8 Finite Differences: Partial Differential Equations

The world is defined by structure in space and time, and it is forever changing in complex ways that can't be solved exactly. Therefore the numerical solution of partial differential equations leads to some of the most important, and computationally intensive, tasks in all of numerical analysis (such as forecasting the weather). This chapter introduces *finite difference* techniques; the next two will look at other ways to discretize partial differential equations (finite elements and cellular automata). Just as we used a Taylor expansion to derive a numerical approximation for ordinary differential equations, the same procedure can be applied to partial differential equations. Because the discretization must be done in space as well as time, there are many more possible strategies for finding good (and bad) approximations.

We will start with two degrees of freedom, say one spatial variable x and a time t. Given a function $u(x,t)$, its spatial derivatives are found from the Taylor expansion

$$u(x+\Delta x,t) = u(x,t) + \Delta x \left.\frac{\partial u}{\partial x}\right|_{x,t} + \frac{(\Delta x)^2}{2!} \left.\frac{\partial^2 u}{\partial x^2}\right|_{x,t} + \mathcal{O}[(\Delta x)^3] \quad . \tag{8.1}$$

The first partial derivative can be approximated by the *forward difference*

$$\frac{u(x+\Delta x,t) - u(x,t)}{\Delta x} = \left.\frac{\partial u}{\partial x}\right|_{x,t} + \mathcal{O}[\Delta x] \quad . \tag{8.2}$$

If we replace Δx with $-\Delta x$, this becomes the equally reasonable *backwards difference* approximation

$$\frac{u(x,t) - u(x-\Delta x,t)}{\Delta x} = \left.\frac{\partial u}{\partial x}\right|_{x,t} + \mathcal{O}[\Delta x] \quad . \tag{8.3}$$

Their order can be raised by taking the difference between two time steps, which subtracts out the quadratic term:

$$\frac{u(x+\Delta x,t) - u(x-\Delta x,t)}{2\Delta x} = \left.\frac{\partial u}{\partial x}\right|_{x,t} + \mathcal{O}[(\Delta x)^2] \quad . \tag{8.4}$$

Although this might appear always to be preferable, we will see that it can have surprising undesirable stability properties.

The straightforward finite difference approximation to the second partial derivative is

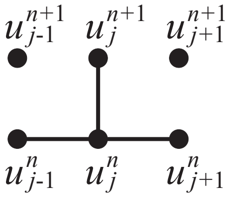

Figure 8.1. A computational cluster.

also correct to first order:

$$\frac{1}{\Delta x}\left[\frac{u(x+\Delta x,t)-u(x,t)}{\Delta x}-\frac{u(x,t)-u(x-\Delta x,t)}{\Delta x}\right]=$$

$$\frac{u(x+\Delta x,t)-2u(x,t)+u(x-\Delta x,t)}{(\Delta x)^2}=\left.\frac{\partial^2 u}{\partial x^2}\right|_{x,t}+\mathcal{O}[(\Delta x)^2] \quad . \qquad (8.5)$$

Numerical methods for partial differential equations are usually classified by the characteristics for the equation that they apply to (Chapter 4), which measure how information from the boundary conditions influences the solution. Characteristics can even be used as the basis for numerical solvers [Ames, 1992], but here we will simply use them as convenient labels for the most common cases: a wave equation (hyperbolic characteristics), diffusive processes (parabolic), and boundary value problems (elliptic). More complex systems can have some or all of these elements.

8.1 HYPERBOLIC EQUATIONS: WAVES

To see how the stability of the solution depends on the finite difference scheme, let's start with a simple first-order hyperbolic PDE for a conserved quantity in one dimension

$$\frac{\partial u}{\partial t}=-v\frac{\partial u}{\partial x} \quad . \qquad (8.6)$$

Substitution readily shows that this is solved by any function of the form

$$u=f(x-vt) \quad . \qquad (8.7)$$

Writing $u(j\Delta x, n\Delta t) = u_j^n$ to make the notation clearer, a simple discretization is first-order in time and second-order in space:

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}=-v\left(\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}\right)$$

$$u_j^{n+1}=u_j^n-\frac{v\Delta t}{2\Delta x}(u_{j+1}^n-u_{j-1}^n) \qquad (8.8)$$

(using a first-order spatial approximation would make it asymmetrical). It can be convenient to represent such approximations by drawing the cluster of values used in the update rule (Figure 8.1). Given an initial distribution u_j^n, it is straightforward to iterate this rule forward in time.

To analyze the stability of a finite difference scheme, the *von Neumann stability* *analysis* locally linearizes the equations (if they are not linear) and then separates the temporal and spatial dependence (Section 4.3) to look at the growth of the linear modes

$$u_j^n = A(k)^n e^{ijk\Delta x} \quad . \tag{8.9}$$

This assumed form has an oscillatory dependence on space, which can be used to synthesize any initial condition, and an exponential dependence on time to test the stability. It's customary to define k with respect to the spatial separation Δx, because that will parameterize the solution, but use an integer n for the time step since that will be used only for checking the stability.

