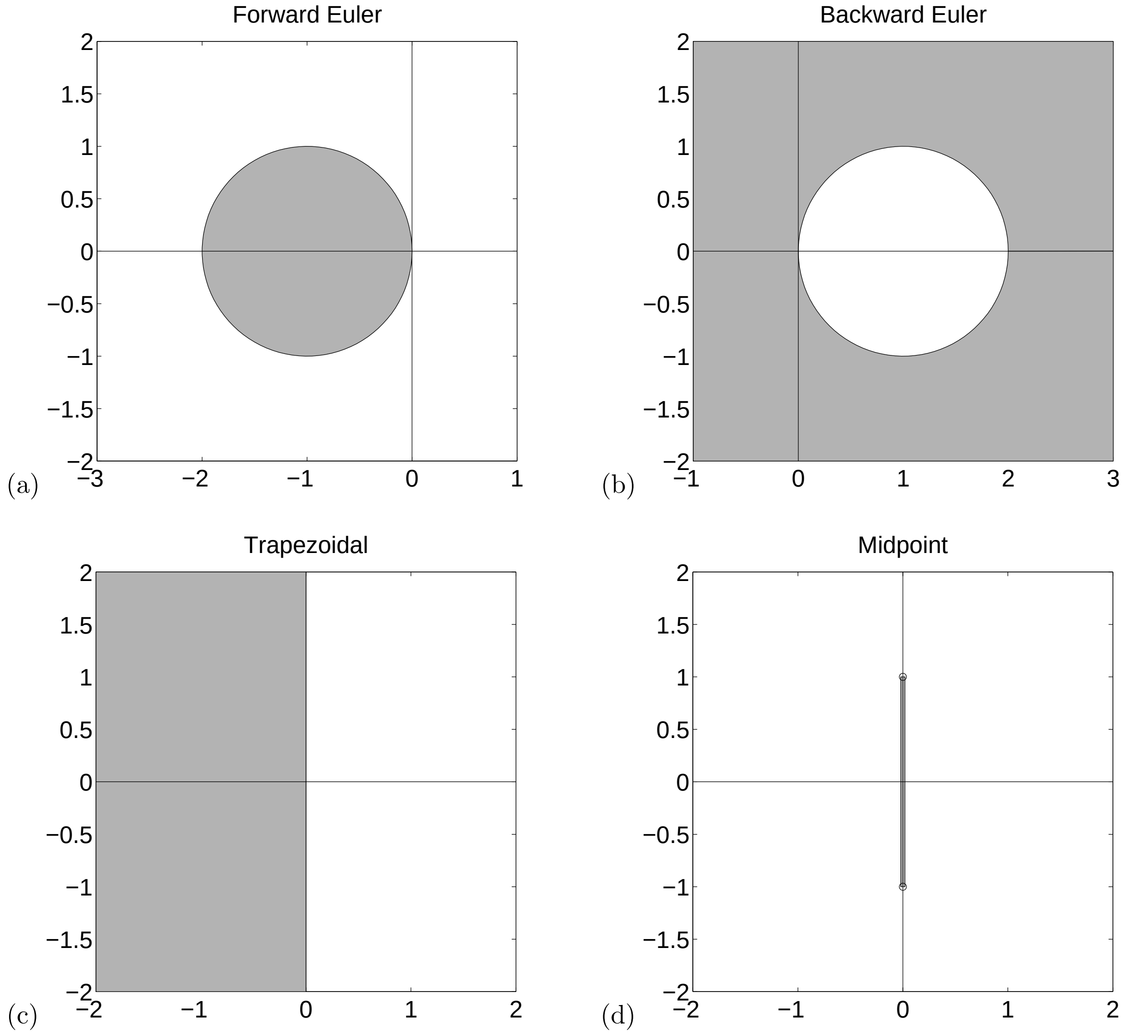


Figure 8.1: Stability regions for (a) Euler — interior of circle, (b) Backward Euler, — exterior of circle (c) Trapezoidal — left half plane, and (d) Midpoint — segment on imaginary axis.

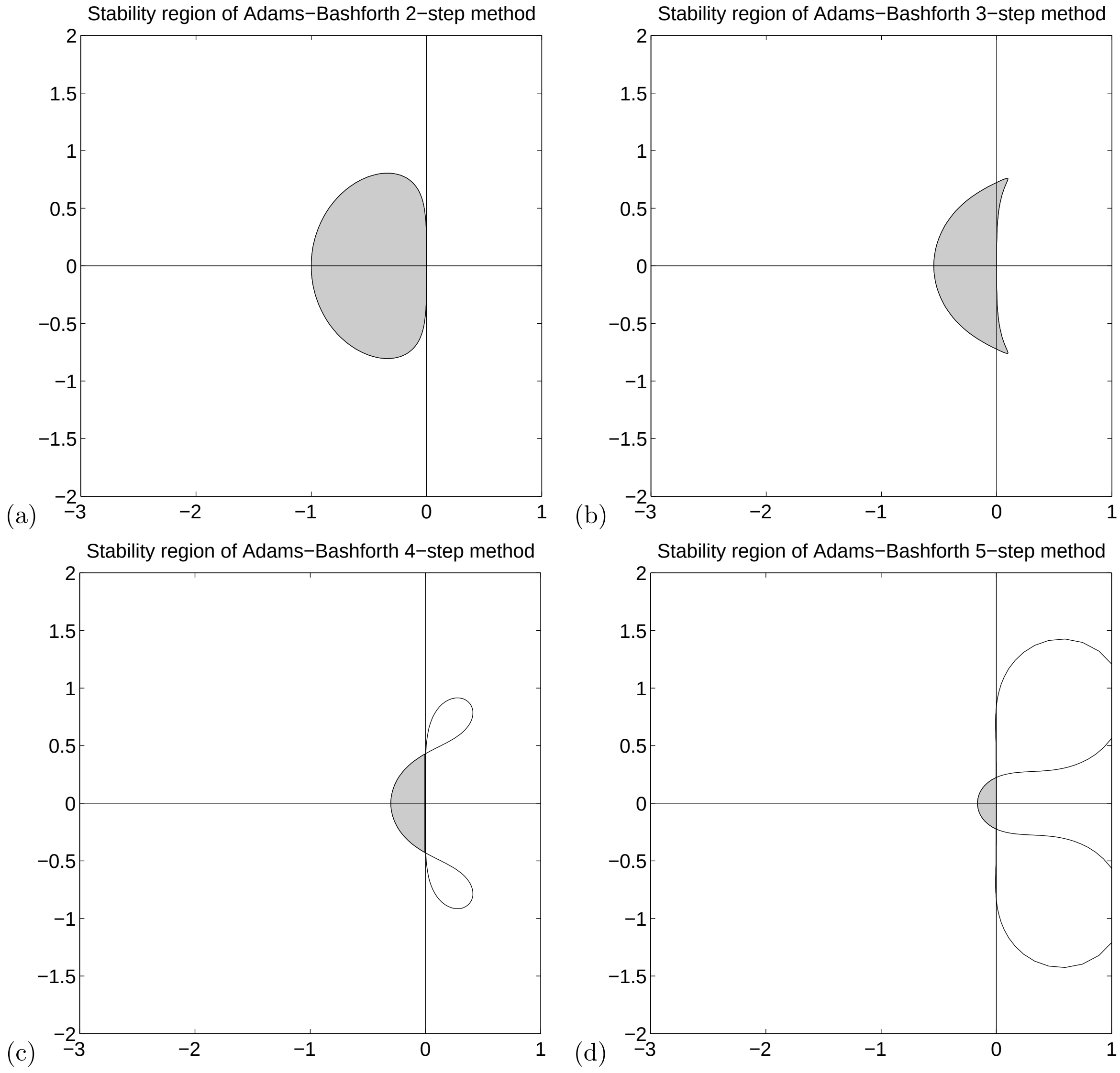


Figure 8.2: Stability regions for some Adams-Bashforth methods. The enclosed region just to the left of the origin is the region of absolute stability. See Section 8.4 for a discussion of the other loops seen in some figures.

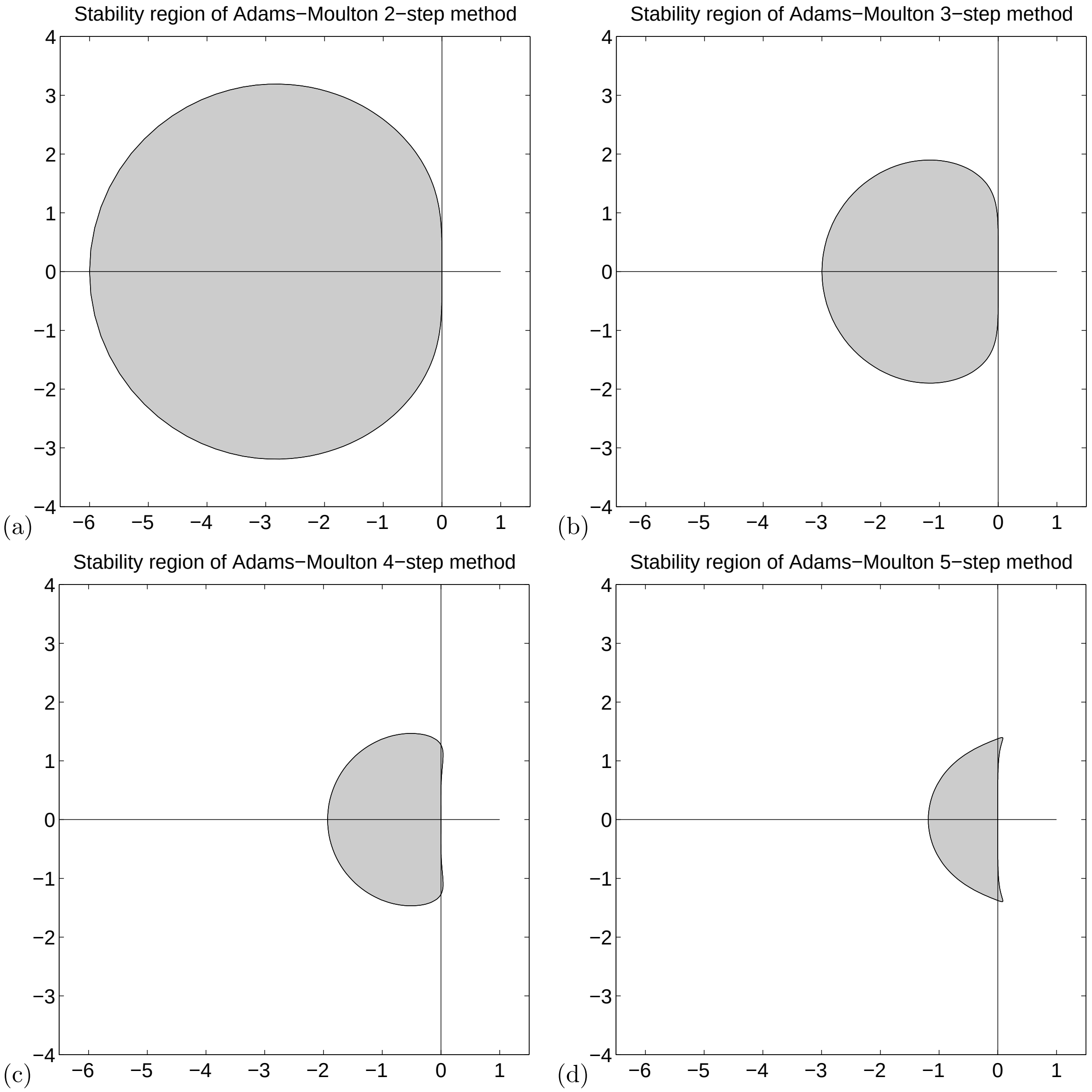


Figure 8.3: Stability regions for some Adams-Moulton methods. The stability region is interior to the curves.

8.4 The Boundary Locus Method

A point $z \in \mathbb{C}$ is in the stability region $\mathcal{S}$ of a linear multistep method 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 $[0, 2\pi]$ (Sorry for 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

$$\rho(e^{i\theta}) - z\sigma(e^{i\theta}) = 0 \tag{8.5}$$

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

$$\tilde{z}(\theta) \equiv \frac{\rho(e^{i\theta})}{\sigma(e^{i\theta})} \tag{8.6}$$

for $0 \leq \theta \leq 2\pi$ 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 8.9. 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 8.1(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. For example, at the center of the circle $z = 1$ we have $\pi(\zeta; 1) = -\zeta$ with root $\zeta_1 = 0$. So π satisfies the root condition here and this point must be in the interior. It follows easily that all other points inside the curve must also be in the interior of $\mathcal{S}$, since the roots are continuous functions of z and so as we vary z a root can potentially move outside the unit circle only when we cross the boundary locus. Checking the roots at a random point outside the circle shows that these points must be exterior to the stability region.

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 8.10. The 5-step Adams-Bashforth method gives the boundary locus seen in Figure 8.4. The numbers 0, 1, 2 in the different regions indicate how many roots are outside of the unit circle in each region, as determined by testing the roots at a random point in each region. 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 into one of the regions marked "2", another root moves outside and the method is still unstable in these regions.

For a linear multistep method of any order the equation (8.5) is linear in z and so the boundary locus is easily plotted by solving for z and letting θ range over $[0, 2\pi]$. The boundary locus isea can be extgended to other methods, including Taylor series and Runge-Kutta methods, but for a pth order

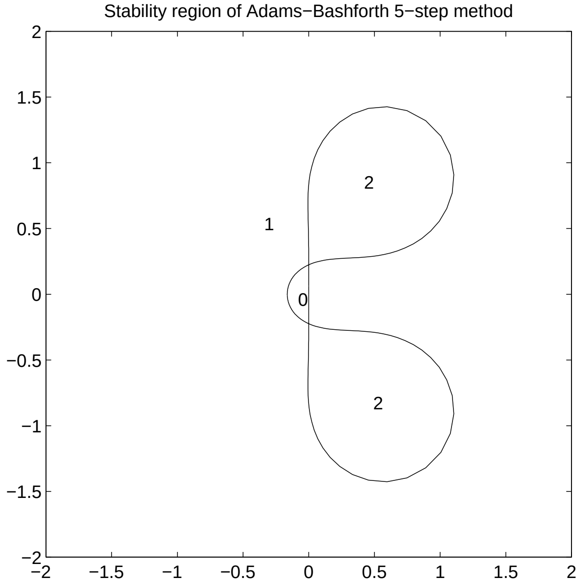


Figure 8.4: Boundary locus for the 5-step Adams-Bashforth method. The numbers indicate how many roots are outside of the unit circle in each region. The region marked "0" is the stability region.

1-step method this will generally lead to a pth order polynomial equation in z that must be solved to obtain points on the boundary locus.

Example 8.11. The second order Taylor series method (see Section 6.5) when applied to the test problem $u' = \lambda u$ gives

$$U^{n+1} = \left(1 + z + \frac{1}{2}z^2\right) U^n, \tag{8.7}$$

where $z = k\lambda$. Note that the 2-step Runge-Kutta method (6.24) when applied to the test problem also gives the expression (8.7), so this method gives exactly the same computational results as the 2nd order Taylor series method on the test problem. Consequently these two 1-step methods have identical regions of absolute stability. These methods are absolutely stable if z satisfies

$$\left|1 + z + \frac{1}{2}z^2\right| \le 1.$$

To determine the boundary locus we set

$$1 + z + \frac{1}{2}z^2 = e^{i\theta}$$

and plot the z values for which this holds for some $\theta \in [0, 2\pi]$. For each θ we now have a quadratic equation to solve for z, yielding

$$z = -1 \pm \sqrt{1 - 2(1 - e^{i\theta})}. \tag{8.8}$$

The stability region is shown in Figure 8.5, along with the stability region for the 3rd order Taylor series method, obtained by finding the roots of a cubic for each value of θ (see Exercise 8.1). These figures indicate the boundary locus by plotting the roots for each of 30 values of θ in $[0, 2\pi]$.

8.5 Linear multistep methods as one-step methods on a system

A linear multistep method can be written in the form of a one-step method applied to a system of equations. Doing so allows us to see how studying absolute stability via the roots of the characteristic

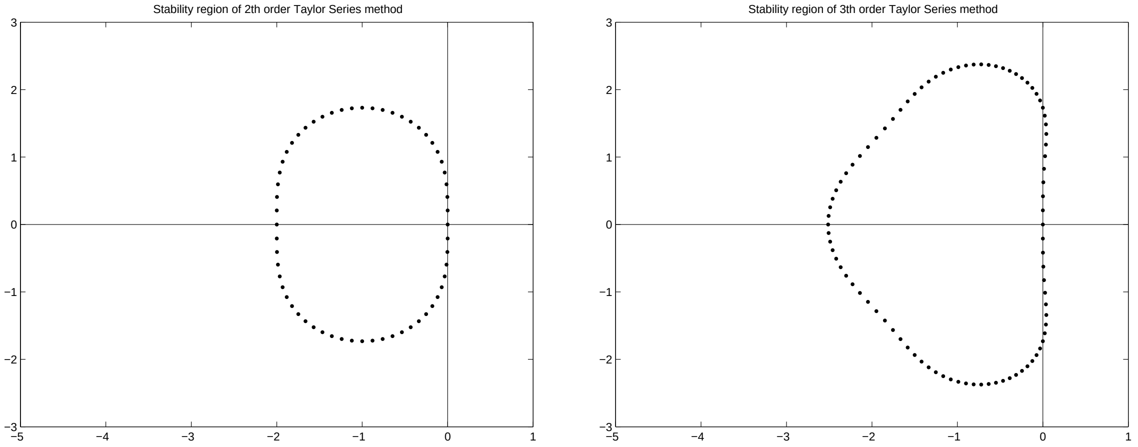


Figure 8.5: Boundary locus for the 2nd and 3rd order Taylor Series methods.

polynomial is related to studying matrix iterations via the eigenvalues of the matrix. Moreover we can see how the condition of zero-stability leads to a convergence proof similar to the proof of convergence of a one-step method seen in Chapter 7.

For simplicity we will only consider *explicit* methods applied to the *linear* problem $u'(t) = \lambda u(t) + g(t)$, for which a general r-step method has the form

$$\sum_{j=0}^{r} \alpha_j U^{n+j} = k \sum_{j=0}^{r-1} \beta_j \left[\lambda U^{n+j} + g(t_{n+j})\right]. \tag{8.9}$$

We can assume without loss of generality that $\alpha_r = 1$, since there is one degree of freedom in the coefficients because we can multiply both sides of (8.9) by an arbitrary constant without changing the numerical method. Then we can solve for U^{n+r} as

$$U^{n+r} = \sum_{j=0}^{r-1} [(-\alpha_j + k\lambda\beta_j)U^{n+j} + k\beta_j g(t_{n+j})].$$

Let $\gamma_j = -\alpha_j + k\lambda\beta_j$ and define the vector V^n by

$$V^n = \begin{bmatrix} U^n \\ U^{n+1} \\ \vdots \\ U^{n+r-1} \end{bmatrix}, \quad \text{so that} \quad V^{n+1} = \begin{bmatrix} U^{n+1} \\ U^{n+2} \\ \vdots \\ U^{n+r} \end{bmatrix}.$$

With this notation we see that V^n is updated by a 1-step method

$$V^{n+1} = AV^n + kb^n$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & & & 1 \\ \gamma_0 & \gamma_1 & \cdots & & & \gamma_{r-1} \end{bmatrix}, \qquad b^n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \sum_{j=0}^{r-1} \beta_j g(t_{n+j}) \end{bmatrix}.$$

Note that we know the initial vector V^0 from the starting values U^0 …, U^{r-1} needed for the r-step method.

In the usual way, we find that the error vector E^n satisfies

$$E^{n+1} = AE^n - k\tau^n,$$

where

$$\tau^n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \tau(t_{n+r}) \end{bmatrix},$$

and hence

$$E^n = A^n E^0 - k\sum_{m=1}^{n} A^{n-m}\tau^{m-1}. \tag{8.10}$$

(Here the superscripts on E and τ are time indices while the superscripts on A are powers.)

8.5.1 Absolute stability

First let's consider the question of absolute stability. How does the error behave as we march forward in time with some fixed k? We want the effect of past errors to not be amplified in future steps, so we want a uniform bound on $\|A^n\|$. A necessary condition is that the spectral radius of A must be no larger than 1. If A is diagonalizable then this is also a sufficient condition.

The eigenvalues ζ of A satisfy

$$\det(\zeta I - A) = 0.$$

It is easy to verify that this determinant is simply

$$\det(\zeta I - A) = \zeta^r - \gamma_{r-1}\zeta^{r-1} - \cdots - \gamma_1\zeta - \gamma_0.$$

Recalling that $\alpha_r = 1$ and $\beta_r = 0$, we see that this is exactly the "stability polynomial" $\pi(\zeta; k\lambda)$ defined in Section **??**. So for stability we require that the roots of this polynomial (which are the eigenvalues of A) be no larger than 1 in modulus. The matrix A is called the *companion matrix* for this polynomial.

Moreover, it turns out that if π has a repeated root then this is a multiple eigenvalue of A, and in this case A is *not* diagonalizable — it is *defective* and has a nontrivial Jordan block. If this eigenvalue has modulus equal to 1 then $\|A^n\|$ will grow with n. Hence for stability we must also require that any multiple root have modulus strictly less than 1.

The upshot is that the condition required for uniform boundedness of $\|A^n\|$ is exactly the "root condition" on the stability polynomial π, *i.e.*, absolute stability of the method.

8.5.2 Convergence and zero-stability

Now let's examine the question of convergence as $k \to 0$. Looking again at (8.10), and following the proof in Section 7.3, we see that the method converges if $\|\tau^n\|_\infty \to 0$ (consistency), $\|E^0\| \to 0$ (good starting values), and $\|A^n\|$ is uniformly bounded as $k \to 0$ for $nk \le T$ (which we will see requires exactly the conditions of zero-stability). In the case of a 1-step method, A was a scalar bounded by $1 + L'k$, and hence $A^n \le e^{L'T}$ could be uniformly bounded.

Note that we are looking for something different than the uniform boundedness of the previous section. Rather than looking at how A^n behaves as $n \to \infty$ with k fixed, we are examining what happens as $k \to 0$ and $n \to \infty$ simultaneously, while $nk = T$ is fixed. To make this clearer, let's now write the matrix as A_k to remind ourselves that it depends on k. As $k \to 0$ the value γ_j in the last row of A_k approaches $-\alpha_j$. It can be shown that $\|A_k^n\|$ is uniformly bounded for all n and k with $nk \le T$

provided the eigenvalues of the limiting matrix A_0 are no greater than 1 in modulus (and striclty less than 1 for multiple eigenvalues). The determinant of $A_0 - \zeta I$ is just the characteristic polynomial $\rho(\zeta) = \sum_{j=0}^{r} \alpha_j \zeta^j$, and so this is just the zero-stability condition.

The proof of this statement follows from the fact that the eigenvalues of a matrix are continuous functions of the matrix elements, and moreover that modifying the elements by $O(k)$ leads to an $O(k)$ perturbation of the eigenvalues (in the nondefective case). Hence if the characteristic polynomial ρ satisfies the root condition then the matrix A_k for $k > 0$ has spectral radius that can be bounded by $1 + Ck$ for some constant C, and hence the n'th power can be bounded by some constant times $(1 + Ck)^n \le e^{CT}$, just as in the case of a scalar 1-step method.

8.6 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 $u(t) \in \mathbb{R}^m$. We will see that for a linear system $u' = Au + b$, where A is an $m \times m$ matrix, it is the eigenvalues of A that are important and we need $k\lambda$ 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 stabilty region for each eigenvalue λ of the Jacobian matrix $f'(u)$. Before discussing this theory, we will review the theory of chemical kinetics, a field where the solution of systems of ordinary differential equations is very important, and where the eigenvalues of the Jacobian matrix often have a physical interpretation in terms of reaction rates.

8.6.1 Chemical Kinetics

Let A, B, $C, \ldots$ 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

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

and the equations then become

$$\begin{aligned} u_1' &= -K_1 u_1 + K_2 u_2 \\ u_2' &= K_1 u_1 - K_2 u_2 \end{aligned} \tag{8.11}$$

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

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

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

$$\begin{aligned} u_1' &= -K_1u_1u_2 + K_2u_3 \\ u_2' &= -K_1u_1u_2 + K_2u_3 \\ u_3' &= K_1u_1u_2 - K_2u_3 \end{aligned} \tag{8.12}$$

Note that this is a nonlinear system of equations, while (8.11) 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

$$\begin{aligned} u_1' &= -K_1u_1u_2 + K_2u_3 - 2K_3u_1^2u_4 + 2K_4u_5 \\ u_2' &= -K_1u_1u_2 + K_2u_3 \\ u_3' &= K_1u_1u_2 - K_2u_3 \\ u_4' &= -K_3u_1^2u_4 + K_4u_5 \\ u_5' &= K_3u_1^2u_4 - K_4u_5 \end{aligned} \tag{8.13}$$

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 9.

Example 8.12. 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

$$A = \begin{bmatrix} -K_1 & 0 & 0 \\ K_1 & -K_2 & 0 \\ 0 & K_2 & 0 \end{bmatrix}. \tag{8.14}$$

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 (though 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 8.6.

8.6.2 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

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

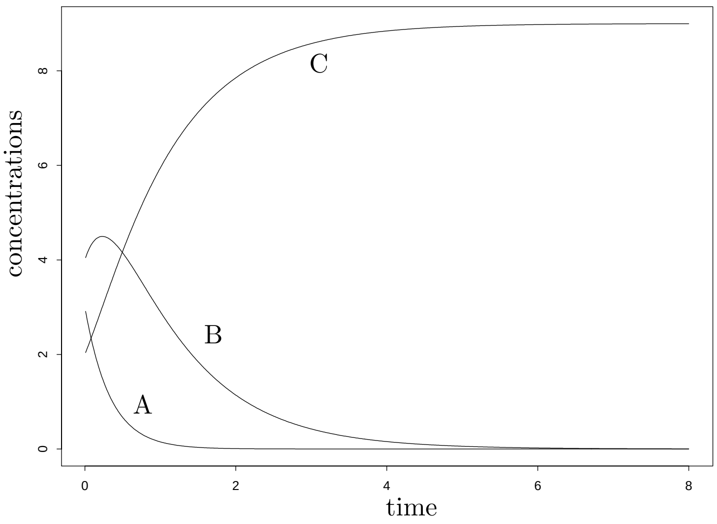


Figure 8.6: Sample solution for the kinetics problem in Example 8.12.

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

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

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 $U^n = RV^n$.

The overall method is stable if each of the scalar problems is 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 a LMM is absolutely stable if $k\lambda_p$ is in the stability region of the method for each eigenvalue λ_p of the matrix A.

Example 8.13. Consider the linear kinetics problem with A given by (8.14). 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 the Euler's method is stable provided $k \max(K_1, K_2) \leq 2$.

Example 8.14. 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

$$A = \begin{bmatrix} 0 & 1 \\ -a & -b \end{bmatrix}. \tag{8.15}$$

The eigenvalues of this matrix are $\lambda = \frac{1}{2}\left(-b \pm \sqrt{b^2 - 4a}\right)$. 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.

8.6.3 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 8.15. Consider the kinetics problem (8.12). 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 $u_1(t) + u_2(t) \leq u_1(0) + u_2(0) + 2u_3(0)$.) So we can determine the possible range of λ_1 along the negative real axis and hence how small k must be to stay within the region of absolute stability.

8.7 Choice of stepsize

As the examples at the beginning of this chapter illustrated, in order to obtain computed results that are within some error tolerance, we need two conditions satisfied:

1. The time step k must be small enough that the local truncation error is acceptably small. This gives a constraint of the form $k \leq k_{\text{acc}}$ 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 occuring in this expansion are.
- 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 size of λ or, more generally, 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 in order 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 9, for which special methods have been developed.

8.8 Exercises

Exercise 8.1 *Use the boundary locus method to determine the region of absolute stability for the following methods:*

(a) The midpoint method (6.17).

(b) The third-order Taylor series method. (You will have to plot roots of a cubic equation for each θ. You can use the MATLAB `roots` *command to compute these.)*

(c) The fourth-order Taylor series method.

Chapter 9

Stiff ODEs

The problem of *stiffness* leads to computational difficulty in many practical problems. The classic example is the case of a stiff ordinary differential equation, 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 towards a smooth solution.

Example 9.1. Consider the ODE (8.2) from the previous chapter,

$$u'(t) = \lambda(u - \cos t) - \sin t. \tag{9.1}$$

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

$$u(t) = e^{\lambda(t-t_0)}(\eta - \cos(t_0)) + \cos t. \tag{9.2}$$

Figure 9.1 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 9.1b shows the corresponding solution curves when $\lambda = -10$.

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

Example 9.2. Consider the kinetics model $A \to B \to C$ developed in Example 8.12. The system of equations is given by (8.14). Suppose that $K_1 \gg K_2$ so that a typical solution appears as in Figure 9.2(a). (Here $K_1 = 20$ and $K_2 = 1$. Compare this to Figure 8.6.) Now suppose at time $t = 1$ we perturb the system by adding more of species A. Then the solution behaves as shown in Figure 9.2(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, though 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.

9.1 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

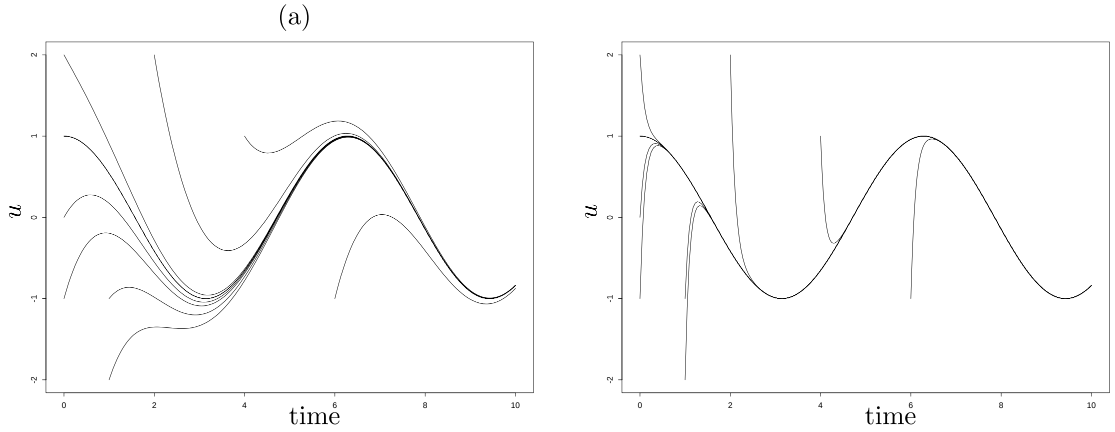


Figure 9.1: Solution curves for the ODE (9.1) for various initial values. (a) With $\lambda = -1$. (b) With $\lambda = -10$ and the same set of initial values.

