Finite Difference Methods for Differential Equations

Randall J. LeVeque

AMath 585, Winter Quarter 2006
University of Washington
Version of January, 2006

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

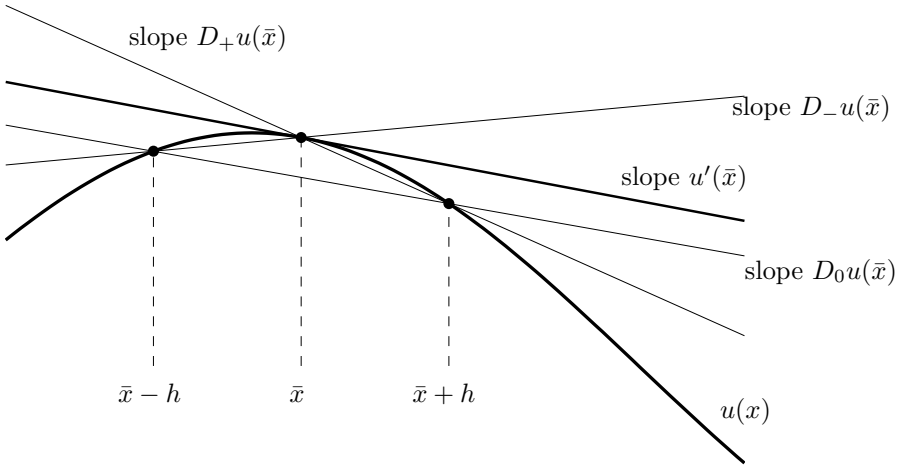


Figure 1.1: Various approximations to $u'(\bar{x})$ interpreted as the slope of secant lines.

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

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

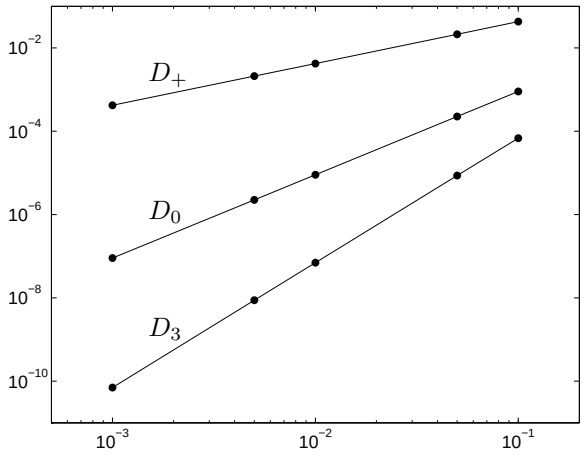


Figure 1.2: The errors in $Du(\bar{x})$ from Table 1.1 plotted against h on a log-log scale.

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

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

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:

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

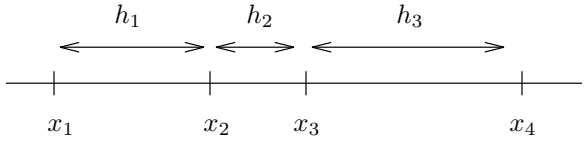
The next approximation is centered and second order accurate:

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

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: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

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

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

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

Then we will have

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

and so $\|E^h\|$ goes to zero at least as fast as $\|\tau^h\|$. This motivates the following definition of *stability* for linear 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 $A^h U^h = F^h$ where h is the mesh width. We say that the method is* stable *if $(A^h)^{-1}$ exists for all h sufficiently small (for $h < h_0$, say) and if there is a constant C, independent of h, such that*

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

2.8 Consistency

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

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

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

2.9 Convergence

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

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

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

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

Although this has been demonstrated only for the linear 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

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

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

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

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

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

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

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

Since $\tau^h_j \approx \frac{1}{12}h^2 u''''(x_j)$, we expect $\|\tau^h\|_2 \approx \frac{1}{12}h^2\|u''''\|_2 = \frac{1}{12}h^2\|f''\|_2$. The 2-norm of the function f'' here means the grid-function norm of this function evaluated at the discrete points x_j, though this is approximately equal to the function space norm of f'' defined using (A1.12).

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

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

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 (A1.17) we know that

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

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

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

since $\|\tau\|_\infty = O(h^2)$. As in the computation of the eigenvalues in the last section, we can 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

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

What is the derivative of this function? For $x \ne 0$ 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 $\phi_\epsilon(x)$.

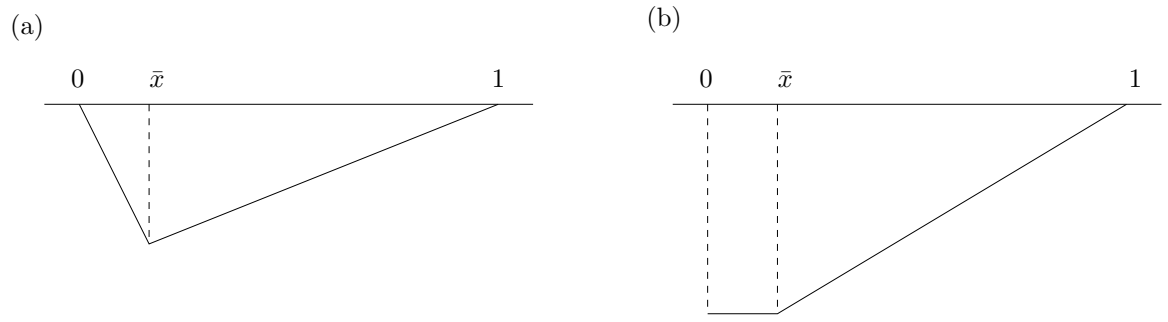


Figure 2.1: (a) The Green's function (2.28) for the Dirichlet problem. (b) The Green's function for the mixed problem with $u'(0) = u(1) = 0$ (see Exercise 2.5).

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 [FJ98]. 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:

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

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

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

which is simply the discrete form of (2.31).

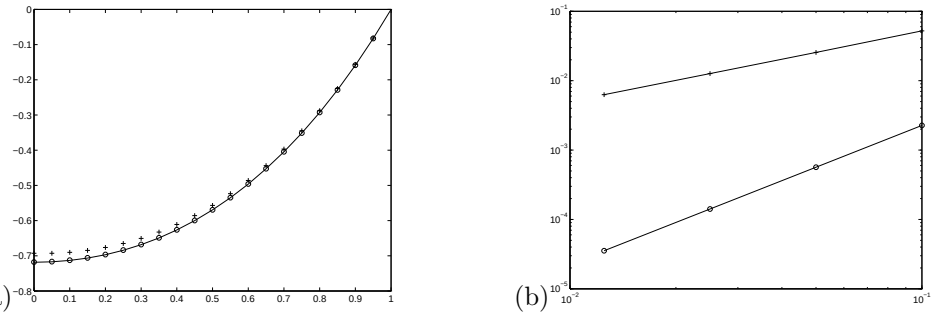


Figure 2.2: (a) Sample solution to the steady-state heat equation with a Neumann boundary condition at the left boundary and Dirichlet at the right. Solid line is the true solution. + shows solution on a grid with 20 points using (2.34). o shows the solution on the same grid using (2.36). (b) A log-log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

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 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,

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

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 $e = [1,\ 1,\ \ldots,\ 1]^T$ 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

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

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 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 Section A3.8). It is generally desirable to have important properties such as these modeled by the discrete approximation to the differential equation. One can then show, for example, that the solution to the difference equations satisfies a *maximum principle* of the same type as the solution to the differential equation: for the homogeneous equation with $f(x) \equiv 0$, the values of $u(x)$ lie between the values of the boundary values α and β everywhere, so the maximum and minimum values of u arise on the boundaries. For the heat conduction problem this is physically obvious: the steady state temperature in the rod won't exceed what's imposed at the boundaries if there is no heat source.

When solving the resulting linear system by iterative methods (see 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

$$\theta''(t) = -(g/L)\sin(\theta(t)) \quad (2.50)$$

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

$$\theta''(t) = -\sin(\theta(t)) \quad (2.51)$$

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

$$\theta''(t) = -\theta(t) \quad (2.52)$$

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

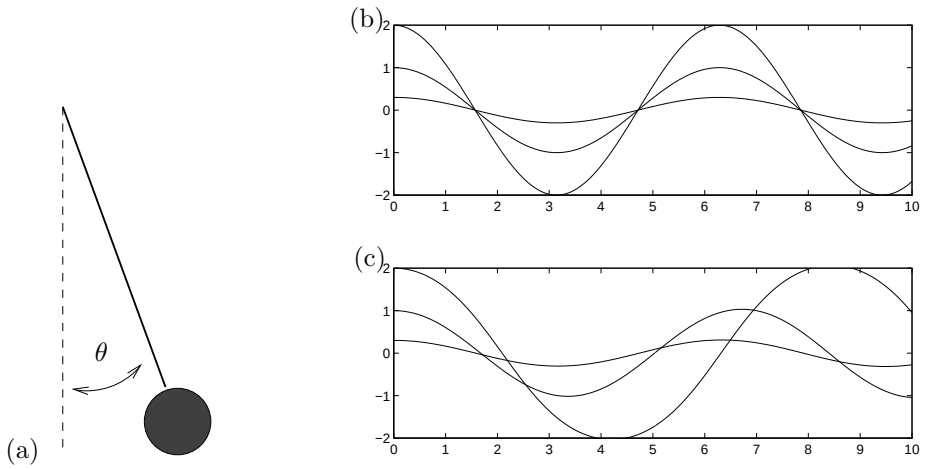


Figure 2.3: (a) Pendulum. (b) Solutions to the linear equation (2.52) for various initial θ and zero initial velocity. (c) Solutions to the nonlinear equation (2.51) for various initial θ and zero initial velocity.