Plugging in this ansatz gives a solution to the finite difference equation for $A(k)$. If $|A(k)| > 1$ for some k, then these modes will diverge and the scheme will be unstable (remember that the exact solution (8.7) does not diverge). For equation (8.8) this gives

$$\begin{aligned} A^{n+1}e^{ijk\Delta x} &= A^n e^{ijk\Delta x} - \frac{v\Delta t}{2\Delta x}\left(A^n e^{i(j+1)k\Delta x} - A^n e^{i(j-1)k\Delta x}\right) \\ A &= 1 - \frac{v\Delta t}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right) \\ &= 1 - i\frac{v\Delta t}{\Delta x}\sin k\Delta x \quad . \end{aligned} \tag{8.10}$$

The absolute magnitude of this is always greater than 1, and so this scheme is always unstable. Any initial condition will diverge!

This disturbing behavior in such a sensible approximation is easily corrected with the *Lax method*, which averages the neighbors for the time derivative:

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{v\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n) \quad . \tag{8.11}$$

Repeating the stability analysis shows that the amplitude of a solution is

$$A = \cos k\Delta x - i\frac{v\Delta t}{\Delta x}\sin k\Delta x \quad . \tag{8.12}$$

Requiring that the magnitude be less than 1,

$$\begin{aligned} |A|^2 = \cos^2 k\Delta x + \left(\frac{v\Delta t}{\Delta x}\right)^2 \sin^2 k\Delta x \le 1 \\ \Rightarrow \frac{|v|\Delta t}{\Delta x} \le 1 \quad . \end{aligned} \tag{8.13}$$

This is the *Courant–Friedrichs–Levy* stability criterion, and it will recur for a number of other schemes. It says that the velocity at which information propogates within the numerical algorithm ($\Delta x/\Delta t$) must be faster than the velocity of the solution v. For space and time steps that satisfy this condition, the Lax method will be stable. Otherwise, there is a "numerical boom" as the real solution tries to out-run the rate at which the numerical solution can advance. The lateral averaging for the time derivative in the Lax method helps the numerical information propagate, compared to the unstable approximation that we started with (equation 8.8). The origin of this stability becomes clearer if the Lax method is rewritten by subtracting u_j^n from both sides:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v\left(\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\right) + \frac{1}{2\Delta t}(u_{j+1}^n - 2u_j^n + u_{j-1}^n) \quad . \tag{8.14}$$

This is just our original equation (8.8), with an extra fictitious diffusion term added that depends on the discretization:

$$\frac{\partial u}{\partial t} = -v\frac{\partial u}{\partial x} + \frac{(\Delta x)^2}{2\Delta t}\frac{\partial^2 u}{\partial x^2} \quad . \tag{8.15}$$

This is an example of an artificial *numerical dissipation*, which can occur (and even be added intentionally) in stable schemes. In this case it is good, because it serves to damp out the spurious high-frequency modes ($k \sim 1$) while preserving the desired long wavelength solutions. In other cases it might be a problem if the goal is to look at the long-term behavior of a nondissipative system.

The Lax method cures the stability problem and is accurate to second order in space, but it is only first-order in time. This means that $v\Delta t$ will need to be much smaller than Δx to have the same accuracy in time and space (even though a much larger time step will be stable). A natural improvement is to go to second order in time:

$$u_j^{n+1} = u_j^{n-1} - \frac{v\Delta t}{\Delta x}\left(u_{j+1}^n - u_{j-1}^n\right) \quad . \tag{8.16}$$

The stability analysis for this equation now leads to a quadratic polynomial for the amplitude, giving two solutions

$$A = -i\frac{v\Delta t}{\Delta x}\sin(k\Delta x) \pm \sqrt{1 - \left[\frac{v\Delta t}{\Delta x}\sin(k\Delta x)\right]^2} \quad . \tag{8.17}$$

If $|v|\Delta t/\Delta x \leq 1$ then the radical will be real, and $|A|^2 = 1$ independent of k. The Courant condition applies again, but now there is no dependence of the amplitude on the spatial wavelength k and so there is no artificial damping (unlike the Lax method). This is called the *leapfrog method* because it separates the space into two interpenetrating lattices that do not influence each other (u_j^{n+1} does not depend on u_j^n). Numerical round-off errors can lead to a divergence of the sublattices over long times, requiring the addition of an artificial coupling term.

Problem 8.1 considers the finite difference approximaton to the wave equation.

8.2 PARABOLIC EQUATIONS: DIFFUSION

We will next look for finite difference approximations for the 1D diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial u}{\partial x}\right) \quad , \tag{8.18}$$

and will assume that the diffusion coefficient is constant

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \quad . \tag{8.19}$$

The methods to be described will have natural generalizations when D is not constant.

