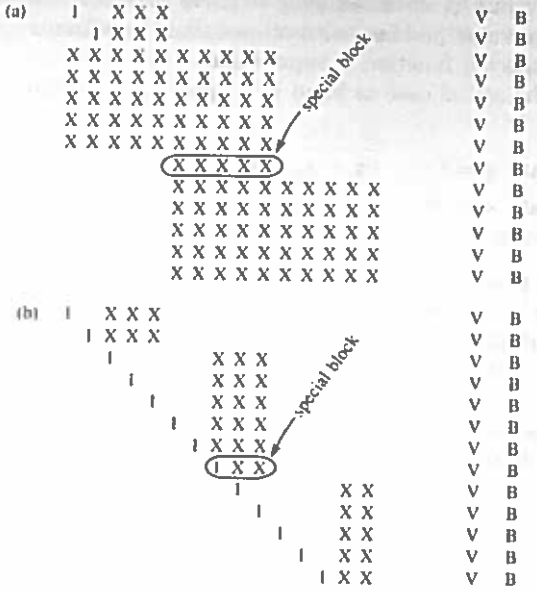

Figure 16.6.1. FDE matrix structure with an internal boundary condition. The internal condition introduces a special block. (a) Original form, compare with Figure 16.3.1; (b) final form, compare with Figure 16.3.2.

REFERENCES AND FURTHER READING:

London, R.A., and Flannery, B.P. 1982, *The Astrophysical Journal*, vol. 258, pp. 260–269.

Chapter 17. Partial Differential Equations

17.0 Introduction

The numerical treatment of partial differential equations is, by itself, a vast subject. Partial differential equations are at the heart of many, if not most, computer analyses or simulations of continuous physical systems, such as fluids, electromagnetic fields, the human body, etc. The intent of this chapter is to give the briefest possible useful introduction. Ideally, there would be an entire second volume of *Numerical Recipes* dealing with partial differential equations alone. (The references below provide, of course, available alternatives.)

In most mathematics books, partial differential equations (PDEs) are classified into the three categories, *hyperbolic*, *parabolic* and *elliptic*, on the basis of their *characteristics*, or curves of information propagation. The prototypical example of a hyperbolic equation is the one-dimensional *wave* equation

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} \tag{17.0.1}$$

where v = constant is the velocity of wave propagation. The prototypical parabolic equation is the *diffusion* equation

$$\frac{\partial u}{\partial t} = -\frac{\partial}{\partial x}\left(D\frac{\partial u}{\partial x}\right) \tag{17.0.2}$$

where D is the diffusion coefficient. The prototypical elliptic equation is the *Poisson* equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y) \tag{17.0.3}$$

where the source term ρ is given. If the source term is equal to zero, the equation is *Laplace's equation*.

From a computational point of view, the classification into these three canonical types is not very meaningful — or at least not as important as some other essential distinctions. Equations (17.0.1) and (17.0.2) both define *initial value* or *Cauchy* problems: If information on u (perhaps including time derivative information) is given at some initial time t_0 for all x, then the equations describe how $u(x,t)$ propagates itself forward in time. In other words, equations (17.0.1) and (17.0.2) describe time evolution. The goal of a numerical code should be to track that time evolution with some desired accuracy.

By contrast, equation (17.0.3) directs us to find a single "static" function $u(x,y)$ which satisfies the equation within some (x,y) region of interest, and which – one must also specify – has some desired behavior on the boundary of the region of interest. These problems are called *boundary value problems*. In general it is not possible stably to just "integrate in from the boundary" in the same sense that an initial value problem can be "integrated forward in time." Therefore, the goal of a numerical code is somehow to converge on the correct solution everywhere at once.

This, then, is the most important classification from a computational point of view: Is the problem at hand an *initial value* (time evolution) problem? or is it a *boundary value* (static solution) problem? Figure 17.0.1 emphasizes the distinction. Notice that while the italicized terminology is standard, the terminology in parentheses is a much better description of the dichotomy from a computational perspective. The subclassification of initial value problems into parabolic and hyperbolic is much less important because (i) many actual problems are of a mixed type, and (ii) as we will see, most hyperbolic problems get parabolic pieces mixed into them by the time one is discussing practical computational schemes.

Initial Value Problems

An initial value problem is defined by answers to the following questions:

- What are the dependent variables to be propagated forward in time?
- What is the evolution equation for each variable? Usually the evolution equations will all be coupled, with more than one dependent variable appearing on the right-hand side of each equation.
- What is the highest time derivative that occurs in each variable's evolution equation? If possible, this time derivative should be put alone on the equation's left-hand side. Not only the value of a variable, but also the value of all its time derivatives — up to the highest one — must be specified to define the evolution.
- What special equations (boundary conditions) govern the evolution in time of points on the boundary of the spatial region of interest? Examples: *Dirichlet conditions* specify the values of the boundary points as a function of time; *Neumann conditions* specify the values of the normal gradients on the boundary; *outgoing-wave boundary conditions* are just what they say.

Sections 17.1–17.3 of this chapter deal with initial value problems of several different forms. We make no pretence of completeness, but rather hope

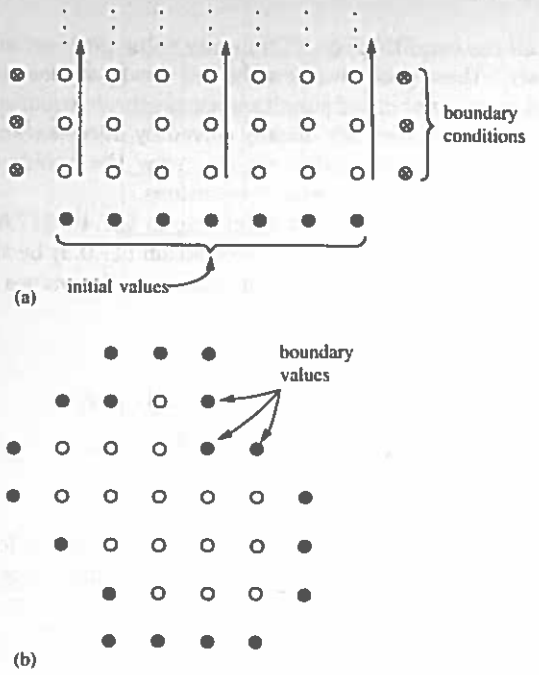

Figure 17.0.1. Initial value problem (a) and boundary value problem (b) are contrasted. In (a) initial values are given on one "time slice", and it is desired to advance the solution in time, computing successive rows of open dots in the direction shown by the arrows. Boundary conditions at the left and right edges of each row (⊗) must also be supplied, but only one row at a time. Only one, or a few, previous rows need be maintained in memory. In (b), boundary values are specified around the edge of a grid, and an iterative process is employed to find the values of all the internal points (open circles). All gridpoints must be maintained in memory.

to convey a certain amount of generalizable information through a few carefully chosen model examples. These examples will illustrate an important point: one's principal *computational* concern must be the *stability* of the algorithm. Many reasonable-looking algorithms for initial value problems just don't work — they are numerically unstable.

Boundary Value Problems

The questions that define a boundary value problem are

- What are the variables?
- What equations are satisfied in the interior of the region of interest?
- What equations are satisfied by points on the boundary of the region of interest. (Here Dirichlet and Neumann conditions are possible choices for elliptic second-order equations, but more complicated boundary conditions can also be encountered.)

In contrast to initial value problems, stability is relatively easy to achieve for boundary value problems. Thus, the *efficiency* of the algorithms, both in computational load and storage requirements, becomes the principal concern.

Because all the conditions on a boundary value problem must be satisfied "simultaneously," these problems usually boil down, at least conceptually, to the solution of large numbers of simultaneous algebraic equations. When such equations are nonlinear, they are usually solved by linearization and iteration; so without much loss of generality we can view the problem as being the solution of special, large linear sets of equations.

As an example, one which we will refer to in §17.4 – §17.6 as our "model problem," let us consider the solution of equation (17.0.3) by the *finite difference method*. We represent the function $u(x, y)$ by its values at the discrete set of points

$$\begin{aligned} x_j &= x_0 + j\Delta, \qquad j = 0, 1, \ldots, J, \\ y_l &= y_0 + l\Delta, \qquad l = 0, 1, \ldots, L \end{aligned} \tag{17.0.4}$$

where Δ is the *grid spacing*. From now on, we will write $u_{j,l}$ for $u(x_j, y_l)$, and $\rho_{j,l}$ for $\rho(x_j, y_l)$. For (17.0.3) we substitute a finite-difference representation (see Figure 17.0.2),

$$\frac{u_{j+1,l} - 2u_{j,l} + u_{j-1,l}}{\Delta^2} + \frac{u_{j,l+1} - 2u_{j,l} + u_{j,l-1}}{\Delta^2} = \rho_{j,l} \tag{17.0.5}$$

or equivalently

$$u_{j+1,l} + u_{j-1,l} + u_{j,l+1} + u_{j,l-1} - 4u_{j,l} = \Delta^2 \rho_{j,l} \tag{17.0.6}$$

To write this system of linear equations in matrix form we need to make a vector out of u. Let us number the two dimensions of grid points in a single one-dimensional sequence by defining

$$i \equiv j(L+1) + l \qquad \text{for} \qquad j = 0, 1, \ldots, J, \qquad l = 0, 1, \ldots, L \tag{17.0.7}$$

In other words, i increases most rapidly along the columns representing y values. Equation (17.0.6) now becomes

$$u_{i+L+1} + u_{i-(L+1)} + u_{i+1} + u_{i-1} - 4u_i = \Delta^2 \rho_i \tag{17.0.8}$$

This equation only holds at the interior points $j = 1, 2, \ldots, J-1; l = 1, 2, \ldots, L-1$.

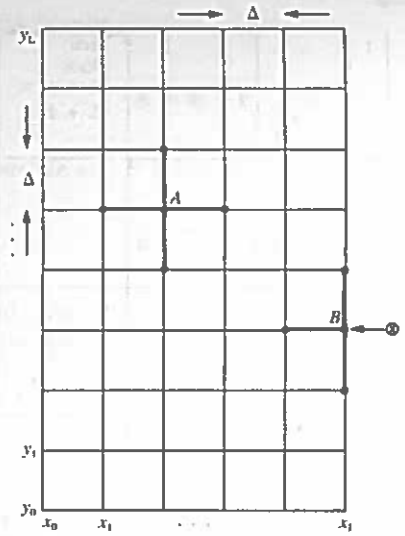

Figure 17.0.2. Finite difference representation of a second-order elliptic equation on a two-dimensional grid. The second derivatives at the point A are evaluated using the points to which A is shown connected. The second derivatives at point B are evaluated using the connected points and also using "right-hand side" boundary information, shown schematically as ⊗.