(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 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

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

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 $\theta^{[0]} \equiv 0.7$.

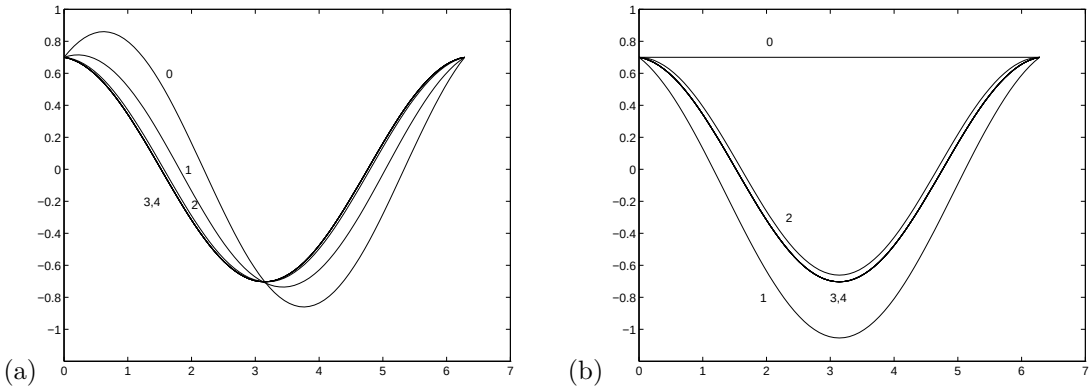


Figure 2.4: Convergence of Newton iterates towards a solution of the pendulum problem. The iterates $\theta^{[k]}$ for $k = 1,\ 2,\ \cdots$ are denoted by the number k in the plots. (a) Starting from $\theta_i^{[0]} = 0.7\cos(t_i) + 0.5\sin(t_i)$ (b) Starting from $\theta_i^{[0]} = 0.7$.

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

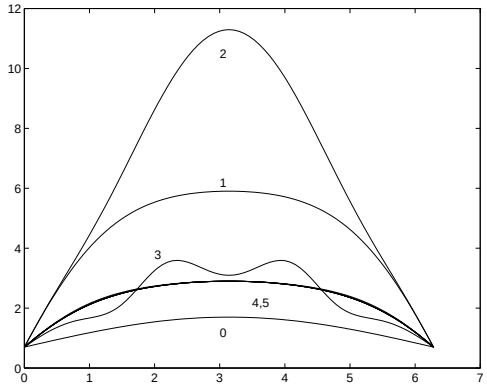


Figure 2.5: Convergence of Newton iterates towards a different solution of the pendulum problem, by starting with initial guess $\theta_i^{[0]} = 0.7 + \sin(t_i/2)$. The iterates k for $k = 1,\ 2,\ \cdots$ are denoted by the number k in the plots.

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 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*

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

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 $\kappa u_{xx} - a u_x = f$ for very small a is a *regular perturbation*. Both of these equations are second order differential equations requiring the same number of boundary conditions. The solution of the perturbed equation looks nearly identical 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 errors in our finite difference approximations. Recall that the error in our approximation to $u''(x)$ is proportional to $h^2 u''''(x)$, for example. If h is not small enough, then the local truncation error will be very large in the boundary layer. Moreover, even if the truncation error is large only in the boundary layer, the resulting global error may be large everywhere. (Recall that the global error E is obtained from the truncation error τ by solving a linear system $AE = -\tau$, which means that each element of E depends on *all* elements of τ since A^{-1} is a dense matrix.) This is clearly seen in Figure 2.6(b) where the numerical solution with $h = 1/10$ is plotted. Errors are large even in regions where the exact solution is nearly linear and $u'''' \approx 0$.

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

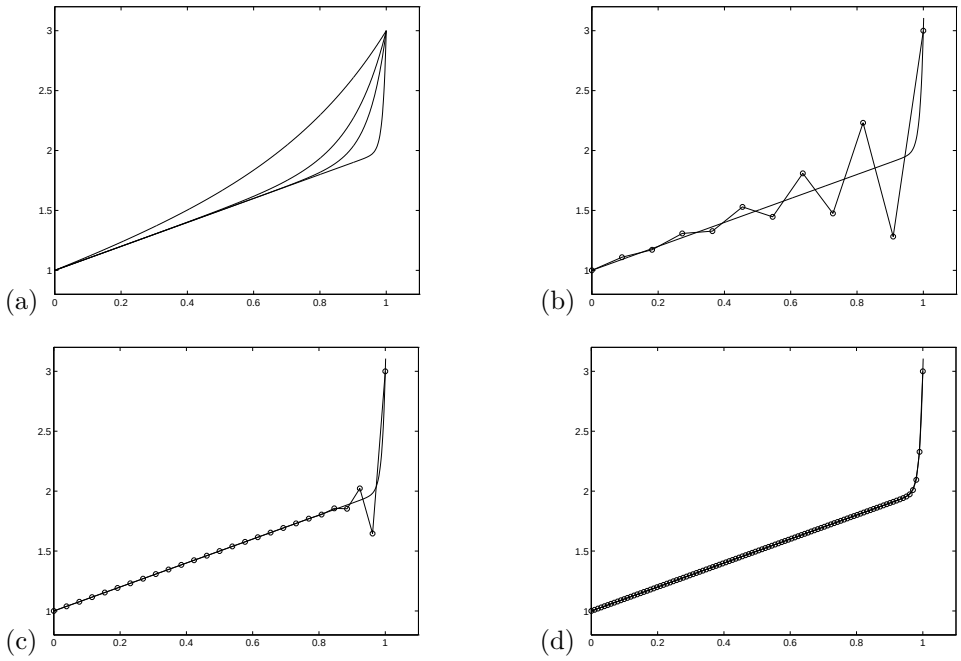


Figure 2.6: (a) Solutions to the steady-state advection-diffusion equation (2.64) for different values of ϵ. The four lines correspond to $\epsilon = 0.3,\ 0.1,\ 0.05$ and 0.01 from top to bottom. (b) Numerical solution with $\epsilon = 0.01$ and $h = 1/10$. (c) $h = 1/25$. (d) $h = 1/100$.

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

$$\bar{U}_j - u(jh) = \frac{1}{3}\left(\frac{1}{4} - 1\right)C_4h^4 + O(h^6). \tag{2.75}$$

The result has fourth order accuracy as h is reduced, and a much smaller error than either U_j or V_{2j} (provided C_4h^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 order	fourth order	extrap.	deferred 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 three 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

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

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

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

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

3.1 Steady-state heat conduction

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

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

where $\kappa(x, y) > 0$ is a diffusion or heat conduction coefficient that may vary with x and y, and $\psi(x, y, t)$ is a source term. The solution $u(x, y, t)$ 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*,

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

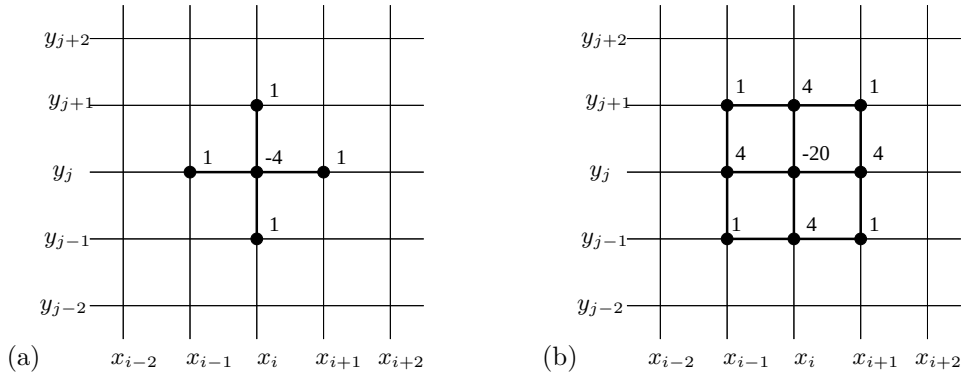


Figure 3.1: Portion of the computational grid for a two-dimensional elliptic equation. (a) The 5-point stencil for the Laplacian about the point (i, j) is also indicated. (b) The 9-point stencil is indicated, which is discussed in Section 3.4.

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 $0 \le x \le 1$, $0 \le y \le 1$ and suppose we have Dirichlet boundary conditions. We will use a uniform Cartesian grid consisting of grid points (x_i, y_j) where $x_i = i\Delta x$ and $y_j = j\Delta y$. A section of such a grid is shown in Figure 3.1.

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 of 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:

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

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.

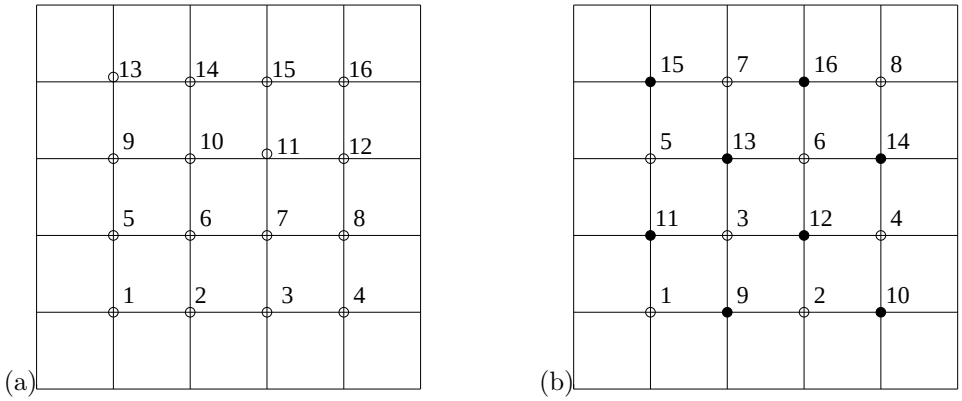


Figure 3.2: (a) The natural rowwise order of unknowns and equations on a 4×4 grid. (b) The red-black ordering.