The straightforward discretization is

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\left[\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}\right]$$

$$u_j^{n+1} = u_j^n + \frac{D\Delta t}{(\Delta x)^2}\left[u_{j+1}^n - 2u_j^n + u_{j-1}^n\right] \quad . \tag{8.20}$$

Solving the stability analysis,

$$\begin{aligned} A &= 1 + \frac{D\Delta t}{(\Delta x)^2} \underbrace{\underbrace{\left[e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right]}_{2\cos k\Delta x - 2}}_{2\left(2\cos^2\frac{k\Delta x}{2} - 1\right) - 2} \\ &= 1 - \frac{4D\Delta t}{(\Delta x)^2}\sin^2\frac{k\Delta x}{2} \end{aligned}$$

$$|A| \le 1 \Rightarrow \frac{4D\Delta t}{(\Delta x)^2} \le 2 \;\Rightarrow\; \frac{2D\Delta t}{(\Delta x)^2} \le 1 \quad . \tag{8.21}$$

The method is stable for small step sizes, but since for a diffusive process the time t to expand a distance L is roughly $t \sim L^2/D$ (Problem 4.2), the number of time steps required to model this will be $\sim L^2/(\Delta x)^2$ (i.e., a *very* large number).

The stability can be improved by evaluating the space derivative forwards in time:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\left[\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{(\Delta x)^2}\right]$$

$$u_j^{n+1} - \frac{D\Delta t}{(\Delta x)^2}\left[u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}\right] = u_j^n \quad . \tag{8.22}$$

The stability analysis for this is

$$\begin{aligned} A - \frac{D\Delta t}{(\Delta x)^2}\left[Ae^{ik\Delta x} - 2A + Ae^{-ik\Delta x}\right] &= 1 \\ A\left[1 + \frac{4D\Delta t}{(\Delta x)^2}\sin^2\frac{k\Delta x}{2}\right] &= 1 \\ A &= \frac{1}{1 + \frac{4D\Delta t}{(\Delta x)^2}\sin^2\frac{k\Delta x}{2}} \le 1 \; . \end{aligned} \tag{8.23}$$

This scheme is stable for all step sizes, but might appear to be useless: how can we implement it since we don't know the forward values used in the space derivative? These future values are implicitly determined by the past values, and the trick is to recognize that the full set of equations can be inverted. The stability follows because peeking into the future in this way helps move information through the solution more quickly.

The boundary conditions are typically given as either *fixed* (u_1 and u_N are specified) or *periodic* ($u_1 = u_{N+1}$, so that the system does not have edges). If we assume fixed boundary conditions and define $\alpha = D\Delta t/(\Delta x)^2$, then equation (8.22) can be written as a matrix problem

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & & & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & \cdots & & 0 \\ & \ddots & \ddots & \ddots & & & \\ \vdots & 0 & -\alpha & 1+2\alpha & -\alpha & 0 & \vdots \\ & & & \ddots & \ddots & \ddots & \\ 0 & & \cdots & 0 & -\alpha & 1+2\alpha & -\alpha \\ 0 & & & \cdots & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_i^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \\ u_N^{n+1} \end{pmatrix} = \begin{pmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_i^n \\ \vdots \\ u_{N-1}^n \\ u_N^n \end{pmatrix}$$

This is a *tridiagonal* matrix (all the elements are zero, except for the diagonal and the adjacent elements), and it can easily be solved to find u^{n+1} in terms of u^n without doing all of the work needed to invert an arbitrary matrix.

The system of equations corresponding to an arbitrary tridiagonal matrix is

$$\begin{aligned} b_1u_1 + c_1u_2 &= d_1 \\ a_iu_{i-1} + b_iu_i + c_iu_{i+1} &= d_i \quad (2 \le i \le N-1) \quad . \\ a_Nu_{N-1} + b_Nu_N &= d_N \end{aligned} \tag{8.24}$$

For us, $a_i = c_i = -\alpha$, $b_i = 1 + 2\alpha$, and the $d_i = u_i^{n+1}$. These can be solved in two passes. In a system of equations, multiplying one equation by a constant and adding it to another one does not change the solution. If we multiply the first row by $-a_2/b_1$ and add it to the second row this will eliminate the a_2 term. If we then divide the second row by the new b_2 term, and repeat these steps (*Gauss elimination*) down the matrix, we will get a new matrix with zeros below the diagonal and ones on the diagonal (this is called an *upper-diagonal* matrix). Then, a reverse pass back up the matrix that multiplies the new Nth row by the new values for $-c_{N-1}/b_N$ and adds it to the previous row, and so forth, converts the matrix to a diagonal one and the solution can be read off. Using primes for the values after the forward pass, a bit of algebra shows that

$$\begin{aligned} c_1' &= \frac{c_1}{b_1} \qquad d_1' = \frac{d_1}{b_1} \\ c_{i+1}' &= \frac{c_{i+1}}{b_{i+1} - a_{i+1}c_i'} \qquad d_{i+1}' = \frac{d_{i+1} - a_{i+1}d_i'}{b_{i+1} - a_{i+1}c_i'} \quad . \end{aligned} \tag{8.25}$$

Then, the reverse pass gives

$$\begin{aligned} u_N &= d_N' \\ u_i &= d_i' - c_i'u_{i+1} \quad . \end{aligned} \tag{8.26}$$

This is an $\mathcal{O}(N)$ steps algorithm, and so there is little performance penalty for using an implicit discretization instead of an explicit one.