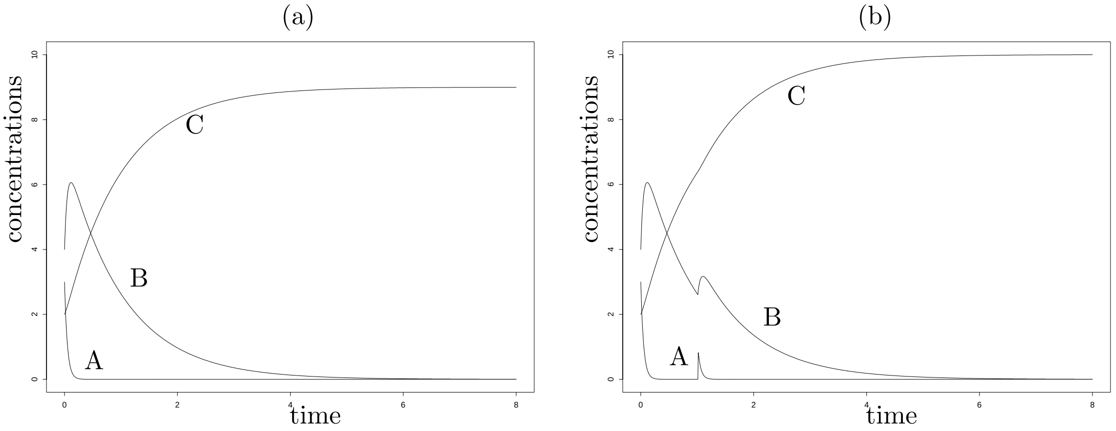


Figure 9.2: Solution curves for the kinetics problem in Example 8.12, with $K_1 = 20$ and $K_2 = 1$. In (b) a perturbation has been made by adding one unit of species A at time $t = 1$.

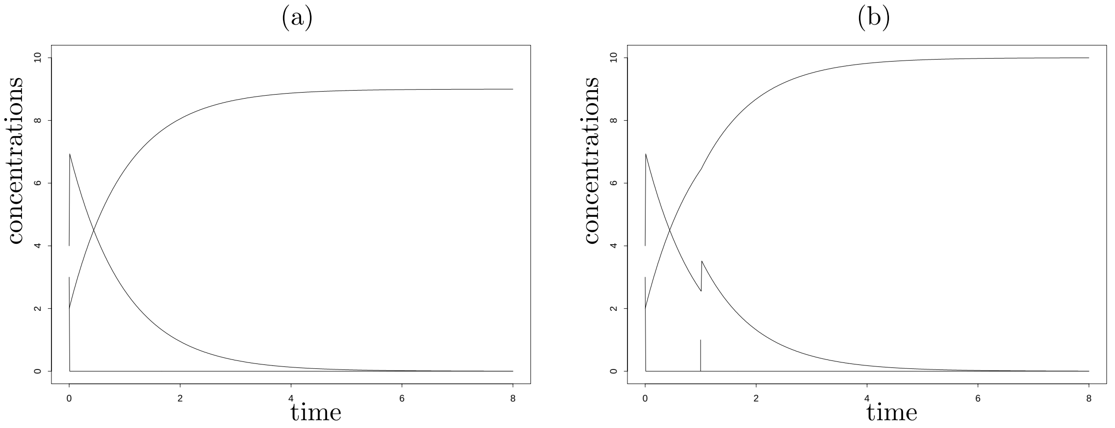


Figure 9.3: Solution curves for the kinetics problem in Example 8.12, with $K_1 = 10^6$ and $K_2 = 1$. In (b) a perturbation has been made by adding one unit of species A at time $t = 1$.

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 (9.1) 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 8.2 were much better than in Example 8.1.)

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 8.7, 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.

9.2 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 ODE's, stiffness is sometimes defined in terms of the "stiffness ratio" of the system, which is the ratio

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

over all eigenvalues of the Jacobian matrix $f'(u)$. If this is large then there is a large range of time scales 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 9.1), even though for a scalar problem the stiffness ratio is always 1 since there is only one eigenvalue. There can still be more than one time scale present. In (9.1) 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 may also be additional time scales arising from inhomogeneous forcing terms or other time-dependent coefficients that are distinct from the scales imposed by the eigenvalues.

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.

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 9.3(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 9.3(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.

9.3 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 method, with a stability region that only extends 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 9.3, 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, even though the solution is very smooth over most of this time.

An analysis of stability regions shows that there are basically two different classes of LMM's: those for which the stability region is bounded and extends distance $O(1)$ from the origin, such as Euler's method, the Midpoint method, or any of the Adams methods (see Figure 8.1 and Figure 9.5), and those for which the stability region is unbounded, such as Backward Euler or trapezoidal. Clearly the first class of methods are inappropriate for stiff problems.

Unfortunately, all explicit methods have bounded stability regions and hence are inefficient on stiff problems. Some implicit methods also have bounded stability regions, such as the Adams-Moulton methods.

9.3.1 A-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. A method with this property is said to be *A-stable*. Unfortunately, a theorem of Dahlquist 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[Hen62]. Higher order A-stable implicit Runge-Kutta methods do exist, but are more difficult to apply [But87].

9.3.2 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. Recall that the root of the stability polynomial for Backward Euler is $\zeta_1 = (1-z)^{-1}$. This approaches 0 as $|z| \to \infty$ in any direction in the complex plane. The point at infinity (on the Riemann sphere, in the sense of complex analysis) is in the *interior* of the stability region for the Backward Euler method.

For the Trapezoidal method, on the other hand,

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

which approaches 1 in modulus as $|z| \to \infty$. It is less than 1 in modulus in the left half plane but greater than 1 in the right half plane. The point at infinity is on the boundary of the stability region for this method.

For a general linear multistep method, the roots of the stability polynomial

$$\pi(\zeta; z) = \rho(\zeta) - z\sigma(\zeta)$$

are the same as the roots of

$$\frac{1}{z}\pi(\zeta; z) = \frac{1}{z}\rho(\zeta) - \sigma(\zeta)$$

and should behave like the roots of $\sigma(\zeta)$ as $|z| \to \infty$. (This is actually true only for implicit methods. What happens for an explicit method?) For Backward Euler the root of σ is at 0, while for Trapezoidal the root is at 1.

A method is called *L-stable* if the method is A-stable and, in addition, the roots of σ are strictly inside the unit circle, $|\zeta_j| < 1$. For an L-stable method the point at infinity is in the interior of the stability region.

I will use the term L-stability in a looser sense and not require A-stability, only that the roots of σ are strictly inside the unit circle, $|\zeta_j| < 1$. This is the crucial property that makes such methods useful for many problems.

If we are solving a stiff equation with initial data such that 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 Rule that is not L-stable.

However, in some situations 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 at time step that is much larger than the true decay time of the transient. For this purpose an L-stable method is crucial. This is best illustrated with an example.

Example 9.3. Again consider the problem $u'(t) = \lambda(u(t) - \cos(t)) - \sin(t)$ with $\lambda = -10^6$ and let's see how Trapezoidal and Backward Euler 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. The following table shows the errors at $T = 3$ with various values of k.

k	Backw. Euler	Trapezoidal
4.0000e-01	4.7770e-02	4.7770e-02
2.0000e-01	9.7731e-08	4.7229e-10
1.0000e-01	4.9223e-08	1.1772e-10
5.0000e-02	2.4686e-08	2.9409e-11
2.5000e-02	1.2360e-08	7.3508e-12
1.2500e-02	6.1837e-09	1.8373e-12
6.2500e-03	3.0928e-09	4.6030e-13
3.1250e-03	1.5466e-09	1.1757e-13

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

k	Backw. Euler	Trapezoidal
4.0000e-01	4.7770e-02	4.5219e-01
2.0000e-01	9.7731e-08	4.9985e-01
1.0000e-01	4.9223e-08	4.9940e-01
5.0000e-02	2.4686e-08	4.9761e-01
2.5000e-02	1.2360e-08	4.9049e-01
1.2500e-02	6.1837e-09	4.6304e-01
6.2500e-03	3.0928e-09	3.6775e-01
3.1250e-03	1.5466e-09	1.4632e-01

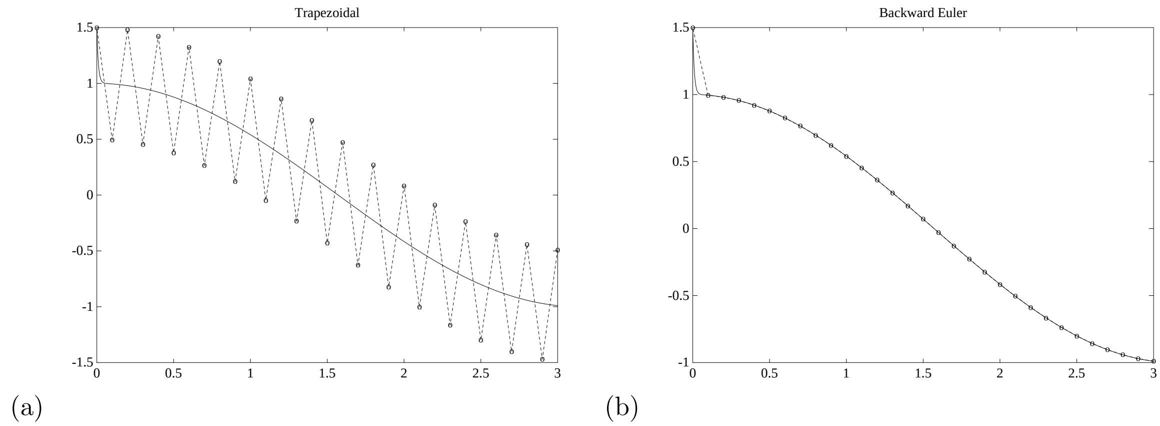


Figure 9.4: Comparison of (a) Trapezoidal method and (b) Backward Euler on a stiff problem with an initial transient (Case 2 of Example 9.3).

To understand what is happening, see Figure 9.4, 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 $\frac{1}{1+k\lambda} \approx -10^{-6}$. This is the proper behavior since the true rapid transient decays in a time period much shorter than a single time step.

9.4 BDF Methods

One class of very effective methods for stiff problems are the BDF methods (Backward Differentiation Formulas). These were first used by Curtis and Hirschfelder[CH52] but are often referred to as Gear's methods since he wrote one of the first software packages for stiff problems based them. See Gear[Gea71] or Lambert[Lam73] for more about these methods.

These methods result from taking $\sigma(\zeta) = \beta_r\zeta^r$, which has all its roots at the origin, resulting in an L-stable method (in the loose sense — these methods are not A-stable for $r > 2$). The method thus has the form

$$\alpha_0 U^n + \alpha_1 U^{n+1} + \cdots + \alpha_r U^{n+r} = k\beta_r f(U^{n+r}), \tag{9.3}$$

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 backwards in time.

It is possible to derive an r-step method that is r'th order accurate. The 1-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. (BDF methods for $r > 6$ fail to be zero-stable.)

$$r = 2: \quad 3U^{n+2} - 4U^{n+1} + U^n = 2kf(U^{n+2})$$

$$r = 3: \quad 11U^{n+3} - 18U^{n+2} + 9U^{n+1} - 2U^n = 6kf(U^{n+3})$$

$$r = 4: \quad 25U^{n+4} - 48U^{n+3} + 36U^{n+2} - 16U^{n+1} + 3U^n = 12kf(U^{n+4})$$

$$r = 5: \quad 137U^{n+5} - 300U^{n+4} + 300U^{n+3} - 200U^{n+2} + 75U^{n+1} - 12U^n = 60kf(U^{n+5})$$

$$r = 6: \quad 147U^{n+6} - 360U^{n+5} + 450U^{n+4} - 400U^{n+3} + 225U^{n+2} - 72U^{n+1} + 10U^n = 60kf(U^{n+6})$$

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. These are shown in Figure 9.5.

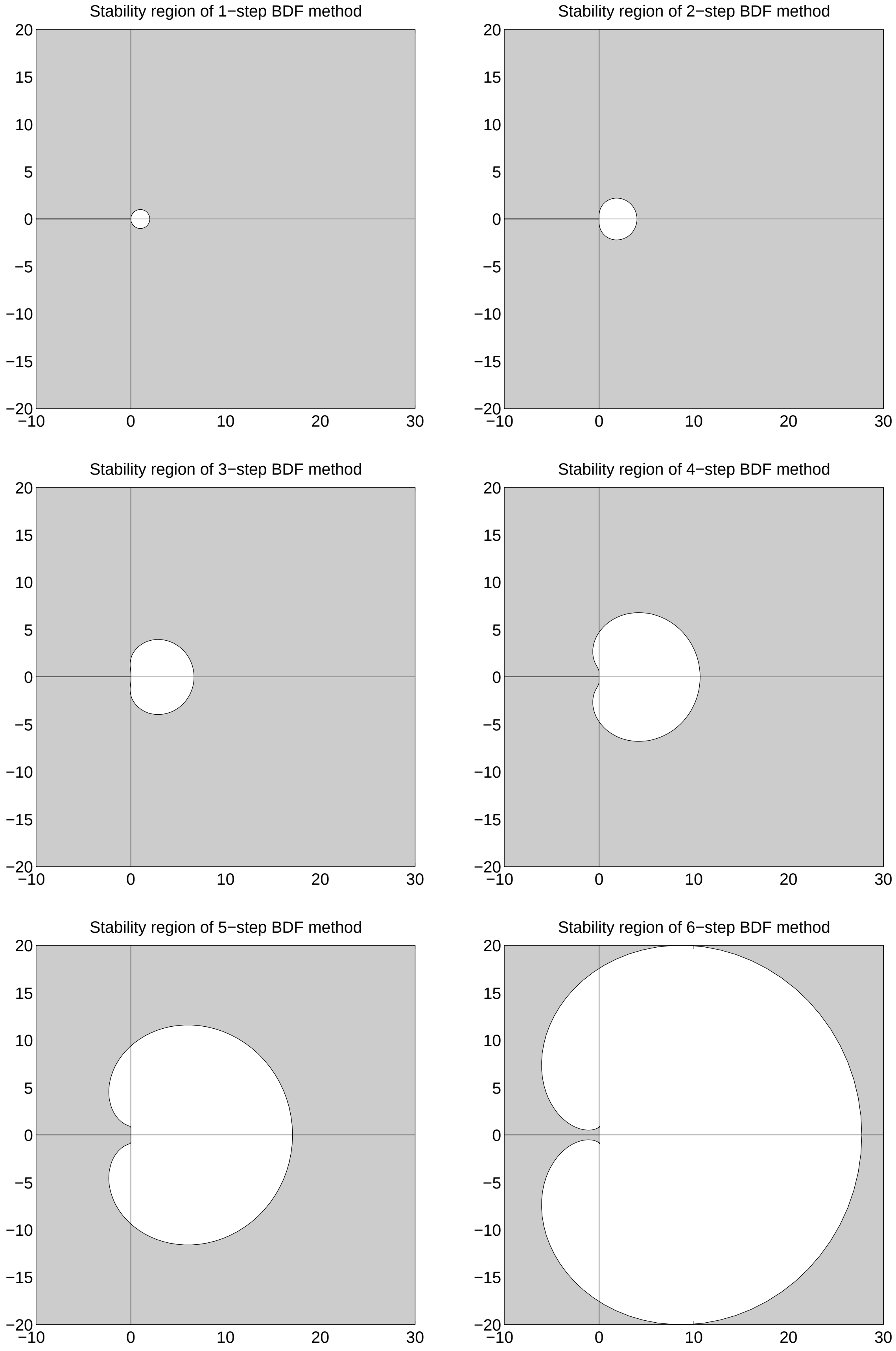
In particular, we need to make sure that the method is zero-stable. Otherwise it would not be convergent. This is not guaranteed from our derivation of the methods, since zero-stability depends only on the polynomial $\rho(\zeta)$, whose coefficients α_j are determined by considering the local truncation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

9.5 The TR-BDF2 method

There are often situations where it is useful to have a one-step method that is L-stable. The backward Euler method is one possibility, but 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 2-stage second-order accurate diagonally implicit method

$$\begin{aligned} U^* &= U^n + \frac{k}{4}(f(U^n) + f(U^*)) \\ U^{n+1} &= \frac{1}{3}(4U^* - U^n + kf(U^{n+1})). \end{aligned} \tag{9.4}$$

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 2-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 (6.29).


Stability region of 1−step BDF method

Stability region of 2−step BDF method

Stability region of 3−step BDF method

Stability region of 4−step BDF method

Stability region of 5−step BDF method

Stability region of 6−step BDF method

Figure 9.5: Stability regions for the BDF methods. The stability region is *exterior* to the curves.

Chapter 10

Some basic PDEs

In this chapter we will briefly develop some of the basic PDEs that will be used to illustrate the development of numerical methods. In solving a partial differential equation, we are looking for a function of more than one variable that satisfies some relations between different partial derivatives.

10.1 Classification of differential equations

First we review the classification of differential equations into elliptic, parabolic, and hyperbolic equations. Not all PDE's 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[Kev90] give further discussion.

10.1.1 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 anology with conic sections. The canonical example 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. There are other equations that 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.

10.1.2 Elliptic equations

The classic example of an elliptic equation is the **Poisson problem**

$$\nabla^2 u = f, \tag{10.1}$$

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 (10.1) 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. 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 (BVP's) 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 sytem 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

$$Lu = f, \tag{10.2}$$

where L is some **elliptic operator**. This notion will not be discussed further here, but the idea is that mathematical conditions are required on the differential operator L which insure that the boundary value problem has a unique solution.

10.1.3 Parabolic equations

If L is an elliptic operator then the time-dependent equation

$$u_t = Lu - f \tag{10.3}$$

is called **parabolic**. If $L = \nabla^2$ is the Laplacian, then (10.3) 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 (10.3), which results in the elliptic equation (10.2).

Marching to steady state by solving the time-dependent equation (10.3) numerically would be one approach to solving the elliptic equation (10.2), but this is typically not the fastest method if all we require is the steady state.

10.1.4 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

$$u_t + Au_x = 0 \tag{10.4}$$

where $u(x, t) \in \mathbb{R}^m$ and A is an $m \times m$ 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.

The simplest example of a hyperbolic equation is the constant-coefficient **advection equation**

$$u_t + au_x = 0, \tag{10.5}$$

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

$$\begin{bmatrix} p \\ u \end{bmatrix}_t + \begin{bmatrix} 0 & \kappa_0 \\ 1/\rho_0 & 0 \end{bmatrix} \begin{bmatrix} p \\ u \end{bmatrix}_x = 0 \tag{10.6}$$

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

$$c = \sqrt{\kappa_0/\rho}$$

is the speed of sound in the material.

10.2 Derivation of PDEs 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, [Kev90],[Whi74].

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:

$$\text{Total mass between } x_1 \text{ and } x_2 \text{ at time } t \quad = \int_{x_1}^{x_2} u(x,t)\,dx.$$

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.

10.3 Advection

If the substance is simply carried along (advected) in a flow at some constant velocity a, then the flux function is

$$f(u) = au. \tag{10.7}$$

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

$$\frac{d}{dt}\int_{x_1}^{x_2} u(x,t)\,dx = f(u(x_1,t)) - f(u(x_2,t)).$$

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

$$\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}f(u(x,t)) = 0. \tag{10.8}$$

This form of equation is called a *conservation law.* (For further discussion see [Lax72], [LeV90], [Whi74].)

For the case considered in Section 10.3, $f(u) = au$ with a constant and this equation becomes

$$u_t + au_x = 0. \tag{10.9}$$

This is called the *advection equation.*

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 only need to specify initial data

$$u(x,0) = \eta(x). \tag{10.10}$$

Physically, we would expect the initial profile of η to simply be carried along with the flow at speed a, so we should find

$$u(x,t) = \eta(x - at). \tag{10.11}$$

It is easy to verify that this function satisfyies the advection equation (10.9) 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 (10.9). Along these curves the PDE reduces to a simple ODE $U' = 0$ and the solution must be constant along each such curve, as is also seen from the solution (10.11).

10.4 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 water 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 water, as a drop of ink spreads. 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 1D) 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

$$f(u_x) = -\kappa u_x, \tag{10.12}$$

where κ is the *diffusion coefficient*. The relation (10.12) is known as *Fick's law*. Using this flux in (10.8) gives

$$u_t = \kappa u_{xx} \tag{10.13}$$

which is known as the *diffusion equation*. It is also called the *heat equation* since heat diffuses in much the same way. In this case we can think of the one-dimensional equation as modeling the conduction of heat in a rod. The heat conduction coefficient κ depends on the material and how well it conducts heat. The variable u is then the temperature and the relation (10.12) is known as *Fourier's law of heat condution.*

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 occuring simultaneously. Then the flux is $f(u, u_x) = au - \kappa u_x$, giving the *advection-diffusion* equation

$$u_t + au_x = \kappa u_{xx}. \tag{10.14}$$

The diffusion and advection-diffusion equations are examples of the general class of PDEs called *parabolic.*

10.5 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

$$u_t(x,t) + f(u(x,t))_x = \psi(x,t). \tag{10.15}$$

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

$$u_t(x,t) = \kappa u_{xx}(x,t) + \alpha(u_0 - u(x,t)).$$

10.5.1 Reaction-diffusion equations

One common form of source terms arises from chemical kinetics. If the components of $u \in \mathbb{R}^m$ represent concentrations of m different species reacting with one another, then the kinetics equations have the form $u_t = \psi(u)$, as described in Section 8.6.1. 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

$$u_t = \kappa u_{xx} + \psi(u). \tag{10.16}$$

The diffusion coefficient could be different for each species, in which case κ would be a diagonal matrix instead of a scalar. Advection terms might also be present if the reactions are taking place in a flowing fluid.

Chapter 11

Fourier Analysis of Linear PDEs

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 *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 finite difference operators. This will be exploited in Section 12.6 where von Neumann stability analysis of finite difference methods is discussed. An understanding of Fourier analysis of PDEs is also required in Section 13.6 where finite difference methods are discussed by derived and analyzed by studying "modified equations".

11.1 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

$$\hat{v}(\xi) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} v(x)e^{-i\xi x}\,dx. \tag{11.1}$$

The function $\hat{v}(\xi)$ is also in L^2 and in fact it has exactly the same 2-norm as v,

$$\|\hat{v}\|_2 = \|v\|_2. \tag{11.2}$$

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

$$v(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \hat{v}(\xi)e^{i\xi x}\,d\xi, \tag{11.3}$$

which is known as the *inverse Fourier transform.* This is analogous to writing a vector as a linear combination of basis vectors.

11.2 Solution of differential equations

The Fourier transform plays a fundamental role in the study of linear PDEs because of the fact that the functions $w(x) = e^{i\xi x}$ (for each fixed wave number ξ) are eigenfunctions of the differential operator ∂_x. In general applying ∂_x to a function gives a new function that is not simply a scalar multiple of the original function. For example, $\partial_x(x^3) = 3x^2$ and these functions are shaped quite differently. However, applying ∂_x to $w(x) = e^{i\xi x}$ gives $i\xi e^{i\xi x}$, which is simply the constant $i\xi$ times the original function $w(x)$. We say that $e^{i\xi x}$ is an *eigenfunction* of the operator ∂_x with *eigenvalue* $i\xi$.

Example 11.1. To see the importance of this fact, let's 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 $\hat{u}(\xi,t)$:

$$\hat{u}(\xi,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} u(x,t)e^{-i\xi x}\,dx. \tag{11.4}$$

Then

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{u}(\xi,t)e^{i\xi x}\,d\xi \tag{11.5}$$

and differentiating this with respect to t and x gives

$$\begin{aligned} u_t(x,t) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{u}_t(\xi,t)e^{i\xi x}\,dx \\ u_x(x,t) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{u}(\xi,t)i\xi e^{i\xi x}\,dx. \end{aligned}$$

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

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} (u_t + au_x)e^{-i\xi x}\,dx = 0$$

thus gives

$$\hat{u}_t(\xi,t) + ai\xi\hat{u}(\xi,t) = 0$$

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.

In fact 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

$$\hat{u}(\xi,t) = e^{-i\xi at}\hat{\eta}(\xi). \tag{11.6}$$

We can now Fourier transform back using (11.5) to get the desired solution $u(x,t)$:

$$\begin{aligned} u(x,t) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-i\xi at}\hat{\eta}(\xi)e^{i\xi x}\,d\xi \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi)e^{i\xi(x-at)}\,d\xi \\ &= \eta(x-at). \end{aligned}$$

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 (10.11) of the advection equation in this manner.

11.3 The heat equation

Now consider the heat equation,

$$u_t = \kappa u_{xx}. \tag{11.7}$$

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 the equation (11.7) gives the ODE

$$\hat{u}_t(\xi,t) = -\kappa\xi^2\hat{u}(\xi,t). \tag{11.8}$$

Again we have initial data $\hat{u}(\xi,0) = \hat{\eta}(\xi)$ from the given initial data on u. Now solving the ODE gives

$$\hat{u}(\xi,t) = e^{-\kappa\xi^2 t}\hat{\eta}(\xi). \tag{11.9}$$

Note that this has a very different character than (11.6), the Fourier transform otained 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, $|\hat{u}(\xi,t)|$ decays in time exponentially fast. The decay rate depends on κ, the diffusion coefficient, and also on ξ, the wavenumber. Highly oscillatory components (with ξ^2 large) decay much faster than those with low wavenumbers. This results in a *smoothing* of the solution as time evolves. (See Figure 12.3.)

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 ODE's in Chapter 9. In Section 12.4 we will see that this is indeed the case, and that implicit methods must generally be used in order to efficiently solve the heat equation.

11.4 Dispersive waves

Now consider the equation

$$u_t = u_{xxx}. \tag{11.10}$$

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

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi)e^{i\xi(x-\xi^2 t)}\,d\xi, \tag{11.11}$$

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 the equation (11.10), the shape of the initial data will in general 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 leads 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[Whi74] for an extensive discussion of dispersive waves.

11.5 Even vs. odd order derivatives

Note that odd order derivatives $\partial_x,\ \partial_x^3,\ \ldots$ (as in the advection equation or the dispersive equation (11.10)) 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.

Chapter 12

Diffusion Equations

We now begin to study finite difference methods for time-dependent partial differential equations, where variations in space are related to variations in time. We begin with the heat equation (or diffusion equation) introduced in Chapter 10,

$$u_t = \kappa u_{xx}. \tag{12.1}$$

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 (12.1) 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 $t_0 = 0$,

$$u(x,0) = \eta(x) \tag{12.2}$$

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

$$\begin{aligned} u(0,t) &= g_0(t) \quad \text{for } t > 0 \\ u(1,t) &= g_1(t) \quad \text{for } t > 0 \end{aligned} \tag{12.3}$$

if $0 \leq x \leq 1$.

We have already studied the steady state version of this equation and spatial discretizations of u_{xx} (Chapter 2). We have also studied discretizations of the time derivatives and some of the stability issues that arise with these discretizations in Chapters 6 through 9. 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).

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 (12.1) would be

$$\frac{U_i^{n+1} - U_i^n}{k} = \frac{1}{h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n). \tag{12.4}$$

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:

$$U_i^{n+1} = U_i^n + \frac{k}{h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n). \tag{12.5}$$

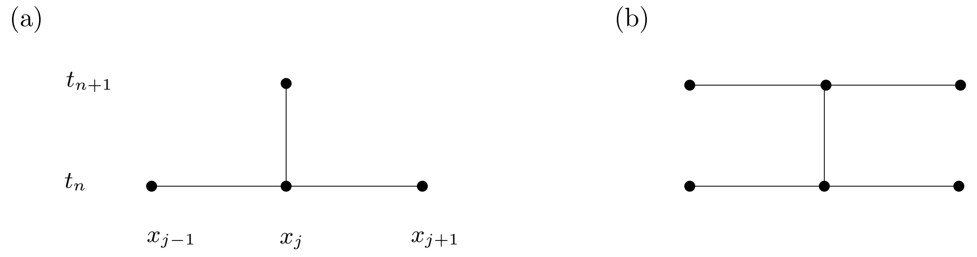


Figure 12.1: Stencils for the methods (12.5) and (12.7).

Figure 12.1(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 PDE's 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,

$$\begin{aligned}\frac{U_i^{n+1}-U_i^n}{k} &= \frac{1}{2}(D^2U_i^n + D^2U_i^{n+1}) \qquad (12.6)\\ &= \frac{1}{2h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n + U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}),\end{aligned}$$

which can be rewritten as

$$U_i^{n+1} = U_i^n + \frac{k}{2h^2}(U_{i-1}^n - 2U_i^n + U_{i+1}^n + U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}) \qquad (12.7)$$

or

$$-rU_{i-1}^{n+1} + (1+2r)U_i^{n+1} - rU_{i+1}^{n+1} = rU_{i-1}^n + (1-2r)U_i^n + rU_{i+1}^n \qquad (12.8)$$

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

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

$$= \begin{bmatrix} r(g_0(t_n) + g_0(t_{n+1})) + (1-2r)U_1^n + rU_2^n \\ rU_1^n + (1-2r)U_2^n + rU_3^n \\ rU_2^n + (1-2r)U_3^n + rU_4^n \\ \vdots \\ rU_{m-2}^n + (1-2r)U_{m-1}^n + rU_m^n \\ rU_{m-1}^n + (1-2r)U_m^n + r(g_1(t_n) + g_1(t_{n+1})) \end{bmatrix}. \qquad (12.9)$$

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 12.4 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 2-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 12.7.

Solving a parabolic equation with an implicit method requires solving a system of equations with the same structure as the 2-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.

12.1 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 12.1. The local truncation error of the method (12.5) is based on the form (12.4): $\tau_i^n = \tau(x_i, t_n)$, 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

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

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 (Exercise 12.1) 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 ODE's), 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 PDE's, the fact that consistency plus stability is equivalent to convergence is known as the *Lax Equivalence Theorem*, and is discussed in Section 12.5 after introducing 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.

12.2 Method of Lines discretizations

To understand how stability theory for time-dependent PDE's relates to the stability theory we have already developed for time-dependent ODE's, 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

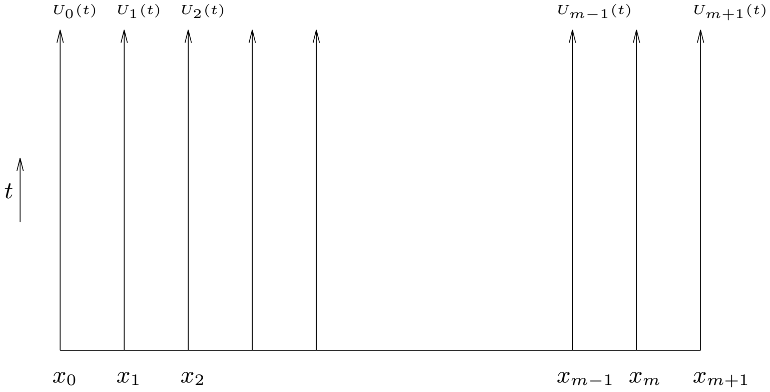


Figure 12.2: Method of lines interpretation. $U_i(t)$ is the solution along the line forward in time at the grid point x_i.

system of ODE's with each component of the system corresponding to the solution at some grid point, as a function of time. The system of ODE's can then be solved using one of the methods for ODE's that we have previously studied.