The points where

$$\begin{aligned} j &= 0 \qquad \text{[i.e., } i = 0, \ldots, L\text{]} \\ j &= J \qquad \text{[i.e., } i = J(L+1), \ldots, J(L+1) + L\text{]} \\ l &= 0 \qquad \text{[i.e., } i = 0, L+1, \ldots, J(L+1)\text{]} \\ l &= L \qquad \text{[i.e., } i = L, L+1+L, \ldots, J(L+1) + L\text{]} \end{aligned} \tag{17.0.9}$$

are boundary points where either u or its derivative has been specified. If we pull all this "known" information over to the right-hand side of equation (17.0.8), then the equation takes the form

$$\mathbf{A} \cdot \mathbf{u} = \mathbf{b} \tag{17.0.10}$$

where $\mathbf{A}$ has the form shown in Figure 17.0.3. The matrix $\mathbf{A}$ is called "tridi-

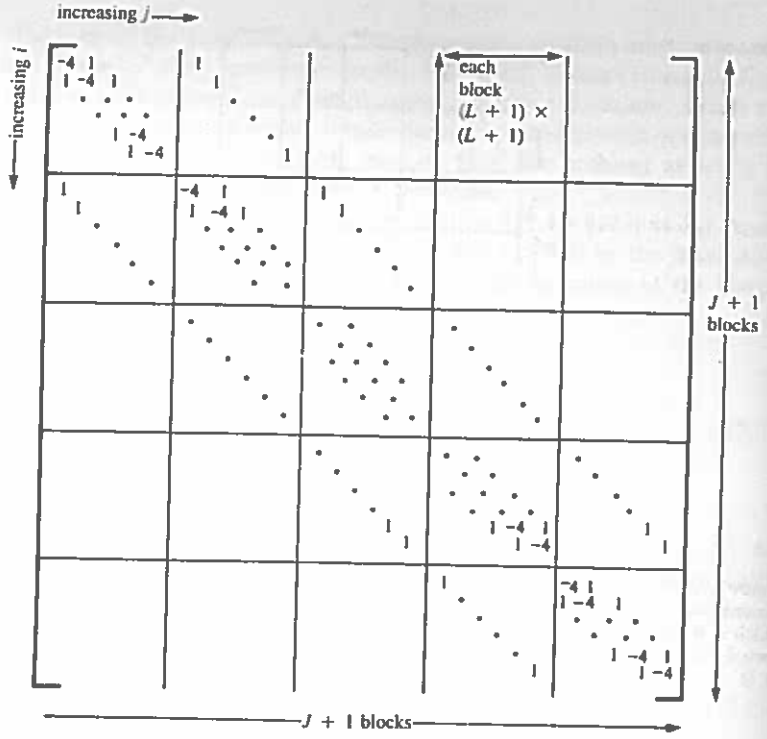

Figure 17.0.3. Matrix structure derived from a second-order elliptic equation (here equation 17.0.6). All elements not shown are zero. The matrix has diagonal blocks that are themselves tridiagonal, and sub- and super-diagonal blocks that are diagonal. This form is called "tridiagonal with fringes." A matrix this sparse would never be stored in its full form as shown here.

agonal with fringes." A general linear second-order elliptic equation

$$a(x,y)\frac{\partial^2 u}{\partial x^2} + b(x,y)\frac{\partial u}{\partial x} + c(x,y)\frac{\partial^2 u}{\partial y^2} + d(x,y)\frac{\partial u}{\partial y} + e(x,y)\frac{\partial^2 u}{\partial x \partial y} + f(x,y)u = g(x,y) \tag{17.0.11}$$

will lead to a matrix of similar structure except that the nonzero entries will not be constants.

As a rough classification, there are three different approaches to the solution of equation (17.0.10), not all applicable in all cases: relaxation methods, "rapid" methods (e.g. Fourier methods), and direct matrix methods.

Relaxation methods make immediate use of the structure of the sparse matrix A. The matrix is split into two parts

$$\mathbf{A} = \mathbf{E} - \mathbf{F} \tag{17.0.12}$$

where E is easily invertible and F is the remainder. Then (17.0.10) becomes

$$\mathbf{E} \cdot \mathbf{u} = \mathbf{F} \cdot \mathbf{u} + \mathbf{b} \tag{17.0.13}$$

The relaxation method involves choosing an initial guess $\mathbf{u}^{(0)}$ and then solving successively for iterates $\mathbf{u}^{(r)}$ from

$$\mathbf{E} \cdot \mathbf{u}^{(r)} = \mathbf{F} \cdot \mathbf{u}^{(r-1)} + \mathbf{b} \tag{17.0.14}$$

Since E is chosen to be easily invertible, each iteration is fast. We will discuss relaxation methods in some detail in §17.5, and from a slightly different point of view ("operator splitting") in §17.6.

Rapid methods apply for only a rather special class of equations: those with constant coefficients, or, more generally, those that are separable in the chosen coordinates. In addition, the boundaries must coincide with coordinate lines. This special class of equations is met quite often in practice. We defer detailed discussion to §17.4.

Matrix methods attempt to solve the equation

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{17.0.15}$$

directly. The degree to which this is practical depends very strongly on the exact structure of the matrix A for the problem at hand, so our discussion can go no farther than a few remarks and references at this point.

Sparseness of the matrix *must* be the guiding force. Otherwise the matrix problem is prohibitively large. For example, the simplest problem on a 100×100 spatial grid would involve 10000 unknown $u_{j,l}$'s, implying a 10000×10000 matrix A, containing 10^8 elements!

As we discussed at the end of §10.6, if A is symmetric and positive definite (as it usually is in elliptic problems), the conjugate-gradient algorithm can be used. In practice, rounding error often spoils the effectiveness of the conjugate gradient algorithm for solving finite-difference equations. However, it is useful when incorporated in methods that first rewrite the equations so that A is transformed to a matrix A′ that is close to the identity matrix. The quadratic surface defined by the equations then has almost spherical contours, and the conjugate gradient algorithm works very well. An example of such an algorithm is the *incomplete Choleski conjugate gradient method (ICCG)* (see Meijerink and van der Vorst; van der Vorst; Kershaw).

Another method that relies on a transformation approach is the *strongly implicit procedure* of Stone. A program called SIPSOL that implements this routine has been published by Jesshope.

A third class of matrix methods is the Analyze-Factorize-Operate approach as described in §2.10.

Generally speaking, when you have the storage available to implement these methods — not nearly as much as the 10^8 above, but usually much more than is required by relaxation methods — then you should consider doing so. Their execution times can be superior to the best relaxation methods by factors of, e.g., five. For grids larger than, say, 100×100, however, it is generally found that only relaxation methods, or "rapid" methods when they are applicable, are possible.

There is More to Life than Finite Differencing

Besides finite differencing, there are other methods for solving PDEs. Most important are finite element, Monte Carlo, spectral, and variational methods. Unfortunately, we shall barely be able to do justice to finite differencing in this chapter, and so shall not be able to discuss these other methods in this book. You can consult the references below for more details of the other techniques. Finite element methods are often preferred by practitioners in solid mechanics and structural engineering; these methods allow considerable freedom in putting computational elements where you want them, important when dealing with highly irregular geometries. Spectral methods are preferred for very regular geometries and smooth functions; they converge more rapidly than finite-difference methods (cf. §17.4), but they do not work well for problems with discontinuities.

REFERENCES AND FURTHER READING:

Ames, W.F. 1977, *Numerical Methods for Partial Differential Equations*, 2nd ed. (New York: Academic Press).

Richtmyer, R.D., and Morton, K.W. 1967, *Difference Methods for Initial Value Problems*, 2nd ed. (New York: Wiley-Interscience).

Roache, P.J. 1976, *Computational Fluid Dynamics* (Albuquerque: Hermosa).

Strang, G., and Fix, G. 1973, *An Analysis of the Finite Element Method* (Englewood Cliffs, N.J.: Prentice-Hall).

Mitchell, A.R., and Griffiths, D.F. 1980, *The Finite Difference Method in Partial Differential Equations* (New York: Wiley) [includes discussion of finite element methods].

Dorr, F.W. 1970, *S.I.A.M. Review*, vol. 12, pp. 248–263 [review of rapid Poisson equation methods].

Jesshope, C.R. 1979, *Computer Physics Communications*, vol. 17, pp. 383–391.

Kershaw, D.S. 1970, *Journal of Computational Physics*, vol. 26, pp. 43–65.

Meijerink, J.A., and van der Vorst, H.A. 1977, *Mathematics of Computation*, vol. 31, pp. 148–162.

Stone, H.J. 1968, *S.I.A.M. Journal of Numerical Analysis*, vol. 5, pp. 530–558.

van der Vorst, H.A. 1981, *Journal of Computational Physics*, vol. 44, pp. 1–19 [review of sparse iterative methods].

17.1 Flux-Conservative Initial Value Problems

A large class of initial value (time-evolution) PDEs in one space dimension can be cast into the form of a *flux-conservative equation*,

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\partial \mathbf{F}(\mathbf{u})}{\partial x} \tag{17.1.1}$$

where $\mathbf{u}$ and $\mathbf{F}$ are vectors, and where (in some cases) $\mathbf{F}$ may depend not only on $\mathbf{u}$ but also on spatial derivatives of $\mathbf{u}$. The vector $\mathbf{F}$ is called the *conserved flux*.

For example, the prototypical hyperbolic equation, the one-dimensional wave equation with constant velocity of propagation v

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} \tag{17.1.2}$$

can be rewritten as a set of two first-order equations

$$\begin{aligned} \frac{\partial r}{\partial t} &= v \frac{\partial s}{\partial x} \\ \frac{\partial s}{\partial t} &= v \frac{\partial r}{\partial x} \end{aligned} \tag{17.1.3}$$

where

$$\begin{aligned} r &\equiv v \frac{\partial u}{\partial x} \\ s &\equiv \frac{\partial u}{\partial t} \end{aligned} \tag{17.1.4}$$

In this case r and s become the two components of $\mathbf{u}$, and the flux is given by the linear matrix relation

$$\mathbf{F}(\mathbf{u}) = \begin{pmatrix} 0 & -v \\ -v & 0 \end{pmatrix} \cdot \mathbf{u} \tag{17.1.5}$$

[The physicist-reader may recognize equations (17.1.3) as analogous to Maxwell's equations for one-dimensional propagation of electromagnetic waves.]

We will consider, in this section, a prototypical example of the general flux-conservative equation (17.1.1), namely the equation for a scalar u,

$$\frac{\partial u}{\partial t} = -v \frac{\partial u}{\partial x} \tag{17.1.6}$$

with v a constant. As it happens, we already know analytically that the general solution of this equation is a wave propagating in the positive x-direction,

$$u = f(x - vt) \tag{17.1.7}$$

where f is an arbitrary function. However, the numerical strategies that we develop will be equally applicable to the more general equations represented by (17.1.1). In some contexts, equation (17.1.6) is called an *advective* equation, because the quantity u is transported by a "fluid flow" with a velocity v.