The accuracy can be improved to second order in time by averaging the spatial derivative at the beginning and the end of the interval:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{D}{2(\Delta x)^2}[(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}) + (u_{j+1}^n - 2u_j^n + u_{j-1}^n)] \quad . \tag{8.27}$$

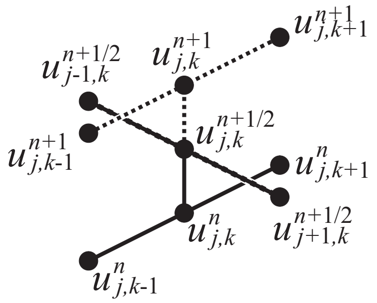

Figure 8.2. Computational clusters for alternating implicit steps for the x (solid) and y (dashed) coordinates.

This is called the *Crank–Nicholson* method, and the stability analysis shows that it is stable for any time step:

$$A = \frac{1-(2D\Delta t/(\Delta x)^2)\sin^2(kx/2)}{1+(2D\Delta t/(\Delta x)^2)\sin^2(kx/2)} \quad . \tag{8.28}$$

To solve a diffusion problem in higher dimensions we could make it implicit in all of the dimensions. This works, but results in a banded matrix that is no longer tridiagonal. Although it can be inverted (with much more effort), a simple trick recovers tridiagonal matrices. This is the *Alternating-Direction Implicit* method (*ADI*), which is an example of the general principle of *Operator Splitting*. In 2D, instead of advancing both coordinates in one step, the x coordinates are advanced in a first implicit (tridiagonal) step of $\Delta t/2$, and then in a second implicit step of $\Delta t/2$ the new y coordinates are found. For the implicit method this is

$$\begin{aligned} u^{n+1/2}_{j,k} &= u^n_{j,k} + \frac{D\Delta t}{2(\Delta x)^2}\,(u^{n+1/2}_{j+1,k} - 2u^{n+1/2}_{j,k} + u^{n+1/2}_{j-1,k} + \\ &\qquad u^n_{j,k+1} - 2u^n_{j,k} + u^n_{j,k-1}) \\ u^{n+1}_{j,k} &= u^{n+1/2}_{j,k} + \frac{D\Delta t}{2(\Delta x)^2}\,(u^{n+1/2}_{j+1,k} - 2u^{n+1/2}_{j,k} + u^{n+1/2}_{j-1,k} + \\ &\qquad u^{n+1}_{j,k-1} - 2u^{n+1}_{j,k} + u^{n+1}_{j,k+1}) \end{aligned} \tag{8.29}$$

(shown in Figure 8.2).

8.3 ELLIPTIC EQUATIONS: BOUNDARY VALUES

The remaining class of partial differential equations to be discussed are of the form of the elliptic boundary value problem

$$\nabla^2 u = \rho \quad . \tag{8.30}$$

This is *Poisson's equation*; if the source term $\rho = 0$ then it becomes *Laplace's equation*. The boundary condition can be specified by giving the value of u on the boundary (*Dirichlet conditions*), the value of the normal derivative of u on the boundary (*Neumann conditions*), or a mixture of these. Poisson's equation and Laplace's equation are among the most commonly solved numerical equations because they apply to so many different areas. It's instructive to look at some important examples to see how they arise:

- *Heat Flow*

  The heat flux $\vec{F}$ in a material is proportional to the gradient of the temperature T by the thermal conductivity K:

  $$\vec{F} = -K\nabla T \quad . \tag{8.31}$$

  The change of heat Q in a volume is related to the temperature change by the specific heat C and the density ρ by

  $$\frac{dQ}{dt} = \int_V C\rho \frac{\partial T}{\partial t}\, dV \quad , \tag{8.32}$$

  and it is also equal to the surface integral of the heat flux

  $$\frac{dQ}{dt} = -\int_S \vec{F} \cdot d\vec{A} \quad . \tag{8.33}$$

  Equating these and using Gauss' theorem,

  $$\begin{aligned} \int C\rho \frac{\partial T}{\partial t}\, dV &= -\int \vec{F} \cdot d\vec{A} \\ &= -\int \nabla \cdot \vec{F}\, dV \\ &= \int K\nabla^2 T\, dV \\ \Rightarrow \nabla^2 T &= \frac{1}{\kappa}\frac{\partial T}{\partial t} \quad , \end{aligned} \tag{8.34}$$

  where $\kappa = K/C\rho$ is the thermal diffusivity. This is a diffusion equation, and for a steady-state problem the time derivative of T will vanish, leaving Laplace's equation for the temperature distribution.