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 two 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 very 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 $100 \times 100 \times 100$ grid 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 eigen-decomposition 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. This approach yields a so-called spectral method, as discussed further in the next section.

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. The discussion in this section 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 $F = [f(x_1),\ f(x_2),\ \ldots,\ f(x_m)]^T$ (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 $M = \text{diag}(-\pi^2,\ -(2\pi)^2,\ \ldots,\ -(m\pi)^2)$. 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 $u(x_1),\ \ldots,\ u(x_m)$ 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 $O(m \log m)$.

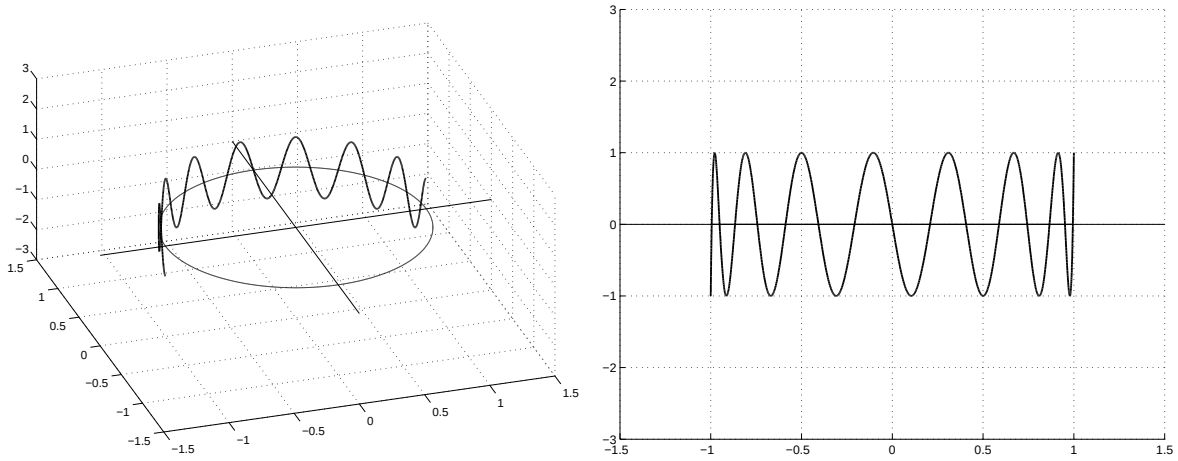


Figure 4.1: The Chebyshev polynomial viewed as a function $C_m(\theta)$ on the unit disk $e^{i\theta}$ and when projected on the x axis, *i.e.*, as a function of $x = \cos(\theta)$. Shown for $m = 15$.

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

$$Kc = F$$

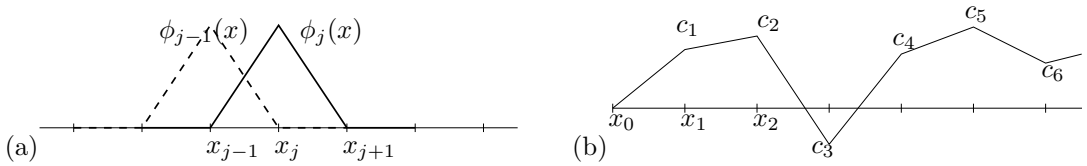


Figure 4.2: (a) Two typical basis functions $\phi_{j-1}(x)$ and $\phi_j(x)$ with continuous piecewise linear elements. (b) $U(x)$, a typical linear combination such basis functions.

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

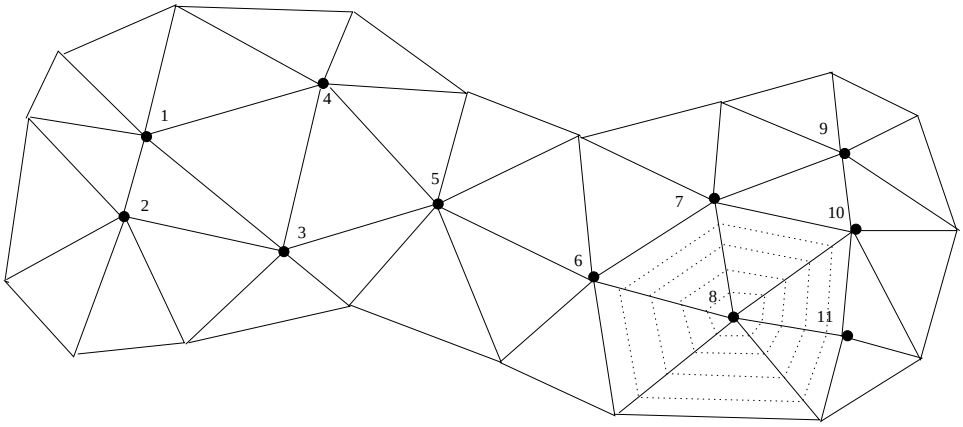


Figure 4.3: Triangulation of a two-dimensional region with 11 nodes. Contourlines for the basis function $\phi_8(x, y)$ are also shown as dashed lines.

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 (see Section 5.4) 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. See Appendix A4 for a more general discussion of the asymptotic properties of matrix powers.

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

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

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 $u'' = f$:

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

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

If $\omega < 1$ then we would be underrelaxing, rather than overrelaxing. This would be even less effective than Gauss-Seidel as a 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

$$\omega_{\mathrm{opt}} = \frac{2}{1 + \sin(\pi h)} \approx 2 - 2\pi h.$$

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

$$\rho_{\mathrm{opt}} = \omega_{\mathrm{opt}} - 1 \approx 1 - 2\pi h,$$

but if ω is changed slightly this can deteriorate substantially.

Even with the optimal ω we see that $\rho_{\mathrm{opt}} \to 1$ 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

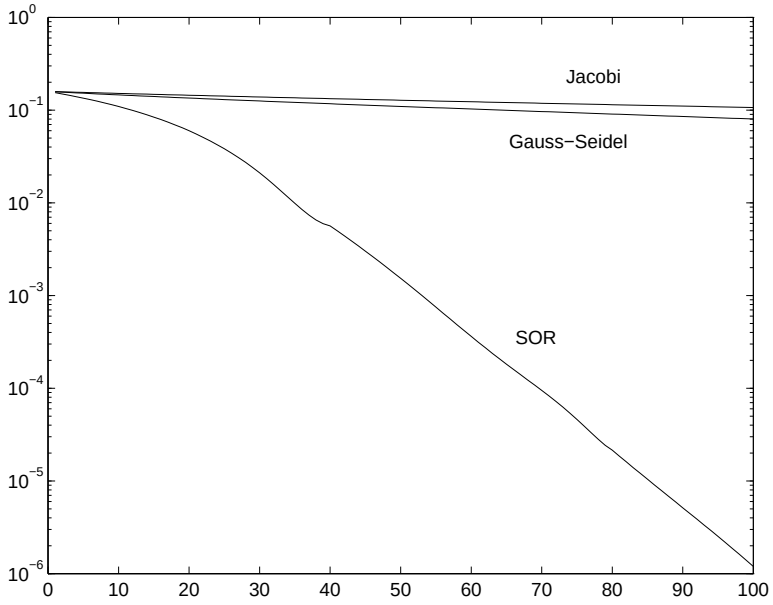


Figure 5.1: Errors vs. k for three methods.

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

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

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

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

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

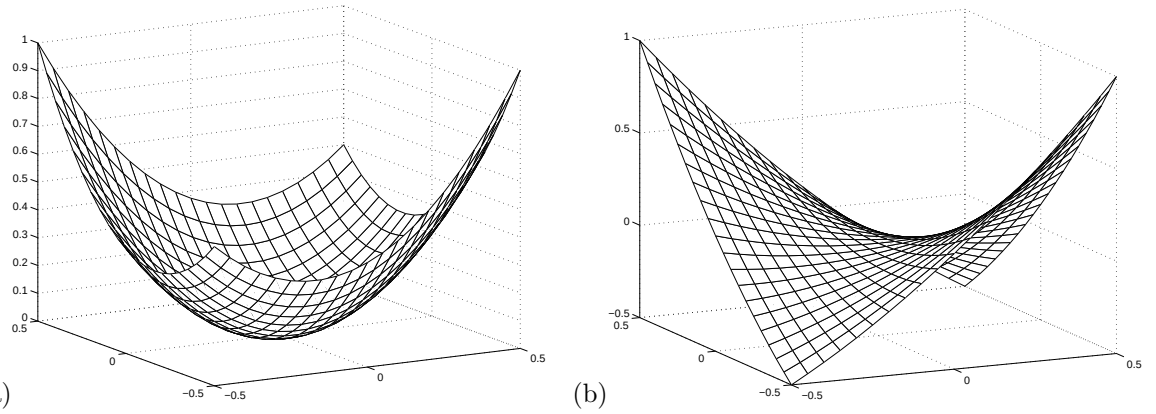


Figure 5.2: (a) The function $\phi(u)$ for $m = 2$ in a case where A is symmetric and positive definite. (b) The function $\phi(u)$ for $m = 2$ in a case where A is symmetric but indefinite.