How do we go about finite differencing equation (17.1.6) (or, analogously, 17.1.1)? The straightforward approach is to choose equally spaced points along both the t- and x-axes. Thus denote

$$\begin{aligned} x_j &= x_0 + j\Delta x, \qquad j = 0, 1, \ldots, J, \\ t_n &= t_0 + n\Delta t, \qquad n = 0, 1, \ldots, N \end{aligned} \tag{17.1.8}$$

Let u_j^n denote $u(t_n, x_j)$. We have several choices for representing the time derivative term. The obvious way is to set

$$\left.\frac{\partial u}{\partial t}\right|_{j,n} = \frac{u_j^{n+1} - u_j^n}{\Delta t} + O(\Delta t) \tag{17.1.9}$$

This is called *forward Euler* differencing (cf. equation 15.1.1). While forward Euler is only first-order accurate in Δt, it has the advantage that one is able to calculate quantities at timestep $n+1$ in terms of only quantities known at timestep n. For the space derivative, we can use a second-order representation still using only quantities known at timestep n:

$$\left.\frac{\partial u}{\partial x}\right|_{j,n} = \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} + O(\Delta x^2) \tag{17.1.10}$$

The resulting finite-difference approximation to equation (17.1.6) is called the FTCS representation (Forward Time Centered Space),

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v\left(\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\right) \tag{17.1.11}$$

which can easily be rearranged to be a formula for u_j^{n+1} in terms of the other quantities. The FTCS scheme is illustrated in Figure 17.1.1. It's a

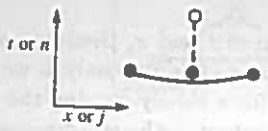

Figure 17.1.1. Representation of the Forward Time Centered Space (FTCS) differencing scheme. In this and subsequent figures, the open circle is the new point at which the solution is desired; filled circles are known points whose function values are used in calculating the new point; the solid lines connect points that are used to calculate spatial derivatives; the dashed lines connect points that are used to calculate time derivatives. The FTCS scheme is generally unstable for hyperbolic problems and cannot usually be used.

fine example of an algorithm that is easy to derive, takes little storage, and executes quickly. Too bad it doesn't work! (See below.)

The FTCS representation is an *explicit* scheme. This means that u_j^{n+1} for each j can be calculated explicitly from the quantities that are already known. Later we shall meet *implicit* schemes, which require us to solve implicit equations coupling the u_j^{n+1} for various j. (Explicit and implicit methods for ordinary differential equations were discussed in §15.6.) The FTCS algorithm is also an example of a *single-level* scheme, since only values at time level n have to be stored to find values at time level $n+1$.

von Neumann Stability Analysis

Unfortunately, equation (17.1.11) is of very limited usefulness. It is an *unstable* method, which can only be used (if at all) to study waves for a short fraction of one oscillation period. To find alternative methods with more general applicability, we must introduce the *von Neumann stability analysis*.

The von Neumann analysis is local: we imagine that the coefficients of the difference equations are so slowly varying as to be considered constant in space and time. In that case, the independent solutions, or *eigenmodes*, of the difference equations are all of the form

$$u_j^n = \xi^n e^{ikj\Delta x} \tag{17.1.12}$$

where k is a real spatial wave number (which can have any value) and $\xi = \xi(k)$ is a complex number that depends on k. The key fact is that the time dependence of a single eigenmode is nothing more than successive integer powers of the complex number ξ. Therefore, the difference equations are unstable (have exponentially growing modes) if $|\xi(k)| > 1$ for *some* k. The number ξ is called the *amplification factor* at a given wave number k.

To find $\xi(k)$, we simply substitute (17.1.12) back into (17.1.11). Dividing by ξ^n, we get

$$\xi(k) = 1 - i\frac{v\Delta t}{\Delta x}\sin k\Delta x \tag{17.1.13}$$

whose modulus is > 1 for *all* k; so the FTCS scheme is unconditionally unstable.

If the velocity v were a function of t and x, then we would write v_j^n in equation (17.1.11). In the von Neumann stability analysis we would still treat v as a constant, the idea being that for v slowly varying the analysis is local. In fact, even in the case of strictly constant v, the von Neumann analysis does not rigorously treat the end effects at $j = 0$ and $j = N$.

More generally, if the equation's right-hand side were nonlinear in u, then a von Neumann analysis would linearize by writing $u = u_0 + \delta u$, expanding to linear order in δu. Assuming that the u_0 quantities already satisfy the difference equation exactly, the analysis would look for an unstable eigenmode of δu.

Despite its lack of rigor, the von Neumann method generally gives valid answers and is much easier to apply than more careful methods. We accordingly adopt it exclusively. (See, for example, Richtmyer and Morton for a discussion of other methods of stability analysis.)

Lax Method

The instability in the FTCS method can be cured by a simple change due to Lax. One replaces the term u_j^n in the time derivative term by its average (Figure 17.1.2):

$$u_j^n \to \frac{1}{2}\left(u_{j+1}^n + u_{j-1}^n\right) \tag{17.1.14}$$

This turns (17.1.11) into

$$u_j^{n+1} = \frac{1}{2}\left(u_{j+1}^n + u_{j-1}^n\right) - \frac{v\Delta t}{2\Delta x}\left(u_{j+1}^n - u_{j-1}^n\right) \tag{17.1.15}$$

Substituting equation (17.1.12), we find for the amplification factor

$$\xi = \cos k\Delta x - i\frac{v\Delta t}{\Delta x}\sin k\Delta x \tag{17.1.16}$$

The stability condition $|\xi|^2 \le 1$ leads to the requirement

$$\frac{|v|\Delta t}{\Delta x} \le 1 \tag{17.1.17}$$

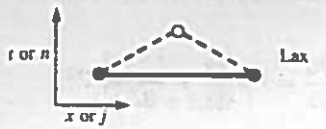

Figure 17.1.2. Representation of the Lax differencing scheme, as in the previous figure. The stability criterion for this scheme is the Courant condition.

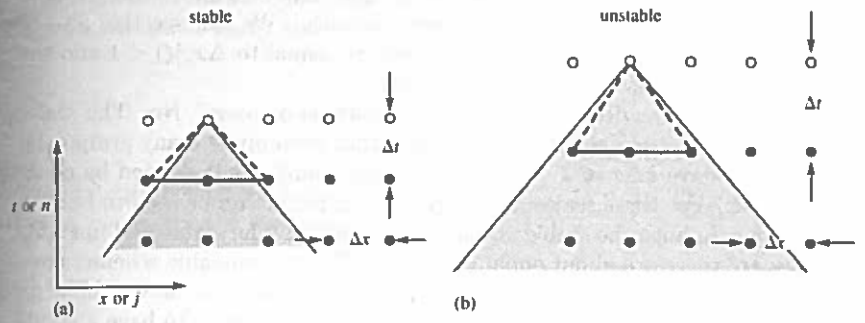

Figure 17.1.3. Courant condition for stability of a differencing scheme. The PDEs of an initial value problem imply that the value at a point depends on information within some domain of dependency to the past, shown here shaded. A differencing scheme has its own domain of dependency determined by the choice of points on one time slice (shown as connected solid dots) whose values are used in determining a new point (shown connected by dashed lines). A differencing scheme is Courant stable if the differencing domain of dependency is larger than that of the PDE's, as in (a), and unstable if the relationship is the reverse, as in (b).

This is the famous Courant-Friedrichs-Lewy stability criterion, often called simply the *Courant condition*. Intuitively, the stability condition can be understood as follows (Figure 17.1.3): The quantity u_j^{n+1} in equation (17.1.15) is computed from information at points $j-1$ and $j+1$ at time n. In other words, x_{j-1} and x_{j+1} are the boundaries of the spatial region that is allowed to communicate information to u_j^{n+1}. Now recall that in the continuum wave equation, information actually propagates with a maximum velocity v. If the point u_j^{n+1} is outside of the shaded region in Figure 17.1.3, then it requires information from points more distant than the differencing scheme allows. Lack of that information gives rise to an instability. Therefore, Δt cannot be made too large.

The surprising result, that the simple replacement (17.1.14) stabilizes the FTCS scheme, is our first encounter with the fact that differencing PDEs is an art as much as a science. To see if we can demystify the art somewhat, let us compare the FTCS and Lax schemes by rewriting equation (17.1.15) so that it is in the form of equation (17.1.11) with a remainder term:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v\left(\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\right) + \frac{1}{2}\left(\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta t}\right) \tag{17.1.18}$$

But this is exactly the FTCS representation of the equation

$$\frac{\partial u}{\partial t} = -v\frac{\partial u}{\partial x} + \frac{(\Delta x)^2}{2\Delta t}\nabla^2 u \tag{17.1.19}$$

where $\nabla^2 = \partial^2/\partial x^2$ in one dimension. We have, in effect, added a diffusion term to the equation, or, if you recall the form of the Navier-Stokes equation for viscous fluid flow, a dissipative term. The Lax scheme is thus said to have *numerical dissipation*, or *numerical viscosity*. We can see this also in the amplification factor. Unless $|v|\Delta t$ is exactly equal to Δx, $|\xi| < 1$ and the amplitude of the wave decreases spuriously.

Isn't a spurious decrease as bad as a spurious increase? No. The scales that we hope to study accurately are those that encompass many gridpoints, so that they have $k\Delta x \ll 1$. (The spatial wave number k is defined by equation 17.1.12.) For these scales, the amplification factor can be seen to be very close to one, in both the stable and unstable scheme. The stable and unstable schemes are therefore about equally accurate. For the unstable scheme, however, short scales with $k\Delta x \sim 1$, *which we are not interested in*, will blow up and swamp the interesting part of the solution. Much better to have a stable scheme in which these short wavelengths die away innocuously. Both the stable and the unstable schemes are *inaccurate* for these short wavelengths, but the inaccuracy is of a tolerable character when the scheme is stable.