- *Fluid Flow*

  The continuity equation for a fluid of density ρ and velocity $\vec{v}$ is

  $$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho\vec{v} = 0 \quad . \tag{8.35}$$

  If the density is constant this reduces to

  $$\nabla \cdot \vec{v} = 0 \quad . \tag{8.36}$$

  A second condition on the velocity field is that if it starts out irrotational ($\nabla \times \vec{v} = 0$) it will remain irrotational (for example, this will be the case if at $-\infty$ the flow is uniform; see any fluids text such as [Batchelor, 1967] for a derivation). If the curl of a vector field vanishes, it can be written as the gradient of a potential:

  $$\nabla \times \vec{v} = 0 \Rightarrow \vec{v} = -\nabla\varphi \quad . \tag{8.37}$$

Combining this with the continuity equation gives Laplace's equation again

$$\nabla \cdot (\nabla \varphi) = \nabla^2 \varphi = 0 \quad . \tag{8.38}$$

- *Electric Fields*
  In MKS units, the electric field $\vec{E}$ is determined in terms of the charge density ρ, the magnetic field $\vec{B}$, and the polarizability ϵ by

$$\nabla \cdot \epsilon \vec{E} = \rho \qquad \nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \quad . \tag{8.39}$$

  For a steady-state problem, the time derivative of $\vec{B}$ is zero, and so the electric field is the gradient of a potential φ

$$\nabla \times \vec{E} = 0 \Rightarrow \vec{E} = -\nabla \varphi \quad . \tag{8.40}$$

  Combining this with the charge equation and assuming that ϵ does not depend on position gives Poisson's equation

$$\nabla^2 \varphi = -\frac{\rho}{\epsilon} \quad . \tag{8.41}$$

A problem closely related to Laplace's equation is *Helmholtz's Equation*, which we found by separating out the time dependence in a wave equation:

$$\nabla^2 u = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2}$$

$$u = Ae^{i\omega t} \Rightarrow e^{i\omega t} \nabla^2 A = -\frac{\omega^2}{c^2} A e^{i\omega t}$$

$$\nabla^2 A + \underbrace{\frac{\omega^2}{c^2}}_{k^2} A = 0 \quad . \tag{8.42}$$

The obvious finite difference approximation for Poisson's equation is (in 1D)

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{(\Delta x)^2} = \rho_j \quad . \tag{8.43}$$

Unlike the initial value problems we have been studying, this has no time dependence. It can be written as a matrix problem which can be solved exactly:

$$\mathbf{A} \cdot \vec{u} = (\Delta x)^2 \vec{\rho} \quad \Rightarrow \quad \vec{u} = (\Delta x)^2 \mathbf{A}^{-1} \cdot \vec{\rho} \quad , \tag{8.44}$$

where

$$\mathbf{A} = \left(\begin{array}{ccccccc} -2 & 1 & 0 & \cdots & & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & & 0 \\ 0 & \ddots & \ddots & \ddots & & & \\ \vdots & 0 & 1 & -2 & 1 & 0 & \vdots \\ & & & \ddots & \ddots & \ddots & 0 \\ 0 & & \cdots & 0 & 1 & -2 & 1 \\ 1 & 0 & & \cdots & 0 & 1 & -2 \end{array} \right) \quad \text{and} \quad \vec{u} = \left(\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_i \\ \vdots \\ u_{N-1} \\ u_N \end{array} \right) \quad . \tag{8.45}$$

This is for periodic boundary conditions; with fixed boundary conditions the upper-right and lower-left corner elements of **A** would be zero. For Helmholtz's equation the matrix problem requires finding the eigenvalues and eigenvectors of **A** to determine the modes.

With fixed boundary conditions **A** is tridiagonal, which we saw in Section 8.2 is easy to solve. For periodic boundary conditions it is almost tridiagonal, but with the extra corner elements added. Fortunately, if a matrix is related to another one by adding the outer product of two vectors

$$\mathbf{A} \rightarrow (\mathbf{A} + \vec{u} \otimes \vec{v}) \qquad (\vec{u} \otimes \vec{v})_{i,j} \equiv u_i v_j \tag{8.46}$$

then there is a simple relationship between their inverses. In our case, $\vec{u} = \vec{v} = (1, 0, \ldots, 0, 1)$, and -1 is subtracted from the upper-left and lower-right elements of **A** before the inversion. The inverses are related by the *Sherman–Morrison* formula, derived by doing a power series expansion of the inverse and then using the associativity of the inner and outer products