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

$$\phi(u+\alpha r) = \left(\frac{1}{2}u^T Au - u^T f\right) + \alpha(r^T Au - r^T f) + \frac{1}{2}\alpha^2 r^T Ar \tag{5.29}$$

and so

$$\frac{d\phi(u+\alpha r)}{d\alpha} = r^T Au - r^T f + \alpha r^T Ar.$$

Setting this to zero and solving for α gives

$$\alpha = \frac{r^T r}{r^T Ar}. \tag{5.30}$$

The steepest descent algorithm thus takes the form:

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

Note that implementing this algorithm requires only that we be able to multiply a vector by A, as with the other iterative methods discussed earlier. We do not need to store the matrix A and if A is very sparse then this multiplication can be done quickly.

It appears that in each iteration we must do two matrix-vector multiplies, Au_{k-1} to compute r_{k-1} and then Ar_{k-1} to compute α_{k-1}. However, note that

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

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

Figure 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.

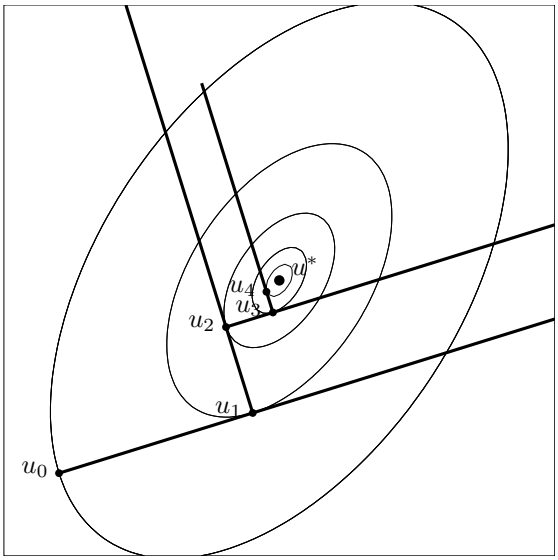


Figure 5.3: Several iterates of the method of steepest descent in the case $m = 2$. The concentric ellipses are level sets of $\phi(u)$.

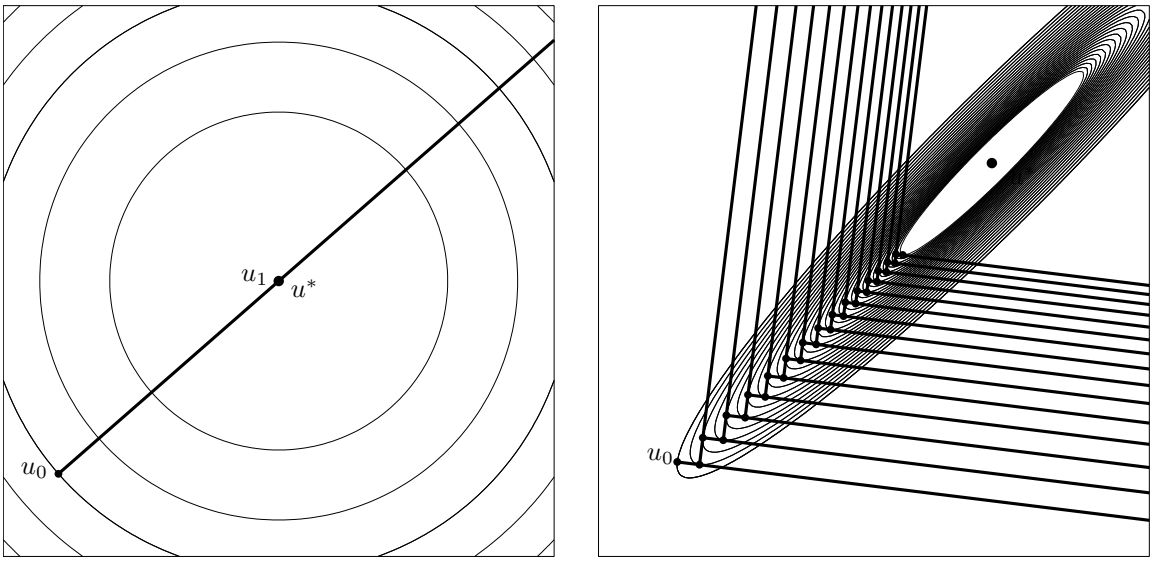


Figure 5.4: (a) If A is a scalar multiple of the identity then the level sets of $\phi(u)$ are circular and steepest descent converges in one iteration from any initial guess u_0. (b) If the level sets of $\phi(u)$ are far from circular then steepest descent may converge slowly.

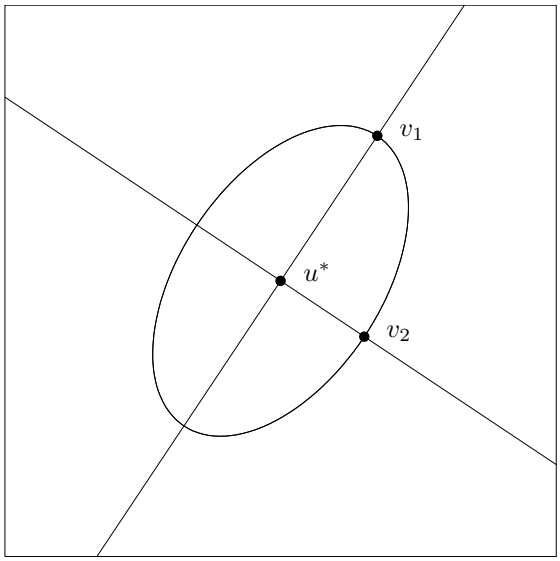


Figure 5.5: The major and minor axes of the elliptical level set of $\phi(u)$ point in the directions of the eigenvectors of A.

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 $\lambda_1 \le \lambda_2$ 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.

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, $u^T A v = 0$. 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

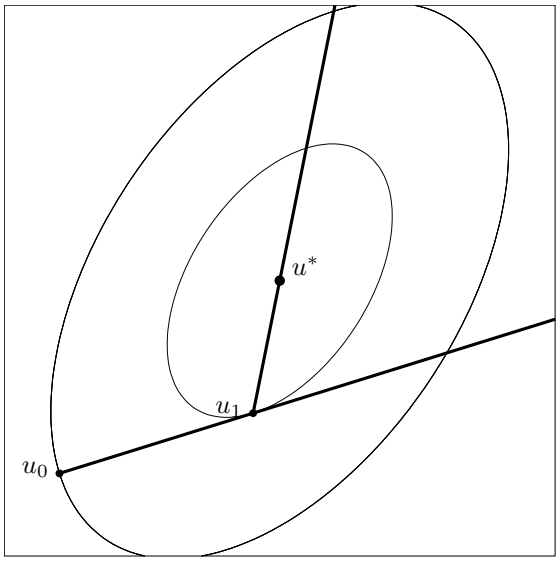


Figure 5.6: The conjugate gradient algoritm converges in 2 iterations from any initial guess u_0 in the case $m = 2$. The two search directions used are A-conjugate.

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 $A \in \mathbb{R}^{m \times m}$ $(n \leq m)$.

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, 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 (after preconditioning the matrix appropriately).

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

As with steepest descent, only one matrix-vector multiply is required 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 $j = 0,\ 1,\ , \ldots,\ k-1$. 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 $j = 0,\ 1,\ , \ldots,\ k-1$.*
2. *The residual r_k is orthogonal to all previous residuals, $r_k^T r_j = 0$ for $j = 0,\ 1,\ , \ldots,\ k-1$.*
3. *The following four subspaces of $\mathbb{R}^m$ are identical:*

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

The subspace $\mathcal{K}_k = \text{span}(f, Af, A^2 f, \ldots, A^{k-1} f)$ 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 $\text{span}(p_0, p_1, \ldots, p_{k-1})$ 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 $\text{span}(f, Af, \ldots, A^{k-1}f)$. Since $u_0 = 0$ we have $f = Au^* = -Ae_0$ and so $e_k - e_0$ also lies in $\text{span}(Ae_0, A^2e_0, \ldots, A^ke_0)$. So $e_k = e_0 + c_1Ae_0 + c_2A^2e_0 + \cdots + c_kA^ke_0$ for some coefficients $c_1, \ \ldots, \ c_k$. In other words,

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

where

$$P_k(A) = I + c_1A + c_2A^2 + \cdots + c_kA^k \tag{5.44}$$

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

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

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

$$e_0 = \sum_{j=1}^{m} a_j r_j$$

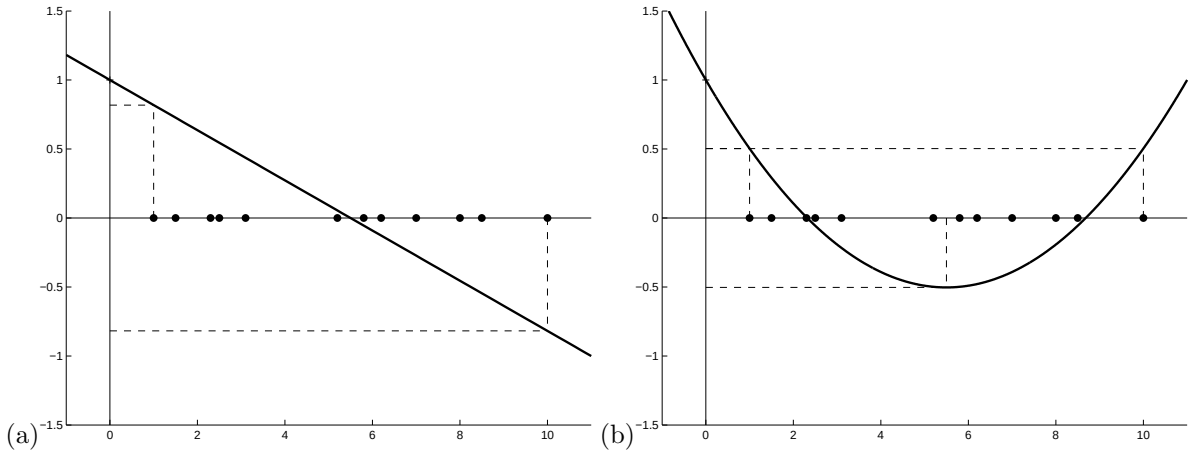


Figure 5.7: (a) The polynomial $\tilde{P}_1(x)$ based on a sample set of eigenvalues marked by dots on the x-axis. (b) The polynomial $\tilde{P}_2(x)$ for the same set of eigenvalues. See Figure 5.8(a) for the polynomial $\tilde{P}_5(x)$.