For example, we might discretize the heat equation (12.1) in space at grid point x_i by

$$U_i'(t) = \frac{1}{h^2}(U_{i-1}(t) - 2U_i(t) + U_{i+1}(t)), \quad \text{for } i = 1,\ 2,\ \ldots,\ m, \tag{12.10}$$

where prime now means differentiation with respect to time. We can view this as a coupled system of m ODE's for the variables $U_i(t)$, which vary continuously in time along the lines shown in Figure 12.2. This system can be written as

$$U'(t) = AU(t) + g(t) \tag{12.11}$$

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 $U_{m+1}(t) \equiv g_1(t)$,

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix}, \qquad g(t) = \frac{1}{h^2}\begin{bmatrix} g_0(t) \\ 0 \\ 0 \\ \vdots \\ 0 \\ g_1(t) \end{bmatrix}. \tag{12.12}$$

This MOL approach is sometimes used in practice by first discretizing in space and then applying a software package for systems of ODE's. 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 PDE's, but the resulting method is often not as efficient as specially designed methods for the PDE.

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 (12.11) 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 (12.11), 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 (12.5). Applying instead the trapezoidal method (6.16) results in the Crank-Nicolson method (12.7). Applying a higher order linear multistep or Runge-Kutta method would give a different method, though with the spatial discretization (12.10) the overall method would be only second order accurate in space. Replacing the right hand side of (12.10) by a higher-order approximation to $u_{xx}(x_i)$ and then using a higher order time discretization would give a more accurate method.

12.3 Stability theory

We can now investigate the stability of schemes like (12.5) or (12.7) since these can be interpreted as standard ODE methods applied to the linear system (12.11). 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 8.

We have determined the eigenvalues of A in (2.23),

$$\lambda_p = \frac{2}{h^2}(\cos(p\pi h) - 1), \quad \text{for } p = 1,\ 2,\ \ldots,\ m, \tag{12.13}$$

where again m and h are related by $h = 1/(m+1)$. Note that there is a new wrinkle here relative to the ODE's we considered in Chapter 8: 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 with, 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 8.

For the matrix (12.12) 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 12.2. If we use Euler's method to obtain the discretization (12.5), then we must require $|1 + k\lambda| \le 1$ for each eigenvalue (see Chapter 8) and hence we require $-2 \le -4k/h^2 \le 0$. This limits the time step allowed to

$$\frac{k}{h^2} \le \frac{1}{2}. \tag{12.14}$$

This is a severe restriction: the time step must decrease like h^2 as we refine the grid, which is much smaller than the spatial width h when h is small.

Example 12.3. If we use the trapezoidal method we obtain the Crank-Nicolson discretization (12.6). The trapezoidal method for the ODE is absolutely stable in the whole left half plane and the eigenvalues (12.13) 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.

12.4 Stiffness of the heat equation

Note that the system of ODE's 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 $4/\pi^2h^2$, which grows rapidly as $h \to 0$. As a result the explicit Euler method is stable only for very small time steps $k \le \frac{1}{2}h^2$. This is typically much smaller than what we would like to use over physically meaningful times, and an implicit 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 modelled 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 decays much more slowly since diffusion over long distances takes much longer. This is apparent from the Fourier analysis of Section 11.3 or is easily seen by writing down the exact solution to the heat equation on $0 \le x \le 1$ 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 ODE's

$$\hat{u}_j'(t) = -j^2\pi^2 \hat{u}_j(t), \qquad \text{for } j = 1,\ 2,\ ,\ldots \tag{12.15}$$

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 the equations (12.15) as an infinite system of ODE's, 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 12.3 for some examples of the time evolution with different sets of data.

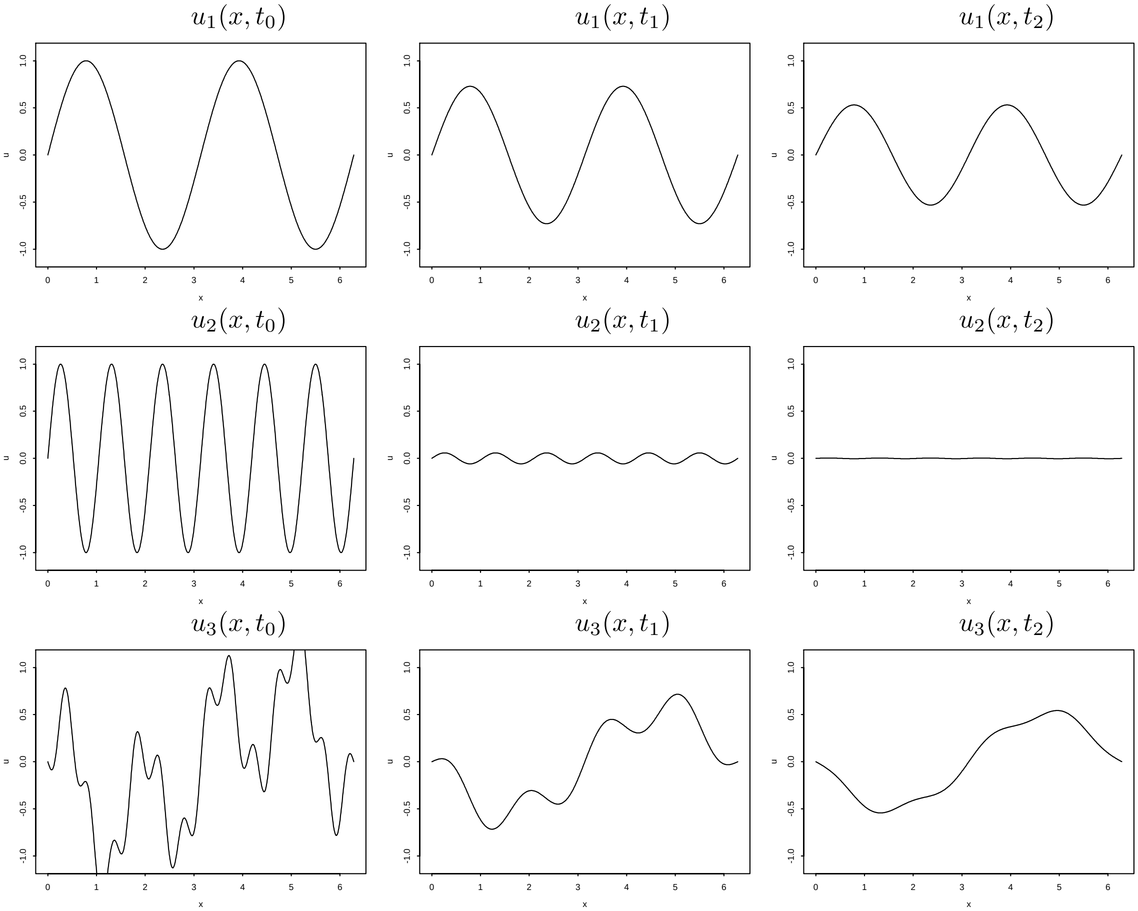
If we are solving the problem over the long time periods needed to track this slow diffusion, then we would ultimately (after any physical transients have decayed) like to use rather large time steps, since typically the variation in time is then on roughly the same scale as variations in space. We would 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 that 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 modelled by a single time step. This is the same difficulty we encountered with stiff ODE's in Chapter 9.

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 (12.5) 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 $k \approx h/\kappa$ 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 (12.11) 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 there are eigenvalues $-j^2\pi^2$ with arbitrarily large magnitude since j can be any integer. Luckily the discrete problem is not this stiff. The reason it is not is that, once we discretize in space, only a finite number of spatial wave numbers can be represented and we obtaine the finite set of eigenvalues (12.13). As we refine the grid we can represent higher and higher wave numbers, leading to the increasing stiffness ratio as $h \to 0$.



$u_1(x, t_0)$ $u_1(x, t_1)$ $u_1(x, t_2)$

$u_2(x, t_0)$ $u_2(x, t_1)$ $u_2(x, t_2)$

$u_3(x, t_0)$ $u_3(x, t_1)$ $u_3(x, t_2)$

Figure 12.3: 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.

12.5 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), $\ldots$, 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 (12.5), 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 (12.5)), 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

$$U^{n+1} = BU^n + b^n \tag{12.16}$$

for some matrix $B \in \mathbb{R}^{m\times m}$ on a grid with $h = 1/(m+1)$ and $b^n \in \mathbb{R}^m$. In the usual way, we can apply the difference equation to the exact solution $u(x,t)$ and obtain

$$u^{n+1} = Bu^n + b^n + k\tau^n \tag{12.17}$$

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 (12.17) from (12.16) gives the difference equation for the global error $E^n = U^n - u^n$:

$$E^{n+1} = BE^n - k\tau^n,$$

and hence, as usual,

$$E^n = B^n E^0 - k \sum_{m=1}^{n} B^{n-m}\tau^{m-1},$$

from which we obtain

$$\|E^n\| \leq \|B^n\| \|E^0\| + k \sum_{m=1}^{n} \|B^{n-m}\| \, \|\tau^{m-1}\|. \tag{12.18}$$

The method *converges* provided it is *consistent*, which requires that $\tau^{n-1} \to 0$ in each step, and *stable*, which now requires that $\|B^n\|$ be uniformly bounded for all k and n with $nk \leq T$. In the context of linear PDE's, the fact that consistency together with this form of stability gives convergence is known as the *Lax Equivalence Theorem.* A complete proof can be found in [RM67], but the essential inequality is (12.18). The form of stability required here, the uniform bound on $\|B^n\|$, is often called *Lax-Richtmyer stability* in the present context.

Recall that B depends on both k and h, but we are assuming some fixed relationship between these. For the methods we analyzed earlier in this Chapter, we found relations that would guarantee $\|B\|_2 \leq 1$ for each pair k, h, from which Lax-Richtmyer stability follows directly (in the 2-norm at least).

12.5.1 PDE vs. 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 ODE's 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 ODE's we were studying a fixed system of ODE's and dimension of the system and the fixed set of eigenvalues λ were 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 in order to prove convergence as $k \to 0$. Hence the need for zero-stabilty. With PDE's, on the other hand, in our MOL discretization the system of ODE's grows as we refine the grid, and the eigenvalues λ grow 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 want to keep k/h fixed as $k, h \to 0$, then $k\lambda \to -\infty$ and we must use an implicit method, and one that includes the entire negative real axis in its stability region.

Although for the methods considered so far we have obtained $\|B\| \leq 1$, this is not really necessary in order to have Lax-Richtmyer stability. If there is a constant α so that a bound of the form

$$\|B\| \leq 1 + \alpha k \tag{12.19}$$

holds in some norm (at least for all k sufficiently small), then we will have Lax-Richtmyer stability in this norm, since

$$\|B^n\| \leq (1 + \alpha k)^n \leq e^{\alpha T}$$

for $nk \leq T$. Since the matrix B depends on k and grows in size as $k \to 0$, the general theory of stability in the sense of uniform power boundedness of such families of matrices is often nontrivial. The *Kreiss Matrix Theorem* is one important tool in many practical problems. This is discussed in [RM67] along with some other techniques. See also [Str89] for a good discussion of stability. The recent review paper [LT98] gives an overview of how ODE and PDE stability theory are related, with a discussion of stability theory based on the "energy method", another important approach.

12.6 von Neumann analysis

Although it is useful to go through the MOL formulation in order to understand how stability theory for PDE's is related to the theory for ODE's, in practice there is another approach that will typically 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 PDE's. 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 $0 \leq x \leq 1$ with $u(0,t) = u(1,t)$ imposed. This is generally equivalent to a Cauchy problem with periodic initial data.

Stability theory for PDE's 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 [Str89], [RM67].

The Cauchy problem for linear PDE's can be solved using Fourier transforms — see Chapter 11 for a review. The basic reason this works is that the functions $e^{i\xi x}$ with wave number ξ = 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 standard 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 just a scalar multiple of V. But for the special case of W, we obtain

$$\begin{aligned} D_0 W_j &= \frac{1}{2h}\left(e^{i(j+1)h\xi} - e^{i(j-1)h\xi}\right) \\ &= \frac{1}{2h}\left(e^{ih\xi} - e^{-ih\xi}\right) e^{ijh\xi} \\ &= \frac{i}{h}\sin(h\xi) e^{ijh\xi} \\ &= \frac{i}{h}\sin(h\xi) W_j. \end{aligned} \tag{12.20}$$

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

$$\begin{aligned} \frac{i}{h}\sin(h\xi) &= \frac{i}{h}\left(h\xi - \frac{1}{6}h^3\xi^3 + O(h^5\xi^5)\right) \\ &= i\xi - \frac{i}{6}h^2\xi^3 + \cdots . \end{aligned}$$

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 analogs 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:

$$\|V\|_2 = \left(h \sum_{j=-\infty}^{\infty} |V_j|^2\right)^{1/2}, \qquad \|\hat{V}\|_2 = \left(\int_{-\pi/h}^{\pi/h} |\hat{V}(\xi)|^2\, d\xi\right)^{1/2}.$$

[1] Note: in this section $i = \sqrt{-1}$ and the index j is used on the grid functions.

In order 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 $\|B\|_2 \leq 1 + \alpha k$ in the notation of (12.19). This amounts to showing that there is a constant α such that

$$\|U^{n+1}\|_2 \leq (1+\alpha k)\|U^n\|_2$$

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 12.5, but this matrix is growing as we refine the grid.

Using Parseval's relation, we see that it is sufficient to instead show that

$$\|\hat{U}^{n+1}\|_2 \leq (1+\alpha k)\|\hat{U}^n\|_2$$

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 2-level method this has the form

$$\hat{U}^{n+1}(\xi) = g(\xi)\hat{U}^n(\xi). \tag{12.21}$$

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

$$|g(\xi)| \leq 1 + \alpha k$$

where α is independent of ξ, then it follows that the method is stable, since then

$$|\hat{U}^{n+1}(\xi)| \leq (1+\alpha k)|\hat{U}^n(\xi)| \quad \text{for all } \xi$$

and so

$$\|\hat{U}^{n+1}\|_2 \leq (1+\alpha k)\|\hat{U}^n\|_2.$$

Fourier analysis allows us to obtain simple scalar recursions of the form (12.21) 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 an 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 $h \to 0$.

Example 12.4. Consider the method (12.5). To apply von Neumann analysis we consider how this method works on a single wavenumber ξ, *i.e.*, we set

$$U_j^n = e^{ijh\xi}. \tag{12.22}$$

Then we expect that

$$U_j^{n+1} = g(\xi)e^{ijh\xi}, \tag{12.23}$$

where $g(\xi)$ is the amplification factor for this wavenumber. Inserting these expressions into (12.5) 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 $-1 \leq \cos(\xi h) \leq 1$ for any value of ξ, we see that

$$1 - 4\frac{k}{h^2} \leq g(\xi) \leq 1$$

for all ξ. We can guarantee that $|g(\xi)| \leq 1$ for all ξ if we require

$$4\frac{k}{h^2} \leq 2.$$

This is exactly the stability restriction (12.14) 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 wavenumbers that go unstable first as k is increased).

Example 12.5. The fact that the Crank-Nicolson method is stable for all k and h can also be shown using von Neumann analysis. Substituting (12.22) and (12.23) into the difference equations (12.7) and cancelling 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

$$g = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z} \tag{12.24}$$

where

$$\begin{aligned} z &= \frac{k}{h^2}(e^{-i\xi h} - 2 + e^{i\xi h}) \\ &= \frac{2k}{h^2}(\cos(\xi h) - 1). \end{aligned} \tag{12.25}$$

Since $z \leq 0$ for all ξ, we see that $|g| \leq 1$ and the method is stable for any choice of k and h.

Note that (12.24) agrees with the root ζ_1 found for the Trapezoidal method in Example 8.6, while the z determined in (12.25), for certain values of ξ, is simply k times an eigenvalue λ_p from (12.13), the eigenvalues of the Method of Lines matrix. So there is a close connection between the von Neumann approach and the MOL reduction to a system of ODE's.

12.7 Multi-dimensional problems

In two space dimensions the heat equation takes the form

$$u_t = u_{xx} + u_{yy} \tag{12.26}$$

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 five-point Laplacian from Section 3.2:

$$\nabla_h^2 U_{ij} = \frac{1}{h^2}(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{ij}). \tag{12.27}$$

If we then discretize in time using the trapezoidal method, we will obtain the two-dimensional version of the Crank-Nicolson method,

$$U_{ij}^{n+1} = U_{ij}^n + \frac{k}{2}[\nabla_h^2 U_{ij}^n + \nabla_h^2 U_{ij}^{n+1}]. \tag{12.28}$$

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 5. This matrix is large and sparse, and we generally do not want to solve the system by a direct method such as Gaussian Elimination. In fact 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 (12.28) as

$$\left(I - \frac{k}{2}\nabla_h^2\right) U_{ij}^{n+1} = \left(I + \frac{k}{2}\nabla_h^2\right) U_{ij}^n. \tag{12.29}$$

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 3.3. 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 3.3. 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 (12.28), 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 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.

12.8 The LOD method

Rather than solving the coupled sparse matrix equation for all the unknowns on the grid simultaneously as in (12.29), an alternative approach is to replace this fully-coupled single time step by 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:

$$U_{ij}^* = U_{ij}^n + \frac{k}{2}(D_x^2 U_{ij}^n + D_x^2 U_{ij}^*) \tag{12.30}$$

$$U_{ij}^{n+1} = U_{ij}^* + \frac{k}{2}(D_y^2 U_{ij}^* + D_y^2 U_{ij}^{n+1}). \tag{12.31}$$

or, in matrix form,

$$\left(I - \frac{k}{2}D_x^2\right)U^* = \left(I + \frac{k}{2}D_x^2\right)U^n \tag{12.32}$$

$$\left(I - \frac{k}{2}D_y^2\right)U^{n+1} = \left(I + \frac{k}{2}D_y^2\right)U^*. \tag{12.33}$$

In (12.30) 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 (12.31) 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.

Numerically there is a great advantage in using (12.32) and (12.33) rather than the fully coupled (12.29). In (12.32) 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 (12.29). 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, (12.31) 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.

12.8.1 Boundary conditions

In solving the second set of systems (12.31), we need boundary values U_{i0}^* and U_{i0}^{n+1} along the bottom boundary and $U_{i,m+1}^*$ and $U_{i,m+1}^{n+1}$ 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 equation (12.30) for $j = 0$ and $j = m+1$ (along the boundaries) in addition to the systems along each row interior to the grid.

In order to solve the first set of systems (12.30), we need boundary values U_{0j}^n and U_{0j}^* along the left boundary and values $U_{m+1,j}^n$ 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 half way 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 degredation of accuracy. The problem is that in the first step, equation (12.30) 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 the equations (12.31) 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_{0j}^{n+1}, which are already known from the boundary conditions at time t_{n+1}. Note that we are solving this equation backwards 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_{0j}^{n+1} from boundary data.

Since we are solving this equation backwards, we can view this as solving the diffusion equation $u_t = u_{yy}$ over a time step of length $-k$, backwards 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. However, since we are only doing this over one time step starting with given values U_{0j}^{n+1} in each time step, this turns out to be a stable procedure.

There is still a difficulty at the corners. In order to solve (12.31) 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 (12.31) for U_{0j}^*, we could simply use an explicit approximation to the backwards heat equation along this boundary,

$$U_{0j}^* = U_{0j}^{n+1} - \frac{k}{h^2}(U_{0,j-1}^{n+1} - 2U_{0j}^{n+1} + U_{0,j+1}^{n+1}), \tag{12.34}$$

for $j = 1,\ 2,\ \ldots,\ m$. This eliminates the need for values of U^* in the corners. Again, since this is not iterated but only done starting with given (and presumably smooth) boundary data U^{n+1} in each time step, this yields a stable procedure.

12.8.2 Accuracy and stability

With proper treatment of the boundary conditions, it can be shown that the LOD method is second order accurate. It can also be shown that this method, like full Crank-Nicolson, is unconditionally stable for any time step.

12.8.3 The ADI 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:

$$U_{ij}^* = U_{ij}^n + \frac{k}{2}(D_y^2 U_{ij}^n + D_x^2 U_{ij}^*) \tag{12.35}$$

$$U_{ij}^{n+1} = U_{ij}^* + \frac{k}{2}(D_x^2 U_{ij}^* + D_y^2 U_{ij}^{n+1}). \tag{12.36}$$

This is called the *Alternating Direction Implicit (ADI)* method and was first introduced by Douglas and Rachford [DR56]. This again gives decoupled tridiagonal systems to solve in each step:

$$\left(I - \frac{k}{2}D_x^2\right)U^* = \left(I + \frac{k}{2}D_y^2\right)U^n \tag{12.37}$$

$$\left(I - \frac{k}{2}D_y^2\right)U^{n+1} = \left(I + \frac{k}{2}D_x^2\right)U^*. \tag{12.38}$$

With this method, each of the two steps involves diffusion in both the x- and y-directions. In the first step the diffusion in x is modelled implicitly while diffusion in y is modelled 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 cancelled out by the second step.

12.9 Exercises

Exercise 12.1 *Compute the dominant term in the truncation error of Crank-Nicolson.*

Chapter 13

Advection Equations

In this chapter we consider numerical methods for the scalar advection equation

$$u_t + au_x = 0 \tag{13.1}$$

where a is a constant. See Section 10.3 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 is so simple that we can write down the exact solution,

$$u(x, t) = \eta(x - at). \tag{13.2}$$

One can verify directly that this is the solution (see also Chapter 11). However, many of the issues that arise more generally in discretizing hyperbolic equations can be most easily seen with this equation. Other hyperbolic systems will be discussed later.

The first approach we might consider is the analog of the method (12.4) for the heat equation. Using the centered difference in space and the forward difference in time results in

$$\frac{U_j^{n+1} - U_j^n}{k} = -\frac{a}{2h}(U_{j+1}^n - U_{j-1}^n), \tag{13.3}$$

which can be rewritten as

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n). \tag{13.4}$$

This again has the stencil shown in Figure 12.1(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 (13.4) by the average $\frac{1}{2}(U_{j-1}^n + U_{j+1}^n)$ then we obtain the *Lax-Friedrichs method*,

$$U_j^{n+1} = \frac{1}{2}(U_{j-1}^n + U_{j+1}^n) - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n). \tag{13.5}$$

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 (13.4) 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 12.5) and convergent provided

$$\left|\frac{ak}{h}\right| \leq 1. \tag{13.6}$$

Note that this stability restriction allows us to use a time step $k = O(h)$ even though 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 (13.6) 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 (13.6) 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 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 (13.6).

13.1 MOL discretization

To investigate stability further we will again introduce the method of lines (MOL) discretization as we did in Section 12.2 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 $0 \leq x \leq 1$, the advection equation can have a boundary condition specified only on one of the two boundaries. If $a > 0$ then we need a boundary condition at $x = 0$, say

$$u(0,t) = g_0(t), \tag{13.7}$$

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 later.

For analysis purposes we can obtain a nice MOL discretization if we consider the special case of *periodic boundary conditions*,

$$u(0,t) = u(1,t) \quad \text{for } t \geq 0.$$

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 only need to model a single period $0 \leq x \leq 1$.

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 $2 \leq j \leq m$ we have the 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:

$$\begin{aligned} U_1'(t) &= -\frac{a}{2h}(U_2(t) - U_{m+1}(t)), \\ U_{m+1}'(t) &= -\frac{a}{2h}(U_1(t) - U_m(t)). \end{aligned}$$

This system can be written as

$$U'(t) = AU(t) \tag{13.8}$$

with

$$A = -\frac{a}{2h}\begin{bmatrix} 0 & 1 & & & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ 1 & & & & -1 & 0 \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}. \tag{13.9}$$

Note that this matrix is skew-symmetric ($A^T = -A$) and so its eigenvalues must be pure imaginary. In fact, the eigenvalues are

$$\lambda_p = -\frac{ia}{h}\sin(2\pi ph), \quad \text{for } p = 1,\ 2,\ \ldots,\ m+1. \tag{13.10}$$

The corresponding eigenvector u^p has components

$$u_j^p = e^{2\pi ipjh}, \quad \text{for } j = 1,\ 2,\ \ldots,\ m+1. \tag{13.11}$$

The eigenvalues lie on the imaginary axis between $-ia/h$ and ia/h. Note how this relates to the eigenvalues and eigenfunctions of $-a\partial_x$.

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 be stable provided $|ak/h| \leq b$.

13.1.1 Forward Euler time discretization

The method (13.4) can be viewed as the forward Euler time discretization of the MOL system of ODE's (13.8). We found in Section 8.3 that this method is only stable if $|1 + k\lambda| \leq 1$ 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 (13.10) are imaginary, the values $k\lambda_p$ will not lie in $\mathcal{S}$. Hence the method (13.4) is *unstable* for any fixed mesh ratio k/h; see Figure 13.1(a).

The method (13.4) is 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 (12.19), $\|B\| \leq 1 + \alpha k$, rather than $\|B\| \leq 1$. Here $B = I + kA$. Suppose we take $k = h^2$, for example. Then we have

$$|1 + k\lambda_p|^2 \leq 1 + (ka/h)^2$$

for each p (using the fact that λ_p is pure imaginary) and so

$$|1 + k\lambda_p|^2 \leq 1 + a^2h^2 = 1 + a^2k.$$

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.

13.1.2 Leapfrog

A better time discretization is to use the midpoint method (6.17),

$$U^{n+1} = U^{n-1} + 2kAU^n,$$

which gives the *Leapfrog method* for the advection equation,

$$U_j^{n+1} = U_j^{n-1} - \frac{ak}{h}(U_{j+1}^n - U_{j-1}^n). \tag{13.12}$$

This is a 3-level explict method, and is second order accurate in both space and time.

Recall from Section 8.3 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 satsified.

13.1.3 Lax-Friedrichs

Now consider the Lax-Friedrichs method (13.5). Note that we can rewrite (13.5) 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

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{1}{2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n). \tag{13.13}$$

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 $\epsilon = h^2/2k$. Actually, we will see later that it is in fact a *second order* accurate method on a slightly different advection-diffusion equation. Viewing Lax-Friedrichs in this way allows us to investigate the diffusive nature of the method quite precisely.

For our present purposes, however, the crucial part is that we can now view (13.13) as resulting from a Forward Euler discretization of the system of ODE's

$$U'(t) = BU(t)$$

with

$$B = -\frac{a}{2h}\begin{bmatrix} 0 & 1 & & & & -1 \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ 1 & & & & -1 & 0 \end{bmatrix} + \frac{\epsilon}{h^2}\begin{bmatrix} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ 1 & & & & 1 & -2 \end{bmatrix} \tag{13.14}$$

where $\epsilon = h^2/2k$. The matrix B differs from the matrix A of (13.9) 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 B 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 (13.11) of the matrix A are also eigenvectors of the second difference operator (with periodic boundary conditions) that appears in (13.14), and hence these are also the eigenvectors of the full matrix B. We can easily compute that the eigenvalues of B are

$$\mu_p = -\frac{ia}{h}\sin(2\pi ph) - \frac{2\epsilon}{h^2}(1-\cos(2\pi ph)). \tag{13.15}$$

The values $k\mu_p$ are plotted in the complex plane for various different values of ϵ in Figure 13.1. 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 $|ak/h| \leq 1$. (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 $|ak/h| \leq 1$.

13.2 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 ODE's (13.8) 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 (13.12) already discussed. However, this is a 3-level method and for various reasons it is often much more convenient to use 2-level methods for PDE's 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 15) may require 2-level methods for each step, to name a few reasons. Moreover, the Leapfrog method is "nondissipative" in a sense that will be discussed in Section 13.6, 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 dicretize the system (13.8), 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 2-stage Runge-Kutta method such as the one in Example 6.11 for the time discretization. This can be done, though some care must be exercised near boundaries and the use of a multi-stage method again typically requires additional storage.

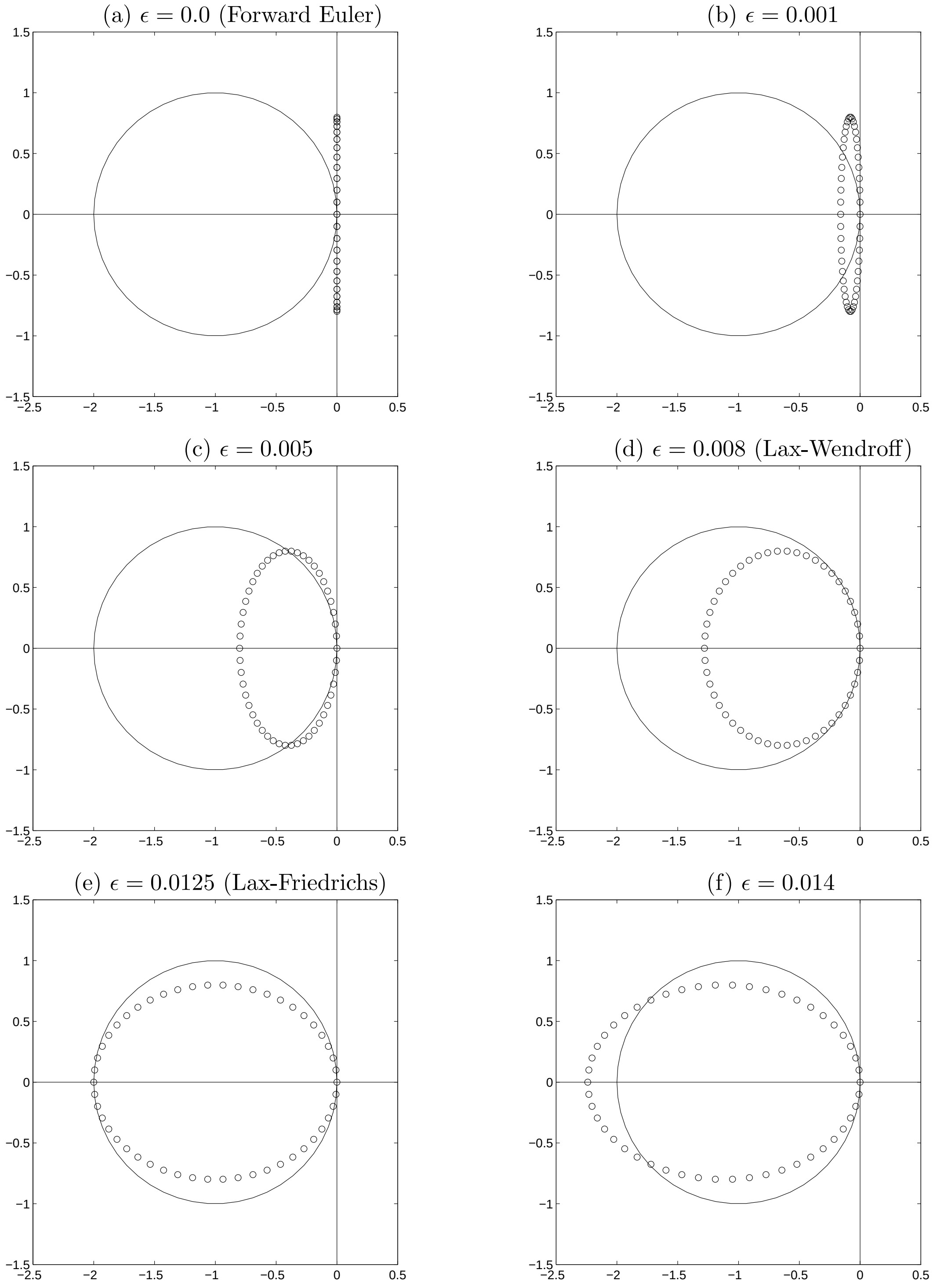
One simple way to achieve a 2-level explicit method with higher accuracy is to use the idea of Taylor series methods, as described in Section 6.5. Applying this directly to the linear system of ODE's $U'(t) = AU(t)$ (and using $U'' = AU' = A^2U$) gives the second order method

$$U^{n+1} = U^n + kAU^n + \frac{1}{2}k^2A^2U^n.$$

Here A is the matrix (13.9) and computing A^2 and writing the method at the typical grid point then gives

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{a^2k^2}{8h^2}(U_{j-2}^n - 2U_j^n + U_{j+2}^n). \tag{13.16}$$

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.