When the independent variable u is a vector, then the von Neumann analysis is slightly more complicated. For example, we can consider equation (17.1.3), rewritten as

$$\frac{\partial}{\partial t}\begin{bmatrix} r \\ s \end{bmatrix} = -\frac{\partial}{\partial x}\begin{bmatrix} vs \\ vr \end{bmatrix} \tag{17.1.20}$$

The Lax method for this equation is

$$\begin{aligned} r_j^{n+1} &= \frac{1}{2}(r_{j+1}^n + r_{j-1}^n) + \frac{v\Delta t}{2\Delta x}(s_{j+1}^n - s_{j-1}^n) \\ s_j^{n+1} &= \frac{1}{2}(s_{j+1}^n + s_{j-1}^n) + \frac{v\Delta t}{2\Delta x}(r_{j+1}^n - r_{j-1}^n) \end{aligned} \tag{17.1.21}$$

The von Neumann stability analysis now proceeds by assuming that the eigenmode is of the following (vector) form,

$$\begin{bmatrix} r_j^n \\ s_j^n \end{bmatrix} = \xi^n e^{ikj\Delta x}\begin{bmatrix} r^0 \\ s^0 \end{bmatrix} \tag{17.1.22}$$

Here the vector on the right-hand side is a constant (both in space and in time) eigenvector, and ξ is a complex number, as before. Substituting (17.1.22) into (17.1.21), and dividing by the power ξ^n, gives the homogeneous vector equation

$$\begin{bmatrix} (\cos k\Delta x) - \xi & i\dfrac{v\Delta t}{\Delta x}\sin k\Delta x \\ i\dfrac{v\Delta t}{\Delta x}\sin k\Delta x & (\cos k\Delta x) - \xi \end{bmatrix} \cdot \begin{bmatrix} r^0 \\ s^0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{17.1.23}$$

This admits a solution only if the determinant of the matrix on the left vanishes, a condition easily shown to yield the two roots ξ

$$\xi = \cos k\Delta x \pm i\frac{v\Delta t}{\Delta x}\sin k\Delta x \tag{17.1.24}$$

The stability condition is that both roots satisfy $|\xi| \le 1$. This again turns out to be simply the Courant condition (17.1.17).

Other Varieties of Error

Thus far we have been concerned with *amplitude error*, because of its intimate connection with the stability or instability of a differencing scheme. Other varieties of error are relevant when we shift our concern to accuracy, rather than stability.

Finite-difference schemes for hyperbolic equations can exhibit dispersion, or *phase errors*. For example, even if we set $v\Delta t/\Delta x = 1$ in equation (17.1.16), we find

$$\xi = e^{-ik\Delta x} \tag{17.1.25}$$

An arbitrary initial wave packet is a superposition of modes with different k's. At each timestep the modes get multiplied by different phase factors (17.1.25), depending on their value of k. After a while, the phase relations of the modes can become hopelessly garbled and the wave packet disperses. Note from (17.1.25) that the dispersion becomes large as soon as the wavelength becomes comparable to the grid spacing Δx.

A third type of error is one associated with nonlinear hyperbolic equations and is therefore sometimes called *nonlinear instability*. For example, a piece of the Euler or Navier-Stokes equations for fluid flow looks like

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} + \dots \tag{17.1.26}$$

The nonlinear term in v can cause a transfer of energy in Fourier space from long wavelengths to short wavelengths. This results in a wave profile steepening until a vertical profile or "shock" develops. Since the von Neumann analysis suggests that the stability can depend on $k\Delta x$, a scheme that was

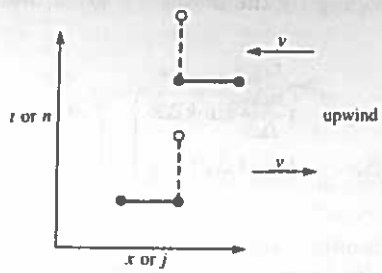

Figure 17.1.4. Representation of upwind differencing schemes. The upper scheme is stable when the advection constant v is negative, as shown; the lower scheme is stable when the advection constant v is positive, also as shown. The Courant condition must, of course, also be satisfied.

stable for shallow profiles can become unstable for steep profiles. This kind of difficulty arises in a differencing scheme where the cascade in Fourier space is halted at the shortest wavelength representable on the grid, that is, at $k \sim 1/\Delta x$. If energy simply accumulates in these modes, it eventually swamps the energy in the long wavelength modes of interest.

Nonlinear instability and shock formation is thus somewhat controlled by numerical viscosity such as that discussed in connection with equation (17.1.18) above. In some fluid problems, however, shock formation is not merely an annoyance, but an actual physical behavior of the fluid whose detailed study is a goal. Then, numerical viscosity alone may not be adequate or sufficiently controllable. While there are some alternative methods, the standard way of handling physical shocks is by adding *artificial viscosity* to the difference equations. For a discussion of this specialized technique, see, for example, Richtmyer and Morton or Roache.

For wave equations, propagation errors (amplitude or phase) are usually most worrisome. For advective equations, on the other hand, *transport errors* are usually of greater concern. In the Lax scheme, equation (17.1.15), a disturbance in the advected quantity u at mesh point j propagates to mesh points $j+1$ and $j-1$ at the next time step. In reality, however, if the velocity v is positive then only mesh point $j+1$ should be affected.

The simplest way to model the transport properties "better" is to use *upwind differencing* (see Figure 17.1.4):

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} = -v_j^n \begin{cases} \dfrac{u_j^n - u_{j-1}^n}{\Delta x}, & v_j^n > 0 \\ \dfrac{u_{j+1}^n - u_j^n}{\Delta x}, & v_j^n < 0 \end{cases} \tag{17.1.27}$$

Note that this scheme is only first-order, not second-order, accurate in the calculation of the spatial derivatives. How can it be "better"? The answer is one that annoys the mathematicians: The goal of numerical simulations is not always "accuracy" in a strictly mathematical sense, but sometimes "fidelity" to the underlying physics in a sense that is looser and more pragmatic. In such contexts, some kinds of error are much more tolerable than others. Upwind differencing generally adds fidelity to problems where the advected variables are liable to undergo sudden changes of state, e.g., as they pass through shocks or other discontinuities. You will have to be guided by the specific nature of your own problem.

For the differencing scheme (17.1.27), the amplification factor (for constant v) is

$$\xi = 1 - \left|\frac{v\Delta t}{\Delta x}\right|(1-\cos k\Delta x) - i\frac{v\Delta t}{\Delta x}\sin k\Delta x \tag{17.1.28}$$

$$|\xi|^2 = 1 - 2\left|\frac{v\Delta t}{\Delta x}\right|\left(1-\left|\frac{v\Delta t}{\Delta x}\right|\right)(1-\cos k\Delta x) \tag{17.1.29}$$

So the stability criterion $|\xi|^2 \le 1$ is (again) simply the Courant condition (17.1.17).

There are various ways of improving the accuracy of first-order upwind differencing. In the continuum equation, material originally a distance $v\Delta t$ away arrives at a given point after a time interval Δt. In the first-order method, the material always arrives from Δx away. If $v\Delta t \ll \Delta x$ (to insure accuracy), this can cause a large error. One way of reducing this error is to interpolate u between $j-1$ and j before transporting it. This gives effectively a second-order method. Various schemes for second-order upwind differencing are discussed and compared by Centrella and Wilson and by Hawley et al.

Second-Order Accuracy in Time

When using a method that is first-order accurate in time but second-order accurate in space, one generally has to take $v\Delta t$ significantly smaller than Δx to achieve desired accuracy, say, by at least a factor of 5. Thus the Courant condition is not actually the limiting factor with such schemes in practice. However, there are schemes that are second-order accurate in both space and time, and these can often be pushed right to their stability limit, with correspondingly smaller computation times.

For example, the *staggered leapfrog* method for the conservation equation (17.1.1) is defined as follows (Figure 17.1.5): Using the values of u^n at time t^n, compute the fluxes F_j^n. Then compute new values u^{n+1} using the time-centered values of the fluxes:

$$u_j^{n+1} - u_j^{n-1} = -\frac{\Delta t}{\Delta x}(F_{j+1}^n - F_{j-1}^n) \tag{17.1.30}$$

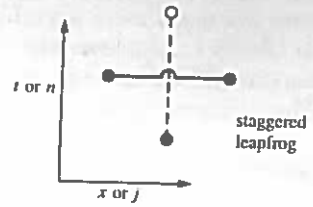

Figure 17.1.5. Representation of the staggered leapfrog differencing scheme. Note that information from two previous time slices is used in obtaining the desired point. This scheme is second-order accurate in both space and time.

The name comes from the fact that the time levels in the time derivative term "leapfrog" over the time levels in the space derivative term. The method requires that u^{n-1} and u^n be stored to compute u^{n+1}.

For our simple model equation (17.1.6), staggered leapfrog takes the form

$$u_j^{n+1} - u_j^{n-1} = -\frac{v\Delta t}{\Delta x}(u_{j+1}^n - u_{j-1}^n) \tag{17.1.31}$$

The von Neumann stability analysis now gives a quadratic equation for ξ, rather than a linear one, because of the occurrence of three consecutive powers of ξ when the form (17.1.12) for an eigenmode is substituted into equation (17.1.31),

$$\xi^2 - 1 = -2i\xi\frac{v\Delta t}{\Delta x}\sin k\Delta x \tag{17.1.32}$$

whose solution is

$$\xi = -i\frac{v\Delta t}{\Delta x}\sin k\Delta x \pm \sqrt{1 - \left(\frac{v\Delta t}{\Delta x}\sin k\Delta x\right)^2} \tag{17.1.33}$$

Thus the Courant condition is again required for stability.

[If u were a vector $\mathbf{u}$, then an eigenmode like (17.1.22) would, in the von Neumann analysis, lead to a determinant whose components were themselves polynomials in ξ. Setting this determinant to zero would give a higher order polynomial equation for the amplification factor ξ.]

In fact, in equation (17.1.33), $|\xi|^2 = 1$ for any $v\Delta t \leq \Delta x$. This is the great advantage of the staggered leapfrog method: There is no amplitude dissipation. For equations more complicated than our simple model equation, especially nonlinear equations, the method usually becomes unstable when the gradients get large. The instability is related to the fact that odd and even mesh points are completely decoupled, like the black and white squares of a chessboard, as shown in Figure 17.1.6. This mesh drifting instability is cured

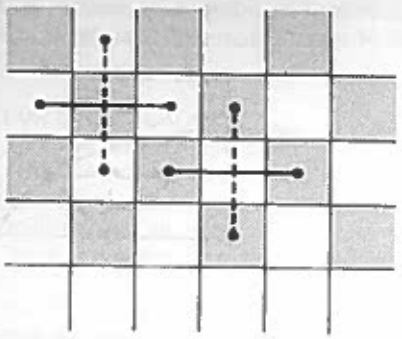

Figure 17.1.6. Origin of mesh-drift instabilities in a staggered leapfrog scheme. If the mesh points are imagined to lie in the squares of a chess board, then white squares couple to themselves, black to themselves, but there is no coupling between white and black. The fix is to introduce a small diffusive mesh-coupling piece.

by coupling the two meshes through a numerical viscosity term, e.g. adding to the right side of (17.1.31) a small coefficient ($\ll 1$) times $u_{j+1}^n - 2u_j^n + u_{j-1}^n$.