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

$$\|e_0\|_A^2 = e_0^T A e_0 = a^T R^T A R e_0 = a^T \Lambda a,$$

which gives (5.49).

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

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

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

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

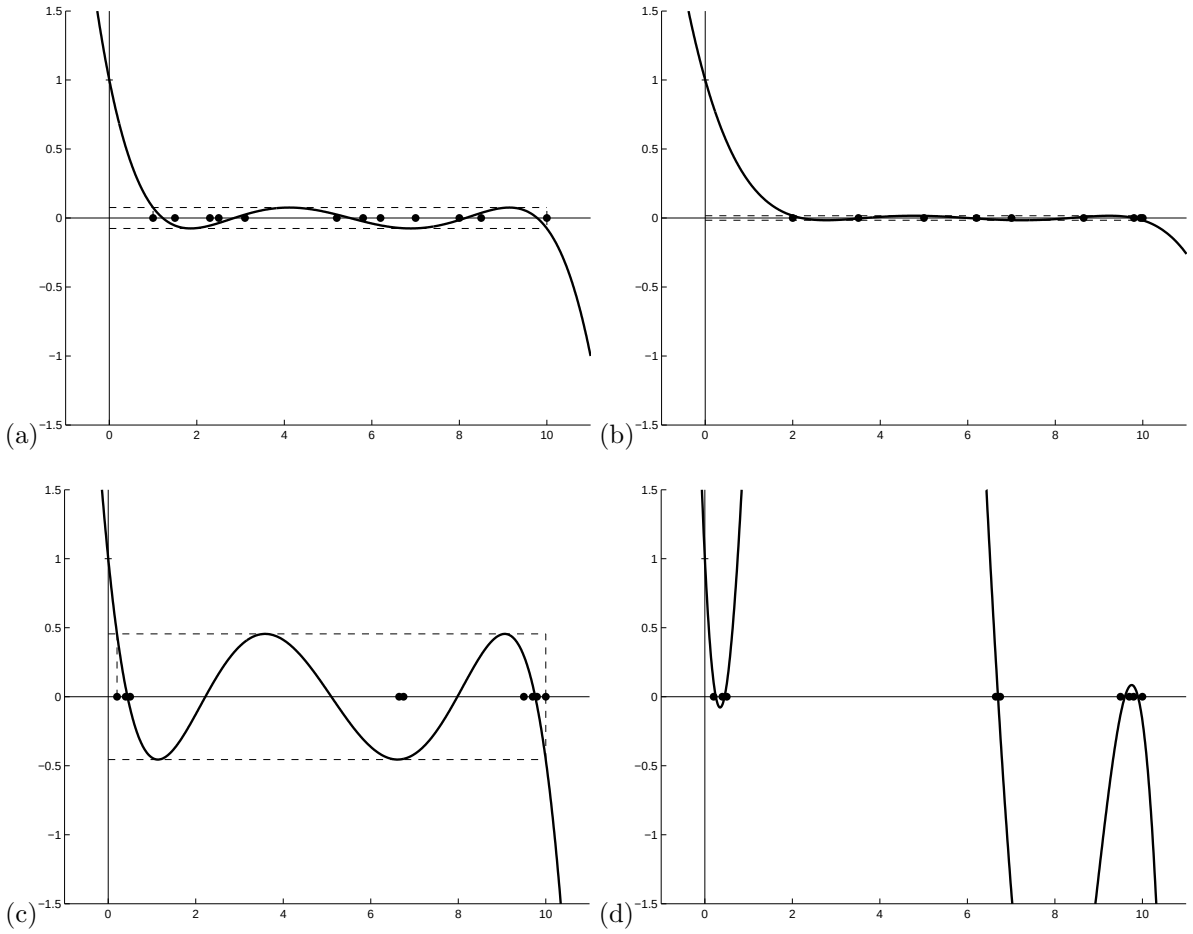


Figure 5.8: (a) The polynomial $\tilde{P}_5(x)$ based on a sample set of eigenvalues marked by dots on the x-axis, the same set as in Figure 5.7. (b) The polynomial $\tilde{P}_5(x)$ for a matrix with smaller κ. (c) The polynomial $\tilde{P}_5(x)$ for a matrix with larger κ. (d) A better polynomial $P(x)$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

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 $\|e_5\|_A/\|e_0\|_A \leq 0.0756$ 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 $\|e_5\|_A/\|e_0\|_A \leq 0.0163$.

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 $\|e_5\|_A/\|e_0\|_A \leq 0.4553$. 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 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].

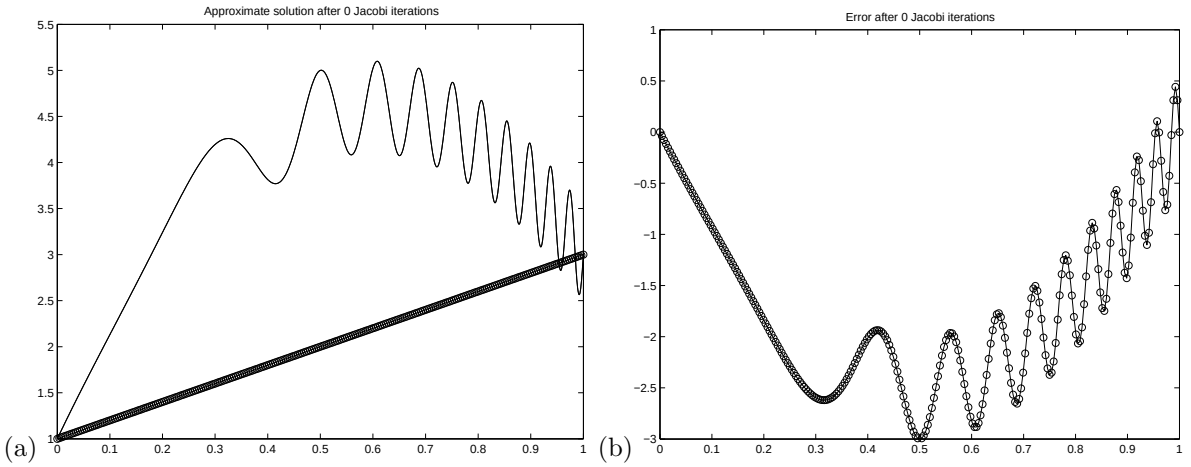


Figure 5.9: (a) The solution $u(x)$ (solid line) and initial guess u_0 (circles). (b) The error e_0 in the initial guess.