(a) $\epsilon = 0.0$ (Forward Euler)

(b) $\epsilon = 0.001$

(c) $\epsilon = 0.005$

(d) $\epsilon = 0.008$ (Lax-Wendroff)

(e) $\epsilon = 0.0125$ (Lax-Friedrichs)

(f) $\epsilon = 0.014$

Figure 13.1: Eigenvalues of the matrix B in (13.14), 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 the forward Euler method (13.4). (d) shows the case $\epsilon = a^2k/2$, the Lax-Wendroff method (13.17). (e) shows the case $\epsilon = h^2/2k$ the Lax-Friedrichs method (13.5). The method is stable for ϵ between $a^2k/2$ and $h^2/2k$, as in figures (d) through (e).

Note that the last term in (13.16) is an approximation to $\frac{1}{2}a^2k^2u_{xx}$ using a centered difference based on stepsize $2h$. 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*:

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{a^2k^2}{2h^2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n). \tag{13.17}$$

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 (13.17). It is also clear how we could obtain higher-order accurate explicit 2-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 PDE's. 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.

13.2.1 Stability analysis

We can analyze the stability of Lax-Wendroff following the same approach used for Lax-Friedrichs in Section 13.1. Note that with periodic boundary conditions, the Lax-Wendroff method (13.17) can be viewed as Euler's method applied to the linear system of ODE's $U'(t) = BU(t)$ where B is given by (13.14) with $\epsilon = a^2k/2$ (instead of the value $\epsilon = h^2/2k$ used in Lax-Friedrichs). The eigenvalues of B are given by (13.15) 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 semiaxes of length $(ak/h)^2$ and $|ak/h|$. If $|ak/h| \le 1$ then all of these values lie inside the stability region of Euler's method. Figure 13.1(d) shows an example in the case $ak/h = 0.8$ The Lax-Wendroff method is stable with exactly the same time step restriction (13.6) as required for Lax-Friedrichs. In Section 13.5 we will see that this is a very natural stability condition to expect for the advection equation, and the best we could hope for when a 3-point method is used.

A close look at Figure 13.1 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 13.1(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 13.1(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 13.1(c) would lead to instability, while more damping as in Figure 13.1(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.

13.2.2 Von Neumann analysis

The stability criterion for the Lax-Wendroff method can also be determined using von Neumann analysis as described in Section 12.6. Setting $\nu = ak/h$ and following the procedure of Example 12.5 for the Lax-Wendroff method (13.17) gives

$$\begin{aligned} g &= 1 - \frac{1}{2}i\nu(e^{i\xi h} - e^{-i\xi h}) + \frac{1}{2}\nu^2(e^{i\xi h} - 2 + e^{-i\xi h}) \\ &= 1 - i\nu\sin(\xi h) + \nu^2(\cos(\xi h) - 1) \\ &= 1 - i\nu[2\sin(\xi h/2)\cos(\xi h/2)] + \nu^2[2\sin^2(\xi h/2)] \end{aligned} \tag{13.18}$$

where we have used two trig identities to obtain the last line. This complex number has modulus

$$\begin{aligned} |g|^2 &= [1 - 2\nu^2\sin^2(\xi h/2)]^2 + 4\sin^2(\xi h/2)\cos^2(\xi h/2) \\ &= 1 - 4\nu^2(1-\nu^2)\sin^4(\xi h/2). \end{aligned} \tag{13.19}$$

Since $0 \le \sin^4(\xi h/2) \le 1$ for all values of ξ, we see that $|g|^2 \le 1$ for all ξ and hence the method is stable provided that $|\nu| \le 1$, which again gives the expected stability bound (13.6).

13.3 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.*,

$$u_x(x_j,t) \approx \frac{1}{h}(U_j - U_{j-1}) \tag{13.20}$$

or

$$u_x(x_j,t) \approx \frac{1}{h}(U_{j+1} - U_j). \tag{13.21}$$

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:

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n) \tag{13.22}$$

or

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_{j+1}^n - U_j^n). \tag{13.23}$$

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 due to the fact that 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. We will see that there are situations where it is best to acknowledge this asymmetry and use one-sided differences in the appropriate direction. In practice we often want to also use more accurate approximations, however, and we will see later how to extend these methods to higher order accuracy.

The choice between the two methods (13.22) and (13.23) 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 (13.22) might be a better choice for $a > 0$ and (13.23) for $a < 0$.

In fact the stability analysis below shows that (13.22) is stable only if

$$0 \le \frac{ak}{h} \le 1. \tag{13.24}$$

Since k and h are positive, we see that this method can only be used 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, (13.23) is stable only if

$$-1 \le \frac{ak}{h} \le 0, \tag{13.25}$$

and can only be used if $a < 0$. In this case (13.23) is the proper upwind method to use.

13.3.1 Stability analysis

The method (13.22) can be written as

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(U_{j+1}^n - U_{j-1}^n) + \frac{ak}{2h}(U_{j+1}^n - 2U_j^n + U_{j-1}^n), \tag{13.26}$$

which puts it in the form (13.14) 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 (13.24).

If we think of (13.26) 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 Chapter 11).

The method (13.23) can also be written in a form similar to (13.26), 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 (13.25).

The three methods Lax-Wendroff, upwind, and Lax-Friedrichs, can all be written in the same form (13.14) 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 13.1.

13.3.2 The Beam-Warming method

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

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(3U_j^n - 4U_{j-1}^n + U_{j-2}^n) + \frac{a^2k^2}{2h^2}(U_j^n - 2U_{j-1}^n + U_{j-2}^n). \tag{13.27}$$

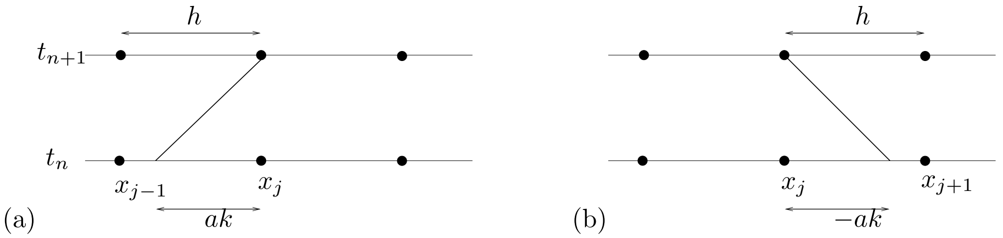


Figure 13.2: Tracing the characteristic of the advection equation back in time from the point (x_j, t_{n+1}) to compute the solution according to (13.29). Interpolating the value at this point from neighboring grid values gives the upwind method (for linear interpolation) or the Lax-Wendroff or Beam-Warming methods (quadratic interpolation). (a) shows the case $a > 0$, (b) shows the case $a < 0$.

For $a < 0$ the Beam-Warming method is one-sided in the other direction:

$$U_j^{n+1} = U_j^n - \frac{ak}{2h}(-3U_j^n + 4U_{j+1}^n - U_{j+2}^n) + \frac{a^2k^2}{2h^2}(U_j^n - 2U_{j+1}^n + U_{j+2}^n). \tag{13.28}$$

These methods are stable for $0 \leq \nu \leq 2$ and $-2 \leq \nu \leq 0$ respectively.

13.4 Characteristic tracing and interpolation

The solution to the advection equation is given by (13.2). The value of u is constant along each characteristic, which for this example are straight lines with constant slope. Over a single time step we have

$$u(x_j, t_{n+1}) = u(x_j - ak, t_n). \tag{13.29}$$

Tracing this characteristic back over time step k from the grid point x_j results in the picture shown in Figure 13.2(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

$$U_j^{n+1} = U_{j-1}^n. \tag{13.30}$$

Actually, all of the 2-level methods that we have considered so far reduce to the formula (13.30) in this special case $ak = h$, and each of these methods happens to be exact in this case.

If $ak/h < 1$ then the point $x_j - ak$ is not exactly at a grid point, as illustrated in Figure 13.2. However, we might attempt to use the relation (13.29) 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

$$p(x) = U_j^n + (x - x_j)\left(\frac{U_j^n - U_{j-1}^n}{h}\right). \tag{13.31}$$

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 (13.22). Note that this can also be interpreted as a linear combination of the two values U_{j-1}^n and U_j^n:

$$U_j^{n+1} = \left(1 - \frac{ak}{h}\right)U_j^n + \frac{ak}{h}U_{j-1}^n. \tag{13.32}$$

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 (13.24) is satisfied, which is also the condition required to insure 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_j^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 (13.17). Note that in this case we are properly interpolating provided that the stability restriction $|ak/h| \leq 1$ 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 (13.27), and we are properly interpolating provided $0 \leq ak/h \leq 2$.

13.5 The CFL Condition

The discussion of Section 13.4 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 $p \leq q$ (negative values are allowed for p and q), then we must have $x_{j+p} \leq x_j - ak \leq x_{j+q}$ or the method cannot be convergent. Since $x_i = ih$, this requires

$$-q \leq \frac{ak}{h} \leq -p.$$

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 essentially the first paper on the stability and convergence of finite difference methods for partial differential equations. (The original paper [CFL28] is in German but an English translation is available in [CFL67].) 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.*, [Kev90], [LeV02] 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 13.8 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

$$\mathcal{D}(X,T) = \{X - \lambda_p T \text{ for } p = 1,\ 2,\ \ldots,\ s\}.$$

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 is felt everywhere at any small time in the future (though its effect of course decays exponentially away from this point).

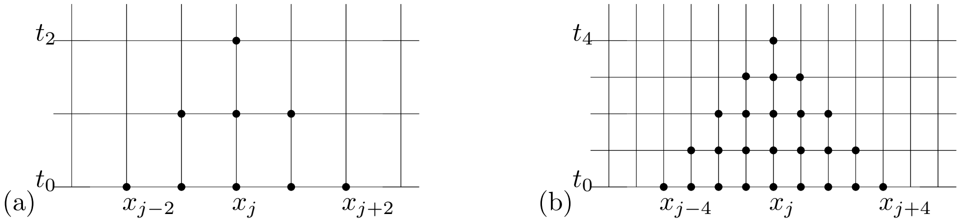


Figure 13.3: (a) Numerical domain of dependence of a grid point when using a 3-point explicit method. (b) On a finer grid.

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 (13.17) 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 13.3(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 13.3(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.

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 only be possible if

$$X - T/r \le X - aT \le X + T/r. \tag{13.33}$$

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 (13.33) 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 like within one grid point of x_j (recall the discussion of interpolation vs. extrapolation in Section 13.4).

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 13.1. The 3-point method (13.4) 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 13.2. The upwind methods (13.22) and (13.23) each have a 2-point stencil and the stability restrictions of these methods, (13.24) and (13.25) respectively, agree precisely with what the CFL condition requires.

Example 13.3. The Beam-Warming method (13.27) has a 3-point one-sided stencil. The CFL condition is satisfied if $0 \le ak/h \le 2$. When $a < 0$ the method (13.28) 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 (Exercise 13.4).

Example 13.4. 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 13.1.1 that the 3-point explicit method (12.5) 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 (12.7) 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).

13.6 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.

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 PDE's 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.

13.6.1 Upwind

This is best illustrated with an example. Consider the upwind method (13.22) for the advection equation $u_t + au_x = 0$ in the case $a > 0$,

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n). \tag{13.34}$$

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 (13.34) 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

$$v_t + av_x = \frac{1}{2}(ahv_{xx} - kv_{tt}) + \frac{1}{6}(ah^2v_{xxx} - k^2v_{ttt}) + \cdots .$$

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:

$$v_t + av_x = \frac{1}{2}(ahv_{xx} - kv_{tt}) \tag{13.35}$$

This involves second derivatives in both x and t, but we can derive a slightly different modified equation with the same accuracy by differentiating (13.35) 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

$$v_{tt} = a^2v_{xx} + O(k).$$

Inserting this in (13.35) gives

$$v_t + av_x = \frac{1}{2}(ahv_{xx} - a^2kv_{xx}) + O(k^2).$$

Since we have already decided to drop terms of $O(k^2)$, we can drop these terms here also to obtain

$$v_t + av_x = \frac{1}{2}ah\left(1 - \frac{ak}{h}\right)v_{xx}. \tag{13.36}$$

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 only give 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 13.4.

Note that the diffusion coefficient in (13.36) 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.

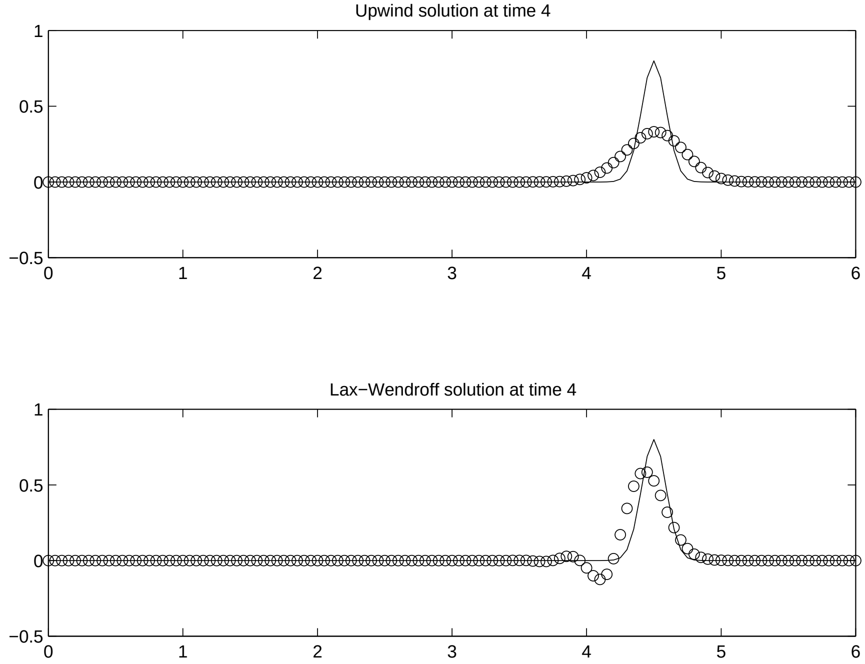


Figure 13.4: Numerical solution using upwind (diffusive) and Lax-Wendroff (dispersive) methods.

13.6.2 Lax-Wendroff

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

$$v_t + av_x + \frac{1}{6}ah^2\left(1 - \left(\frac{ak}{h}\right)^2\right)v_{xxx} = 0. \tag{13.37}$$

The Lax-Wendroff method produces a *third order* accurate solution to this equation. This equation has a very different character from (13.35). The v_{xxx} term leads to *dispersive* behavior rather than diffusion. This is clearly seen in Figure 13.4, 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*.

The group velocity for wave number ξ under Lax-Wendroff 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. (The concept of group velocity is explained in Section 13.7.) 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.

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

$$v_t + av_x + \frac{1}{6}ah^2\left(1 - \left(\frac{ak}{h}\right)^2\right)v_{xxx} = -\epsilon v_{xxxx}, \tag{13.38}$$

where the ϵ in the fourth order dissipative term is $O(h^3)$ and positive when the stability bound holds. This higher order dissipation causes the highest wave numbers to be damped, 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 disasterous, for example if the value of u represents a concentration that cannot go negative or exceed some limit without difficulties arising elsewhere in the modeling process.

13.6.3 Beam-Warming

The Beam-Warming method (13.27) has a similar modified equation,

$$v_t + av_x = \frac{1}{6}ah^2\left(2 - \frac{3ak}{h} + \left(\frac{ak}{h}\right)^2\right)v_{xxx}. \tag{13.39}$$

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.

13.7 Dispersive waves

This section contains a short introduction to the theory of dispersive waves, useful in understanding the behavior of many finite difference methods. See, e.g, [Kev90], [Str89], [Whi74] for further details.

Section 11.4 contains a brief discussion of Fourier analysis of the dispersive equation $u_t = u_{xxx}$. Extending that analysis to an equation of the form

$$u_t + au_x + bu_{xxx} = 0, \tag{13.40}$$

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.

13.7.1 The dispersion relation

Consider a more general PDE of the form

$$u_t + a_1u_x + a_3u_{xxx} + a_5u_{xxxxx} + \cdots = 0 \tag{13.41}$$

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

$$\omega = \omega(\xi) = a_1\xi - a_3\xi^3 + a_5\xi^5 - \cdots . \tag{13.42}$$

The solution can thus be written as

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi) e^{i(\xi x - \omega(\xi)t)}\, d\xi. \tag{13.43}$$

The relation (13.42) 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

$$u(x,t) = e^{-i\omega t} e^{i\xi x} \tag{13.44}$$

into the PDE and cancelling the common terms in order to find the relation between ω and ξ. This is similar to what we found when applying von Neumann analysis in Section 12.6. 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 also 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 11.3, 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 (13.41) the single Fourier mode (13.44) 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,

$$c_p(\xi) = \frac{\omega(\xi)}{\xi}. \tag{13.45}$$

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,

$$\eta(x) = e^{-\beta x^2} \tag{13.46}$$

The Fourier transform of this function is a Gaussian in ξ,

$$\hat{\eta}(\xi) = \frac{1}{\sqrt{2\beta}} e^{-\xi^2/4\beta}. \tag{13.47}$$

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. Note the nice duality here, since $\eta(x)$ can also be viewed as essentially (up to the sign in the exponent) the Fourier transform of $\hat{\eta}(\xi)$.

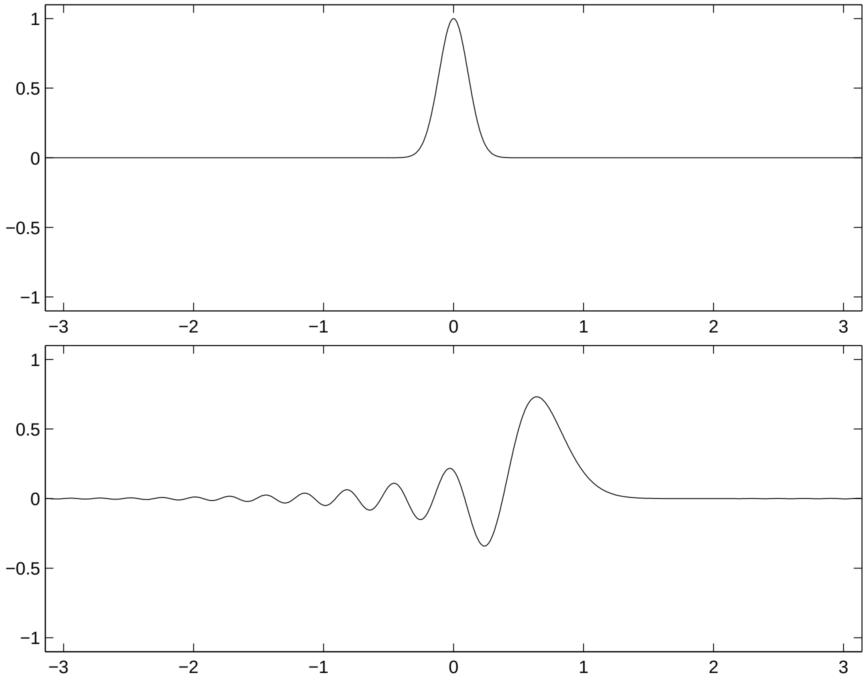


Figure 13.5: Gaussian initial data propagating with dispersion.

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 13.5, 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*

$$c_g(\xi) = \frac{d\omega(\xi)}{d\xi}. \tag{13.48}$$

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 (13.40), $\omega(\xi) = a\xi - b\xi^3$ and we find that

$$c_g(\xi) = a - 3b\xi^2$$

whereas

$$c_p(\xi) = a - b\xi^2.$$

13.7.2 Wave packets

The notion and importance of group velocity is easiest to appreciate by considering a "wave packet" with data of the form

$$\eta(x) = e^{i\xi_0 x}e^{-\beta x^2} \tag{13.49}$$

or the real part of such a wave,

$$\eta(x) = \cos(\xi_0 x)e^{-\beta x^2}. \tag{13.50}$$

This is a single Fourier mode modulated by a Gaussian.

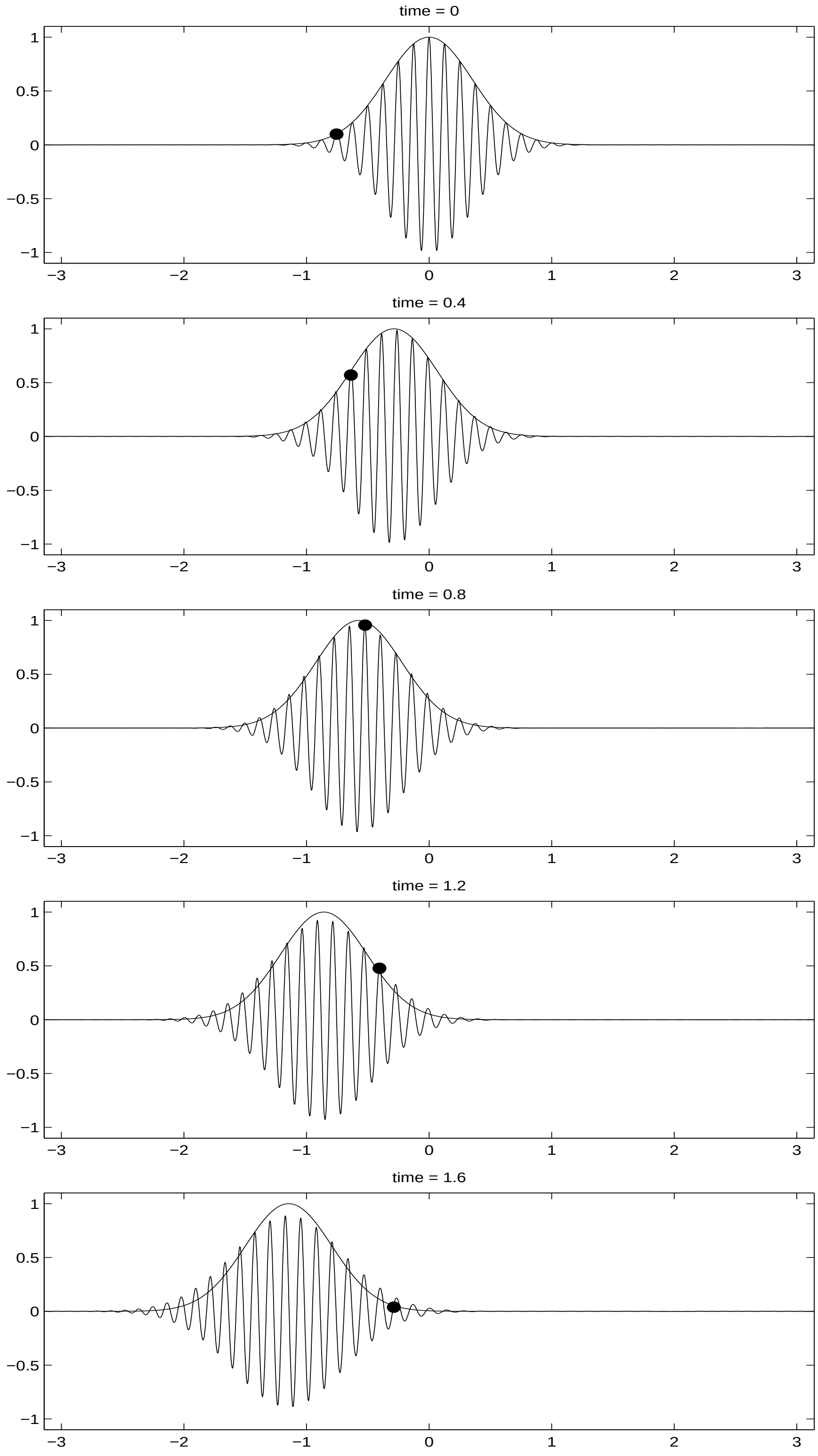


Figure 13.6: The oscillatory wave packet satisfies the dispersive equation $u_t + au_x + bu_{xxx} = 0$. Also shown is a black dot, translating at the phase velocity $c_p(\xi_0)$ and a Gaussian that is translating at the group velocity $c_g(\xi_0)$.

The Fourier transform of (13.49) is

$$\hat{\eta}(\xi) = \frac{1}{\sqrt{2\beta}} e^{-(\xi-\xi_0)^2/4\beta}, \tag{13.51}$$

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 evelope 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 (13.43),

$$u(x,t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{\eta}(\xi) e^{i(\xi x - \omega(\xi)t)}\, d\xi.$$

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 that the oscillatory part is oscillating rapidly enough in ξ that it averages out to zero, even though 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.

13.8 Hyperbolic systems

The advection equation $u_t + au_x = 0$ can be generalized to a first order linear system of equations of the form

$$\begin{aligned} &u_t + Au_x = 0 \\ &u(x,0) = u_0(x) \end{aligned} \tag{13.52}$$

where $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times m}$ is a constant matrix. This is a system of conservation laws with the flux function $f(u) = Au$. This system is called **hyperbolic** if A is diagonalizable with real eigenvalues, so that we can decompose

$$A = R\Lambda R^{-1} \tag{13.53}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$ is a diagonal matrix of eigenvalues and $R = [r_1|r_2|\cdots|r_m]$ is the matrix of right eigenvectors. Note that $AR = R\Lambda$, i.e.,

$$Ar_p = \lambda_p r_p \quad \text{for } p = 1,\ 2,\ \ldots,\ m. \tag{13.54}$$

The system is called **strictly hyperbolic** if the eigenvalues are distinct.

13.8.1 Characteristic variables

We can solve (13.52) by changing to the "characteristic variables"

$$w = R^{-1}u, \tag{13.55}$$

in much the same way we solved linear systems of ODEs in Section 8.6.2. Multiplying (13.52) by R^{-1} and using (13.53) gives

$$R^{-1}u_t + \Lambda R^{-1}u_x = 0 \tag{13.56}$$

or, since R^{-1} is constant,

$$v_t + \Lambda v_x = 0. \tag{13.57}$$

Since Λ is diagonal, this decouples into m independent scalar equations

$$(w_p)_t + \lambda_p(w_p)_x = 0, \qquad p = 1,\ 2,\ \ldots,\ m. \tag{13.58}$$

Each of these is a constant coefficient linear advection equation, with solution

$$w_p(x,t) = w_p(x - \lambda_p t, 0). \tag{13.59}$$

Since $w = R^{-1}u$, the initial data for w_p is simply the pth component of the vector

$$w(x,0) = R^{-1}u_0(x). \tag{13.60}$$

The solution to the original system is finally recovered via (13.55):

$$u(x,t) = Rw(x,t). \tag{13.61}$$

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., (13.61) can be written out as

$$u(x,t) = \sum_{p=1}^{m} w_p(x,t)r_p. \tag{13.62}$$

Combining this with the solutions (13.59) of the decoupled scalar equations gives

$$u(x,t) = \sum_{p=1}^{m} w_p(x - \lambda_p t, 0)r_p. \tag{13.63}$$

Note that $u(x,t)$ depends only on the initial data at the m points $x - \lambda_p t$. This set of points is called the *Domain of Dependence* of the point (x,t) and will be denoted by $\mathcal{D}(x,t)$.

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, m 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 (13.62) of $u(x,t)$ is constant along any p-characteristic.

13.9 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 by A in the formulas. For example, the Lax-Wendroff method becomes

$$U_j^{n+1} = U_j^n - \frac{k}{2h}A(U_{j+1}^n - U_{j-1}^n) + \frac{k^2}{2h^2}A^2(U_{j-1}^n - 2U_j^n + U_{j+1}^n). \tag{13.64}$$

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

$$\nu = \max_{1\leq p\leq m} |\lambda_p k/h|. \tag{13.65}$$

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 $\nu \leq 1$.

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 (13.22) and (13.23) generalize naturally to

$$U_j^{n+1} = U_j^n - \frac{k}{h} A(U_j^n - U_{j-1}^n) \tag{13.66}$$

and

$$U_j^{n+1} = U_j^n - \frac{k}{h} A(U_{j+1}^n - U_j^n). \tag{13.67}$$

For a system of equations, however, neither one of these is useful unless all of the eigenvalues of A have the same sign, so that the upwind direction is the same for all characteristic variables. The method (13.66) is stable only if

$$0 \leq \frac{k\lambda_p}{h} \leq 1 \qquad \text{for all } p = 1,\ 2,\ ,\ldots,\ m, \tag{13.68}$$

while (13.67) is stable only if

$$-1 \leq \frac{k\lambda_p}{h} \leq 0 \qquad \text{for all } p = 1,\ 2,\ ,\ldots,\ m. \tag{13.69}$$

It possible to generalize the upwind method to more general systems with eigenvalues of both sign, 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 the book [LeV02].

13.10 Exercises

Exercise 13.1 *Compute the local truncation error of the Lax-Friedrichs method (13.5) and show that it is first order accurate in both space and time.*

Exercise 13.2 *Verify the expression (13.15) for the eigenvalues of* B*.*

Exercise 13.3 *Compute the local truncation error for Beam-Warming.*

Exercise 13.4 *Show that Beam-Warming is stable for* $0 \leq ak/h \leq 2$*.*

Exercise 13.5 *Verify that Lax-Wendroff and Beam-Warming can be obtained by tracing back the characteristic and using quadratic interpolation as described in Section 13.4*

Exercise 13.6 *View (13.34) as a numerical method for the equation (13.35). Compute the local truncation error and verify that it is* $O(k^2)$*.*

Exercise 13.7 *Determine the modified equation for (13.3) and show that the diffusion coefficient is always negative.*

Exercise 13.8 *Consider the following implicit upwind method for the advection equation $u_t + au_x = 0$ on $0 \le x \le 1$ with boundary conditions $u(0,t) = g_0(t)$:*

$$\begin{aligned} U_0^{n+1} &= g_0(t_{n+1}) \\ U_i^{n+1} &= U_i^n - \nu(U_{i+1}^{n+1} - U_i^{n+1}), \qquad i = 0,\ 1,\ \ldots,\ m \end{aligned} \tag{13.70}$$

where $h = 1/(m+1)$ and $\nu = ak/h$. Note that this can be rewritten as

$$\begin{aligned} U_0^{n+1} &= g_0(t_{n+1}) \\ U_i^{n+1} &= \frac{1}{\nu}(U_i^n - (1-\nu)U_i^{n+1}), \qquad i = 0,\ 1,\ \ldots,\ m \end{aligned} \tag{13.71}$$

so that one can explicitly solve the implicit equation and simply sweep from left to right. This method has the interesting property that it is only stable if the Courant number ν is sufficiently large.