The *Two-Step Lax-Wendroff* scheme is a second-order in time method that avoids large numerical dissipation and mesh drifting. One defines intermediate values $u_{j+1/2}$ at the half timesteps $t_{n+1/2}$ and the half mesh points $x_{j+1/2}$. These are calculated by the Lax scheme:

$$u_{j+1/2}^{n+1/2} = \frac{1}{2}(u_{j+1}^n + u_j^n) - \frac{\Delta t}{2\Delta x}(F_{j+1}^n - F_j^n) \tag{17.1.34}$$

Using these variables, one calculates the fluxes $F_{j+1/2}^{n+1/2}$. Then the updated values u_j^{n+1} are calculated by the properly-centered expression

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}(F_{j+1/2}^{n+1/2} - F_{j-1/2}^{n+1/2}) \tag{17.1.35}$$

The provisional values $u_{j+1/2}^{n+1/2}$ are now discarded. (See Figure 17.1.7.)

Let us investigate the stability of this method for our model advective equation, where $F = vu$. Substitute (17.1.34) in (17.1.35) to get

$$\begin{aligned} u_j^{n+1} = u_j^n - \alpha\Big[&\frac{1}{2}(u_{j+1}^n + u_j^n) - \frac{1}{2}\alpha(u_{j+1}^n - u_j^n) \\ &-\frac{1}{2}(u_j^n + u_{j-1}^n) + \frac{1}{2}\alpha(u_j^n - u_{j-1}^n)\Big] \end{aligned} \tag{17.1.36}$$

where

$$\alpha \equiv \frac{v\Delta t}{\Delta x} \tag{17.1.37}$$

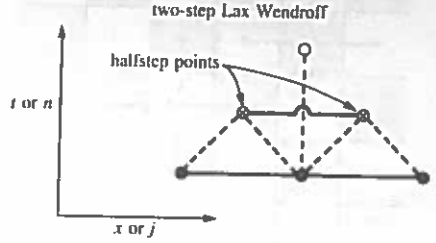

Figure 17.1.7. Representation of the two-step Lax-Wendroff differencing scheme. Two halfstep points (⊗) are calculated by the Lax method. These, plus one of the original points, produce the new point via staggered leapfrog. Halfstep points are used only temporarily and do not require storage allocation on the grid. This scheme is second-order accurate in both space and time.

Then

$$\xi = 1 - i\alpha \sin k\Delta x - \alpha^2(1 - \cos k\Delta x) \tag{17.1.38}$$

so

$$|\xi|^2 = 1 - \alpha^2(1 - \alpha^2)(1 - \cos k\Delta x)^2 \tag{17.1.39}$$

The stability criterion $|\xi|^2 \leq 1$ is therefore $\alpha^2 \leq 1$, or $v\Delta t \leq \Delta x$ as usual. Incidentally, you should not think that the Courant condition is the only stability requirement that ever turns up in PDEs. It keeps doing so in our model examples just because those examples are so simple in form. The method of analysis is, however, general.

Except when $\alpha = 1$, $|\xi|^2 < 1$ in (17.1.39), so some amplitude damping does occur. The effect is relatively small, however, for wavelengths large compared with the mesh size Δx. If we expand (17.1.39) for small $k\Delta x$, we find

$$|\xi|^2 = 1 - \alpha^2(1 - \alpha^2)\frac{(k\Delta x)^4}{4} + \ldots \tag{17.1.40}$$

The departure from unity occurs only at fourth order in k. This should be contrasted with equation (17.1.16) for the Lax method, which shows that

$$|\xi|^2 = 1 - (1 - \alpha^2)(k\Delta x)^2 + \ldots \tag{17.1.41}$$

for small $k\Delta x$.

In summary, our recommendation for initial value problems that can be cast in flux-conservative form is to use either the staggered leapfrog or the Two-Step Lax-Wendroff method. For problems sensitive to transport errors, upwind differencing or one of its refinements should be considered.

REFERENCES AND FURTHER READING:

Ames, W.F. 1977, *Numerical Methods for Partial Differential Equations*, 2nd ed. (New York: Academic Press), Chapter 4.

Richtmyer, R.D., and Morton, K.W. 1967, *Difference Methods for Initial Value Problems*, 2nd ed. (New York: Wiley-Interscience).

Roache, P.J. 1976, *Computational Fluid Dynamics* (Albuquerque: Hermosa).

Centrella, J., and Wilson, J. R. 1984, *Astrophysical Journal Supplement*, vol. 54, pp. 229–249, Appendix B.

Hawley, J. F., Smarr. L. L., and Wilson, J. R. 1984, *Astrophysical Journal Supplement*, vol. 55, pp. 211–246, §2c.

17.2 Diffusive Initial Value Problems

Recall the model parabolic equation, the diffusion equation in one space dimension,

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial u}{\partial x}\right) \tag{17.2.1}$$

where D is the diffusion coefficient. Actually, this equation is a flux-conservative equation of the form considered in the previous section, with

$$F = -D\frac{\partial u}{\partial x} \tag{17.2.2}$$

the flux in the x-direction. We will assume $D \geq 0$, otherwise equation (17.2.1) has physically unstable solutions: A small disturbance evolves to become more and more concentrated instead of dispersing. (Don't make the mistake of trying to find a stable differencing scheme for a problem whose underlying PDEs are themselves unstable!)

Even though (17.2.1) is of the form already considered, it is useful to consider it as a model in its own right. The particular form of flux (17.2.2), and its direct generalizations, occur quite frequently in practice. Moreover, we have already seen that numerical viscosity and artificial viscosity can introduce diffusive pieces like the right-hand side of (17.2.1) in many other situations.

Consider first the case when D is a constant. Then the equation

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \tag{17.2.3}$$

can be differenced in the obvious way:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\left[\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}\right] \tag{17.2.4}$$

This is the FTCS scheme again, except that it is a second derivative that has been differenced on the right-hand side. But this makes a world of difference! The FTCS scheme was unstable for the hyperbolic equation; however, a quick calculation shows that the amplification factor for equation (17.2.4) is

$$\xi = 1 - \frac{4D\Delta t}{(\Delta x)^2}\sin^2\left(\frac{k\Delta x}{2}\right) \tag{17.2.5}$$

The requirement $|\xi| \le 1$ leads to the stability criterion

$$\frac{2D\Delta t}{(\Delta x)^2} \le 1 \tag{17.2.6}$$

The physical interpretation of the restriction (17.2.6) is that the maximum allowed timestep is, up to a numerical factor, the diffusion time across a cell of width Δx.

More generally, the diffusion time τ across a spatial scale of size λ is of order

$$\tau \sim \frac{\lambda^2}{D} \tag{17.2.7}$$

Usually we are interested in modeling accurately the evolution of features with spatial scales $\lambda \gg \Delta x$. If we are limited to timesteps satisfying (17.2.6), we will need to evolve through of order $\lambda^2/(\Delta x)^2$ steps before things start to happen on the scale of interest. This number of steps is usually prohibitive. We must therefore find a stable way of taking timesteps comparable to, or perhaps — for accuracy — somewhat smaller than, the time scale of (17.2.7).

This goal poses an immediate "philosophical" question. Obviously the large timesteps that we propose to take are going to be woefully inaccurate for the small scales that we have decided not to be interested in. We want those scales to do something stable, "innocuous," and perhaps not too physically unreasonable. We want to build this innocuous behavior into our differencing scheme. What should it be?

There are two different answers, each of which has its pros and cons. The first answer is to seek a differencing scheme that drives small scale features to their *equilibrium* forms, e.g. satisfying equation (17.2.3) with the left-hand side set to zero. This answer generally makes the best physical sense; but, as we will see, it leads to a differencing scheme ("fully implicit") that is only *first-order* accurate in time for the scales that we are interested in. The second

answer is to let small scale features *maintain* their initial amplitudes, so that the evolution of the larger scale features of interest takes place superposed with a kind of "frozen in" (though fluctuating) background of small scale stuff. This answer gives a differencing scheme ("Crank-Nicholson") that is *second-order* accurate in time. Toward the end of an evolution calculation, however, one might want to switch over to some steps of the other kind, to drive the small-scale stuff into equilibrium. Let us now see where these distinct differencing schemes come from:

Consider the following differencing of (17.2.3),

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\left[\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{(\Delta x)^2}\right] \tag{17.2.8}$$

This is exactly like the FTCS scheme (17.2.4), except that the spatial derivatives on the right-hand side are evaluated at timestep $n+1$. Schemes with this character are called *fully implicit* or *backward time*, by contrast with FTCS (which is called *fully explicit*). To solve equation (17.2.8) one has to solve a set of simultaneous linear equations at each timestep for the u_j^{n+1}. Fortunately, this is a simple problem because the system is tridiagonal: Just group the terms in equation (17.2.8) appropriately:

$$-\alpha u_{j-1}^{n+1} + (1+2\alpha)u_j^{n+1} - \alpha u_{j+1}^{n+1} = u_j^n, \qquad j = 1, 2 \ldots J-1 \tag{17.2.9}$$

where

$$\alpha \equiv \frac{D\Delta t}{(\Delta x)^2} \tag{17.2.10}$$

Supplemented by Dirichlet or Neumann boundary conditions at $j = 0$ and $j = J$, equation (17.2.9) is clearly a tridiagonal system, which can easily be solved at each timestep by the method of §2.6.

What is the behavior of (17.2.8) for very large timesteps? The answer is seen most clearly in (17.2.9), in the limit $\alpha \to \infty$ ($\Delta t \to \infty$). Dividing by α, we see that the difference equations are just the finite-difference form of the equilibrium equation

$$\frac{\partial^2 u}{\partial x^2} = 0 \tag{17.2.11}$$

What about stability? The amplification factor for equation (17.2.8) is

$$\xi = \frac{1}{1 + 4\alpha\sin^2\left(\frac{k\Delta x}{2}\right)} \tag{17.2.12}$$

Clearly $|\xi| < 1$ for any stepsize Δt. The scheme is unconditionally stable. The details of the small-scale evolution from the initial conditions are obviously inaccurate for large Δt. But, as advertised, the correct equilibrium solution is obtained. This is the characteristic feature of implicit methods.

Here, on the other hand, is how one gets to the second of our above philosophical answers, combining the stability of an implicit method with the accuracy of a method that is second-order in both space and time. Simply form the average of the explicit and implicit FTCS schemes:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{D}{2}\left[\frac{(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}) + (u_{j+1}^n - 2u_j^n + u_{j-1}^n)}{(\Delta x)^2}\right] \tag{17.2.13}$$

Here both the left- and right-hand sides are centered at timestep $n + \frac{1}{2}$, so the method is second-order accurate in time as claimed. The amplification factor is

$$\xi = \frac{1 - 2\alpha \sin^2\left(\frac{k\Delta x}{2}\right)}{1 + 2\alpha \sin^2\left(\frac{k\Delta x}{2}\right)} \tag{17.2.14}$$

so the method is stable for any size Δt. This scheme is called the *Crank-Nicholson* scheme, and is our recommended method for any simple diffusion problem (perhaps supplemented by a few fully implicit steps at the end). (See Figure 17.2.1.)