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

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

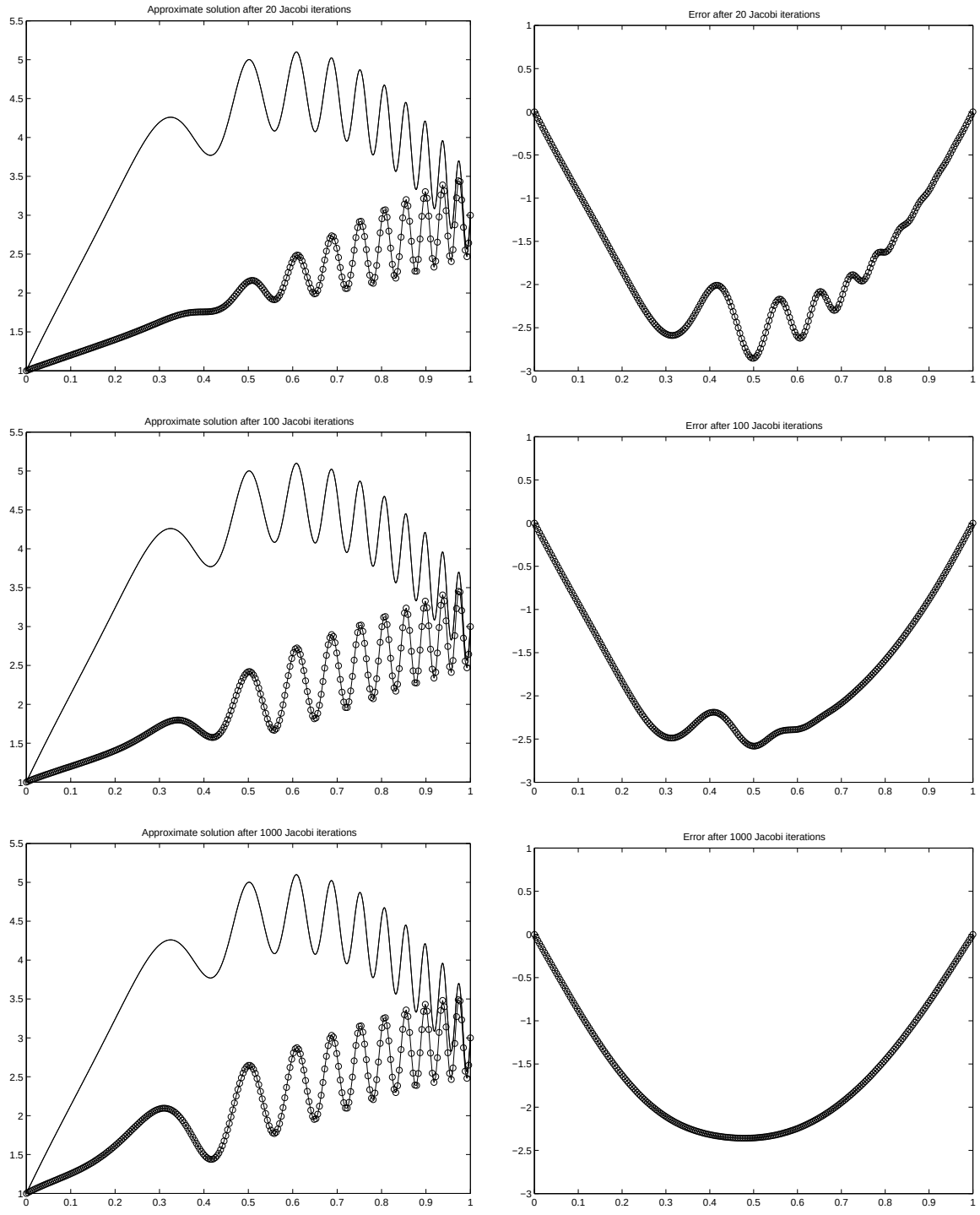


Figure 5.10: On the left: The solution $u(x)$ (solid line) and Jacobi iterate u_k. On the right: The error e_k. Shown for $k = 20$ (top), $k = 100$ (middle), and $k = 1000$ (bottom).

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.

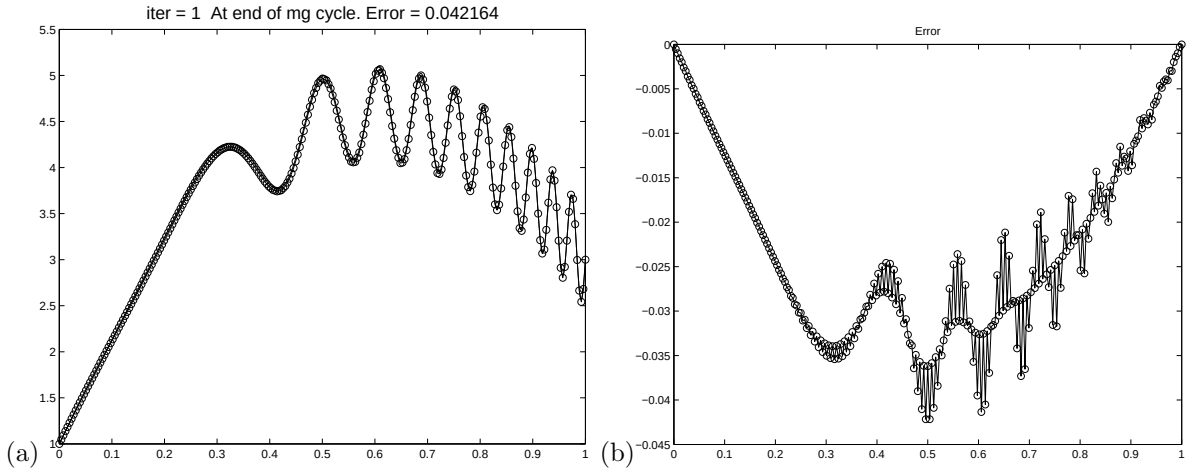


Figure 5.11: (a) The solution $u(x)$ (solid line) and approximate solution (circles) obtained after one V-cycle of the multigrid algorithm with $\nu = 10$. (b) The error in this approximation. Note the change in scale from Figure 5.10.

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,

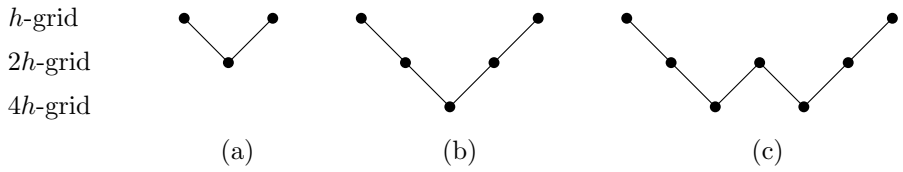


Figure 5.12: (a) One V-cycle with 2 levels. (b) One V-cycle with 3 levels. (c) One W-cycle with 3 levels.

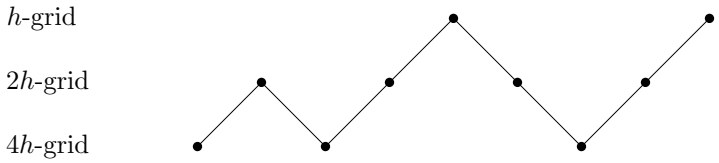


Figure 5.13: Full multigrid (FMG) with one V-cycle on 3 levels.

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.

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:

$$2h^3 = O(h^2) \quad \text{as } h \to 0, \quad \text{since } \frac{2h^3}{h^2} = 2h < 1 \quad \text{for all } h < 1/2.$$

$$2h^3 = o(h^2) \quad \text{as } h \to 0, \quad \text{since } 2h \to 0 \quad \text{as } h \to 0.$$

$$\sin(h) = O(h) \quad \text{as } h \to 0, \quad \text{since } \sin h = h - \frac{h^3}{3} + \frac{h^5}{5} + \cdots < h \quad \text{for all } h > 0.$$

$$\sin(h) = h + o(h) \quad \text{as } h \to 0, \quad \text{since } (\sin h - h)/h = O(h^2).$$

$$\sqrt{h} = O(1) \quad \text{as } h \to 0, \quad \text{and also } \sqrt{h} = o(1), \quad \text{but } \sqrt{h} \text{ is not } O(h) \ .$$

$$1 - \cos h = o(h) \quad \text{and } 1 - \cos h = O(h^2) \quad \text{as } h \to 0.$$

$$e^{-1/h} = o(h^q) \quad \text{as } h \to 0 \quad \text{for every value of } q.$$

To see this, let $x = 1/h$ then $\dfrac{e^{-1/h}}{h^q} = e^{-x}x^q \to 0 \quad \text{as } x \to \infty.$

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. $\|x\| \geq 0$ 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 $\|x + y\| \le \|x\| + \|y\|$. (Triangle inequality)

One common choice is the max-norm (or infinity-norm) denoted by $\|\cdot\|_\infty$:

$$\|e\|_\infty = \max_{1\le i\le m} |e_i|.$$

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

$$C_1\|x\|_\alpha \le \|x\|_\beta \le C_2\|x\|_\alpha \tag{A1.5}$$

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:

$$\|x\|_\infty \le \|x\|_1 \le m\|x\|_\infty \tag{A1.6a}$$
$$\|x\|_\infty \le \|x\|_2 \le \sqrt{m}\|x\|_\infty \tag{A1.6b}$$
$$\|x\|_2 \le \|x\|_1 \le \sqrt{m}\|x\|_2. \tag{A1.6c}$$

Now suppose that $\|e^h\|_\alpha \le Ch^q$ as $h \to 0$ in some norm $\|\cdot\|_\alpha$. Then we have

$$\|e^h\|_\beta \le C_2\|e^h\|_\alpha \le C_2 C h^q$$

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 $A \in \mathbb{R}^{m\times m}$ is denoted by $\|A\|$ and has the property that $C = \|A\|$ is the *smallest* value of the constant C for which the bound

$$\|Ax\| \le C\|x\| \tag{A1.7}$$

holds for *every* vector $x \in \mathbb{R}^m$. Hence $\|A\|$ is defined by

$$\|A\| = \max_{\substack{x\in\mathbb{R}^m \\ x\neq\vec{0}}} \frac{\|Ax\|}{\|x\|} = \max_{\substack{x\in\mathbb{R}^m \\ \|x\|=1}} \|Ax\|. \tag{A1.8}$$

It would be rather difficult to calculate $\|A\|$ from the above definitions, but for the most commonly used norms there are simple formulas for computing $\|A\|$ directly from the matrix:

$$\|A\|_1 = \max_{1\le j\le m} \sum_{i=1}^{m} |a_{ij}| \qquad \text{(maximum column sum)} \tag{A1.9a}$$

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{m} |a_{ij}| \qquad \text{(maximum row sum)} \tag{A1.9b}$$

$$\|A\|_2 = \sqrt{\rho(A^TA)}. \tag{A1.9c}$$

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 $a \le x \le b$ rather than a single value or vector. Some numerical methods, such as finite element or collocation methods, produce an approximate solution $\hat{u}(x)$ 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

$$\|e\|_\infty = \max_{a\le x\le b} |e(x)|. \tag{A1.10}$$

The 1-norm and 2-norms are given by integrals over $[a, b]$ rather than by sums over the vector elements:

$$\|e\|_1 = \int_a^b |e(x)|\,dx, \tag{A1.11}$$

$$\|e\|_2 = \left(\int_a^b |e(x)|^2\,dx\right)^{1/2}. \tag{A1.12}$$

These are again special cases of the general p-norm, defined by

$$\|e\|_p = \left(\int_a^b |e(x)|^p\,dx\right)^{1/p}. \tag{A1.13}$$

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.*,

$$\|e\|_1 = \sum_{i=0}^{N} |e_i|. \tag{A1.14}$$

However, this choice would give a very misleading idea of the magnitude of the error. The quantity in (A1.14) 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 (A1.14) 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 (A1.11), which is motivated by considering the vector $(e_0,\ \ldots,\ e_N)$ as a discretization of some error function $e(x)$. This suggests defining

$$\|e\|_1 = h\sum_{i=0}^{N} |e_i| \tag{A1.15}$$

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 (A1.11). The norm (A1.15) 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

$$\|e\|_p = \left(h\sum_{i=0}^{N} |e_i|^p\right)^{1/p}. \tag{A1.16}$$

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 (A1.10).