1. *If ν is fixed as the grid is refined, show that the CFL condition is satisfied only if $\nu \ge 1$.*
2. *Use von Neumann analysis for the Cauchy problem to show that this method is stable only if $\nu \ge 1$.*

Chapter 14

Higher-Order Methods

Higher order methods are often most easily derived by a method of lines approach — use a higher-order spatial discretization to obtain a semi-discrete system of ODEs, and then apply a suitably accurate time discretization method to this system.

14.1 Higher-order centered differences

We first consider the problem of obtaining more accurate approximations to spatial derivatives. Given data U_j for $j = 1,\ 2,\ \ldots,\ m$, we would like to compute corresponding values $W_j \approx u_x(x_j)$. So far we have worked with one-sided and centered difference operators, *e.g.*,

$$\begin{aligned} W_j &= (U_j - U_{j-1})/h = u_x + O(h), && (14.1)\\ W_j &= (U_{j+1} - U_{j-1})/(2h) = u_x + O(h^2). && (14.2) \end{aligned}$$

These can be derived by determining an interpolating polynomial $p_j(x)$ and setting $W_j = p_j'(x_j)$. For (14.1), we use a linear interpolant through U_{j-1} and U_j while for (14.2) we use a quadratic interpolant through U_{j-1}, U_j and U_{j+1}. This idea can be extended to obtain higher order approximations. For example, interpolating with a polynomial of degree 4 through $U_{j-2},\ U_{j-1},\ U_j,\ U_{j+1},\ U_{j+2}$ results in the fourth order formula

$$W_j = \frac{4}{3}\left(\frac{U_{j+1} - U_{j-1}}{2h}\right) - \frac{1}{3}\left(\frac{U_{j+2} - U_{j-2}}{4h}\right). \tag{14.3}$$

Similarly, a sixth-order accurate formula is

$$W_j = \frac{3}{2}\left(\frac{U_{j+1} - U_{j-1}}{2h}\right) - \frac{3}{5}\left(\frac{U_{j+2} - U_{j-2}}{4h}\right) + \frac{1}{10}\left(\frac{U_{j+3} - U_{j-3}}{6h}\right). \tag{14.4}$$

Order of accuracy is not the entire story, however, since this gives information primarily on how good the approximation is in the limit $h \to 0$ when the solution is very well resolved. Often we must compute on grids where the solution is not very well resolved, particularly in more than one space dimension. In an advection problem, for example, there may be high wavenumber components of the solution that are not well resolved but that we hope will propagate at the correct velocity.

Recall that on a grid with spacing h we can resolve wavenumbers ξ for which $|h\xi| \leq \pi$. (The highest wavenumber produces a sawtooth wave on this grid.) Order of accuracy tells us how well the method works in the $h\xi \to 0$ limit, but to get a feel for how the method handles wavenumbers that are not so well resolved, it is often more useful to look at the dispersion relation over the full range $0 \leq \xi h \leq \pi$.

Note that this is philosophically similar to the way we earlier studied stability of ODE methods. For convergence all one needs is zero-stability (good behavior as $k\lambda \to 0$), but recognizing that we must

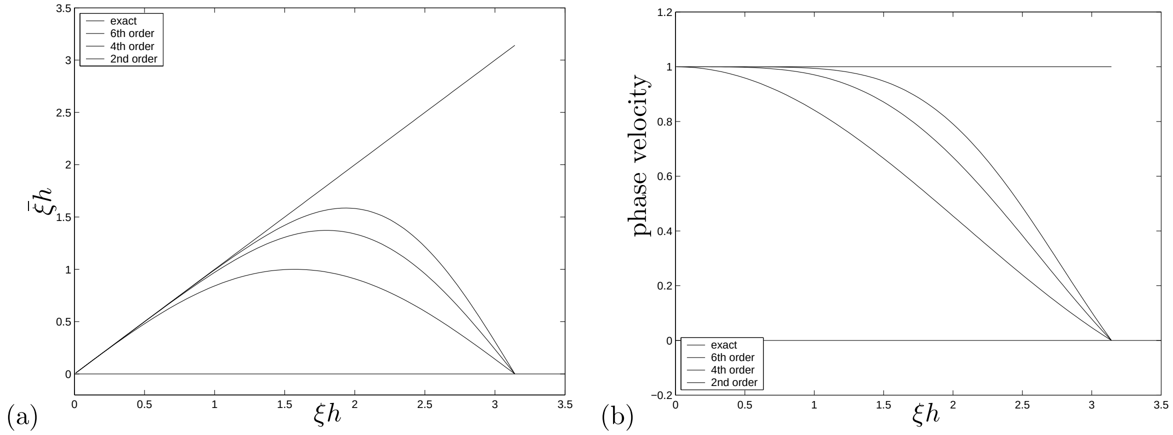


Figure 14.1: (a) Dispersion relation $\bar{\xi}h$ vs. ξh for explicit centered schemes. (b) Phase velocity c_p from (14.10) plotted as a function of ξh for the same schemes.

often compute on grids far from this limit led us to also consider absolute stability and use the stability region as a guide to selecting methods for particular problems.

To study the dispersive properties of a spatial discretization method (without discussion of the time stepping yet), it is convenient to consider how the formula works when applied to a single Fourier mode $U_j = e^{i\xi x_j}$. Applying the true differential operator ∂_x to Fourier mode $e^{i\xi x}$ results in $i\xi e^{i\xi x}$. Applying a linear difference operator to the grid function $e^{i\xi x_j}$ generally results in an approximation to the derivative of the form

$$W_j = i\bar{\xi}e^{i\xi x_j}, \tag{14.5}$$

where $\bar{\xi}$ is a *modified wave number* depending on the particular discretization. (The factor $i\bar{\xi}$ will be pure imaginary only for centered even-order approximations to the first derivative, which we consider here. Uncentered approximations such as (14.1) would be dissipative and this factor would have negative real part.)

All of the centered approximations listed above are of the form

$$W_j = a\left(\frac{U_{j+1} - U_{j-1}}{2h}\right) + b\left(\frac{U_{j+2} - U_{j-2}}{4h}\right) + c\left(\frac{U_{j+3} - U_{j-3}}{6h}\right) \tag{14.6}$$

for which we find that

$$\bar{\xi} = a\frac{\sin(\xi h)}{h} + b\frac{\sin(2\xi h)}{2h} + c\frac{\sin(3\xi h)}{3h}. \tag{14.7}$$

Plotting $\bar{\xi}h$ vs. ξh gives an indication of how well the difference operator works for each wavenumber. These are shown in Figure 14.1(a) for the 2nd, 4th, and 6th order methods above. The straight line $\bar{\xi}h = \xi h$ is the ideal behavior. Each curve lies along this line for small ξh, with increasing order of tangency, but all begin to deviate significantly at some point and drop to zero at $\xi h = \pi$. (Note that these centered approximations to the derivative all produce $W_j \equiv 0$ when applied to a sawtooth wave.)

Now consider the semi-discretized method for the advection equation $u_t + u_x = 0$,

$$U_j'(t) = -W_j(t), \tag{14.8}$$

using some approximation $W_j(t) \approx u_x(x_j, t)$ of the form considered above. We can derive a dispersion relation for this semi-discrete method. Taking initial data to be a single Fourier mode, $U_j(0) = e^{i\xi x_j}$, it is easy to verify that the solution to (14.8) will have the form

$$U_j(t) = e^{i(\xi x_j - \omega t)}, \tag{14.9}$$

where $\omega(\xi) = \bar{\xi}$, *i.e.*, the modified wavenumber discussed above gives directly the dispersion relation for the semi-discrete method. From this we can calculate the phase velocity for a pure mode of wavenumber ξ,

$$c_p(\xi) = \frac{\omega(\xi)}{\xi} = \frac{\bar{\xi}}{\xi} = \frac{\bar{\xi}h}{\xi h}. \tag{14.10}$$

Plotting this as a function of ξh gives the curves shown in Figure 14.1(b). Now the ideal behavior is the horizontal line $c_p \equiv 1$.

14.2 Compact schemes

With a *compact scheme*, we determine the values $W_j \approx u_x(x_j)$ by an implicit formula in which several values of W_j are coupled together, rather than by an explicit formula of the form (14.6). One general class of compact schemes have the form

$$\alpha W_{j-1} + W_j + \alpha W_{j+1} = a\left(\frac{U_{j+1} - U_{j-1}}{2h}\right) + b\left(\frac{U_{j+2} - U_{j-2}}{4h}\right) + c\left(\frac{U_{j+3} - U_{j-3}}{6h}\right). \tag{14.11}$$

If $\alpha = 0$ then this is just (14.6). If $\alpha \neq 0$ then we must solve a tridiagonal system of equations to determine the values W_j from the data U_j. This is more complicated but only slightly more work. The payoff is that compact schemes can have much better accuracy and dispersion characteristics than explicit methods.

If $c = 0$ then (14.11) simplifies but it can be shown that there is a one-parameter family of methods of this form that are fourth order accurate. For any choice of α we can set

$$a = \frac{2}{3}(\alpha + 2), \qquad b = \frac{1}{3}(4\alpha - 1), \qquad c = 0, \tag{14.12}$$

and the resulting method will be fourth order. The choice $\alpha = 0$ gives the explicit method (14.3). Choosing $\alpha = 1/4$ gives $a = 3/2$ and $b = 0$, in which case (14.11) reduces to simply

$$\frac{1}{4}W_{j-1} + W_j + \frac{1}{4}W_{j+1} = \frac{3}{2}\left(\frac{U_{j+1} - U_{j-1}}{2h}\right). \tag{14.13}$$

The right-hand side now only involves data at the two points U_{j-1} and U_{j+1} and yet the method is fourth-order accurate, hence the term "compact", though of course in reality all values of U come into determining each W_j since the inverse of a tridiagonal matrix is dense.

If we choose $\alpha = 1/3$, then $a = 14/9$ and $b = 1/9$ (and $c = 0$), and it turns out that the method is formally sixth-order accurate. This is a special case of a one-parameter family of sixth-order methods obtained by allowing c to be nonzero in (14.11) in general, and for any given α setting

$$a = \frac{1}{6}(\alpha + 9), \qquad b = \frac{1}{15}(32\alpha - 9), \qquad c = \frac{1}{10}(-3\alpha + 1). \tag{14.14}$$

When $\alpha = 1/3$, $c = 0$ and we obtain the method mentioned above. Setting $\alpha = 0$ gives the explicit sixth-order method (14.4). The special choice $\alpha = 3/8$ gives an eighth-order method. Other eighth-order methods and even a tenth-order method can be obtained by using a wider stencil on the left hand side of (14.11). See Lele [Lel92] for additional methods and more discussion of the topics covered in this section.

The dispersion relation for the method (14.11) is easily determined. If $U_j = e^{i\xi x_j}$ then $W_j = i\bar{\xi}e^{i\xi x_j}$ and using these in (14.11) gives

$$\left(\alpha(e^{-i\xi h} + e^{i\xi h}) + 1\right)\bar{\xi} = a\frac{\sin(\xi h)}{h} + b\frac{\sin(2\xi h)}{2h} + c\frac{\sin(3\xi h)}{3h} \tag{14.15}$$

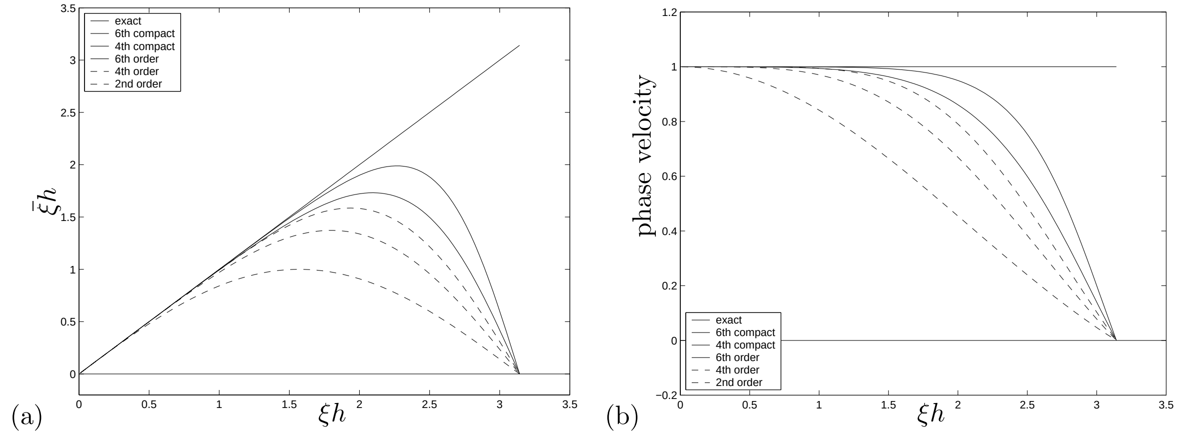


Figure 14.2: (a) Dispersion relation $\bar{\xi}h$ vs. ξh for two compact schemes, the fourth-order scheme with $\alpha = 1/4$ and the sixth-order scheme with $\alpha = 1/3$. (b) Phase velocity c_p from (14.10) plotted as a function of ξh for the same schemes. The curves from Figure 14.1 are plotted as dashed lines.

and hence

$$\bar{\xi} = \frac{(a/h)\sin(\xi h) + (b/2h)\sin(2\xi h) + (c/3h)\sin(3\xi h)}{1 + 2\alpha\cos(\xi h)}. \tag{14.16}$$

We can produce plots analogous to those shown in Figure 14.1 for any of these compact schemes, and a few are shown in Figure 14.2. The dispersion relations for the explicit schemes from Figure 14.1 are also shown in these plots, as dashed lines. Note that the compact schemes are better for larger ξh. In fact the fourth-order compact scheme outperforms the sixth-order explicit scheme at resolving high wavenumbers.

The use of a compact scheme requires some additional formulas near the boundaries if the given boundary conditions are not periodic. One-sided compact schemes that can be used in conjunction with the above methods are discussed in [Lel92].

14.3 Spectral methods

The explicit methods introduced at the beginning of this chapter can be derived by fitting a polynomial $p_j(x)$ through a few data points centered about x_j and then evaluating $p_j'(x_j)$. The more points used, the better is the order of accuracy. This suggests that we might do best by instead interpolating with a single function $p(x)$ through all the data points and then evaluating $p'(x_j)$. Under some circumstances this can work very well, particularly if the underlying function giving the data U_j is very smooth and if we are careful in our choice of interpolation points and the form of $p(x)$. Such methods are called *pseudospectral* methods or *spectral collocation* methods since $p(x)$ is chosen to collocate (interpolate) at the grid points. Often they are simply called spectral methods, though technically this term is reserved for related methods in which $p(x)$ is chosen by different criteria than interpolation. A good introduction to pseudospectral methods is presented in Trefethen [Tre00], which includes a set of simple but very useful MATLAB routines that make it easy to get started using these methods. In this section notation consistent with that book is used.

The most classical form of spectral method is based on the approximation of a periodic function by trigonometric polynomials or Fourier series. In this case equally spaced points are typically used and the computations required can be done efficiently by using the FFT. But many problems involve boundary conditions that are not simply periodic. In this case one can use spectral methods based on polynomial interpolation. Using equally spaced points in this case is generally disasterous, however. Even if the underlying function is very smooth, the high order polynomial that interpolates it at a large number of equally spaced points will tyically be highly oscillatory and have large errors between the grid points. Very good results can be obtained, however, by interpolating at a different set of points that are more concentrated near the endpoints of the interval. If we are working on the interval $-1 \le x \le 1$, then a good choice of $N+1$ points is

$$x_j = \cos(j\pi/N) \qquad j = 0,\ 1,\ \ldots,\ N. \tag{14.17}$$

Note that $x_0 = 1$ and $x_N = -1$ so these are ordered backwards from the normal convention. These points are called the *Chebyshev points* or more properly the *Chebyshev extreme points* since they are the points where the Chebyshev polynomial $T_N(x)$ takes its extreme values ± 1. A simple way to visualize where these points are is to draw a half circle in the upper half plane of radius 1 about the origin, place $N+1$ points equally spaced around the half circle, including both ends, and then project these points down to the x-axis. This gives the x_j. Near the center of the interval (around $x = 0$), the spacing between points is roughly π/N, whereas a uniform grid of $N+1$ points on $[-1, 1]$ would have spacing $2/N$ everywhere. So this grid is only coarser by a factor of $\pi/2 \approx 1.5$ near the center. Near the edges the spacing is more like π^2/N^2, however, and considerably finer than a uniform grid with the same number of points.

There is a beautiful mathematical theory of interpolation and approximation that shows that interpolating functions at the Chebyshev points gives nearly the best possible approximation of the function by a polynomial. If the function is infinitely differentiable, then this approximation is “spectrally accurate”, meaning the error goes to zero faster than any fixed power of $1/N$ as $N \to \infty$. The same is true of approximations to the derivative based on this interpolation.

Given data U_j for $j = 0,\ 1,\ \ldots,\ N$ (at the Chebyshev points x_j), we compute the $W_j \approx u_x(x_j)$ by interpolating by a polynomial $p(x)$ of degree N and setting $W_j = p'(x_j)$. This gives

$$W_j = u_x(x_j) + O((1/N)^m) \qquad \text{for all } m \ge 0 \text{ as } N \to \infty. \tag{14.18}$$

Note that W_j is a linear combination of all the U values, and hence we can write

$$W = D_N U \tag{14.19}$$

for some matrix D_N. There are explicit formulas for the elements of D_N, given in [Tre00] and computed by the simple MATLAB code `cheb.m` from that book, which may be found at

`http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/spectral.html`

The matrix D_N is a dense matrix because each W_j depends on all U values, and so a numerical method that uses derivative evaluations based on (14.19) would appear to require $O(N^2)$ operations every time a derivative must be calculated (typically at least once per time step, perhaps more if a multi-stage Runge-Kutta method is used, for example). However, it turns out that the FFT can be applied to this polynomial interpolation problem to reduce this to $O(N \log N)$. This is because Chebyshev polynomials on the interval $[-1, 1]$ are closely related to Fourier series on the unit circle, and data at the Chebyshev points can be mapped out to equally spaced points on the unit circle using the correspondence mentioned above (see Figure 4.1). This allows Chebyshev spectral methods to be very efficiently applied on larger problems, and so they are competitive with methods based on explicit or compact differences in terms of the time required. The excellent accuracy of the these methods (for sufficiently smooth problems) makes them the method of choice for many problems.

It is interesting to investigate the dispersion relation for the Chebyshev spectral method. Unfortunately this is not as easy to calculate as for the methods considered previously, which were translation-invariant methods for which application to $U_j = e^{i\xi x_j}$ results in a W_j of the form (14.5). On a finite grid (with periodic boundary conditions, say) those methods give a differentiation matrix D that is a Toeplitz matrix (constant along each diagonal) and for which the eigenvectors have components of the form $e^{i\xi x_j}$. This is not true of the Chebyshev differentiation matrix D_N. In fact this matrix is not even skew-symmetric and so the eigenvalues are not pure imaginary. Moreover the eigenvalues are very sensitive to small perturbations, an issue that makes stability analysis for methods based on D_N much more subtle than the analysis of typical finite difference methods. See, for example, [RT90] and [TT87] for some discussions.

To produce plots similar to those shown in Figures 14.1 and 14.2 for the Chebyshev method, we can apply D_N to the vectors with components $e^{i\xi x_j}$ for various values of ξ. We hope that for ξ sufficiently small, this will produce a vector that is nearly a scalar multiple of the original vector, with a scale factor that is nearly pure imaginary and of the form $i\bar{\xi}$ for some $\bar{\xi}$. In fact this is nearly so for $|\xi| \le N$ or so, and over this range $\bar{\xi}$ is remarkably close to ξ. To get a picture of this, we do the following:

```
for $k = 1 : (N/2)$
    $\xi = k\pi$
    $W = D_N U$
    $\bar{\xi}_j = W_j/(iU_j)$ for each $j$
    $\bar{\xi}_{min} = \min_j \mathrm{Re}(\bar{\xi}_j)$
    $\bar{\xi}_{max} = \max_j \mathrm{Re}(\bar{\xi}_j)$
    end
```

For a translation-invariant method we would have $\bar{\xi}_j = \bar{\xi}$ for each j and so $\bar{\xi}_{min} = \bar{\xi}_{max} = \bar{\xi}$. In Figure 14.3(a) we plot $\bar{\xi}_{min}h$ and $\bar{\xi}_{max}h$ as functions of ξh. For $\xi h < 2$ these two values are nearly identical and in fact lie along the ideal curves, indicating that the method resolves these waves very well.

Since we are using a nonuniform grid there is no single grid spacing h. In Figure 14.3 we have used the value $h = 2/N$, the average grid spacing over the entire interval $[-1, 1]$ and the grid spacing we would have if we uniformly distributed the same number of points. This seems to give the fairest comparison to the previous pictures, since typically we want the maximum resolution possible for a given number of grid points.

For $\xi h < 2$ the approximation is very good; the real part of each $\bar{\xi}_j$ is nearly equal to ξ and the imaginary part (not shown) is nearly zero. Things fall apart rapidly at $\xi h = 2$, and beyond this point the $\bar{\xi}_j$ are scattered around in the complex plane quite far from ξ. The reason we have no resolution beyond this point is that the grid spacing is roughly π/N near the center of the interval. The best we can hope to do is resolve wavenumbers that are sawtooths on the grid, and near the center of the interval a sawtooth corresponds to wavenumber $\xi = N$ and hence $\xi h = 2$. But viewed in this light the spectral method does very well indeed: it is almost perfectly resolving every wavenumber up to the sawtooth and we cannot expect better than that. (Note that if we had defined h to be π/N instead of $2/N$ in Figure 14.3 then it would show perfect resolution up to nearly $\xi h = \pi$.)

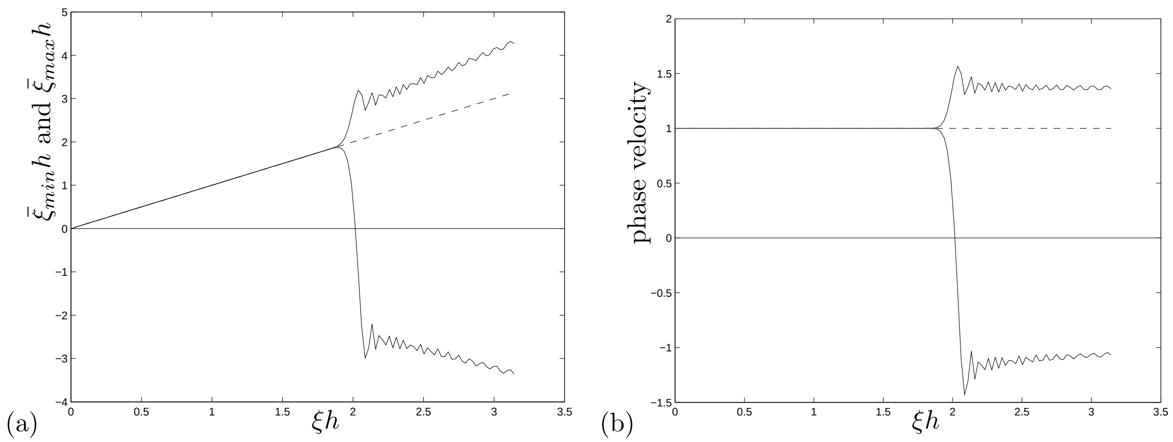


Figure 14.3: (a) Approximate dispersion relation $\bar{\xi}_{min}h$ and $\bar{\xi}_{max}h$ vs. ξh for the Chebyshev differentiation matrix, with $h = 2/N$. (b) Corresponding approximations to the phase velocity c_p from (14.10) plotted as a function of ξh for the same method. The dashed line shows the desired exact relation in each case.

Chapter 15

Mixed Equations and Fractional Step Methods

We have now studied the solution of various types of time-dependent equations — ODE's such as those arising in chemical kinetics, diffusion equations, and advection equations. 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 a mixture. In this chapter we discuss mixed equations such as reaction-diffusion equations or advection-diffusion equations. There are various ways to handle mixed equations, and we will consider two basic approaches:

- Unsplit methods, in which a single finite-difference formula is developed to advance the full mixed equation over one time step.
- Fractional step (splitting) methods, in which the problem is broken down into pieces corresponding to the different processes, and a numerical method appropriate for each separate piece is applied independently. This approach is also often used to split multi-dimensional problems into a sequence of one-dimensional problems (*dimensional splitting*). We have seen an example of this for the heat equation in the LOD method of Section 12.8.

15.1 Advection-reaction equations

Example 15.1. We begin with a simple advection-reaction equation of the form

$$u_t + au_x = -\lambda u, \tag{15.1}$$

with data $u(x,0) = \eta(x)$. This would model, for example, the transport of a radioactive material in a fluid flowing at constant speed a down a pipe. The material decays as it flows along, at rate λ. We can easily compute the exact solution of (15.1), since along the characteristic $dx/dt = a$ we have $du/dt = -\lambda u$, and hence

$$u(x,t) = e^{-\lambda t}\eta(x-at). \tag{15.2}$$

15.1.1 Unsplit methods

It is easy to develop unsplit methods for (15.1). For example, an obvious extension of the upwind method for advection would be (assuming $a > 0$),

$$U_j^{n+1} = U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n) - k\lambda U_j^n. \tag{15.3}$$

This method is first order accurate and stable for $0 < ak/h < 1$.

A second order Lax-Wendroff style method can be developed by using the Taylor series

$$u(x, t+k) \approx u(x,t) + ku_t(x,t) + \frac{1}{2}k^2 u_{tt}(x,t). \tag{15.4}$$

As in the derivation of Lax-Wendroff, we must compute u_{tt} from the PDE, obtaining

$$u_{tt} = -au_{xt} - \lambda u_t.$$

Since

$$u_{tx} = -au_{xx} - \lambda u_x,$$

we obtain

$$u_{tt} = a^2 u_{xx} + 2a\lambda u_x + \lambda^2 u. \tag{15.5}$$

Note that this is more easily obtained by using

$$\partial_t u = (-a\partial_x - \lambda)u,$$

and hence

$$\partial_t^2 u = (-a\partial_x - \lambda)^2 u = (a^2\partial_x^2 - 2a\lambda\partial_x + \lambda^2)u. \tag{15.6}$$

Using this expression for u_{tt} in (15.4) gives

$$\begin{aligned} u(x,t+k) &\approx u - k(au_x + \lambda u) + \frac{1}{2}k^2(a^2u_{xx} + 2a\lambda u_x + \lambda^2 u) \\ &= \left(1 - k\lambda + \frac{1}{2}k^2\lambda^2\right)u - ka\left(1 - \frac{1}{2}k\lambda\right)u_x + \frac{1}{2}k^2a^2u_{xx}. \end{aligned} \tag{15.7}$$

We can now approximate x-derivatives by finite differences to obtain the second-order method

$$U_j^{n+1} = \left(1 - k\lambda + \frac{1}{2}k^2\lambda^2\right)U_j^n - \frac{ka}{2h}\left(1 - \frac{1}{2}k\lambda\right)(U_{j+1}^n - U_{j-1}^n) + \frac{k^2a^2}{2h^2}(U_{j-1}^n - 2U_j^n + U_{j+1}^n). \tag{15.8}$$

Note that in order to correctly model the equation (15.1) to second order accuracy, we must properly model the interaction between the au_x and the λu terms, which brings in the mixed term $-\frac{1}{2}k^2 a\lambda u_x$ in the Taylor series expansion.

For future use we also note that the full Taylor series expansion can be written as

$$u(x,t+k) = \sum_{j=0}^{\infty} \frac{k^j}{j!}(-a\partial_x - \lambda)^j u(x,t), \tag{15.9}$$

which can be written formally as

$$u(x,t+k) = e^{-k(a\partial_x + \lambda)}u(x,t) \tag{15.10}$$

The operator $e^{-k(a\partial_x+\lambda)}$, which is defined via the Taylor series in (15.9), is called the *solution operator* for the PDE (15.1). Note that for any value of k we have

$$u(x,k) = e^{-k(a\partial_x+\lambda)}u(x,0).$$

15.1.2 Fractional step methods

A fractional step method for (15.1) is applied by first splitting the equation into two *subproblems*

$$\text{Problem A:} \quad u_t + au_x = 0, \tag{15.11}$$

$$\text{Problem B:} \quad u_t = -\lambda u. \tag{15.12}$$

We have developed methods for each of these two equations separately. The idea with the fractional step method is to combine these by applying the two methods in an alternating manner. As a simple example, suppose we use the upwind method for the A-step and forward Euler for the ODE in the B-step. Then the simplest fractional step method over one time step would consist of the following 2 stages:

$$\begin{aligned} &\text{A-step:} \quad U_j^* = U_j^n - \tfrac{ak}{h}(U_j^n - U_{j-1}^n), && (15.13)\\ &\text{B-step:} \quad U_j^{n+1} = U_j^* - k\lambda U_j^*. && (15.14)\end{aligned}$$

Note that we first take a time step of length k with upwind, starting with initial data U_j^n to obtain the intermediate value U_j^*. Then we take a time step of length k using forward Euler, starting with the data U^* obtained from the first stage.

It may seem that we have advanced the solution by time $2k$ after taking these two steps of length k. However, in each stage we used only some of the terms in the original PDE, and the two stages combined give a consistent approximation to solving the original PDE (15.1) over a single time step of length k.

To check this consistency, we can combine the two stages by eliminating U^* to obtain a method in a more familiar form:

$$\begin{aligned} U_j^{n+1} &= (1-k\lambda)U_j^* \\ &= (1-k\lambda)\left[U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n)\right] \\ &= U_j^n - \frac{ak}{h}(U_j^n - U_{j-1}^n) - k\lambda U_j^n + \frac{ak^2\lambda}{h}(U_j^n - U_{j-1}^n). \end{aligned} \tag{15.15}$$

The first three terms on the right-hand side agree with the unsplit method (15.3). The final term is $O(k^2)$ (since $(U_j^n - U_{j-1}^n)/h \approx u_x = O(1)$) and so a local truncation error analysis will show that this method, though slightly different from (15.3), is also consistent and first order accurate on the original equation (15.1).

A natural question is whether we could improve the accuracy by using a more accurate method in each step. For example, suppose we use Lax-Wendroff in the A-step and the trapezoidal method, or the 2-stage Runge-Kutta method from Example 6.11, in the B-step. Would we then obtain a second order accurate method for the original equation? For this particular equation, the answer is yes. In fact if we use p'th order accurate methods for each step, the result will be a p'th order accurate method for the full original equation. But this equation is very special in this regard, and this claim should seem surprising. One would think that splitting the equation into pieces in this manner would introduce some error that depends on the size of the time step k and is indepenendent of how well we then approximate the subproblem in each step. In general this is true — there is a "splitting error" that in general would be $O(k)$ for the type of splitting used above, and so the resulting fractional step method will be only first order accurate, no matter how well we then approximate each step. This will be analyzed in more detail below.