Now turn to some generalizations of the simple diffusion equation (17.2.3). Suppose first that the diffusion coefficient D is not constant, say $D = D(x)$. We can adopt either of two strategies. First, we can make an analytic change of variable

$$y = \int \frac{dx}{D(x)} \tag{17.2.15}$$

Then

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} D(x) \frac{\partial u}{\partial x} \tag{17.2.16}$$

becomes

$$\frac{\partial u}{\partial t} = \frac{1}{D(y)} \frac{\partial^2 u}{\partial y^2} \tag{17.2.17}$$

and we evaluate D at the appropriate y_j. Heuristically, the stability criterion

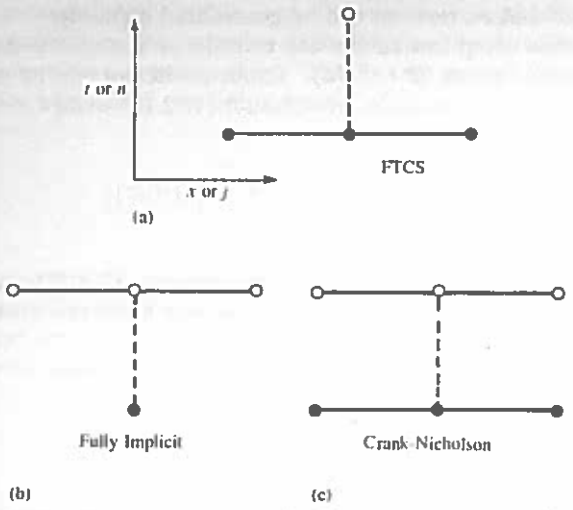

Figure 17.2.1. Three differencing schemes for diffusive problems (shown as in Figure 17.1.2). (a) Forward Time Center Space is first-order accurate, but stable only for sufficiently small timesteps. (b) Fully Implicit is stable for arbitrarily large timesteps, but is still only first-order accurate. (c) Crank-Nicholson is second-order accurate, and is usually stable for large timesteps.

(17.2.6) in an explicit scheme becomes

$$\Delta t \le \min_j \left[\frac{(\Delta y)^2}{2D_j^{-1}}\right] \tag{17.2.18}$$

Note that constant spacing Δy in y does not imply constant spacing in x.

An alternative method that does not require analytically tractable forms for D is simply to difference equation (17.2.16) as it stands, centering everything appropriately. Thus the FTCS method becomes

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{D_{j+1/2}(u_{j+1}^n - u_j^n) - D_{j-1/2}(u_j^n - u_{j-1}^n)}{(\Delta x)^2} \tag{17.2.19}$$

where

$$D_{j+1/2} \equiv D(x_{j+1/2}) \tag{17.2.20}$$

and the heuristic stability criterion is

$$\Delta t \le \min_j \left[\frac{(\Delta x)^2}{2D_{j+1/2}}\right] \tag{17.2.21}$$

The Crank-Nicholson method can be generalized similarly.

The second complication one can consider is a nonlinear diffusion problem, for example where $D = D(u)$. Explicit schemes can be generalized in the obvious way. For example, in equation (17.2.19) write

$$D_{j+1/2} = \frac{1}{2}\left[D(u^n_{j+1}) + D(u^n_j)\right] \tag{17.2.22}$$

Implicit schemes are not as easy. The replacement (17.2.22) with $n \to n+1$ leaves us with a nasty set of coupled nonlinear equations to solve at each timestep. Instead, we linearize them by writing

$$D(u^{n+1}_j) = D(u^n_j) + (u^{n+1}_j - u^n_j)\left.\frac{\partial D}{\partial u}\right|_{j,n} \tag{17.2.23}$$

This reduces the problem to tridiagonal form again and in practice usually retains the stability advantages of implicit differencing.

Schrödinger Equation

Sometimes the physical problem being solved imposes constraints on the differencing scheme that we have not yet taken into account. For example, consider the time-dependent Schrödinger equation of quantum mechanics. This is basically a parabolic equation for the evolution of a complex quantity ψ. For the scattering of a wavepacket by a one-dimensional potential $V(x)$, the equation has the form

$$i\frac{\partial \psi}{\partial t} = -\frac{\partial^2 \psi}{\partial x^2} + V(x)\psi \tag{17.2.24}$$

(Here we have chosen units so that Planck's constant $\hbar = 1$ and the particle mass $m = 1/2$.) One is given the initial wavepacket, $\psi(x, t = 0)$, together with boundary conditions that $\psi \to 0$ at $x \to \pm\infty$. Suppose we content ourselves with first-order accuracy in time, but want to use an implicit scheme, for stability. A slight generalization of (17.2.8) leads to

$$i\left[\frac{\psi^{n+1}_j - \psi^n_j}{\Delta t}\right] = -\left[\frac{\psi^{n+1}_{j+1} - 2\psi^{n+1}_j + \psi^{n+1}_{j-1}}{(\Delta x)^2}\right] + V_j\psi^{n+1}_j \tag{17.2.25}$$

for which

$$\xi = \frac{1}{1 + i\left[\dfrac{4\Delta t}{(\Delta x)^2}\sin^2\left(\dfrac{k\Delta x}{2}\right) + V_j\Delta t\right]} \tag{17.2.26}$$

This is unconditionally stable, but unfortunately is not *unitary*. The underlying physical problem requires that the total probability of finding the particle somewhere remains unity. This is represented formally by the modulus-square norm of ψ remaining unity:

$$\int_{-\infty}^{\infty} |\psi|^2 dx = 1 \tag{17.2.27}$$

The initial wave function $\psi(x, 0)$ is normalized to satisfy (17.2.27). The Schrödinger equation (17.2.24) then guarantees that this condition is satisfied at all later times.

Let us write equation (17.2.24) in the form

$$i\frac{\partial \psi}{\partial t} = H\psi \tag{17.2.28}$$

where the operator H is

$$H = -\frac{\partial^2}{\partial x^2} + V(x) \tag{17.2.29}$$

The formal solution of equation (17.2.28) is

$$\psi(x, t) = e^{-iHt}\psi(x, 0) \tag{17.2.30}$$

where the exponential of the operator is defined by its power series expansion.

The explicit FTCS scheme approximates (17.2.30) as

$$\psi^{n+1}_j = (1 - iH\Delta t)\psi^n_j \tag{17.2.31}$$

where H is represented by a centered finite-difference approximation in x. The stable implicit scheme (17.2.25) is, by contrast,

$$\psi^{n+1}_j = (1 + iH\Delta t)^{-1}\psi^n_j \tag{17.2.32}$$

These are both first-order accurate in time, as can be seen by expanding equation (17.2.30). However, neither operator in (17.2.31) or (17.2.32) is unitary.

The correct way to difference Schrödinger's equation is to use *Cayley's form* for the finite-difference representation of e^{-iHt}, which is second-order accurate *and* unitary:

$$e^{-iHt} \simeq \frac{1 - \frac{1}{2}iH\Delta t}{1 + \frac{1}{2}iH\Delta t} \tag{17.2.33}$$

In other words,

$$(1+\frac{1}{2}iH\Delta t)\psi_j^{n+1} = (1-\frac{1}{2}iH\Delta t)\psi_j^n \tag{17.2.34}$$

On replacing H by its finite-difference approximation in x, we have a complex tridiagonal system to solve. The method is stable, unitary, and second-order accurate in space and time. In fact, it is simply the Crank-Nicholson method once again!

REFERENCES AND FURTHER READING:

Ames, W.F. 1977, *Numerical Methods for Partial Differential Equations*, 2nd ed. (New York: Academic Press), Chapter 2.

Goldberg, A., Schey, H.M., and Schwartz, J.L. 1967, *American Journal of Physics*, vol. 35, pp. 177–186. [Schrödinger equation].

Galbraith, I., Ching, Y.S., and Abraham, E. 1984, *American Journal of Physics*, vol. 52, pp. 60–68. [Schrödinger equation].

17.3 Initial Value Problems in Multidimensions

The methods described in §17.1 and §17.2 for problems in 1 + 1 dimension (one space and one time dimension) can easily be generalized to $N+1$ dimensions. However, the computing power necessary to solve the resulting equations is enormous. If you have solved a one-dimensional problem with 100 spatial grid points, solving the two-dimensional version with 100×100 mesh points requires *at least* 100 times as much computing. You generally have to be content with very modest spatial resolution in multidimensional problems.

Indulge us in offering a bit of advice about the development and testing of multidimensional PDE codes: You should always develop and test your programs on *very small* grids, e.g., 8×8, even though the resulting accuracy is so poor as to be useless. When your program is all debugged and demonstrably stable, *then* you can increase the grid size to a reasonable one and start looking at the results. We have actually heard someone protest, "my program would be unstable for a crude grid, but I am sure the instability will go away on a larger grid." That is nonsense of a most pernicious sort, evidencing total confusion between accuracy and stability. In fact, new instabilities sometimes do show up on *larger* grids; but old instabilities never (in our experience) just go away.

Forced to live with modest grid sizes, some people recommend going to higher order methods in an attempt to improve accuracy. This is very dangerous. Unless the solution you are looking for is known to be smooth, and the high order method you are using is known to be extremely stable, we do not recommend anything higher than second-order in time (for sets of first-order equations). For spatial differencing, we recommend the order of the underlying PDEs, perhaps allowing second-order spatial differencing for first-order-in-space PDEs. When you increase the order of a differencing method to greater than the order of the original PDEs, you introduce spurious solutions to the difference equations. This does not create a problem if they all happen to decay exponentially; otherwise you are going to see all hell break loose!

Lax Method for a Flux-Conservative Equation

As an example, we show how to generalize the Lax method (17.1.15) to two dimensions for the conservation equation

$$\frac{\partial u}{\partial t} = -\nabla \cdot \mathbf{F} = -\left(\frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}\right) \tag{17.3.1}$$

Use a spatial grid with

$$\begin{aligned} x_j &= x_0 + j\Delta \\ y_l &= y_0 + l\Delta \end{aligned} \tag{17.3.2}$$

We have chosen $\Delta x = \Delta y \equiv \Delta$ for simplicity. Then the Lax scheme is

$$\begin{aligned} u_{j,l}^{n+1} = {} & \frac{1}{4}(u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n) \\ & - \frac{\Delta t}{2\Delta}(F_{j+1,l}^n - F_{j-1,l}^n + F_{j,l+1}^n - F_{j,l-1}^n) \end{aligned} \tag{17.3.3}$$

Note that as an abbreviated notation F_{j+1} and F_{j-1} refer to F_x, while F_{l+1} and F_{l-1} refer to F_y.