In two space dimensions we have analogous norms of functions and grid functions, *e.g.*,

$$\begin{aligned} \|e\|_p &= \left(\iint |e(x,y)|^p \, dx \, dy \right)^{1/p} \qquad \text{for functions} \\ \|e\|_p &= \left(\Delta x \, \Delta y \sum_i \sum_j |e_{ij}|^p \right)^{1/p} \qquad \text{for grid functions} \end{aligned}$$

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

$$h\|e\|_\infty \le \|e\|_1 \le Nh\|e\|_\infty = (b-a)\,\|e\|_\infty, \tag{A1.17a}$$

$$\sqrt{h}\,\|e\|_\infty \le \|e\|_2 \le \sqrt{Nh}\,\|e\|_\infty = \sqrt{b-a}\,\|e\|_\infty, \tag{A1.17b}$$

$$\sqrt{h}\,\|e\|_2 \le \|e\|_1 \le \sqrt{Nh}\,\|e\|_2 = \sqrt{b-a}\,\|e\|_2. \tag{A1.17c}$$

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

$$e_i^h = U_i^h - u(x_i) = \begin{cases} \frac{1}{2} & i = N/2 \\ 0 & \text{otherwise.} \end{cases}$$

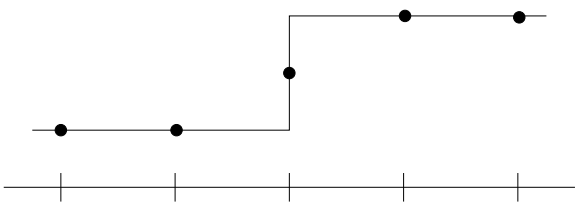


Figure A1.1: The function $u(x)$ and the discrete approximation.

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 (A1.15) 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.

If the function we are approximating is sufficiently smooth, and if we expect the error to be roughly the same magnitude at all points, then it typically does not matter so much which norm is chosen. The convergence rate for a given method, as observed on sufficiently smooth functions, will often be the same in any p-norm. The norm chosen for analysis is then often determined by the nature of the problem and the availability of mathematical techniques for estimating different error norms. For example, for linear problems where Fourier analysis can be applied, the 2-norm is often a natural choice. For conservation laws where integrals of the solution are studied, the 1-norm is often simplest.

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.

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

$$\begin{aligned} Ar_1 = \lambda r_1 \quad &\Longrightarrow \quad (A - \lambda I)r_1 = 0 \\ Ar_2 = r_1 + \lambda r_2 \quad &\Longrightarrow \quad (A - \lambda I)r_2 = r_1 \quad \Longrightarrow \quad (A - \lambda I)^2 r_2 = 0 \\ Ar_3 = r_2 + \lambda r_3 \quad &\Longrightarrow \quad (A - \lambda I)r_3 = r_2 \quad \Longrightarrow \quad (A - \lambda I)^3 r_3 = 0. \end{aligned} \tag{A3.13}$$

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	$\iff$	$u^H A u > 0$ for all $u \neq 0$	$\iff$	all eigenvalues are positive,
positive semi-definite	$\iff$	$u^H A u \geq 0$ for all $u \neq 0$	$\iff$	all eigenvalues are nonnegative,
negative definite	$\iff$	$u^H A u < 0$ for all $u \neq 0$	$\iff$	all eigenvalues are negative,
negative semi-definite	$\iff$	$u^H A u \leq 0$ for all $u \neq 0$	$\iff$	all eigenvalues are nonpositive,
indefinite	$\iff$	$u^H A u$ indefinite	$\iff$	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.

Eigenvalue analysis is particularly useful for normal matrices, since they can be diagonalized by a unitary matrix. A unitary matrix R satisfies $\|R\|_2 = \|R^{-1}\|_2 = 1$, and hence the behavior of powers of A is very closely related to the powers of the eigenvalues, for example. Non-normal matrices can be harder to analyze, particularly if R is very poorly conditioned. In this case studying only the eigenvalues of A can be misleading. See Section A4.3 for more discussion of this.

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 Chapter 2, since such matrices arise in studying 3-point difference equations on the unit interval with periodic boundary conditions. Then $h = 1/(m+1)$ is the mesh spacing between grid points and the unknowns are $U_1,\ \ldots,\ U_{m+1}$.

The pth eigenvalue of the matrix (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

$$e^{2\pi ip(j\pm1)h} = e^{\pm 2\pi iph}e^{2\pi ipjh} \tag{A3.19}$$

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 A3.2).

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,*

$$D_i = \{z \in \mathbb{C} : \ |z - a_{ii}| \leq r_i\}.$$

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

$$D_1 = \{z : |z - 5| \leq 0.7\}, \qquad D_2 = \{z : |z - 6| \leq 1.1\}, \qquad D_3 = \{z : |z - 2| \leq 1.0\},$$

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

$$D_1 = \{z : |z - 5| \leq 2.0\}, \qquad D_2 = \{z : |z - 6| \leq 0.6\}, \qquad D_3 = \{z : |z - 2| \leq 0.2\},$$

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.

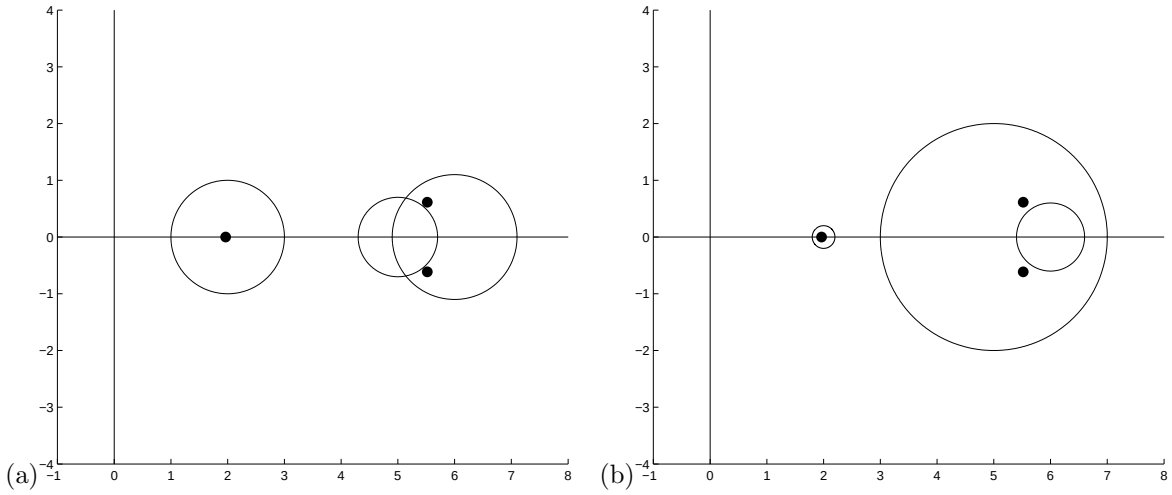


Figure A3.1: Gerschgorin circles containing the eigenvalues of A for Example A3.5.

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 (A1.8) 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 5.3.4) and T could be defined as a "square root" of A, *e.g.*, $T = R\Lambda^{1/2}R^{-1}$ if $A = R\Lambda R^{-1}$.

In studying iterations of the form $U^{n+1} = AU^n$, and variants such as $U^{n+1} = A_n U^n$ (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 (A1.8):

$$\begin{aligned}
\|A\|_T &= \sup_{u\neq 0} \frac{\|Au\|_T}{\|u\|_T} = \sup_{w\neq 0} \frac{\|ATw\|_T}{\|Tw\|_T} \\
&= \sup_{w\neq 0} \frac{\|T^{-1}ATw\|_2}{\|w\|_2} \\
&= \|T^{-1}AT\|_2.
\end{aligned} \tag{A3.35}$$

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

$$D_i = \mathrm{diag}(1,\ \delta_i,\ \ \ldots,\ \delta_i^{k_i-1}).$$

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

$$\begin{aligned} |u(t)| \to 0 \text{ as } t \to \infty \text{ for any } u(0) &\iff \mathrm{Re}(a) < 0, \\ |u(t)| \text{ remains bounded as } t \to \infty \text{ for any } u(0) &\iff \mathrm{Re}(a) \le 0, \\ |u(t)| \to \infty \text{ as } t \to \infty \text{ for any } u(0) \ne 0 &\iff \mathrm{Re}(a) > 0 \end{aligned} \tag{A4.1}$$

The first order scalar linear difference equation $U^{n+1} = bU^n$ has the solution $U^n = b^n U^0$ and so

$$\begin{aligned} |U^n| \to 0 \text{ as } n \to \infty \text{ for any } U^0 &\iff |b| < 1, \\ |U^n| \text{ remains bounded as } n \to \infty \text{ for any } U^0 &\iff |b| \le 1, \\ |U^n| \to \infty \text{ as } n \to \infty \text{ for any } U^0 \ne 0 &\iff |b| > 1 \end{aligned} \tag{A4.2}$$

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

$$\begin{aligned} |b| < 1 &\iff \mathrm{Re}(a) < 0, \\ |b| = 1 &\iff \mathrm{Re}(a) = 0, \\ |b| > 1 &\iff \mathrm{Re}(a) > 0. \end{aligned} \tag{A4.3}$$

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.