For the case of equation (15.1) there is no splitting error. This follows from the observation that we can solve (15.1) over any time period k by first solving equation (15.11) over time k, and then using the result as data to solve the equation (15.12) over time k. To verify this, let $u^*(x,k)$ be the exact solution to the A-problem,

$$\begin{aligned} &u_t^* + au_x^* = 0 \\ &u^*(x,0) = \eta(x). \end{aligned} \tag{15.16}$$

We use a different symbol $u^*(x,t)$ for the solution to this problem rather than $u(x,t)$, which we reserve for the exact solution to the original problem.

Then we have

$$u^*(x,k) = \eta(x-ak).$$

If we now use this as data in solving the B-problem (15.12), we will be solving

$$u_t^{**} = -\lambda u^{**} \tag{15.17}$$

with initial data

$$u^{**}(x,0) = \eta(x-ak).$$

This is just an ODE at each point x, and the solution is

$$u^{**}(x,k) = e^{-\lambda k}\eta(x-ak).$$

Comparing this with (15.2), we see that we have indeed recovered the solution to the original problem by this 2-stage procedure.

Physically we can interpret this as follows. Think of the original equation as modeling a radioactive tracer that is advecting with constant speed a (carried along in a fluid, say) and also decaying with rate λ. Since the decay properties are independent of the position x, we can think of first advecting the tracer over time k without allowing any decay, and then holding the fluid and tracer stationary while we allow it to decay for time k. We will get the same result, and this is what we have done in the fractional step method.

Another way to examine the splitting error, which must be used more generally when we do not know the exact solution to the equations involved, is to use Taylor series expansions. If we look at a time step of length k, then solving the A-equation gives

$$\begin{aligned} u^*(x,k) &= u^*(x,0) + ku_t^*(x,0) + \frac{1}{2}k^2u_{tt}^*(x,0) + \cdots \\ &= u^*(x,0) - aku_x^*(x,0) + \frac{1}{2}a^2k^2u_{xx}^*(x,0) - \cdots \end{aligned} \tag{15.18}$$

Similarly, if we solve the problem (15.17) with general initial data we obtain

$$\begin{aligned} u^{**}(x,k) &= u^{**}(x,0) + ku_t^{**}(x,0) + \frac{1}{2}k^2u_{tt}^{**}(x,0) + \cdots \\ &= \left(1 - k\lambda + \frac{1}{2}k^2\lambda^2 + \cdots\right)u^{**}(x,0). \end{aligned} \tag{15.19}$$

If we now use the result from (15.18) as the initial data in (15.19), we obtain

$$\begin{aligned} u^{**}(x,k) &= \left(1 - k\lambda + \frac{1}{2}k^2\lambda^2 - \cdots\right)\left(u^*(x,0) - aku_x^*(x,0) + \frac{1}{2}a^2k^2u_{xx}^*(x,0) + \cdots\right) \\ &= u^* - (aku_x^* + \lambda u^*) + \frac{1}{2}k^2(a^2u_{xx}^* + 2a\lambda u_x^* + \lambda^2u^*) + \cdots . \end{aligned} \tag{15.20}$$

Comparing this with the Taylor series expansion (15.7) that we used in deriving the unsplit Lax-Wendroff method shows that this agrees with $u(x,k)$, at least for the 3 terms shown, and in fact to all orders.

Note that the mixed term $k^2a\lambda u_x$ needed in the u_{tt} term from (15.5) now arises naturally from taking the product of the two Taylor series (15.18) and (15.19). In fact, we see that for this simple equation we can write (15.19) as

$$u^{**}(x,k) = e^{-k\lambda}u^{**}(x,0)$$

while (15.18) can be written formally as

$$u^*(x,k) = e^{-ak\partial_x}\eta(x).$$

If we now use $u^*(x,k)$ as the data $u^{**}(x,0)$ as we do in the fractional step method, we obtain

$$u^{**}(x,k) = e^{-k\lambda}e^{-ak\partial_x}\eta(x).$$

Multiplying out the Taylor series as we did in (15.20) verifies that these exponentials satisfy the usual rule, that to take the product we need only add the exponents, *i.e.*,

$$u^{**}(x,k) = e^{-k(a\partial_x+\lambda)}\eta(x).$$

The exponential appearing here is exactly the solution operator for the original equation, and so again we see that $u^{**}(x,k) = u(x,k)$.

The fact that there is no splitting error for the problem (15.1) is a reflection of the fact that, for this problem, the solution operator for the full problem is exactly equal to the product of the solution operators of the two subproblems (15.11) and (15.12). This is not generally the case in other problems.

Example 15.2. Suppose we modify the equation slightly so that the decay rate λ depends on x,

$$u_t + au_x = -\lambda(x)u. \tag{15.21}$$

Then our previous argument for the lack of a splitting error breaks down — advecting the tracer a distance ak and then allowing it to decay, with rates given by the values of λ at the final positions, will not in general give the same result as if the decays occurs continuously as it advects, using the instantaneous rate given by $\lambda(x)$ at each point passed.

This can be analyzed formally using Taylor series expansions again. Rather than going through this for this particular example, we will first examine the more general case and then apply it to this problem.

15.2 General formulation of fractional step methods

Consider a more general PDE of the form

$$u_t = (\mathcal{A}+\mathcal{B})u \tag{15.22}$$

where $\mathcal{A}$ and $\mathcal{B}$ may be differential operators, *e.g.*, $\mathcal{A} = -a\partial_x$ and $\mathcal{B} = \lambda(x)$ in the previous example. For simplicity suppose that $\mathcal{A}$ and $\mathcal{B}$ do not depend explicitly on t, *e.g.*, $\lambda(x)$ is a function of x but not of t. Then we can compute that

$$u_{tt} = (\mathcal{A}+\mathcal{B})u_t = (\mathcal{A}+\mathcal{B})^2u,$$

and in general

$$\partial_t^j u = (\mathcal{A}+\mathcal{B})^j u.$$

We have used this idea before in calculating Taylor series, *e.g.*, in (15.6).

Note that if $\mathcal{A}$ or $\mathcal{B}$ do depend on t, then we would have to use the product rule,

$$u_{tt} = (\mathcal{A}+\mathcal{B})u_t + (\mathcal{A}_t+\mathcal{B}_t)u$$

and everything would become more complicated.

In our simple case we can write the solution at time t using Taylor series as

$$\begin{aligned} u(x,k) &= u(x,0) + k(\mathcal{A}+\mathcal{B})u(x,0) + \frac{1}{2}k^2(\mathcal{A}+\mathcal{B})^2u(x,0) + \cdots \\ &= \left(I + k(\mathcal{A}+\mathcal{B}) + \frac{1}{2}k^2(\mathcal{A}+\mathcal{B})^2 + \cdots\right)u(x,0) \\ &= \sum_{j=0}^{\infty} \frac{k^j}{j!}(\mathcal{A}+\mathcal{B})^j u(x,0), \end{aligned} \tag{15.23}$$

which formally could be written as

$$u(x,k) = e^{k(\mathcal{A}+\mathcal{B})}u(x,0).$$

With the fractional step method, we instead compute

$$u^*(x,k) = e^{k\mathcal{A}}u(x,0),$$

and then

$$u^{**}(x,k) = e^{k\mathcal{B}}e^{k\mathcal{A}}u(x,0),$$

and so the *splitting error* is

$$u(x,k) - u^{**}(x,k) = \left(e^{k(\mathcal{A}+\mathcal{B})} - e^{k\mathcal{B}}e^{k\mathcal{A}}\right)u(x,0). \tag{15.24}$$

This should be calculated using the Taylor series expansions. We have (15.23) already, while

$$\begin{aligned} u^{**}(x,k) &= \left(I + k\mathcal{B} + \frac{1}{2}k^2\mathcal{B}^2 + \cdots\right)\left(I + k\mathcal{A} + \frac{1}{2}k^2\mathcal{A}^2 + \cdots\right)u(x,0) \\ &= \left(I + k(\mathcal{A}+\mathcal{B}) + \frac{1}{2}k^2(\mathcal{A}^2 + 2\mathcal{B}\mathcal{A} + \mathcal{B}^2) + \cdots\right)u(x,0). \end{aligned} \tag{15.25}$$

The $I + k(\mathcal{A}+\mathcal{B})$ terms agree with (15.23). In the k^2 term, however, the term from (15.23) is

$$\begin{aligned} (\mathcal{A}+\mathcal{B})^2 &= (\mathcal{A}+\mathcal{B})(\mathcal{A}+\mathcal{B}) \\ &= \mathcal{A}^2 + \mathcal{A}\mathcal{B} + \mathcal{B}\mathcal{A} + \mathcal{B}^2. \end{aligned} \tag{15.26}$$

In general this is *not* the same as

$$\mathcal{A}^2 + 2\mathcal{B}\mathcal{A} + \mathcal{B}^2,$$

and so the splitting error is

$$u(x,k) - u^{**}(x,k) = \frac{1}{2}k^2(\mathcal{A}\mathcal{B} - \mathcal{B}\mathcal{A})u(x,0) + O(k^3). \tag{15.27}$$

The splitting error is zero only in the special case when the differential operators $\mathcal{A}$ and $\mathcal{B}$ commute (in which case it turns out that all the higher order terms in the splitting error also vanish).

Example 15.3. For the problem considered in Example 15.1

$$\mathcal{A} = -a\partial_x \quad \text{and} \quad \mathcal{B} = -\lambda.$$

We then have $\mathcal{A}\mathcal{B}u = \mathcal{B}\mathcal{A}u = a\lambda u_x$. These operators commute for λ constant and there is no splitting error.

Example 15.4. Now suppose $\lambda = \lambda(x)$ depends on x as in Example 15.2. Then we have

$$\mathcal{A}\mathcal{B}u = a\partial_x(\lambda(x)u) = a\lambda(x)u_x + a\lambda'(x)u$$

while

$$\mathcal{B}\mathcal{A}u = \lambda(x)au_x.$$

These are not the same unless $\lambda'(x) = 0$. In general the splitting error will be

$$u(x,k) - u^{**}(x,k) = \frac{1}{2}k^2a\lambda'(x)u(x,0) + O(k^3).$$

If we now design a fractional step method based on this splitting, we will see that the splitting error alone will introduce an $O(k^2)$ error in each time step, which can be expected to accumulate to an $O(k)$ error after the T/k time steps needed to reach some fixed time T (in the best case, assuming the method is stable). Hence even if we solve each subproblem *exactly* within the fractional step method, the resulting method will be only first order accurate. If the subproblems are actually solved with numerical methods that are p'th order accurate, the solution will still only be first order accurate no matter how large p is.

15.3 Strang splitting

It turns out that a slight modification of the splitting idea will yield second order accuracy quite generally (assuming each subproblem is solved with a method of at least this accuracy). The idea is to solve the first subproblem $u_t = \mathcal{A}u$ over only a half time step of length $k/2$. Then we use the result as data for a full time step on the second subproblem $u_t = \mathcal{B}u$, and finally take another half time step on $u_t = \mathcal{A}u$. We can equally well reverse the roles of $\mathcal{A}$ and $\mathcal{B}$ here. This approach is often called *Strang splitting* as it was popularized in a paper by Strang[Str68] on solving multi-dimensional problems.

To analyze the Strang splitting, note that we are now approximating the solution operator $e^{k(\mathcal{A}+\mathcal{B})}$ by $e^{\frac{1}{2}k\mathcal{A}}e^{k\mathcal{B}}e^{\frac{1}{2}k\mathcal{A}}$. Taylor series expansion of this product shows that

$$\begin{aligned} e^{\frac{1}{2}k\mathcal{A}}e^{k\mathcal{B}}e^{\frac{1}{2}k\mathcal{A}} &= \left(I + \frac{1}{2}k\mathcal{A} + \frac{1}{8}k^2\mathcal{A}^2 + \cdots\right)\left(I + k\mathcal{B} + \frac{1}{2}k^2\mathcal{B}^2 + \cdots\right)\left(I + \frac{1}{2}k\mathcal{A} + \frac{1}{8}k^2\mathcal{A}^2 + \cdots\right) \\ &= I + k(\mathcal{A}+\mathcal{B}) + \frac{1}{2}k^2(\mathcal{A}^2 + \mathcal{A}\mathcal{B} + \mathcal{B}\mathcal{A} + \mathcal{B}^2) + O(k^3). \end{aligned} \tag{15.28}$$

Comparing with (15.23), we seee that the $O(k^2)$ term is now captured correctly. The $O(k^3)$ term is not correct in general, however, unless $\mathcal{A}\mathcal{B} = \mathcal{B}\mathcal{A}$.

Exercise 15.1 *Compute the $O(k^3)$ term in the splitting error for the Strang splitting.*

Note that over several time steps we can simplify the expression obtained with the Strang splitting. After n steps we have

$$U^n = \left(e^{\frac{1}{2}k\mathcal{A}}e^{k\mathcal{B}}e^{\frac{1}{2}k\mathcal{A}}\right)\left(e^{\frac{1}{2}k\mathcal{A}}e^{k\mathcal{B}}e^{\frac{1}{2}k\mathcal{A}}\right)\cdots\left(e^{\frac{1}{2}k\mathcal{A}}e^{k\mathcal{B}}e^{\frac{1}{2}k\mathcal{A}}\right)U^0 \tag{15.29}$$

repeated n times. Dropping the parentheses and noting that $e^{\frac{1}{2}k\mathcal{A}}e^{\frac{1}{2}k\mathcal{A}} = e^{k\mathcal{A}}$, we obtain

$$U^n = e^{\frac{1}{2}k\mathcal{A}}e^{k\mathcal{B}}e^{k\mathcal{A}}e^{k\mathcal{B}}e^{k\mathcal{A}}\cdots e^{k\mathcal{B}}e^{\frac{1}{2}k\mathcal{A}}U^0. \tag{15.30}$$

This differs from the first order splitting only in the fact that we start and end with a half time step on $\mathcal{A}$, rather than starting with a full step and ending with $\mathcal{B}$.

Another way to achieve this same effect is to simply take steps of length k on each problem, as in the first-order splitting, but to alternate the order of these steps in alternate time steps, *e.g.*,

$$\begin{aligned} U^1 &= e^{k\mathcal{B}}e^{k\mathcal{A}}U^0 \\ U^2 &= e^{k\mathcal{A}}e^{k\mathcal{B}}U^1 \\ U^3 &= e^{k\mathcal{B}}e^{k\mathcal{A}}U^2 \\ U^4 &= e^{k\mathcal{A}}e^{k\mathcal{B}}U^3 \\ &\text{etc.} \end{aligned}$$

If we take an even number of time steps, then we obtain

$$\begin{aligned} U^n &= \left(e^{k\mathcal{A}}e^{k\mathcal{B}}\right)\left(e^{k\mathcal{B}}e^{k\mathcal{A}}\right)\left(e^{k\mathcal{A}}e^{k\mathcal{B}}\right)\left(e^{k\mathcal{B}}e^{k\mathcal{A}}\right)\cdots\left(e^{k\mathcal{A}}e^{k\mathcal{B}}\right)\left(e^{k\mathcal{B}}e^{k\mathcal{A}}\right)U^0 \\ U^n &= e^{k\mathcal{A}}\left(e^{k\mathcal{B}}e^{k\mathcal{B}}\right)\left(e^{k\mathcal{A}}e^{k\mathcal{A}}\right)\left(e^{k\mathcal{B}}e^{k\mathcal{B}}\right)\cdots\left(e^{k\mathcal{B}}e^{k\mathcal{B}}\right)e^{k\mathcal{A}}U^0. \end{aligned}$$

Since $e^{k\mathcal{B}}e^{k\mathcal{B}} = e^{2k\mathcal{B}}$, this is essentially the same as (15.29) but with $\frac{1}{2}k$ replaced by k.

The fact that the Strang splitting is so similar to the first order splitting suggests that the first order splitting is not really so bad, and in fact it is not. While formally only first order accurate, the coefficient of the $O(k)$ term may be much smaller than coefficients in the second order terms arising from discretization of $e^{k\mathcal{A}}$ and $e^{k\mathcal{B}}$.

Part II

Appendices

Appendix A1

Measuring Errors

In order to discuss the accuracy of a numerical solution, or the relative virtues of one numerical method, vs. 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 accuaracy of an approximate solution.

A1.1 Errors in a scalar value

First consider a problem in which the answer is a single value $z \in \mathbb{R}$. Consider, for example, the scalar 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 $z = u(T)$. Denote the computed solution by $\hat{z}$. Then the error in this computed solution is

$$E = \hat{z} - z.$$

A1.1.1 Absolute error

A natural measure of this error would be the absolute value of E,

$$|E| = |\hat{z} - z|.$$

This is called the *absolute error* in the approximation.

As an example, suppose that $z = 2.2$ while some numerical method produced a solution $\hat{z} = 2.20345$. Then the absolute error is

$$|\hat{z} - z| = 0.00345 = 3.45 \times 10^{-3}.$$

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 z. For example, suppose the z discussed above were measured in meters, so $z = 2.2$ 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 $z = 2.2 \times 10^9$ and the approximate solution is $\hat{z} = 2.20345 \times 10^9$, giving an absolute error of

$$|\hat{z} - z| = 3.45 \times 10^{6}.$$

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 z in kilometers then $z = 2.2 \times 10^{-3}$ and $\hat{z} = 2.20345 \times 10^{-3}$ so

$$|\hat{z} - z| = 3.45 \times 10^{-6}.$$

The error seems much smaller and yet there are still only three correct digits.

A1.1.2 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

$$\left|\frac{\hat{z} - z}{z}\right|.$$

The size of the error is scaled by the size of the value being computed. For the above examples, the relative error in $\hat{z}$ 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 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 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 insure 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.

A1.2 "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 $f(h) = O(g(h))$ though 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:

$$2h^3 = O(h^2) \quad \text{as } h \to 0, \quad \text{since } \frac{2h^3}{h^2} = 2h < 1 \quad \text{for all } h < 1/2.$$

$$2h^3 = o(h^2) \quad \text{as } h \to 0, \quad \text{since } 2h \to 0 \quad \text{as } h \to 0.$$

$$\sin(h) = O(h) \quad \text{as } h \to 0, \quad \text{since } \sin h = h - \frac{h^3}{3} + \frac{h^5}{5} + \cdots < h \quad \text{for all } h > 0.$$

$$\sin(h) = h + o(h) \quad \text{as } h \to 0, \quad \text{since } (\sin h - h)/h = O(h^2).$$

$$\sqrt{h} = O(1) \quad \text{as } h \to 0, \quad \text{and also } \sqrt{h} = o(1), \quad \text{but } \sqrt{h} \text{ is not } O(h)\ .$$

$$1 - \cos h = o(h) \quad \text{and } 1 - \cos h = O(h^2) \quad \text{as } h \to 0.$$

$$e^{-1/h} = o(h^q) \quad \text{as } h \to 0 \quad \text{for every value of } q.$$

$$\text{To see this, let } x = 1/h \text{ then } \frac{e^{-1/h}}{h^q} = e^{-x}x^q \to 0 \quad \text{as } x \to \infty.$$

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 $O(h^2)$. Though 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

$$E(h) = 0.003\, h \tag{A1.1}$$

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

$$E(h) = 10^6\, h^2 \tag{A1.2}$$

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 (A1.1) over a second order method with error (A1.2).

A1.3 Errors in vectors

Now suppose $z \in \mathbb{R}^m$ is a vector with m components, for example the solution to a system of m ODE's at some particular fixed time T. Then $\hat{z}$ is a vector of approximate values and the error $e = \hat{z} - z$ is also a vector in $\mathbb{R}^m$. 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 $\mathbb{R}^m$ to nonnegative real numbers, satisfying the following conditions (which generalize important properties of the absolute value for scalars):

1. $\|x\| \ge 0$ for any $x \in \mathbb{R}^m$, 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 $\|\cdot\|_\infty$:

$$\|e\|_\infty = \max_{1\leq i\leq m} |e_i|.$$

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,

$$\|e\|_1 = \sum_{i=1}^m |e_i| \qquad \text{and} \qquad \|e\|_2 = \sqrt{\sum_{i=1}^m |e_i|^2}. \tag{A1.3}$$

These are special cases of the general family of p-norms, defined by

$$\|e\|_p = \left[\sum_{i=1}^m |e_i|^p\right]^{1/p}. \tag{A1.4}$$

Note that the max-norm can be obtained as the limit as $p \to \infty$ of the p-norm.

A1.3.1 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 $\|e^h\| = O(h^q)$ in some norm, so that the method is qth 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 $\mathbb{R}^m$. (Note that this result is only valid as long as the dimension m of the vector is fixed as $h \to 0$. See Section A1.5 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 $\mathbb{R}^m$. Then there exist two constants C_1 and C_2 such that

$$C_1\|x\|_\alpha \leq \|x\|_\beta \leq C_2\|x\|_\alpha \tag{A1.5}$$

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:

$$\|x\|_\infty \leq \|x\|_1 \leq m\|x\|_\infty \tag{A1.6a}$$

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{m}\|x\|_\infty \tag{A1.6b}$$

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{m}\|x\|_2. \tag{A1.6c}$$

Now suppose that $\|e^h\|_\alpha \leq Ch^q$ as $h \to 0$ in some norm $\|\cdot\|_\alpha$. Then we have

$$\|e^h\|_\beta \leq C_2\|e^h\|_\alpha \leq C_2 C h^q$$

and so $\|e^h\|_\beta = O(h^q)$ 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 A1.4, where we consider approximating functions rather than vectors.

A1.3.2 Matrix norms

For any vector norm $\|\cdot\|$ we can define a corresponding matrix norm. The norm of a matrix $A \in \mathrm{I\!R}^{m\times m}$ is denoted by $\|A\|$ and has the property that $C = \|A\|$ is the *smallest* value of the constant C for which the bound

$$\|Ax\| \leq C\|x\| \tag{A1.7}$$

holds for *every* vector $x \in \mathrm{I\!R}^m$. Hence $\|A\|$ is defined by

$$\|A\| = \max_{\substack{x\in\mathrm{I\!R}^m \\ x\neq\vec{0}}} \frac{\|Ax\|}{\|x\|} = \max_{\substack{x\in\mathrm{I\!R}^m \\ \|x\|=1}} \|Ax\|.$$

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:

$$\begin{aligned} \|A\|_1 &= \max_{1\leq j\leq m} \sum_{i=1}^m |a_{ij}| \qquad \text{(maximum column sum)} && \text{(A1.8a)} \\ \|A\|_\infty &= \max_{1\leq i\leq m} \sum_{j=1}^m |a_{ij}| \qquad \text{(maximum row sum)} && \text{(A1.8b)} \\ \|A\|_2 &= \sqrt{\rho(A^TA)}. && \text{(A1.8c)} \end{aligned}$$

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 $A = A^T$ is symmetric, then $\|A\|_2 = \rho(A)$.

A1.4 Errors in functions

Now consider a problem in which the solution is a function $u(x)$ over some interval $a \leq x \leq b$ rather than a single value or vector. Some numerical methods, such as finite element or collocation methods, produce an approximate solution $\hat{u}(x)$ which is also a function. Then the error is given by a function

$$e(x) = \hat{u}(x) - u(x).$$

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

$$\|e\|_\infty = \max_{a\leq x\leq b} |e(x)|. \tag{A1.9}$$

The 1-norm and 2-norms are given by integrals over $[a,b]$ rather than by sums over the vector elements:

$$\begin{aligned} \|e\|_1 &= \int_a^b |e(x)|\,dx, && \text{(A1.10)} \\ \|e\|_2 &= \left(\int_a^b |e(x)|^2\,dx\right)^{1/2}. && \text{(A1.11)} \end{aligned}$$

These are again special cases of the general p-norm, defined by

$$\|e\|_p = \left(\int_a^b |e(x)|^p\,dx\right)^{1/p}. \tag{A1.12}$$

A1.5 Errors in grid functions

Finite difference methods do not produce a function $\hat{u}(x)$ 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 $N+1$ equally spaced points with spacing $h = (b-a)/N$,

$$x_i = a + ih, \qquad i = 0,\ 1,\ \ldots,\ N,$$

our approximation to $u(x)$ would consist of the $N+1$ values $(U_0,\ U_1,\ \ldots,\ U_N)$. 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 $e = (e_0,\ e_1,\ \ldots,\ e_N)$ 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.*,

$$U_i \approx \frac{1}{h} \int_{x_{i-1}}^{x_i} u(x)\,dx, \qquad i = 1,\ 2,\ \ldots,\ N.$$

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 (see Example 3.1 below), so it is important to make the proper choice for the method being studied.

Once we have defined the vector of errors $(e_0,\ \ldots,\ e_N)$, we can measure its magnitude using some norm. Since this is simply a vector with $N+1$ components, it would be tempting to simply use one of the vector norms discussed above, *e.g.*,

$$\|e\|_1 = \sum_{i=0}^{N} |e_i|. \tag{A1.13}$$

However, this choice would give a very misleading idea of the magnitude of the error. The quantity in (A1.13) 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 (A1.13) 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 (A1.10), which is motivated by considering the vector $(e_0,\ \ldots,\ e_N)$ as a discretization of some error function $e(x)$. This suggests defining

$$\|e\|_1 = h \sum_{i=0}^{N} |e_i| \tag{A1.14}$$

with the factor of h corresponding to the dx in the integral. Note that since $h = (b-a)/N$, 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 (A1.10). The norm (A1.14) will be called a *grid-function norm* and is distinct from the related vector norm. The set of values $(e_0,\ \ldots,\ e_N)$ 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 p-norm should be scaled by $h^{1/p}$, so that the p-norm for grid functions is

$$\|e\|_p = \left(h \sum_{i=0}^{N} |e_i|^p \right)^{1/p}. \tag{A1.15}$$

Since $h^{1/p} \to 1$ as $p \to \infty$, the max-norm remains unchanged:

$$\|e\|_\infty = \max_{0\le i\le N} |e_i|,$$

which makes sense from (A1.9).

In two space dimensions we have analogous norms of functions and grid functions, *e.g.*,

$$\begin{aligned} \|e\|_p &= \left(\iint |e(x,y)|^p \, dx\, dy\right)^{1/p} && \text{for functions} \\ \|e\|_p &= \left(\Delta x\, \Delta y \sum_i \sum_j |e_{ij}|^p\right)^{1/p} && \text{for grid functions} \end{aligned}$$

with the obvious extension to more dimensions.

A1.5.1 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 $e \in \mathbb{R}^{N+1}$. For example, translating (A1.6a) to the context of grid-function norms gives the bounds

$$\begin{aligned} h\|e\|_\infty \le \quad & \|e\|_1 \quad \le Nh\|e\|_\infty = (b-a)\,\|e\|_\infty, && \text{(A1.16a)} \\ \sqrt{h}\,\|e\|_\infty \le \quad & \|e\|_2 \quad \le \sqrt{Nh}\,\|e\|_\infty = \sqrt{b-a}\,\|e\|_\infty, && \text{(A1.16b)} \\ \sqrt{h}\,\|e\|_2 \le \quad & \|e\|_1 \quad \le \sqrt{Nh}\,\|e\|_2 = \sqrt{b-a}\,\|e\|_2. && \text{(A1.16c)} \end{aligned}$$

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, and 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 $m = N+1$ 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

$$u(x) = \begin{cases} 0 & x \le \frac{1}{2} \\ 1 & x > \frac{1}{2} \end{cases}$$

Let N be even and let

$$U_i^h = \begin{cases} 0 & i < N/2 \\ \frac{1}{2} & i = N/2 \\ 1 & i > N/2 \end{cases}$$

be the discrete approximation on the grid with spacing $h = 1/N$ on the interval $0 \le x \le 1$. This is illustrated in Figure A1.1 for $N = 8$. Define the error e_i^h by

$$e_i^h = U_i^h - u(x_i) = \begin{cases} \frac{1}{2} & i = N/2 \\ 0 & \text{otherwise.} \end{cases}$$

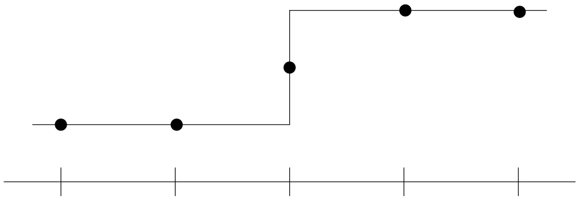


Figure A1.1: The function $u(x)$ and the discrete approximation.

No matter how fine the grid is, there is always an error of magnitude 1/2 at $i = N/2$ and hence

$$\|e^h\|_\infty = \frac{1}{2} \quad \text{for all } \ h.$$

On the other hand, in the 1-norm (A1.14) 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 are looking for. If it is really 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 really 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 A1.1 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

$$e_i^h = U_i^h - \frac{1}{h}\int_{x_i-h/2}^{x_i+h/2} u(x)\,dx,$$

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.

Appendix A2

Estimating errors in numerical solutions

When developing a computer program to solve a differential eqution, 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 are seen. Of course one must be aware of some of the issues raised in Appendix A1, *e.g.*, that the expected order may only appear 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 Attempt 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 there may be new errors introduced. How do we estimate the error in a numerical solution if we do not have the exact solution to compare it with?

The standard approach, when we can afford to, 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 A2.2.

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 A2.3 below.

A2.1 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 chapter 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 p'th 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

$$E(h) \approx Ch^p. \tag{A2.1}$$

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*

$$R(h) = E(h) \,/\, E(h/2), \tag{A2.2}$$

we expect

$$R(h) \approx 2^p, \tag{A2.3}$$

and hence

$$p \approx \log_2(R(h)). \tag{A2.4}$$

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

$$p \approx \frac{\log(E(h_1)/E(h_2))}{\log(h_1/h_2)}. \tag{A2.5}$$

Hence we can estimate the order p based on any two calculations. (This will only be valid if h is small enough that (A2.1) holds, of course.)

Note that we can also estimate the error constant C by

$$C \approx E(h)/h^p$$

once p is known.

A2.2 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. In order 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

$$u^h - \bar{u}^h \quad = \quad (u^h - \hat{u}^h) + (\hat{u}^h - \bar{u}^h)$$

If the method is supposed to be p'th 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 confirms that the code is converging to *some* function with the desired rate. It is perfectly possible that the code is converging very nicely to the *wrong* function. Consider a second-order accurate method applied to a two-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!

A2.3 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 in order to test the order of accuracy. Suppose, for example, that we are only willing to run the calculation on grids with spacing h, $h/2$ and $h/4$, and 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 p'th order accurate and that h is small enough that (A2.1) 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

$$\bar{R}(h) = \bar{E}(h)/\bar{E}(h/2) \approx \frac{4^p - 1}{2^p - 1} = 2^p + 1.$$

This differs significantly from (A2.3). 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 (A2.1) is valid), but to do so we must calculate p by

$$p \approx \log_2(\bar{R}(h) - 1)$$

rather than by (A2.4). The approximation (A2.4) 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 insure 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

$$\tilde{E}(h) = E(h) - E(h/2) \approx C\left(1 - \frac{1}{2^p}\right) h^p$$

and

$$\tilde{E}(h/2) = E(h/2) - E(h/4) \approx C\left(1 - \frac{1}{2^p}\right) \frac{h^p}{2^p}$$

and so

$$\tilde{E}(h)/\tilde{E}(h/2) \approx 2^p.$$

In this case the approximate error goes down by the same factor we would expect if the true solution is used as the reference solution on each grid.