Let us carry out a stability analysis for the model advective equation (analog of 17.1.6) with

$$F_x = v_x u, \qquad F_y = v_y u \tag{17.3.4}$$

This requires an eigenmode with two dimensions in space, though still only a simple dependence on powers of ξ in time,

$$u_{j,l}^n = \xi^n e^{ik_x j\Delta} e^{ik_y l\Delta} \tag{17.3.5}$$

Substituting in equation (17.3.3), we find

$$\xi = \frac{1}{2}(\cos k_x\Delta + \cos k_y\Delta) - i\alpha_x \sin k_x\Delta - i\alpha_y \sin k_y\Delta \tag{17.3.6}$$

where

$$\alpha_x = \frac{v_x \Delta t}{\Delta}, \qquad \alpha_y = \frac{v_y \Delta t}{\Delta} \tag{17.3.7}$$

The expression for $|\xi|^2$ can be manipulated into the form

$$\begin{aligned} |\xi|^2 = 1 &- (\sin^2 k_x\Delta + \sin^2 k_y\Delta)\left[\frac{1}{2} - (\alpha_x^2 + \alpha_y^2)\right] \\ &- \frac{1}{4}(\cos k_x\Delta - \cos k_y\Delta)^2 \\ &- (\alpha_y \sin k_x\Delta - \alpha_x \sin k_y\Delta)^2 \end{aligned} \tag{17.3.8}$$

The last two terms are negative, and so the stability requirement $|\xi|^2 \le 1$ becomes

$$\frac{1}{2} - (\alpha_x^2 + \alpha_y^2) \ge 0 \tag{17.3.9}$$

or

$$\Delta t \le \frac{\Delta}{\sqrt{2}(v_x^2 + v_y^2)^{1/2}} \tag{17.3.10}$$

This is an example of the general result for the N-dimensional Courant condition: If $|v|$ is the maximum propagation velocity in the problem, then

$$\Delta t \le \frac{\Delta}{\sqrt{N}|v|} \tag{17.3.11}$$

is the Courant condition.

Diffusion Equation in Multidimensions

Let us consider the two-dimensional diffusion equation,

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{17.3.12}$$

An explicit method, such as FTCS, can be generalized from the one-dimensional case in the obvious way. However, we have seen that diffusive problems are usually best treated implicitly. Suppose we try to implement the Crank-Nicholson scheme in two dimensions. This would give us

$$u_{j,l}^{n+1} = u_{j,l}^n + \frac{1}{2}\alpha\left(\delta_x^2 u_{j,l}^{n+1} + \delta_x^2 u_{j,l}^n + \delta_y^2 u_{j,l}^{n+1} + \delta_y^2 u_{j,l}^n\right) \tag{17.3.13}$$

Here

$$\alpha \equiv \frac{D\Delta t}{\Delta^2} \qquad \Delta \equiv \Delta x = \Delta y \tag{17.3.14}$$

$$\delta_x^2 u_{j,l}^n \equiv u_{j+1,l}^n - 2u_{j,l}^n + u_{j-1,l}^n \tag{17.3.15}$$

and similarly for $\delta_y^2 u_{j,l}^n$. This is certainly a viable scheme; the problem arises in solving the coupled linear equations. Whereas in one space dimension the system was tridiagonal, that is no longer true, though the matrix is still very sparse. One possibility is to use a suitable sparse matrix technique (see §2.10 and §17.0).

Another possibility, which we generally prefer, is a slightly different way of generalizing the Crank-Nicholson algorithm. It is still second-order accurate in time and space, and unconditionally stable, but the equations are easier to solve than (17.3.13). Called the *alternating-direction implicit method (ADI)*, this scheme is our first mention of the powerful concept of *operator splitting* or *time splitting*. We will discuss that concept in more detail in §17.6. As a preview here, the idea is to divide each timestep into two steps of size $\Delta t/2$. In each substep, a different dimension is treated implicitly:

$$\begin{aligned} u_{j,l}^{n+1/2} &= u_{j,l}^n + \frac{1}{2}\alpha\left(\delta_x^2 u_{j,l}^{n+1/2} + \delta_y^2 u_{j,l}^n\right) \\ u_{j,l}^{n+1} &= u_{j,l}^{n+1/2} + \frac{1}{2}\alpha\left(\delta_x^2 u_{j,l}^{n+1/2} + \delta_y^2 u_{j,l}^{n+1}\right) \end{aligned} \tag{17.3.16}$$

The advantage of this method is that each substep requires only the solution of a simple tridiagonal system.

It is at this point that we turn our attention from initial value problems to boundary value problems. These will occupy us for the remainder of the chapter.

REFERENCES AND FURTHER READING:

Ames, W.F. 1977, *Numerical Methods for Partial Differential Equations*, 2nd ed. (New York: Academic Press).

17.4 Fourier and Cyclic Reduction Methods for Boundary Value Problems

As discussed in §17.0, most boundary value problems (elliptic equations, for example) reduce to solving large sparse linear systems of the form

$$\mathbf{A} \cdot \mathbf{u} = \mathbf{b} \tag{17.4.1}$$

either once, for boundary value equations that are linear, or iteratively, for boundary value equations that are nonlinear.

Two important techniques lead to "rapid" solution of equation (17.4.1) when the sparse matrix is of certain frequently occurring forms. The *Fourier transform method* is directly applicable when the equations have coefficients that are constant in space. The *cyclic reduction* method is somewhat more general; its applicability is related to the question of whether the equations are separable (in the sense of "separation of variables"). Both methods require the boundaries to coincide with the coordinate lines. Finally, for some problems, there is a powerful combination of these two methods called *FACR (Fourier Analysis and Cyclic Reduction)*. We now consider each method in turn, using equation (17.0.3), with finite difference representation (17.0.6), as a model example.

Fourier Transform Method

The discrete inverse Fourier transform in both x and y is

$$u_{jl} = \frac{1}{JL} \sum_{m=0}^{J-1} \sum_{n=0}^{L-1} \hat{u}_{mn} e^{-2\pi i j m/J} e^{-2\pi i l n/L} \tag{17.4.2}$$

This can be computed using the FFT independently in each dimension, or else all at once via the routine FOURN of §12.11. Similarly,

$$\rho_{jl} = \frac{1}{JL} \sum_{m=0}^{J-1} \sum_{n=0}^{L-1} \hat{\rho}_{mn} e^{-2\pi i j m/J} e^{-2\pi i l n/L} \tag{17.4.3}$$

If we substitute expressions (17.4.2) and (17.4.3) in our model problem (17.0.6), we find

$$\hat{u}_{mn} \left(e^{2\pi i m/J} + e^{-2\pi i m/J} + e^{2\pi i n/L} + e^{-2\pi i n/L} - 4 \right) = \hat{\rho}_{mn} \Delta^2 \tag{17.4.4}$$

or

$$\hat{u}_{mn} = \frac{\hat{\rho}_{mn} \Delta^2}{2 \left(\cos \dfrac{2\pi m}{J} + \cos \dfrac{2\pi n}{L} - 2 \right)} \tag{17.4.5}$$

Thus the strategy for solving equation (17.0.6) by FFT techniques is:

- Compute $\hat{\rho}_{mn}$ as the Fourier transform

$$\hat{\rho}_{mn} = \sum_{j=0}^{J-1} \sum_{l=0}^{L-1} \rho_{jl} e^{2\pi i m j/J} e^{2\pi i n l/L} \tag{17.4.6}$$

- Compute $\hat{u}_{mn}$ from equation (17.4.5).
- Compute u_{jl} by the inverse Fourier transform (17.4.2).

The above procedure is valid for periodic boundary conditions. In other words, the solution satisfies

$$u_{jl} = u_{j+J,l} = u_{j,l+L} \tag{17.4.7}$$

Consider now a Dirichlet boundary condition $u = 0$ on the rectangular boundary. Instead of the expansion (17.4.2), we now need an expansion in sine waves:

$$u_{jl} = \frac{2}{J} \frac{2}{L} \sum_{m=1}^{J-1} \sum_{n=1}^{L-1} \hat{u}_{mn} \sin \frac{\pi j m}{J} \sin \frac{\pi l n}{L} \tag{17.4.8}$$

This satisfies the boundary conditions that $u = 0$ at $j = 0, J$ and at $l = 0, L$. If we substitute this expansion and the analogous one for ρ_{jl} into equation (17.0.6), we find that the solution procedure parallels that for periodic boundary conditions:

- Compute $\hat{\rho}_{mn}$ by the sine transform

$$\hat{\rho}_{mn} = \sum_{j=1}^{J-1} \sum_{l=1}^{L-1} \rho_{jl} \sin \frac{\pi j m}{J} \sin \frac{\pi l n}{L} \tag{17.4.9}$$

(A fast sine transform algorithm was given in §12.3.)

- Compute $\hat{u}_{mn}$ from the expression analogous to (17.4.5),

$$\hat{u}_{mn} = \frac{\Delta^2 \hat{\rho}_{mn}}{2 \left(\cos \dfrac{\pi m}{J} + \cos \dfrac{\pi n}{L} - 2 \right)} \tag{17.4.10}$$

- Compute u_{jl} by the inverse sine transform (17.4.8).

If we have inhomogeneous boundary conditions, for example $u = 0$ on all boundaries except $u = f(y)$ on the boundary $x = J\Delta$, we have to add to the above solution a solution u^H of the homogeneous equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{17.4.11}$$

that satisfies the required boundary conditions. In the continuum case, this would be an expression of the form

$$u^H = \sum_n A_n \sinh \frac{n\pi x}{J\Delta} \sin \frac{n\pi y}{L\Delta} \tag{17.4.12}$$

where A_n would be found by requiring that $u = f(y)$ at $x = J\Delta$. In the discrete case, we have

$$u^H_{jl} = \frac{2}{L} \sum_{n=1}^{L-1} A_n \sinh \frac{\pi n j}{J} \sin \frac{\pi n l}{L} \tag{17.4.13}$$

If $f(y = l\Delta) \equiv f_l$, then we get A_n from the inverse formula

$$A_n = \frac{1}{\sinh \pi n} \sum_{l=1}^{L-1} f_l \sin \frac{\pi n l}{L} \tag{17.4.14}$$

The complete solution to the problem is

$$u = u_{jl} + u^H_{jl} \tag{17.4.15}$$

By adding appropriate terms of the form (17.4.12), we can handle inhomogeneous terms on any boundary surface.

A much simpler procedure for handling inhomogeneous terms is to note that whenever boundary terms appear on the left-hand side of (17.0.6), they can be taken over to the right-hand side since they are known. The effective source term is therefore ρ_{jl} plus a contribution from the boundary terms. For example, instead of equation (17.4.13), we write down equation (17.0.6) at $j = J - 1$:

$$u_{J,l} + u_{J-2,l} + u_{J-1,l+1} + u_{J-1,l-1} - 4u_{J-1,l} = \Delta^2 \rho_{J-1,l} \tag{17.4.16}$$

The term $u_{J,l}$ is just the boundary value f_l, so we take it over to the right-hand side. The problem is now equivalent to the case of zero boundary conditions, except that one row of the source term is modified by the replacement

$$\Delta^2 \rho_{J-1,l} \to \Delta^2 \rho_{J-1,l} - f_l \tag{17.4.17}$$

Note that it is crucial that we use the sine transform to automatically impose the zero boundary condition for this method to work.