$$\begin{aligned}
(\mathbf{A} + \vec{u} \otimes \vec{v})^{-1} &= (\mathbf{A} \cdot (\mathbf{1} + \mathbf{A}^{-1} \cdot \vec{u} \otimes \vec{v}))^{-1} \\
&= (\mathbf{1} + \mathbf{A}^{-1} \cdot \vec{u} \otimes \vec{v})^{-1} \cdot \mathbf{A}^{-1} \\
&= [\mathbf{1} - (\mathbf{A}^{-1} \cdot \vec{u} \otimes \vec{v}) + \\
&\quad (\mathbf{A}^{-1} \cdot \vec{u} \otimes \vec{v}) \cdot (\mathbf{A}^{-1} \cdot \vec{u} \otimes \vec{v}) - \cdots] \cdot \mathbf{A}^{-1} \\
&= [\mathbf{1} - (\mathbf{A}^{-1} \cdot \vec{u} \otimes \vec{v}) + \\
&\quad \mathbf{A}^{-1} \cdot \vec{u} \otimes \underbrace{(\vec{v} \cdot \mathbf{A}^{-1} \cdot \vec{u})}_{\beta} \otimes \vec{v} - \cdots] \cdot \mathbf{A}^{-1} \\
&= [\mathbf{1} - (\mathbf{A}^{-1} \cdot \vec{u} \otimes \vec{v})(1 - \beta + \beta^2 - \cdots)] \cdot \mathbf{A}^{-1} \\
&= \mathbf{A}^{-1} - \frac{(\mathbf{A}^{-1} \cdot \vec{u}) \otimes (\vec{v} \cdot \mathbf{A}^{-1})}{1 + \beta} \quad .
\end{aligned} \tag{8.47}$$

In 2D, the finite difference approximation for Poisson's equation is

$$\frac{u_{j+1,k} + u_{j-1,k} + u_{j,k+1} + u_{j,k-1} - 4u_{j,k}}{(\Delta x)^2} = \rho_{j,k} \quad . \tag{8.48}$$

This can be also be solved by a matrix inversion:

$$\mathbf{A} \cdot \begin{pmatrix} u_{1,1} \\ u_{2,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ \vdots \\ u_{N,2} \\ \vdots \\ u_{N,N} \end{pmatrix} = \Delta x^2 \begin{pmatrix} \rho_{1,1} \\ \rho_{2,1} \\ \vdots \\ \rho_{N,1} \\ \rho_{1,2} \\ \vdots \\ \rho_{N,2} \\ \vdots \\ \rho_{N,N} \end{pmatrix} \quad , \tag{8.49}$$

where

$$\mathbf{A} = \begin{pmatrix} \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & & \\ 0 & 1 & 0 & \cdots & 0 & 1 & -4 & 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & \\ \cdots & 0 & 1 & 0 & \cdots & 0 & 1 & -4 & 1 & 0 & \cdots & 0 & 1 & 0 & \cdots \\ & \cdots & 0 & 1 & 0 & \cdots & 0 & 1 & -4 & 1 & 0 & \cdots & 0 & 1 & 0 \\ & & & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots \end{pmatrix} \tag{8.50}$$

The generalization to higher dimensions is straightforward, making a column vector from a multidimensional array by sequentially reading down the axes.

Matrix (8.50) is banded-diagonal, having 1s offset by N elements on either side of the diagonal, and in higher dimensions there will be more bands. Although these are sparse matrices which can be solved directly (see [Press *et al.*, 2007] for techniques), the effort required in two or more dimensions can quickly become prohibitive.

One alternative that is applicable for constant-coefficient linear problems is *Fourier Transform* methods. In 2D, the discrete Fourier transform of the field is

$$\hat{u}_{m,n} = \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} u_{j,k} e^{2\pi i m j/M} e^{2\pi i n k/N} \quad , \tag{8.51}$$

and the inverse transform is

$$u_{j,k} = \frac{1}{N^2} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \hat{u}_{m,n} e^{-2\pi i j m/M} e^{-2\pi i k n/N} \quad . \tag{8.52}$$

Plugging the transforms of u and ρ into equation (8.48), and recognizing that the transform of a function can vanish everywhere only if the function itself is equal to zero, gives

$$\begin{aligned} \hat{u}_{m,n} \left(e^{2\pi i m/M} + e^{-2\pi i m/N} + e^{2\pi i n/M} + e^{-2\pi i n/N} - 4 \right) \\ = \hat{\rho}_{m,n} (\Delta x)^2 \quad . \end{aligned} \tag{8.53}$$

Rearranging terms and simplifying the complex exponentials,

$$\hat{u}_{m,n} = \frac{\hat{\rho}_{m,n} (\Delta x)^2}{2 \cos \dfrac{2\pi m}{M} + 2 \cos \dfrac{2\pi n}{N} - 4} \quad . \tag{8.54}$$

Therefore, the forward transform of the source term can be calculated, this can be used to find the $\hat{u}$, and then the inverse transform can be taken to find u. This solution imposes periodic boundary conditions; some other boundary conditions can be imposed by choosing the form of the expansion (for example, using only sines if the solution vanishes on the boundary).

The Fourier transform method is so simple only for linear constant-coefficient problems with boundary conditions along the coordinate axes; some kind of iterative algorithm is needed for arbitrary geometries. An important class of techniques is found by remembering that Poisson's equation is the steady-state solution of a diffusion equation

$$\frac{\partial u}{\partial t} = \nabla^2 u - \rho \tag{8.55}$$

whether or not it originally arose from diffusion. This means that all of the techniques for solving diffusion problems can be applied here, with the asymptotic answer giving the solution to Poisson's equation.