Appendix A3

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 A4. An understanding of this is crucial in order to analyze the behavior and stability properties of differential or finite difference equations, as discussed in Appendix A5 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, though 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

$$1 \le m_g(\lambda) \le m_a(\lambda). \tag{A3.1}$$

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 A3.1. If the eigenvalues of A are all *distinct* then $m_g = m_a = 1$ for every eigenvalue and the matrix is not defective.

Example A3.2. 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 2-dimensional eigenspace for $\lambda = 3$ is spanned by $e_1 = (1, 0, 0)^T$ and $e_2 = (0, 1, 0)^T$. The 1-dimensional eigenspace for $\lambda = 5$ is spanned by $e_3 = (0, 0, 1)^T$.

Example A3.3. 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 2-dimensional space of eigenvectors associated with $\lambda = 3$, spanned by e_1 and e_3, so $m_g = 2$.

A3.1 Similarity transformations

Let S be any nonsingular matrix and set

$$B = S^{-1}AS. \tag{A3.2}$$

Then B has the same eigenvalues as A. To see this, suppose

$$Ar = \lambda r \tag{A3.3}$$

for some vector r and scalar λ. Let $w = S^{-1}r$ and multiply (A3.3) 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 (A3.2) 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, [GL96] for introductory discussions of such algorithms.

A3.2 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*

$$R = [r_1|r_2|\cdots|r_m]. \tag{A3.4}$$

Then

$$AR = R\Lambda \tag{A3.5}$$

where

$$\Lambda = \text{diag}(\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_m). \tag{A3.6}$$

This follows by viewing the matrix multiplication column-wise. Since the vectors r_j are linearly independent, the matrix R is invertible and so from (A3.5) we obtain

$$R^{-1}AR = \Lambda \tag{A3.7}$$

and hence we can *diagonolize* A by a similarity transformation. We can also write

$$A = R\Lambda R^{-1}, \tag{A3.8}$$

which is sometimes called the *eigen-decomposition 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 row-wise 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.

A3.3 The Jordan Canonical Form

If A is diagonalizable we have just seen in (A3.8) 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

$$J(\lambda,k) = \lambda I_k + S_k \tag{A3.9}$$

where I_k is the $k \times k$ identity matrix and S_k is the $k \times k$ shift matrix

$$S_k = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \qquad \text{for } k > 1 \qquad (\text{with } S_1 = 0), \tag{A3.10}$$

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 1-dimensional eigenspace of this block.

Theorem A3.3.1 *Every $m \times m$ matrix $A \in \mathbb{C}^{m\times m}$ can be transformed into the form*

$$A = RJR^{-1}, \tag{A3.11}$$

where J is a block diagonal matrix of the form.

$$J = \begin{bmatrix} J(\lambda_1, k_1) & & & \\ & J(\lambda_2, k_2) & & \\ & & \ddots & \\ & & & J(\lambda_s, k_s) \end{bmatrix}. \tag{A3.12}$$

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.

For the proof, see for example [**?**]. 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 *principle vectors.*

Example A3.4. 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

$$\begin{aligned} Ar_1 &= \lambda r_1 &&\Longrightarrow (A - \lambda I)r_1 = 0 \\ Ar_2 &= r_1 + \lambda r_2 &&\Longrightarrow (A - \lambda I)r_2 = r_1 \Longrightarrow (A - \lambda I)^2 r_2 = 0 \\ Ar_3 &= r_2 + \lambda r_3 &&\Longrightarrow (A - \lambda I)r_3 = r_2 \Longrightarrow (A - \lambda I)^3 r_3 = 0. \end{aligned} \tag{A3.13}$$

The vector r_1 forms a basis for the 1-dimensional eigenspace. The vectors r_2 and r_3 are principle 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 by 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

$$D = \begin{bmatrix} 1 & & & & \\ & \delta & & & \\ & & \delta^2 & & \\ & & & \ddots & \\ & & & & \delta^{k-1} \end{bmatrix}, \qquad D^{-1} = \begin{bmatrix} 1 & & & & \\ & \delta^{-1} & & & \\ & & \delta^{-2} & & \\ & & & \ddots & \\ & & & & \delta^{-(k-1)} \end{bmatrix}. \tag{A3.14}$$

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 JCF with $\delta \ll 1$, we can find a matrix R so that $R^{-1}AR$ is close to diagonal with 0's or δ's at each location on the superdiagonal. 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.

A3.4 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 analog 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 Hermition 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

$$u^H A u > 0 \tag{A3.15}$$

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 $u \neq 0$	$\Longleftrightarrow$	all eigenvalues are positive,
positive semi-definite	$\Longleftrightarrow$	$u^H A u \geq 0$ for all $u \neq 0$	$\Longleftrightarrow$	all eigenvalues are nonnegative,
negative definite	$\Longleftrightarrow$	$u^H A u < 0$ for all $u \neq 0$	$\Longleftrightarrow$	all eigenvalues are negative,
negative semi-definite	$\Longleftrightarrow$	$u^H A u \leq 0$ for all $u \neq 0$	$\Longleftrightarrow$	all eigenvalues are nonpositive,
indefinite	$\Longleftrightarrow$	$u^H A u$ indefinite	$\Longleftrightarrow$	if there are eigenvalues of both signs.

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 $w = R^H u$.

A3.5 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 13). 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, $R^H R = I$.

A3.6 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.

A3.7 Toeplitz and circulant matrices

A matrix is said to be *Toeplitz* if the value along each diagonal of the matrix 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

$$A = \begin{bmatrix} d_0 & d_1 & & & d_{-1} \\ d_{-1} & d_0 & d_1 & & \\ & d_{-1} & d_0 & d_1 & \\ & & & \ddots & \\ & & & & d_1 \\ d_1 & & & d_{-1} & d_0 \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}. \tag{A3.16}$$

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 Section **??**, for example, where 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 (A3.16) is given by

$$\lambda_p = d_{-1}e^{-2\pi iph} + d_0 + d_1 e^{2\pi iph}, \tag{A3.17}$$

where $i = \sqrt{-1}$, and the jth element of the corresponding eigenvector r_p is given by

$$r_{jp} = e^{2\pi ipjh}. \tag{A3.18}$$

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

$$e^{2\pi ip(j\pm 1)h} = e^{\pm 2\pi iph}e^{2\pi ipjh} \tag{A3.19}$$

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 (A3.18) 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

$$\lambda_p = \sum_{k=0}^{m} d_k e^{2\pi ipkh}. \tag{A3.20}$$

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 (A3.20) 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 (A3.18). Note that the jth component of r_p can be rewirtten as

$$r_{jp} = e^{2\pi i p x_j} = \phi_p(x_j)$$

where $x_j = jh$ is the jth grid point and $\phi_p(x) = e^{2\pi i p x}$. The function $\phi_p(x)$ is the pth eigenfunction of the differentiation operator ∂_x on the unit interval with periodic boundary conditions,

$$\partial_x \phi_p(x) = (2\pi i p)\phi_p(x).$$

It is also the eigenfunction of any higher order derivative ∂_x^s, with eigenvalue $(2\pi i p)^s$. 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 12.6 and 13.2.2.

Now consider the symmetric tridiagonal Toeplitz matrix (now truly tridiagonal)

$$A = \begin{bmatrix} d_0 & d_1 & & & & \\ d_1 & d_0 & d_1 & & & \\ & d_1 & d_0 & d_1 & & \\ & & & \ddots & & \\ & & & & & d_1 \\ & & & & d_1 & d_0 \end{bmatrix} \in \mathbb{R}^{m\times m}. \tag{A3.21}$$

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

$$\lambda_p = d_0 + 2d_1\cos(p\pi h), \qquad p = 1,\ 2,\ \ldots,\ m, \tag{A3.22}$$

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

$$r_{jp} = \sin(p\pi jh), \qquad j = 1,\ 2,\ \ldots,\ m. \tag{A3.23}$$

Again it is easy to verify that (A3.22) gives the eigenvalue once the form of the eigenvector is known. In this case we use the fact that, for any p,

$$\sin(p\pi jh) = 0 \quad \text{for } j = 0 \text{ and } j = m+1$$

in order to verify that $(Ar_p)_j = \lambda_p r_{jp}$ for $j = 0$ and $j = m+1$ as well as in the interior (Exercise **??**).

If A is not symmetric,

$$A = \begin{bmatrix} d_0 & d_1 & & & & \\ d_{-1} & d_0 & d_1 & & & \\ & d_{-1} & d_0 & d_1 & & \\ & & & \ddots & & \\ & & & & & d_1 \\ & & & & d_{-1} & d_0 \end{bmatrix} \in \mathbb{R}^{m\times m}. \tag{A3.24}$$

then the eigenvalues are

$$\lambda_p = d_0 + 2d_1\sqrt{d_{-1}/d_1}\,\cos(p\pi h), \qquad p = 1,\ 2,\ \ldots,\ m, \tag{A3.25}$$

and the corresponding eigenvector r_p has jth component

$$r_{jp} = \left(\sqrt{d_{-1}/d_1}\right)^j \sin(p\pi jh), \qquad j = 1,\ 2,\ \ldots,\ m. \tag{A3.26}$$

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

$$\lambda_p = 2i\cos(p\pi h). \tag{A3.27}$$

A3.8 The Gerschgorin 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 A3.8.1 *Let $A \in \mathbb{C}^{m\times m}$ and let D_i be the closed disc 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 discs D_i for $i = 1,\ 2,\ \ldots,\ m$.*

2. *If some set of k overlapping discs is disjoint from all the other discs, then exactly k eigenvalues lie in the union of these k discs.*

Note the following:

- If a disc D_i is disjoint from all other discs then it contains exactly one eigenvalue of A.
- If a disc D_i overlaps other discs then it need not contain any eigenvalues (though the union of the overlapping discs 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 disc radii can be defined by summing elements of columns rather than rows).

For a proof of this theorem see Wilkinson [Wil65], for example.

Example A3.5. Let

$$A = \begin{bmatrix} 5 & 0.6 & 0.1 \\ -1 & 6 & -0.1 \\ 1 & 0 & 2 \end{bmatrix}.$$

Applying the Gerschgoring 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 A3.1(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 A3.1(a), and are

$$\lambda = 1.9639,\ 5.518 \pm 0.6142i.$$

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 A3.1(b). Note that this gives a tighter bound on the eigenvalue near 2 but a larger region around the complex pair.

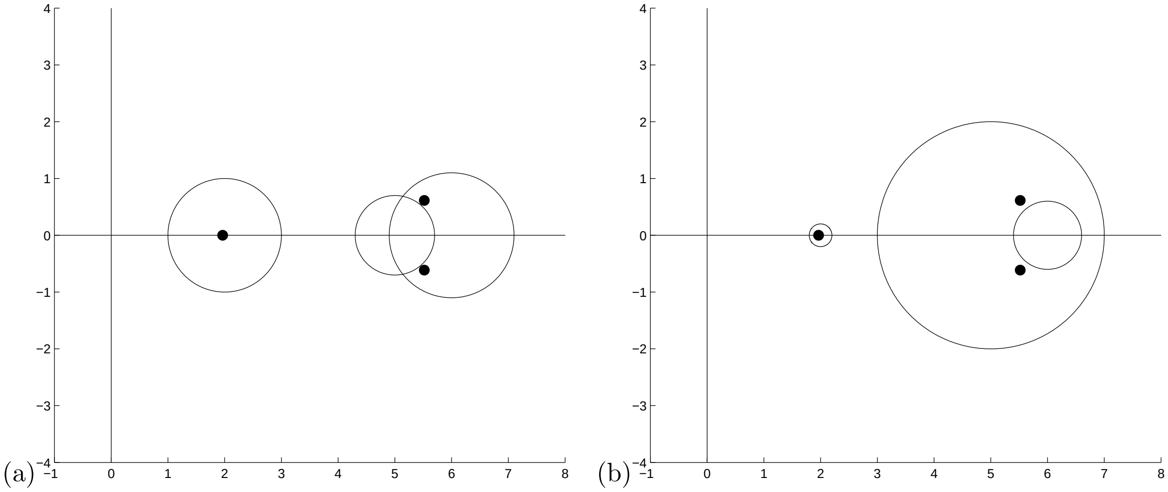


Figure A3.1: Gerschgorin circles containing the eigenvalues of A for Example A3.5.

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

$$PAP^{-1} = \begin{bmatrix} A_{11} & 0 \\ A_{12} & A_{22} \end{bmatrix}$$

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 A3.5 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 discs can be an eigenvalue only if it is on the boundary of *all* discs.

Example A3.6. 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 disc since the matrix is irreducible and the first and last row of A give discs 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 nonsigularity 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 A3.7. Consider the matrix (2.49) with all $\kappa > 0$. The Gerschgorin discs all lie in the left half plane and the discs 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).

A3.9 Inner-product norms

Some standard vector norms and the corresponding matrix norms were introduced in Section A1.3. 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

$$\|u\|_2^2 = u^H u = \sum_{i=1}^m |u_i|^2 = \langle u, u\rangle, \tag{A3.28}$$

where $\langle\cdot,\cdot\rangle$ is the standard inner product. The 2-norm of the matrix A is defined by the formula (**??**) 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

$$\begin{aligned}\|A\|_2 &= \sup_{\|u\|_2=1} (u^H B u)^{1/2}\\ &= \sup_{\|w\|_2=1} (w^H R^H B R w)^{1/2}\\ &= \sup_{\|w\|_2=1} (w^H M w)^{1/2}\\ &= \max_{p=1,2,\ldots,m} |\mu_p|^{1/2} = \sqrt{\rho(A^H A)}.\end{aligned} \tag{A3.29}$$

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 A3.6). 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

$$\begin{aligned}\|A\|_2 &= \|R\Lambda R^{-1}\|_2\\ &\leq \|R\|_2 \|R^{-1}\|_2 \max_{p=1,2,\ldots,m} |\lambda_p| = \kappa_2(R)\rho(A)\end{aligned} \tag{A3.30}$$

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

$$\rho(A) \leq \|A\|_2 \leq \kappa_2(A)\rho(A), \tag{A3.31}$$

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 $U^n = A^n U^0$ (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

$$\|A^n\|_2 \le \kappa_2(R)(\rho(A))^n \tag{A3.32}$$

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.

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 (*i.e.*, if $\kappa_2(R)$ is large). This topic is discussed in more detail in Appendix A4 along with a discussion of the non-diagonalizable (defective) case.

A3.10 Other inner-product norms

If T is any nonsingular matrix then we can define an inner-product norm based on T (the T-norm of A) by

$$\|u\|_T = \|T^{-1}u\|_2 = (u^H T^{-H} T^{-1} u)^{1/2} = \langle T^{-1}u, T^{-1}u\rangle. \tag{A3.33}$$

This satisfies the requirements of a norm summarized in Section A1.3. If we let $G = T^{-H}T^{-1}$ then we can also write $\|u\|_T$ as

$$\|u\|_T = (u^H G u)^{1/2} = \langle u, Gu\rangle. \tag{A3.34}$$

The matrix G is Hermitian positive definite (SPD if T is real) 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 **??**) 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 $U^{n+1} = A_n U^n$ (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 (**??**):

$$\begin{aligned} \|A\|_T &= \sup_{u\neq 0} \frac{\|Au\|_T}{\|u\|_T} = \sup_{w\neq 0} \frac{\|ATw\|_T}{\|Tw\|_T} \\ &= \sup_{w\neq 0} \frac{\|T^{-1}ATw\|_2}{\|w\|_2} \\ &= \|T^{-1}AT\|_2. \end{aligned} \tag{A3.35}$$

Now suppose A is a diagonalizable matrix, wtih $R^{-1}AR = \Lambda$. Then choosing $T = R$ yields $\|A\|_R = \|R^{-1}AR\|_2 = \rho(A)$. Recall that $\|A\| \ge \rho(A)$ 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 A3.10.1 *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 A3.10.2 below.

Note that in general the T-norm, for any nonsingular T, is "equivalent" to the 2-norm in the sense of Section A1.3.1 with the equivalence inequalities

$$\|T\|_2^{-1}\|u\|_2 \le \|u\|_T \le \|T^{-1}\|_2\|u\|_2 \tag{A3.36}$$

for the vector norm and

$$\kappa_2(T)^{-1}\|A\|_2 \le \|A\|_T \le \kappa_2(T)\|A\|_2 \tag{A3.37}$$

for the matrix norm, where $\kappa_2(T)$ is the 2-norm condition number of T. Applying this last inequality in conjunction with Theorem A3.10.1 gives

$$\rho(A) = \|A\|_R \le \kappa_2(R)\|A\|_2, \tag{A3.38}$$

which agrees with the bound (A3.30) obtained earlier.

For general matrices $A \in \mathbb{C}^{m\times m}$ that are not necessarily diagonalizable, Theorem A3.10.1 can be generalized to the following.

Theorem A3.10.2 *(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

$$\|A\|_{T(\epsilon)} < \rho(A) + \epsilon. \tag{A3.39}$$

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 A3.3. Let $R^{-1}AR = J$ have the form (A3.12), 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

$$D_i = \text{diag}(1,\ \delta_i,\ \ldots,\ \delta_i^{k_i-1}).$$

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).

A3.11 Exercises

Exercise A3.1 *True or false? If A and B commute then they are simultaneously diagonalizable.*

Exercise A3.2 *Verify the expressions given in the text for the eigenvalues and eigenvectors of the matrices (A3.16), (A3.21), and (A3.24).*

Exercise A3.3 *Use the Gerschgorin theorem to show that the following matrix is nonsingular:*

$$A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -3 & 1 \\ 0 & 1 & 2 \end{bmatrix}.$$

Appendix A4

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

$$\begin{aligned} |u(t)| \to 0 \text{ as } t \to \infty \text{ for any } u(0) &\iff \text{Re}(a) < 0, \\ |u(t)| \text{ remains bounded as } t \to \infty \text{ for any } u(0) &\iff \text{Re}(a) \le 0, \\ |u(t)| \to \infty \text{ as } t \to \infty \text{ for any } u(0) \neq 0 &\iff \text{Re}(a) > 0 \end{aligned} \tag{A4.1}$$

The first order scalar linear difference equation $U^{n+1} = bU^n$ has the solution $U^n = b^n U^0$ and so

$$\begin{aligned} |U^n| \to 0 \text{ as } n \to \infty \text{ for any } U^0 &\iff |b| < 1, \\ |U^n| \text{ remains bounded as } n \to \infty \text{ for any } U^0 &\iff |b| \le 1, \\ |U^n| \to \infty \text{ as } n \to \infty \text{ for any } U^0 \neq 0 &\iff |b| > 1 \end{aligned} \tag{A4.2}$$

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 A3 on eigen-decompositions 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 (A4.1) and (A4.2) 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 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

$$\begin{aligned} |b| < 1 &\iff \text{Re}(a) < 0, \\ |b| = 1 &\iff \text{Re}(a) = 0, \\ |b| > 1 &\iff \text{Re}(a) > 0. \end{aligned} \tag{A4.3}$$

Similarly, for the system cases we will see that boundedness of the solutions depends on eigenvalues μ of B lying inside the unit circle, or eigenvalues λ of A lying in the left half plane, though the possibility of multiple eigenvalues makes the analysis somewhat more complicated.

A4.1 Powers of matrics

Consider the linear difference equation

$$U^{n+1} = BU^n \tag{A4.4}$$

for some iteration matrix B. 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 (A4.4) we obtain

$$\|U^{n+1}\| \le \|B\|\,\|U^n\|$$

in any norm and hence

$$\|U^n\| \le \|B\|^n \|U^n\|. \tag{A4.5}$$

Alternatively, we can start with $U^{n+1} = B^n U^0$ to obtain

$$\|U^n\| \le \|B^n\|\,\|U^n\|. \tag{A4.6}$$

Since $\|B^n\| \le \|B\|^n$ this again leads to (A4.5), but in many cases $\|B^n\|$ is much smaller than $\|B\|^n$ and the form (A4.6) may be more powerful (see ?? for an example).

If there exists any norm in which $\|B\| < 1$ then (A4.5) shows that $\|U^n\| \to 0$ as 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 ??. 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 $\|B\| = 1$ in some norm then $\|U^n\|$ remains uniformly bounded as $n \to \infty$ in any equivalent norm.

From Theorem A3.10.2 we thus obtain directly the following results.

Theorem A4.1.1 *(a) Suppose $\rho(B) < 1$. Then $\|U^n\| \to 0$ as $n \to \infty$ in any vector norm. (b) Suppose $\rho(B) = 1$ and B 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 the result (a) holds even if B has defective eigenvalues of modulus $\rho(B)$ by choosing $\epsilon < 1 - \rho(B)$ in Theorem A3.10.2.

If B is diagonalizable then the results of Theorem A4.1.1 can also be obtained directly from the eigen-decompositioin of B. Write $B = RMR^{-1}$ where $M = \text{diag}(\mu_1,\ \mu_2,\ \ldots,\ \mu_m)$ is the diagonal matrix of eigenvalues of B and R is the matrix of right eigenvectors of B. Then the nth power of B is given by $B^n = RM^nR^{-1}$ and

$$\begin{aligned}\|U^n\| &\le \|B^n\|\,\|U^0\|\\ &\le \kappa(R)\rho(B)^n\|U^0\|,\end{aligned} \tag{A4.7}$$

where $\kappa(R) = \|R\|\,\|R^{-1}\|$ is the condition number of B 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 $\rho(B) < 1$ (see Example A4.2).

If B is a normal matrix and we use the 2-norm, then $\kappa_2(R) = 1$ and we have the nice result that

$$\|U^n\|_2 \le \rho(B)^n\|U^0\|_2 \qquad \text{if } B \text{ is normal.} \tag{A4.8}$$

For normal matrices, or for those close to normal, $\rho(B)$ gives a good indication of the behavior of $\|U^n\|_2$. For matrices that are far from being normal (in the sense that $\kappa_2(R)$ is large), the spectral radius may give a poor indication of how $\|U^n\|$ behaves. The non-normal case is considered further in Section A4.3.

More detailed information about the behavior of $\|U^n\|$ can be obtained by decomposing U^0 into eigencomponents. Still assuming B 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 B and the vector W^0 is given by $W^0 = R^{-1}U^0$. Multiplying U^0 by B multiplies each r_p by μ_p and so

$$U^n = B^n U^0 = W_1^0 \mu_1^n r_1 + W_2^0 \mu_2^n r_2 + \cdots + W_m^0 \mu_m^n r_m = RM^n W^0. \tag{A4.9}$$

For large n this is dominated by the terms corresponding to the largest eigenvalues, and hence the norm of this vector is proportional to $\rho(B)^n$, at least for generic initial data. This also shows that if $\rho(B) < 1$ then $U^n \to 0$ while if $\rho(B) > 1$ we expect U^n to blow up.

Note that for certain initial data $\|U^n\|$ may behave differently then $\rho(B)^n$, at least in exact arithmetic. For example, if U^0 is void in the dominant eigencomponents then these terms will be missing from (A4.9), and the asymptotic growth or decay rate will be different. In particular, it could happen that $\rho(B) > 1$ and yet $\|U^n\| \to 0$ for special data U^0, if some eigenvalues of B have modulus less than 1 and U^0 contains only these eigencomponents. However, this is generally not releveant for the stability and convergence issues considered in this book, where arbitrary initial data must usually be considered. Moreover, even if U^0 is void of some eigencomponents, rounding errors introduced computationally in each iteration $U^{n+1} = BU^n$ will typically be random and will contain all eigencomponents. The growing modes may start out at the level of rounding error, but if $\rho(B) > 1$ they will grow exponentially and eventually dominate the solution so that the asymptotic behavior will still be governed by $\rho(B)^n$.

If B is defective then we cannot express an arbitrary initial vector U^0 as a linear combiniation of eigenvectors. However, using the Jordan Canonical Form $B = RJR^{-1}$, we can still write $U^0 = RW^0$ where $W^0 = R^{-1}U^0$. The nonsingular matrix R now contains principle vectors as well as eigenvectors, as discussed in Section A3.3.

Example A4.1. Consider the iteration $U^{n+1} = AU^n$ where A is the 3×3 matrix from Example A3.4, 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

$$\begin{aligned} U^1 &= W_1^0 \lambda r_1 + W_2^0 (r_1 + \lambda r_2) + W_3^0 (r_2 + \lambda r_3) \\ &= (W_1^0 \lambda + W_2^0) r_1 + (W_2^0 \lambda + W_3^0) r_2 + W_3^0 \lambda r_3. \end{aligned} \tag{A4.10}$$

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.

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 A4.1.1. If $\rho(B) < 1$ then $\|B^n\| \to 0$ even if B has defective eigenvalues of modulus $\rho(B)$, since the exponential decay overpowers the algebraic growth. However, if $\rho(B) = 1$ then the boundedness of $\|B^n\|$ depends on whether there are defective eigenvalues of modulus 1. If so, then $\|B^n\|$ grows algebraically (but not exponentially).

Recall also from Theorem A3.10.2 that in this latter case we can find, for any $\epsilon > 0$, a norm in which $\|B\| < 1 + \epsilon$. This implies that $\|B^n\| < (1+\epsilon)^n$. 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 only have 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 $B = RJR^{-1}$ then $B^n = RJ^nR^{-1}$, and 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 (A3.9) of order k,

$$J(\lambda, k) = \lambda I_k + S_k$$

where S_k is the shift matrix (A3.10). Powers of $J(\lambda, k)$ can be found using the binomial expansion and the fact that I_k and S_k commute,

$$\begin{aligned} J(\lambda,k)^n &= (\lambda I_k + S_k)^n \\ &\lambda^n I_k + n\lambda^{n-1}S_k + \left(\begin{array}{c} n \\ 2 \end{array}\right)\lambda^{n-2}S_k^2 + \left(\begin{array}{c} n \\ 3 \end{array}\right)\lambda^{n-3}S_k^3 \\ &+ \cdots + \left(\begin{array}{c} n \\ n-1 \end{array}\right)\lambda S_k^{n-1} + S_k^n. \end{aligned} \tag{A4.11}$$

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 (A4.11) always terminates after at most k terms, and when $n > k$ reduces to

$$\begin{aligned} J(\lambda,k)^n = \lambda^n I_k + n\lambda^{n-1}S_k + \left(\begin{array}{c} n \\ 2 \end{array}\right)\lambda^{n-2}S_k^2 + \left(\begin{array}{c} n \\ 3 \end{array}\right)\lambda^{n-3}S_k^3 \\ + \cdots + \left(\begin{array}{c} n \\ k-1 \end{array}\right)\lambda^{n-k+1}S_k^{k-1}. \end{aligned} \tag{A4.12}$$

Since $\left(\begin{array}{c} n \\ j \end{array}\right) = O(n^j)$ 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 A3.4 where $k = 3$, we have

$$\begin{aligned} J(\lambda,3)^n &= \lambda^n I_3 + n\lambda^{n-1}S_3 + \frac{n(n-1)}{2}\lambda^{n-2}S_3^2 \\ &= \left[\begin{array}{ccc} \lambda^n & n\lambda^{n-1} & \frac{n(n-1)}{2}\lambda^{n-2} \\ 0 & \lambda^n & n\lambda^{n-1} \\ 0 & 0 & \lambda^n \end{array}\right]. \end{aligned} \tag{A4.13}$$

This shows that $\|J^n\| \approx n^2\lambda^n$, the same result obtained in Example A4.1,

If B is not normal, *i.e.*, if $B^HB \neq BB^H$ (see Section **??**), then $\|B\|_2 > \rho(B)$ and $\rho(B)^n$ may not give a very good indication of the behavior of $\|B^n\|$. If B is diagonalizable, then we have

$$\|B^n\|_2 \leq \|R\|_2\|M^n\|_2\|R^{-1}\|_2 \leq \kappa_2(R)\rho(B)^n, \tag{A4.14}$$

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 may tell us little or nothing about how $\|B^n\|_2$ is behaving for the values of n we care about in a computation. See Section A4.3 for more about this.

A4.2 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 **??**.

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

$$e^{\Lambda t} = \text{diag}(e^{\lambda_1 t},\ e^{\lambda_2 t},\ \ldots,\ e^{\lambda_m t}). \tag{A4.15}$$

Then we have $v(t) = e^{\Lambda t}v(0)$ and hence

$$u(t) = Rv(t) = Re^{\Lambda t}R^{-1}u(0),$$

so

$$u(t) = e^{At}u(0) \tag{A4.16}$$

where

$$e^{At} = Re^{\Lambda t}R^{-1}. \tag{A4.17}$$

This is one natural way to define the *matrix exponential* e^{At}, at least in the diagonalizable case. Another way to define it is by the Taylor series

$$e^{At} = I + At + \frac{1}{2!}A^2t^2 + \frac{1}{3!}A^3t^3 + \cdots = \sum_{j=0}^{\infty} \frac{1}{j!}A^jt^j. \tag{A4.18}$$

These definitions agree since all powers A^j have the same eigenvector matrix R and so (A4.18) gives

$$\begin{aligned} e^{At} &= R\left[I + \Lambda t + \frac{1}{2!}\Lambda^2t^2 + \frac{1}{3!}\Lambda^3t^3 + \cdots\right]R^{-1} \\ &= Re^{\Lambda t}R^{-1}, \end{aligned} \tag{A4.19}$$

resulting in (A4.17). To go from the first to second line of (A4.19), 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 $e^{\lambda_j t}$.

Computationally there are many other ways to compute the matrix e^{At} (if this matrix is actually needed) that are often better in practice than either of the definitions given above. See the review paper by Moler and van Loan [MV78] and the recent update [MV03] for some alternatives and more discussion of the issues.

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 $|e^{\lambda_j t}| \leq 1$ for all j.

It is useful to introduce the *spectral abscissa* $\alpha(A)$, defined by

$$\alpha(A) = \max_{1\leq j\leq m} \text{Re}(\lambda_j). \tag{A4.20}$$

Then $u(t)$ remains bounded provided $\alpha(A) \leq 0$ and $u(t) \to 0$ if $\alpha(A) < 0$.

Note that for integer values of $t = n$, we have $e^{An} = B^n$ with $B = e^A$ and $\rho(B) = e^{\alpha(A)}$, so this result is consistent with what was found in the last sections for matrix powers.