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 (A1.5). 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,

$$\begin{aligned} J(\lambda,k)^n &= (\lambda I_k + S_k)^n \\ &\quad \lambda^n I_k + n\lambda^{n-1} S_k + \begin{pmatrix} n \\ 2 \end{pmatrix} \lambda^{n-2} S_k^2 + \begin{pmatrix} n \\ 3 \end{pmatrix} \lambda^{n-3} S_k^3 \\ &\qquad + \cdots + \begin{pmatrix} n \\ n-1 \end{pmatrix} \lambda S_k^{n-1} + S_k^n. \end{aligned} \tag{A4.11}$$

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

$$\begin{aligned} J(\lambda,k)^n &= \lambda^n I_k + n\lambda^{n-1} S_k + \begin{pmatrix} n \\ 2 \end{pmatrix} \lambda^{n-2} S_k^2 + \begin{pmatrix} n \\ 3 \end{pmatrix} \lambda^{n-3} S_k^3 \\ &\qquad + \cdots + \begin{pmatrix} n \\ k-1 \end{pmatrix} \lambda^{n-k+1} S_k^{k-1}. \end{aligned} \tag{A4.12}$$

Since $\begin{pmatrix} n \\ j \end{pmatrix} = O(n^j)$ 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

$$\begin{aligned} J(\lambda,3)^n &= \lambda^n I_3 + n\lambda^{n-1} S_3 + \frac{n(n-1)}{2} \lambda^{n-2} S_3^2 \\ &= \begin{bmatrix} \lambda^n & n\lambda^{n-1} & \frac{n(n-1)}{2}\lambda^{n-2} \\ 0 & \lambda^n & n\lambda^{n-1} \\ 0 & 0 & \lambda^n \end{bmatrix}. \end{aligned} \tag{A4.13}$$

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 $B^HB \neq BB^H$, 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 A4.3.

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 $|e^{\lambda_j t}| \le 1$ for all j.

It is useful to introduce the *spectral abscissa* $\alpha(A)$, defined by

$$\alpha(A) = \max_{1\le j\le m} \text{Re}(\lambda_j). \tag{A4.20}$$

Then $u(t)$ remains bounded provided $\alpha(A) \le 0$ 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

$$\begin{aligned} e^{J(\lambda,k)t} &= \sum_{j=0}^{\infty} \frac{t^j}{j!} \left[\lambda^j I + j\lambda^{j-1} S_k + \binom{j}{2} \lambda^{j-2} S_k^2 + \cdots + \binom{j}{j-1} \lambda S_k^{j-1} + S_k^j \right] \\ &= \sum_{j=0}^{\infty} \frac{t^j}{j!} \lambda^j I + t \sum_{j=1}^{\infty} \frac{t^{j-1}}{(j-1)!} \lambda^{j-1} S_k + \frac{t^2}{2!} \sum_{j=2}^{\infty} \frac{t^{j-2}}{(j-2)!} \lambda^{j-2} S_k^2 + \cdots \\ &= e^{\lambda t} I + t e^{\lambda t} S_k + \frac{t^2}{2!} e^{\lambda t} S_k^2 + \cdots + \frac{t^{(k-1)}}{(k-1)!} e^{\lambda t} S_k^{k-1} \\ &= \begin{bmatrix} e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \cdots & \frac{t^{(k-1)}}{(k-1)!}e^{\lambda t} \\ & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \cdots \\ & & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} \\ & & & \ddots & \ddots \\ & & & & e^{\lambda t} \end{bmatrix}. \end{aligned} \tag{A4.22}$$

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

$$U^0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad U^1 = \begin{bmatrix} 100 \\ 0.9 \end{bmatrix}, \quad U^2 = \begin{bmatrix} 170 \\ 0.81 \end{bmatrix}, \quad U^3 = \begin{bmatrix} 217 \\ 0.729 \end{bmatrix}, \quad \ldots$$

We have the bound $\|U^n\|_2 \le \kappa_2(R)(0.9)^n\|U^0\|_2$ but in this case

$$R = \begin{bmatrix} 1 & 1 \\ 0 & 0.001 \end{bmatrix}, \qquad R^{-1} = \begin{bmatrix} 1 & -1000 \\ 0 & 1000 \end{bmatrix}.$$

so $\kappa_2(R) = 2000$. Figure A4.1 shows $\|U^n\|_2$ for $n = 1,\ \ldots,\ 30$ along with the bound.

Clearly this example could be made much more extreme by replacing $a_{22} = 100$ by a larger value. Larger matrices can exhibit similar growth before decay even if all the elements of the matrix are modest.

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.

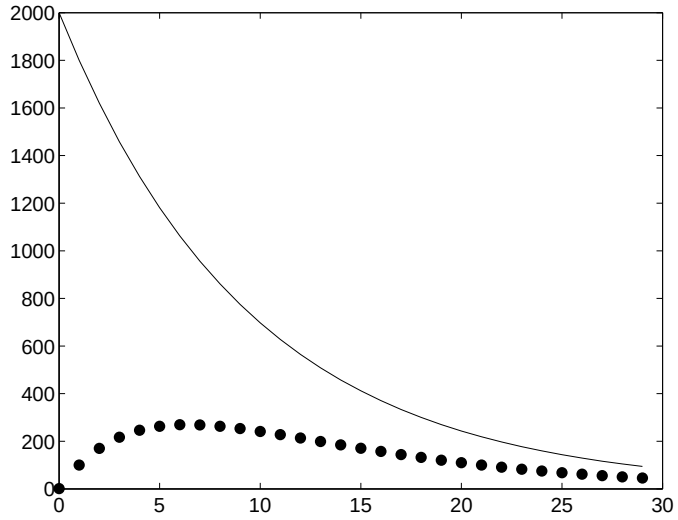


Figure A4.1: The points show $\|U^n\|_2$ for Example A4.2 and the line shows the upper bound $2000(0.9)^n$.

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 $|t_{11}| \le |t_{22}| \le \cdots \le |t_{mm}|$.)

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

$$\tilde{A} = \begin{bmatrix} 0.8 & 100 \\ 0.001 & 0.9 \end{bmatrix} \tag{A4.27}$$

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 [Wri] 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 $\|A^n\| \le C$ for all n and for all $A \in \mathcal{F}$. (The family is power bounded.)

(b) There exists a constant C_1 such that, for all $A \in \mathcal{F}$ and all $z \in \mathbb{C}$ with $|z| > 1$, the resolvent $(A - zI)^{-1}$ exists and is bounded by

$$\|(A - zI)^{-1}\| \le \frac{C_1}{|z| - 1}. \tag{A4.29}$$

(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) $\|S\| \le C_2, \quad \|S^{-1}\| \le C_2,$

ii) $B = S^{-1}AS$ is upper triangular with off-diagonal elements bounded by

$$|b_{ij}| \le C_3 \min(1 - |b_{ii}|,\ 1 - |b_{jj}|). \tag{A4.30}$$

(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

$$\begin{aligned} &C_4^{-1} I \le G \le C_4 I \\ &A^H G A \le G. \end{aligned} \tag{A4.31}$$

In condition (d) we say that two Hermitian matrices A and B satisfy $A \le B$ if $u^H A u \le u^H B u$ for any vector u. This condition can be rewritten in a more familiar form as

(d') There exists a constant C_5 so that for each $A \in \mathcal{F}$ there is a nonsingular matrix T such that

$$\|A\|_T \le 1 \quad \textit{and} \quad \kappa(T) \le C_5. \tag{A4.32}$$

Here the T-norm of A is defined as in (A3.33) in terms of the 2-norm,

$$\|A\|_T = \|T^{-1} A T\|.$$

Condition (d') is related to (d) by setting $G = T^{-H} T^{-1}$, and (d') states that we can define a set of norms, one for each $A \in \mathcal{F}$, for which the norm of A is less than 1 and therefore

$$\|A^n\|_T \le 1 \quad \text{for all } n \ge 0.$$

From this we can obtain uniform power boundedness by noting that

$$\|A^n\| \le \kappa(T) \|A^n\|_T \le C_5.$$

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 $\mathrm{Re}(\zeta_j) \leq 0,\ \ j = 1,\ 2,\ \ldots,\ r$,
b) Any root with $\mathrm{Re}(\zeta_j) = 0$ 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 $|\zeta_j| \leq 1 \;\; j = 1,\; 2,\; \ldots,\; r$,
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, $\zeta_i \neq \zeta_j$ for $i \neq j$, 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 $c_1\zeta_1^n + c_2 n\zeta_1^n$. 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.

[FJ98] F. G. Friedlander and M. Joshi. *Introduction to the Theory of Distributions*. Cambridge University Press, 1998.

[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.

[Wri] T. Wright. Eigtool. `http://web.comlab.ox.ac.uk/projects/pseudospectra/eigtool/`.

[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.