The simplest is *Jacobi's method*, which just takes forward time differences

$$u_{j,k}^{n+1} = u_{j,k}^n + \frac{\Delta t}{(\Delta x)^2}(u_{j+1,k}^n + u_{j-1,k}^n + u_{j,k+1}^n + u_{j,k-1}^n - 4u_{j,k}^n) - \Delta t \rho_{j,k} \quad . \tag{8.56}$$

Here the time step does not have any physical significance; we just want the largest possible step that converges to the solution. In 2D the Courant condition is $\Delta t/(\Delta x)^2 \leq 1/4$, leading to

$$u_{j,k}^{n+1} = \frac{1}{4}(u_{j+1,k}^n + u_{j-1,k}^n + u_{j,k+1}^n + u_{j,k-1}^n) - \frac{(\Delta x)^2}{4}\rho_{j,k} \quad . \tag{8.57}$$

This has a very natural interpretation: starting from a random guess, at each time step each lattice site is set to the average of its neighbors and then a source term is added. This process is repeated until the solution stops changing, a technique called *relaxation*. A related algorithm, the *Gauss–Seidel* method, uses updated values as soon as they become available:

$$u_{j,k}^{n+1} = \frac{1}{4}(u_{j+1,k}^n + u_{j-1,k}^{n+1} + u_{j,k+1}^n + u_{j,k-1}^{n+1}) - \frac{(\Delta x)^2}{4}\rho_{j,k} \tag{8.58}$$

(assuming that the updating proceeds down rows).

These both work, but the convergence is too slow for them to be useful. This can be seen by rewriting them in terms of the matrix problem

$$\mathbf{A} \cdot \vec{u} = \vec{\rho} \tag{8.59}$$

and then separating **A** into lower-triangular, diagonal, and upper-triangular parts

$$(\mathbf{L} + \mathbf{D} + \mathbf{U}) \cdot \vec{u} = \vec{\rho} \quad . \tag{8.60}$$

If $\vec{u}_n$ is the solution after the nth iteration, the Jacobi method moves the lower- and upper-triangular parts to the right hand side to update $\vec{u}$:

$$\mathbf{D} \cdot \vec{u}_{n+1} = -(\mathbf{L} + \mathbf{U}) \cdot \vec{u}_n + \vec{\rho} \tag{8.61}$$

The convergence rate will be determined by the eigenvalues of the iteration matrix $-\mathbf{D}^{-1} \cdot (\mathbf{L} + \mathbf{U})$. The magnitude of all of the eigenvalues must be less than 1 for stability, and the largest eigenvalue determines the overall convergence rate (the largest eigenvalue is called the *spectral radius* r_s). For a large $N \times N$ square lattice problem, the spectral radius is asymptotically equal to [Ames, 1992]

$$r_{\text{Jacobi}} \simeq 1 - \frac{\pi^2}{2N^2} \quad . \tag{8.62}$$

Therefore, reducing the error by a factor of 10 requires $-\ln 10/\ln r_s \simeq N^2$ steps. In the Gauss-Seidel method, the lower-triangular part is moved over to the left side:

$$(\mathbf{L} + \mathbf{D}) \cdot \vec{u}_{n+1} = -\mathbf{U} \cdot \vec{u}_n + \vec{\rho} \quad . \tag{8.63}$$

For the square 2D lattice, this has a spectral radius of

$$r_{\text{Gauss-Seidel}} \simeq 1 - \frac{\pi^2}{N^2} \quad , \tag{8.64}$$

and so the number of steps needed to reduce the error by a factor of 10 is half that required by the Jacobi method. For a 100 × 100 lattice, both of these methods require $\sim 10^4$ steps for an improvement of a factor of 10 in the answer, which is usually prohibitive. The Gauss-Seidel method is preferable to the Jacobi method because it converges faster and does not require auxiliary storage, but something better than both is needed.

The Gauss-Seidel method can be rewritten in a suggestive form as follows:

$$\begin{aligned}(\mathbf{L}+\mathbf{D})\cdot\vec{u}_{n+1} &= -\mathbf{U}\cdot\vec{u}_n+\vec{\rho}\\ \vec{u}_{n+1} &= (\mathbf{L}+\mathbf{D})^{-1}[-\mathbf{U}\cdot\vec{u}_n+\vec{\rho}]\\ &= \vec{u}_n-(\mathbf{L}+\mathbf{D})^{-1}\cdot[\mathbf{U}\cdot\vec{u}_n-\vec{\rho}]-\vec{u}_n\\ &= \vec{u}_n-(\mathbf{L}+\mathbf{D})^{-1}\cdot[(\mathbf{L}+\mathbf{D}+\mathbf{U})\cdot\vec{u}_n-\vec{\rho}]\\ &= \vec{u}_n-(\mathbf{L}+\mathbf{D})^{-1}\cdot[\mathbf{A}\cdot\vec{u}_n-\vec{\rho}]\\ &= \vec{u}_n-(\mathbf{L}+\mathbf{D})^{-1}\cdot\vec{E}_n \quad ,\end{aligned} \tag{8.65}$$

where $\vec{E}_n$ is the error at the nth time step. In each update, the error gets multiplied by $(\mathbf{L}+\mathbf{D})^{-1}$ and subtracted from the state. The idea of *Successive Over-Relaxation* (*SOR*) is to extrapolate this correction and subtract a larger change

$$\vec{u}_{n+1}=\vec{u}_n-\alpha(\mathbf{L}+\mathbf{D})^{-1}\cdot\vec{E}_n \quad . \tag{8.66}$$