If A is not diagonalizable, then the case $\alpha(A) = 0$ is more subtle, as is the case $\rho(B) = 1$. If A has a defective eigenvalue λ with $\text{Re}(\lambda) = 0$ then the solution may still grow, though 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

$$e^{At} = Re^{Jt}R^{-1}.$$

If J has the block structure (A3.12) then e^{Jt} is also block diagonal,

$$e^{Jt} = \begin{bmatrix} e^{J(\lambda_1,k_1)t} & & & \\ & e^{J(\lambda_2,k_2)t} & & \\ & & \ddots & \\ & & & e^{J(\lambda_s,k_s)t} \end{bmatrix}. \tag{A4.21}$$

For a single Jordan block the Taylor series expansion (A4.18) can be used in conjunction with the expansion (A4.11) for powers of the Jordan block. We find that

$$\begin{aligned} e^{J(\lambda,k)t} &= \sum_{j=0}^{\infty} \frac{t^j}{j!}\left[\lambda^j I + j\lambda^{j-1}S_k + \binom{j}{2}\lambda^{j-2}S_k^2 + \cdots + \binom{j}{j-1}\lambda S_k^{j-1} + S_k^j\right] \\ &= \sum_{j=0}^{\infty}\frac{t^j}{j!}\lambda^j I + t\sum_{j=1}^{\infty}\frac{t^{j-1}}{(j-1)!}\lambda^{j-1}S_k + \frac{t^2}{2!}\sum_{j=2}^{\infty}\frac{t^{j-2}}{(j-2)!}\lambda^{j-2}S_k^2 + \cdots \\ &= e^{\lambda t}I + te^{\lambda t}S_k + \frac{t^2}{2!}e^{\lambda t}S_k^2 + \cdots + \frac{t^{(k-1)}}{(k-1)!}e^{\lambda t}S_k^{k-1} \\ &= \begin{bmatrix} e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \cdots & \frac{t^{(k-1)}}{(k-1)!}e^{\lambda t} \\ & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \cdots \\ & & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} \\ & & & \ddots & \ddots \\ & & & & e^{\lambda t} \end{bmatrix}. \end{aligned} \tag{A4.22}$$

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 powers B^n. If $\mathrm{Re}(\lambda) < 0$ then $t^j e^{\lambda t} \to 0$ in spite of 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 A4.1.1.

Theorem A4.2.1 *Let $A \in \mathbb{C}^{m\times m}$ be an arbitrary square matrix, and let $\alpha(A) = \max Re(\lambda)$ 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 $Re(\lambda) = 0$ 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 $\|e^{At}\|_2 = \|e^{\Lambda t}\|_2 = e^{\alpha(A)t}$ since $\|R\|_2 = 1$. In this case the spectral abscissa gives precise information on the behavior of e^{At}. If A is non-normal 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 $\rho(B)^n$ may not give a good indication of $\|B^n\|$ if B is not normal.

A4.3 Non-normal 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:

$$\begin{aligned} \|A^n\| &\le \kappa(R)\|J^n\| \quad \text{in general, and} \\ \|A^n\| &\le \kappa(R)\rho(A)^n \quad \text{if } A \text{ is diagonalizable,} \end{aligned} \tag{A4.23}$$

$$\begin{aligned} \|e^{At}\| &\le \kappa(R)\|e^{Jt}\| \quad \text{in general, and} \\ \|e^{At}\| &\le \kappa(R)e^{\alpha(A)t} \quad \text{if } A \text{ is diagonalizable.} \end{aligned} \tag{A4.24}$$

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.

In this section we consider the influence of the factor $\kappa(R)$ on these bounds, which can be an important factor whether or not J is diagonal.

If A is a normal matrix, $A^HA = AA^H$, then A is diagonalizable and the eigenvector matrix R can be chosen as an unitary matrix, for which $R^HR = I$ 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 (A4.23) 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 A4.2. Consider the non-normal matrix

$$B = \begin{bmatrix} 0.8 & 100 \\ 0 & 0.9 \end{bmatrix}. \tag{A4.25}$$

This matrix is diagonalizable and the spectral radius is $\rho(B) = 0.9$. We expect $\|B^n\| \sim C(0.9)^n$ 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 $U^n = B^nU^0$ for $n = 1,\ 2,\ \ldots$ we find

$$U^0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad U^1 = \begin{bmatrix} 100 \\ 0.9 \end{bmatrix}, \quad U^2 = \begin{bmatrix} 170 \\ 0.81 \end{bmatrix}, \quad U^3 = \begin{bmatrix} 217 \\ 0.729 \end{bmatrix}, \quad \cdots$$

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 A4.1 shows $\|U^n\|_2$ for the $n = 1 : 30$ 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.

A4.3.1 Measures of non-normality

The size of the condition number $\kappa(R)$ is one way to measure the "non-normality" of a diagonalizable matrix. This measure isn't meaningful for a defective matrix however (for example, if A is a nontrivial Jordan block then it is non-normal but $R = I$ in the Jordan form so $\kappa(R) = 1$).

A more robust way to measure non-normality is in terms of the *Schur decomposition* of the matrix.

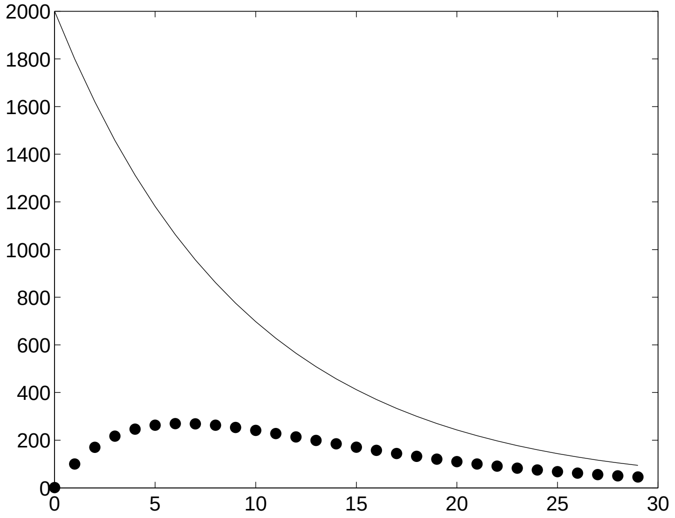


Figure A4.1: The points show $\|U^n\|_2$ for Example A4.2 and the line shows the upper bound $2000(0.9)^n$.

Theorem A4.3.1 *Every matrix* $A \in \mathbb{C}^{m\times m}$ *can be written as*

$$A = QTQ^H$$

where Q *is a unitary matrix* ($Q^HQ = I$) *and* T *is upper triangular.*

The diagonal elements of T are the eigenvalues of A. (In fact Q can be chosen to put these eigenvalues on the diagonal in any desired order, *e.g.*, so that $|t_{11}| \leq |t_{22}| \leq \cdots \leq |t_{mm}|$.)

If A is normal then T can be chosen to be diagonal and Q is the matrix of eigenvectors. More generally the Schur decomposition gives the "best attempt" at diagonalizing A by means of a unitary similarity transformation. The size of the off-diagonal elements of T gives a measure of the non-normality of A. The value

$$\sum_{i=1}^{m} \sum_{j=i+1}^{m} |T_{ij}|^2 \tag{A4.26}$$

is called A's *departure from normality.*

A4.4 Pseudo-eigenvalues

Various tools have been developed to better understand the behavior of matrix powers or exponentials in the case of non-normal matrices. One powerful approach is to investigate the *pseudospectrum* of the matrix. Roughly speaking, this is the set of eigenvalues of all "nearby" matrices. For a highly non-normal matrix a small perturbation to the matrix can give a very large change in the eigenvalues of the matrix. For example, perturbing the matrix (A4.25) to

$$\tilde{A} = \left[\begin{array}{cc} 0.8 & 100 \\ 0.001 & 0.9 \end{array}\right] \tag{A4.27}$$

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 Λ_ϵ of A for $\epsilon = 10^{-3}$.

The eigenvalues of A are isolated points in the complex plane where $(A - zI)$ 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 Λ_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 A4.4.1 *For each $\epsilon \geq 0$, the ϵ-pseudospectrum $\Lambda_\epsilon(A)$ 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 $\|E\| \leq \epsilon$,
(b) $\|Au - zu\| \leq \epsilon$ for some vector u with $\|u\| = 1$,
(c) $\|(A - zI)^{-1}\| \geq \epsilon^{-1}$.

In all these conditions the 2-norm is used. Condition (a) is the easiest to understand and the one already illustrated above by example: z is an ϵ-pseudo-eigenvalue of A if it is a genuine eigenvalue of some ϵ-sized perturbation of A. Condition (b) says that z is an ϵ-pseudo-eigenvalue if there is a unit vector that is almost an eigenvector for this z. Condition (c) is a condition on the *resolvent* $(A-zI)^{-1}$, a matrix-valued function of z that plays an important role in linear analysis. Note that if z is an eigenvalue of A then $A - zI$ is singular and by convention we say that $\|(A - zI)^{-1}\| = \infty$ in this case. The value z is an ϵ-pseudo-eigenvalue of A if the resolvent is sufficiently large at z. This fits with the notion of expanding the singular points λ_i into regions Λ_ϵ where $A - zI$ is nearly singular.

The MATLAB package `eigtool` developed by Wright [**?**] provides tools for computing and plotting the pseudospectra of matrices. See the recent book by Trefethen and Embree [TE05] for an in-depth discussion of pseudospectra with many examples of their use.

A4.5 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,

$$\|A^n\| \leq C \quad \text{for all } n, \tag{A4.28}$$

for some constant C. If A is normal then it fact $\|A^n\| \leq \|A\|$. 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 (A4.28) 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 Δx and/or time step Δt leads to a particular matrix A, but to study stability and prove convergence we need to let $\Delta x, \Delta t \to 0$ and study the whole family of resulting matrices.

Let $\mathcal{F}$ represent a family of matrices, say all the matrices that result 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 (A4.28) 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.

When the matrices are not normal it can be more difficult to establish such a bound. Obviously a necessary condition is that $\rho(A) \leq 1$ for all $A \in \mathcal{F}$, and that any eigenvalues of modulus 1 must be non-defective. 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

$$A_\epsilon = \begin{bmatrix} 1-\epsilon & 1 \\ 0 & 1-\epsilon \end{bmatrix}$$

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 Kriess Matrix Theorem:

Theorem A4.5.1 *The following conditions on a family $\mathcal{F}$ of matrices are equivalent:*

(a) There exists a constant C such that $\|A^n\| \leq C$ 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 $(A - zI)^{-1}$ exists and is bounded by

$$\|(A - zI)^{-1}\| \leq \frac{C_1}{|z| - 1}. \tag{A4.29}$$

(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) $\|S\| \leq C_2$, $\|S^{-1}\| \leq C_2$,
ii) $B = S^{-1}AS$ is upper triangular with off-diagonal elements bounded by

$$|b_{ij}| \leq C_3 \min(1 - |b_{ii}|,\ 1 - |b_{jj}|). \tag{A4.30}$$

(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

$$\begin{aligned} &C_4^{-1} I \leq G \leq C_4 I \\ &A^H G A \leq G. \end{aligned} \tag{A4.31}$$

In condition (d) we say that two Hermitian matrices A and B satisfy $A \leq B$ if $u^H A u \leq u^H B u$ for any vector u. This condition can be rewritten in a more familiar form as

(d') There exists a constant C_5 so that for each $A \in \mathcal{F}$ there is a nonsingular matrix T such that

$$\|A\|_T \leq 1 \quad \textit{and} \quad \kappa(T) \leq C_5. \tag{A4.32}$$

Here the T-norm of A is defined as in (A3.33) in terms of the 2-norm,

$$\|A\|_T = \|T^{-1}AT\|.$$

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

$$\|A^n\|_T \leq 1 \quad \text{for all } n \geq 0.$$

From this we can obtain uniform power boundedness by noting that

$$\|A^n\| \leq \kappa(T)\|A^n\|_T \leq C_5.$$

There are several other equivalent conditions that have been identified and are sometimes more useful in practice. The equivalence of the conditions in Theorem A4.5.1 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 Richtmyer & Morton [RM67] or Strikwerda & Wade [SW97].

Appendix A5

Linear Differential and Difference Equations

Consider the rth order ordinary differential equation

$$a_0 v(t) + a_1 v'(t) + a_2 v''(t) + \cdots + a_r v^{(r)}(t) = 0, \quad \text{for } t \geq 0. \tag{A5.1}$$

This is a constant coefficient linear ODE, and is said to be *homogeneous* since the right hand side is zero. In this Appendix we first review the standard procedure for finding the general solution to this equation, and also for finding the unique solution that satisfies the equation along with r specified initial conditions, $v(0),\ v'(0),\ v''(0),\ \ldots,\ v^{(r-1)}(0)$. By rewriting this rth order equation as a system of r first order equations, we can obtain an alternative view of the solution in terms of the matrix exponential derived in Appendix A4.

Similar techniques are then applied to the linear difference equation (or recurrence relation)

$$a_0 V^n + a_1 V^{n+1} + a_2 V^{n+2} + \cdots + a_r V^{n+r} = 0, \quad \text{for } n \geq 0 \tag{A5.2}$$

to obtain both the general solution and the unique solution that satisfies r initial conditions in which $V^0,\ V^1,\ \ldots,\ V^{r-1}$ are specified. By rewriting this as a system of r first order equations we will see that this solution can also be obtained in terms of the expressions for matrix powers derived in Appendix A4.

We present these theories in parallel since they have so much in common. The main result needed in the text is the general solution to a linear difference equation. This is needed in order to study the stability of finite difference methods, particularly linear multistep methods where higher order recurrence relations are used to approximate first order differential euqations. Of particular interest is the asymptoptic behavior of solutions as $n \to \infty$. In Chapters 7 and 8 we study zero-stability and absolute stability of linear multistep methods. Stability depends on the solution to some difference equation of the form (A5.2) remaining bounded as $n \to \infty$ for arbitrary initial data.

The key to solving either (A5.1) or (A5.2) is to determine the roots $\zeta_1,\ \zeta_2,\ \ldots,\ \zeta_r$ of the *characteristic polynomial*

$$p(\zeta) = a_0 + a_1\zeta + a_2\zeta^2 + \cdots + a_r\zeta^r = \sum_{j=0}^{r} a_j \zeta^j. \tag{A5.3}$$

We will see that solutions to (A5.1) remain bounded as $t \to \infty$, for any set of initial data, provided that

a) All roots ζ_j satisfy $\text{Re}(\zeta_j) \leq 0, \quad j = 1,\ 2,\ \ldots,\ r$,
b) Any root with $\text{Re}(\zeta_j) = 0$ has multiplicity 1.

For the difference equation (A5.2), we will see that all solutions remain bounded as $n \to \infty$, for any set of initial data, provided that

a) All roots ζ_j satisfy $|\zeta_j| \le 1 \;\; j = 1,\ 2,\ \ldots,\ r$,
b) Any root with $|\zeta_j| = 1$ has multiplicity 1.

Later in this Appendix we will see how these conditions follow directly from Theorems A4.1.1 and A4.2.1 respectively.

A5.1 Linear differential equations

We begin with the standard approach to solving (A5.1), which is to suppose that (A5.1) has a solution of the form $v(t) = e^{\zeta t}$ for some $\zeta \in\mathbb{C}$. Since the jth derivative is $v^{(j)}(t) = \zeta^j e^{\zeta t}$, inserting this in (A5.1) yields

$$\sum_{j=1}^{r} a_j \zeta^j v(t) = 0.$$

This function of t can be identically zero only if the constant value $\sum a_j\zeta^j$ is zero, and hence $e^{\zeta t}$ solves (A5.1) only if ζ is a root of the characteristic polynomial $p(\zeta)$ from (A5.3). If the r roots of this polynomial are distinct, $\zeta_i \ne \zeta_j$ for $i \ne j$, then the general solution to (A5.1) is an arbitrary linear combination of exponentials of the form

$$v(t) = c_1 e^{\zeta_1 t} + c_2 e^{\zeta_2 t} + \cdots + c_r e^{\zeta_r t} = \sum_{j=1}^{r} c_j e^{\zeta_j t}. \tag{A5.4}$$

If a root is repeated, say $\zeta_1 = \zeta_2$, then $e^{\zeta_1 t}$ and $e^{\zeta_2 t}$ are not linearly independent. In this case the function $te^{\zeta_1 t}$ can also be shown to be a solution to the differential equation, and is linearly independent from $e^{\zeta_1 t}$, and so $c_1 e^{\zeta_1 t} + c_2 e^{\zeta_2 t}$ is replaced by $c_1 e^{\zeta_1 t} + c_2 t e^{\zeta_1 t}$. If the multiplicity is $k > 2$, then additional linearly indpendent solutions are $t^2 e^{\zeta_1 t},\ \ldots,\ t^{k-1} e^{\zeta_1 t}$.

To detemine the particular solution for specified initial data we set up an $r \times r$ linear system of equations for $c_1,\ \ldots,\ c_r$ by requiring that the function $v(t)$ in (A5.4) satisfy the initial conditions.

To relate the solution just found to the matrix exponential discussed in Section A4.2, we rewrite the rth order differential equation (A5.1) as a system of r first order differential equations. Let $u_1(t) = v(t)$, $u_2(t) = v'(t),\ \ldots,\ u_r(t) = v^{(r-1)}(t)$. Then these r functions satisfy the system of ODEs

$$\begin{aligned} u_1'(t) &= u_2(t), \\ u_2'(t) &= u_3(t), \\ &\vdots \\ u_{r-1}'(t) &= u_r(t), \\ u_r'(t) &= -\frac{1}{a_r}(a_0 u_1(t) + a_1 u_2(t) + \cdots + a_{r-1} u_r(t)). \end{aligned} \tag{A5.5}$$

This system can be written as

$$u'(t) = Cu(t) \tag{A5.6}$$

where $u(t)$ denotes the vector with components $u_j(t)$ and C is the *companion matrix*

$$C = \begin{bmatrix} 0 & 1 & & & & \\ & 0 & 1 & & & \\ & & 0 & 1 & & \\ & & & \ddots & \ddots & \\ & & & & 0 & 1 \\ -\frac{a_0}{a_r} & -\frac{a_1}{a_r} & -\frac{a_2}{a_r} & \cdots & & -\frac{a_{r-1}}{a_r} \end{bmatrix}. \tag{A5.7}$$

The initial data is

$$u(0) = [v(0),\ v'(0),\ v''(0),\ \ldots,\ v^{(r-1)}(0)]^T.$$

The solution to this system of ODEs is

$$u(t) = e^{Ct}u(0)$$

where the matrix exponential is described in Section A4.2. The eigenvalues of C satisfy the characteristic equation $\tilde{p}(\lambda)$ for the matrix. It can be shown that

$$\begin{aligned}\tilde{p}(\lambda) &= \frac{a_0}{a_r} + \frac{a_1}{a_r}\lambda + \frac{a_2}{a_r}\lambda^2 + \cdots + \frac{a_{r-1}}{a_r}\lambda^{r-1} + \lambda^r \\ &= \frac{1}{a_r}p(\lambda)\end{aligned} \tag{A5.8}$$

where $p(\lambda)$ is the characteristic polynomial (A5.3) for the rth order differential equation. Hence the eigenvalues of C are the roots $\zeta_1,\ \ldots,\ \zeta_r$ previously used to solve the ODE. If these roots are distinct then C is diagonalizable, $C = R\Lambda R^{-1}$, where $\Lambda = \text{diag}(\zeta_1, \ldots, \zeta_r)$. The solution is

$$u(t) = Re^{\Lambda t}R^{-1}u(0). \tag{A5.9}$$

Multiplying this out shows that each component of $u(t)$ (and in particular the first component $u_1(t) = v(t)$) is some linear combination of $e^{\zeta_1 t},\ \ldots,\ e^{\zeta_r t}$, as already found in (A5.4).

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 $[1,\ \zeta_j,\ \zeta_j^2,\ \ldots,\ \zeta_j^{r-1}]^T$. Hence if $p(\zeta)$ has repeated roots the Jordan form $A = RJR^{-1}$ must be used and the solution

$$u(t) = Re^{Jt}R^{-1}u(0) \tag{A5.10}$$

will involve exponentiated Jordan blocks of the form shown in (A4.22). Multiplying this out shows that $v(t)$ will then involve factors $e^{\zeta_j t},\ te^{\zeta_j t},\ \ldots,\ t^{k-1}e^{\zeta_j t}$, as discussed above.

A5.2 Linear difference equations

We now turn to solving the linear difference equation (A5.2). Just as with the rth order differential equation, this rth order difference equation can be solved either by "guessing" the correct form of the solution and plugging it into the difference equation, or more systematically by rewriting (A5.2) as a first order system of equations and explicitly computing the matrix power. We will show both approaches. Note how closely the discussion below parallels that of Section A5.1.

We start by guessing that (A5.2) has solutions of the form $V^n = \zeta^n$ if ζ is chosen appropriately (where V^n has an index and ζ^n is the nth power of ζ). Plugging this into (A5.2) and factoring out the common factor of ζ^n gives

$$\left(\sum_{j=0}^{r} a_j\zeta^j\right)\zeta^n = 0.$$

This can be identically zero only if $\sum a_j\zeta^j = 0$, so $V^n = \zeta^n$ solves the differential equation only if ζ is a root of the characteristic polynomial $p(\zeta)$ in (A5.3). If the roots ζ_j are distinct then the general solution to (A5.2) is a linear combination of such solutions,

$$V^n = c_1\zeta_1^n + c_2\zeta_2^n + \cdots + c_r\zeta_r^n. \tag{A5.11}$$

If a root is repeated, say $\zeta_1 = \zeta_2$, then ζ_1^n and ζ_2^n are not linearly independent. In this case the function $n\zeta_1^n$ can also be shown to be a solution to the difference equation, and is linearly independent from ζ_1^n, and so $c_1\zeta_1^n + c_2\zeta_2^n$ is replaced by $c_1\zeta_1^n + c_2n\zeta_1^n$. If the multiplicity is $k > 2$, then additional linearly indpendent solutions are $n^2\zeta_1^n,\ \ldots,\ n^{k-1}\zeta_1^n$.

To detemine the particular solution for specified initial data we set up an $r \times r$ linear system of equations for $c_1, \ \ldots, \ c_r$ by requiring that the function V^n in (A5.11) satisfy the initial conditions.

To relate the solution just found to powers of a matrix as discussed in Section A4.1, we rewrite the rth order difference equation (A5.2) as a system of r first order difference equations. Let $U_1^n = V^n$, $U_2^n = V^{n+1}, \ \ldots, \ U_r^n = V^{n+r-1}$. Then these r functions satisfy the system of equations

$$\begin{aligned} U_1^{n+1} &= U_2^n, \\ U_2^{n+1} &= U_3^n, \\ &\vdots \\ U_{r-1}^{n+1} &= U_r^n, \\ U_r^{n+1} &= -\frac{1}{a_r}(a_0 U_1^n + a_1 U_2^n + \cdots + a_{r-1} U_r^n). \end{aligned} \tag{A5.12}$$

This system can be written as

$$U^{n+1} = CU^n \tag{A5.13}$$

where U^n denotes the vector with components U_j^n and C is the companion matrix of (A5.7). The initial data is now

$$U^0 = [V^0, \ V^1, \ \ldots, \ V^{r-1}]^T.$$

The solution to this system of first order difference equations is

$$U^n = C^n U^0 = RJ^n R^{-1} U^0 \tag{A5.14}$$

where $C = RJR^{-1}$ is the JCF of C (with J diagonal if and only if the roots of $p(\zeta)$ are distinct). Multiplying this out gives the solution and using (A4.13) we see that each component of U^n (and in particular the first component $U_1^n = V^n$) is a linear combination of functions of the form ζ_j^n, with additional functions $n\zeta_j^n, \ \ldots, \ n^{k-1}\zeta_j^n$ if ζ_j is a root of multiplicity k.

A5.3 Exercises

Exercise A5.1 *(a)Find the general solution of the linear difference equation*

$$U^{n+2} - U^{n+1} + 0.25U^n = 0.$$

(b) Solve a linear system of 2 equations to determine the particular solution with initial data $U^0 = 2, \ U^1 = 3$. *What is* U^{10}*?*

(c) Write down the companion matrix C *for this problem and determine the the matrix* C^n *by using the Jordan Canonical Form. Use this to determine the solution of part (b).*

Exercise A5.2 *Repeat parts (a), (b), (c) of Exercise A5.1 for the difference equation*

$$2U^n - U^{n+1} - 2U^{n+2} + U^{n+3} = 0$$

with initial data $U^0 = 2$, $U^1 = 7$, $U^2 = 5$.

Bibliography

[ALY88] L. M. Adams, R. J. LeVeque, and D. M. Young. Analysis of the SOR iteration for the 9-point Laplacian. *SIAM J. Num. Anal.*, 25:1156–1180, 1988.

[AMR88] U. Ascher, R. Mattheij, and R. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations.* Prentice-Hall, 1988.

[AP98] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, Philadelphia, 1998.

[BEM01] W. L. Briggs, V. Emden Henson, and S. F. McCormick. *A Multigrid Tutorial, Second Edition.* SIAM, Philadelphia, 2001.

[Bri87] W. L. Briggs. *A Multigrid Tutorial.* SIAM, 1987.

[But87] J. C. Butcher. *The Numerical Analysis of Ordinary Differential Equations : Runge-Kutta and General Linear Methods.* J. Wiley, Chichester, 1987.

[CFL28] R. Courant, K. O. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Ann.*, 100:32–74, 1928.

[CFL67] R. Courant, K. O. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM Journal*, 11:215–234, 1967.

[CH52] C. F. Curtiss and J. O. Hirschfelder. Integration of stiff equations. *Proc. Nat. Acad. Sci. U.S.A.*, 38:235–23, 1952.

[CHQZ88] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral Methods in Fluid Dynamics.* Springer, New York, 1988.

[CL55] E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations.* McGraw-Hill, New York, 1955.

[CL88] T. F. Coleman and C. F. Van Loan. *Handbook for matrix computations.* SIAM, 1988.

[Dav63] P. J. Davis. *Interpolation and Approximation.* Blaisdell Pub. Co., New York, 1963.

[DER86] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, 1986.

[DR56] 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.

[Dur99] D. R. Durran. *Numerical Methods for Wave Equations in Geophysical Fluid Dynamics.* Springer, New York, 1999.

[For96] B. Fornberg. *A Practical Guide to Pseudospectral Methods.* Cambridge University Press, 1996.

[FW60] G. E. Forsyth and W. R. Wasow. *Finite Difference Methods for Partial Differential Equations.* Wiley, 1960.

[Gea71] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice-Hall, 1971.

[GL81] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive-definite Systems.* Prentice-Hall, 1981.

[GL96] G. H. Golub and C. F. Van Loan. *Matrix computations.* Johns Hopkins Press, third edition, 1996.

[GO77] D. Gottlieb and S. A. Orszag. *Numerical Analysis of Spectral Methods.* CBMS-NSF Regional Conference Series in Applied Mathematics, #26, SIAM, Philadelphia, 1977.

[GO89] G. H. Golub and D. P. O'Leary. Some history of the conjugate gradient and Lanczos algorithms: 1948-1976. *SIAM Review*, 31:50–102, 1989.

[GO92] G. H. Golub and J. M. Ortega. *Scientific computing and differential equations : an introduction to numerical methods.* Academic Press, 1992.

[Gre97] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia, 1997.

[Hen62] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations.* John Wiley and Sons, New York, 1962.

[HNW87] E. Hairer, S. P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations I. Nonstiff Problems.* Springer-Verlag, Berlin, Heidelberg, 1987.

[HNW93] E. Hairer, S. P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems.* Springer-Verlag, New York, 1993.

[HS52] M. R. Hestenes and E. Stiefel. Method of conjugate gradients for solving linear equations. *J. Res. of the National Bureau of Standards*, 49:409–436, 1952.

[HY81] L. A. Hageman and D. M. Young. *Applied Iterative Methods.* Academic Press, 1981.

[Ise96] A. Iserles. *Numerical Analysis of Differential Equations.* Cambridge University Press, Cambridge, 1996.

[Jes84] D. C. Jesperson. Multigrid methods for partial differential equations. In G. H. Golub, editor, *Studies in Numerical Analysis*, pages 270–317. MAA Studies in Mathematics, Vol. 24, 1984.

[Joh87] C. Johnson. *Numerical solution of Partial Differential Equations by the Finite Element Method.* Cambridge University Press, 1987.

[KC81] J. Kevorkian and J. D. Cole. *Perturbation Methods in Applied Mathematics.* Springer, New York, 1981.

[Kel76] H. B. Keller. *Numerical Solution of Two Point Boundary Value Problems.* SIAM, 1976.

[Kev90] J. Kevorkian. *Partial Differential Equations.* Wadsworth & Brooks/Cole, 1990.

[Lam73] J. D. Lambert. *Computational Methods in Ordinary Differential Equations.* Wiley, 1973.

[Lax72] P. D. Lax. *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves.* SIAM Regional Conference Series in Applied Mathematics, #11, 1972.

[Lel92] S. K. Lele. Compact difference schemes with spectral-like resolution. *J. Comput. Phys.*, 103:16–42, 1992.

[LeV90] R. J. LeVeque. *Numerical Methods for Conservation Laws.* Birkhäuser-Verlag, 1990.

[LeV02] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge University Press, 2002.

[Loa97] C. F. Van Loan. *Introduction to Scientific Computing.* Prentice Hall, Upper Saddle River, NJ, 1997.

[LT88] R. J. LeVeque and L. N. Trefethen. Fourier analysis of the SOR iteration. *IMA J. Numer. Anal.*, 8:273–279, 1988.

[LT98] D. Levy and E. Tadmor. From semidiscrete to fully discrete: stability of Runge-Kutta schemes by the energy method. *SIAM Review*, 40:40–73, 1998.

[McC89] S. F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations.* SIAM, 1989.

[MG80] A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations.* Wiley, 1980.

[MM94] K. W. Morton and D. F. Mayers. *Numerical Solution of Partial Differential Equations.* Cambridge University Press, Cambridge, 1994.

[MV78] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20:801–836, 1978.

[MV03] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45:3–49, 2003.

[RM67] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-value Problems.* Wiley-Interscience, 1967.

[RT90] S. C. Reddy and L. N. Trefethen. Lax-stability of fully discrete spectral methods via stability regions and pseudo-eigenvalues. *Comp. Meth. Appl. Mech. Eng.*, 80:147–164, 1990.

[SF73] G. Strang and G. J. Fix. *An Analysis of the Finite Element Method.* Prentice-Hall, 1973.

[She94] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report, 1994.

[Str68] G. Strang. On the construction and comparison of difference schemes. *SIAM J. Num. Anal.*, 5:506–517, 1968.

[Str89] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations.* Wadsworth & Brooks/Cole, 1989.

[SW97] 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*, volume 38, pages 329–360, Warsaw, 1997. Banach Center Publications, Inst. Math. Pol. Acad. Sciences.

[Swa84] Paul N. Swarztrauber. Fast Poisson solvers. In Gene H. Golub, editor, *Studies in Numerical Analysis*, volume 24, pages 319–370, Washington, D.C., 1984. The Mathematical Association of America.

[TB97] L. N. Trefethen and D. Bau. *Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[TE05] L. N. Trefethen and M. Embree. *Spectra and Pseudospectra.* Princeton University Press, 2005.

[Tre00] L. N. Trefethen. *Spectral Methods in Matlab*. SIAM, Philadelphia, 2000.

[TT87] L. N. Trefethen and M. R. Trummer. An instability phenomenon in spectral methods. *SIAM J. Numer. Anal.*, 24:1008–1023, 1987.

[Var62] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, 1962.

[Whi74] G. Whitham. *Linear and Nonlinear Waves*. Wiley-Interscience, 1974.

[Wil65] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965.

[You50] D. M. Young. *Iterative Methods for Solving Partial Differential Equations of Elliptic Type*. PhD thesis, Harvard University, 1950.

[You71] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, 1971.