It can be shown that this converges for $0<\alpha<2$ [Ames, 1992]. When $\alpha=1$ this is just the Gauss–Seidel method, $\alpha<1$ is underrelaxation (which slows the convergence), and $1<\alpha<2$ is overrelaxation. The convergence rate depends on the value of α; choosing a value that is too large is as bad as choosing one that is too small because the solution will overshoot the final value. The optimal relaxation rate is

$$\alpha=\frac{2}{1+\sqrt{1-\rho^2_{\text{Jacobi}}}} \quad , \tag{8.67}$$

which leads to an asymptotic spectral radius of

$$r_{\text{SOR}}\simeq 1-\frac{2\pi}{N} \quad . \tag{8.68}$$

This reduces the number of steps needed to reduce the error by a factor of 10 to $\mathcal{O}(N)$, which is now proportional to the grid size rather than the square of the grid size. Written out in components, for the 2D case SOR is

$$u^{n+1}_{j,k}=(1-\alpha)u^n_{j,k}+\frac{\alpha}{4}(u^n_{j+1,k}+u^{n+1}_{j-1,k}+u^n_{j,k+1}+u^{n+1}_{j,k-1})-\frac{\alpha(\Delta x)^2}{4}\rho_{j,k} \tag{8.69}$$

(remember that this is not an implicit method, because the forward time steps on the right-hand side come from preceeding spatial updates).

SOR is very easy to program, but does require determining the relaxation parameter α (although this can be estimated empirically, since if α is too large the solution will oscillate). An alternative is to use ADI, which permits larger time steps at the expense of algorithm complexity. For large problems that require repeated fast solution both techniques have been superseded by *multigrid* methods [Press *et al.*, 2007], which find the final solution on N grid points in $\mathcal{O}(N)$ steps. These methods are based on iteratively coarse-graining the problem to produce a simpler one that can be solved quickly, and then interpolating to find the approximate solution at higher resolution. This is analogous to the Richardson extrapolation methods for ODEs, but because of the extra dimensions they are more complicated to implement.

Finally, for a problem such as weather forecasting, the simulation must contend with not just numerical errors but also uncertainty in the initial conditions and the model selection. Although these can't be eliminated, they can be reduced by combining predictions from ensembles of models [Krishnamurti *et al.*, 1999]. This theme will return in Part Three.

8.4 SELECTED REFERENCES

[Ames, 1992] Ames, William F. (1992). *Numerical Methods for Partial Differential Equations*. 3rd edn. Boston, MA: Academic Press.

Ames is a classic reference for numerical methods for PDEs, with a bit more emphasis on mathematical rigor than on practical advice.

8.5 PROBLEMS

(8.1) Consider the 1D wave equation

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} \quad . \tag{8.70}$$

(*a*) Write down the straightforward finite-difference approximation.

(*b*) What order approximation is this in time and in space?

(*c*) Use the von Neumann stability criterion to find the mode amplitudes.

(*d*) Use this to find a condition on the velocity, time step, and space step for stability (hint: consider the product of the two amplitude solutions).

(*e*) Do different modes decay at different rates for the stable case?

(*f*) Numerically solve the wave equation for the evolution from an initial condition with $u = 0$ except for one nonzero node, and verify the stability criterion.

(*g*) If the equation is replaced by

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} + \gamma \frac{\partial}{\partial t} \frac{\partial^2 u}{\partial x^2} \quad , \tag{8.71}$$

assume that

$$u(x,t) = Ae^{i(kx-\omega t)} \tag{8.72}$$

and find a relationship between k and ω, and simplify it for small γ. Comment on the relationship to the preceeding question.

(*h*) Repeat the numerical solution of the wave equation with the same initial conditions, but include the damping term.

(8.2) Write a program to solve a 1D diffusion problem on a lattice of 500 sites, with an initial condition of zero at all the sites, except the central site which starts at the value 1.0. Take $D = \Delta x = 1$, and use fixed boundary conditions set equal to zero.

(*a*) Use the explicit finite difference scheme, and look at the behavior for $\Delta t = 1$, 0.5, and 0.1. What step size is required by the Courant condition?

(*b*) Now repeat this using implicit finite differences and compare the stability.

(8.3) Use ADI to solve a 2D diffusion problem on a lattice, starting with randomly seeded values.

(8.4) Use SOR to solve Laplace's equation in 2D, with boundary conditions $u_{j,1} = u_{1,k} = 0, u_{N,k} = -1, u_{j,N} = 1$, and explore how the convergence rate depends on α, and how the best choice for α depends on the lattice size.