The case of Neumann boundary conditions $\nabla u = 0$ is handled by the cosine expansion (see §12.3)

$$u_{jl} = \frac{1}{2}\hat{u}_{00} + \frac{2}{J}\frac{2}{L} \sum_{m=1}^{J-1} \sum_{n=1}^{L-1} \hat{u}_{mn} \cos \frac{\pi j m}{J} \cos \frac{\pi l n}{L} \tag{17.4.18}$$

Inhomogeneous terms $\nabla u = g$ can be again included by adding a suitable solution of the homogeneous equation, or more simply by taking boundary terms over to the right-hand side. For example, the condition

$$\frac{\partial u}{\partial x} = g(y) \qquad \text{at} \quad x = 0 \tag{17.4.19}$$

becomes

$$\frac{u_{1,l} - u_{-1,l}}{2\Delta} = g_l \tag{17.4.20}$$

where $g_l \equiv g(y = l\Delta)$. This equation must be satisfied in conjunction with equation (17.0.6) at $j = 0$:

$$u_{1,l} + u_{-1,l} + u_{0,l+1} + u_{0,l-1} - 4u_{0,l} = \Delta^2 \rho_{0,l} \tag{17.4.21}$$

Define a new variable

$$u^*_{-1,l} = u_{-1,l} + 2\Delta g_l \tag{17.4.22}$$

Then equations (17.4.20) and (17.4.21) can be rewritten as

$$u_{1,l} - u^*_{-1,l} = 0 \tag{17.4.23}$$

$$u_{1,l} + u^*_{-1,l} + u_{0,l+1} + u_{0,l-1} - 4u_{0,l} = \Delta^2 \rho_{0,l} + 2\Delta g_l \tag{17.4.24}$$

Equations (17.4.23) and (17.4.24) are the equations for a zero gradient problem, with the source term modified by the replacement

$$\Delta^2 \rho_{0,l} \to \Delta^2 \rho_{0,l} + 2\Delta g_l \tag{17.4.25}$$

Again it is crucial that we use the cosine transform here to automatically impose the zero gradient boundary condition.

Cyclic Reduction

Evidently the FFT method works only when the original PDE has constant coefficients, and boundaries that coincide with the coordinate lines. An alternative algorithm, which can be used on somewhat more general equations, is called *cyclic reduction (CR)*.

We illustrate cyclic reduction on the equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + b(y)\frac{\partial u}{\partial y} + c(y)u = g(x,y) \tag{17.4.26}$$

This form arises very often in practice from the Helmholtz or Poisson equations in polar, cylindrical or spherical coordinate systems. More general separable equations are treated by Swartzrauber.

The finite-difference form of equation (17.4.26) can be written as a set of vector equations

$$\mathbf{u}_{j-1} + \mathbf{T}\cdot\mathbf{u}_j + \mathbf{u}_{j+1} = \mathbf{g}_j\Delta^2 \tag{17.4.27}$$

Here the index j comes from differencing in the x-direction, while the y-differencing (denoted by the index l previously) has been left in vector form. The matrix $\mathbf{T}$ has the form

$$\mathbf{T} = \mathbf{B} - 2\mathbf{1} \tag{17.4.28}$$

where the $2\mathbf{1}$ comes from the x-differencing and the matrix $\mathbf{B}$ from the y-differencing. The matrix $\mathbf{B}$, and hence $\mathbf{T}$, is tridiagonal with variable coefficients.

The CR method is derived by writing down three successive equations like (17.4.27):

$$\begin{aligned}
\mathbf{u}_{j-2} + \mathbf{T}\cdot\mathbf{u}_{j-1} + \mathbf{u}_j &= \mathbf{g}_{j-1}\Delta^2 \\
\mathbf{u}_{j-1} + \mathbf{T}\cdot\mathbf{u}_j + \mathbf{u}_{j+1} &= \mathbf{g}_j\Delta^2 \\
\mathbf{u}_j + \mathbf{T}\cdot\mathbf{u}_{j+1} + \mathbf{u}_{j+2} &= \mathbf{g}_{j+1}\Delta^2
\end{aligned} \tag{17.4.29}$$

Matrix-multiplying the middle equation by $-\mathbf{T}$ and then adding the three equations, we get

$$\mathbf{u}_{j-2} + \mathbf{T}^{(1)}\cdot\mathbf{u}_j + \mathbf{u}_{j+2} = \mathbf{g}_j^{(1)}\Delta^2 \tag{17.4.30}$$

This is an equation of the same form as (17.4.27), with

$$\begin{aligned}
\mathbf{T}^{(1)} &= 2\mathbf{1} - \mathbf{T}^2 \\
\mathbf{g}_j^{(1)} &= \Delta^2(\mathbf{g}_{j-1} - \mathbf{T}\cdot\mathbf{g}_j + \mathbf{g}_{j+1})
\end{aligned} \tag{17.4.31}$$

After one level of CR, we have reduced the number of equations by a factor of two. Since the resulting equations are of the same form as the original equation, we can repeat the process. Taking the number of mesh points to be a power of 2 for simplicity, we finally end up with a single equation for the central line of variables:

$$\mathbf{T}^{(f)}\cdot\mathbf{u}_{J/2} = \Delta^2\mathbf{g}_{J/2}^{(f)} - \mathbf{u}_0 - \mathbf{u}_J \tag{17.4.32}$$

Here we have written $\mathbf{u}_0$ and $\mathbf{u}_J$ on the right-hand side because they are known boundary values. Equation (17.4.32) can be solved for $\mathbf{u}_{J/2}$ by the standard tridiagonal algorithm. The two equations at level $f-1$ involve $\mathbf{u}_{J/4}$ and $\mathbf{u}_{3J/4}$. The equation for $\mathbf{u}_{J/4}$ involves $\mathbf{u}_0$ and $\mathbf{u}_{J/2}$, both of which are known, and hence can be solved by the usual tridiagonal routine. A similar result holds true at every stage, so we end up solving $J-1$ tridiagonal systems.

In practice, equations (17.4.31) should be rewritten to avoid numerical instability. For these and other practical details, we refer you to the review paper of Buzbee et al.

FACR Method

The *best* way to solve equations of the form (17.4.26), including the constant coefficient problem (17.0.3), is a combination of Fourier analysis and cyclic reduction, the FACR method (Hockney 1965, 1971; Temperton). If at the r'th stage of CR we Fourier analyze the equations of the form (17.4.30) along y, that is, with respect to the suppressed vector index, we will have a tridiagonal system in the x-direction for each y-Fourier mode:

$$\hat{u}_{j-2^r}^k + \lambda_k^{(r)}\hat{u}_j^k + \hat{u}_{j+2^r}^k = \Delta^2 g_j^{(r)k} \tag{17.4.33}$$

Here $\lambda_k^{(r)}$ is the eigenvalue of $\mathbf{T}^{(r)}$ corresponding to the k'th Fourier mode. For the equation (17.0.3), equation (17.4.5) shows that $\lambda_k^{(r)}$ will involve terms like $\cos\frac{2\pi k}{L} - 2$ raised to a power. Solve the tridiagonal systems for $\hat{u}_j^k$ at the levels $j = 2^r, 2\times 2^r, 4\times 2^r, \ldots, J-2^r$. Fourier synthesize to get the y-values on these x-lines. Then fill in the intermediate x-lines as in the original CR algorithm.

The trick is to choose the number of levels of CR so as to minimize the total number of arithmetic operations. One can show that for a typical case of a 128×128 mesh, the optimal level is $r = 2$; asymptotically, $r \to \log_2(\log_2 J)$.

A rough estimate of running times for these algorithms for equation (17.0.3) is as follows: the FFT method (in both x and y) and the CR method are roughly comparable. FACR with $r = 0$ (that is, FFT in one dimension and solve the tridiagonal equations by the usual algorithm in the other dimension) gives about a factor of two gain in speed. The optimal FACR with $r = 2$ gives another factor of two gain in speed.

REFERENCES AND FURTHER READING:

Swartzrauber, P.N. 1977, *S.I.A.M Review*, vol. 19, pp. 490 – 501.

Buzbee, B.L, Golub, G.H., and Nielson, C.W. 1970, *S.I.A.M. Journal of Numerical Analysis*, vol. 7, pp. 627–656; see also *op. cit.* vol. 11, pp. 753–763.

Hockney, R.W. 1965, *Journal Assn. Comp. Mach.*, vol. 12, p. 95 ff.

Hockney, R.W. 1970, in *Methods of Computational Physics*, vol. 9 (New York: Academic Press), pp. 135–211.

Hockney, R.W., and Eastwood, J.W. 1981, *Computer Simulation Using Particles* (New York: McGraw Hill), Chapter 6.

Temperton, C. 1980, *Journal of Computational Physics*, vol. 34, pp. 314–329.

17.5 Relaxation Methods for Boundary Value Problems

As we mentioned in §17.0, relaxation methods involve splitting the sparse matrix that arises from finite differencing and then iterating until a solution is found.

There is another way of thinking about relaxation methods that is somewhat more physical. Suppose we wish to solve the elliptic equation

$$\mathcal{L}u = \rho \tag{17.5.1}$$

where $\mathcal{L}$ represents some elliptic operator and ρ is the source term. Rewrite the equation as a diffusion equation,

$$\frac{\partial u}{\partial t} = \mathcal{L}u - \rho \tag{17.5.2}$$

An initial distribution u *relaxes* to an equilibrium solution as $t \to \infty$. This equilibrium has all time derivatives vanishing. Therefore it is the solution of the original elliptic problem (17.5.1). We see that all the machinery of §17.2, on diffusive initial value equations, can be brought to bear on the solution of boundary value problems by relaxation methods.

Let us apply this idea to our model problem (17.0.3). The diffusion equation is

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - \rho \tag{17.5.3}$$

If we use FTCS differencing (cf. equation 17.2.4), we get

$$u_{j,l}^{n+1} = u_{j,l}^n + \frac{\Delta t}{\Delta^2}\left(u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n - 4u_{j,l}^n\right) - \rho_{j,l}\Delta t \tag{17.5.4}$$

Recall from (17.2.6) that FTCS differencing is stable in one spatial dimension only if $\Delta t/\Delta^2 \le \frac{1}{2}$. In two dimensions this becomes $\Delta t/\Delta^2 \le \frac{1}{4}$. Suppose we try to take the largest possible timestep, and set $\Delta t = \Delta^2/4$. Then equation (17.5.4) becomes

$$u_{j,l}^{n+1} = \frac{1}{4}\left(u_{j+1,l}^n + u_{j-1,l}^n + u_{j,l+1}^n + u_{j,l-1}^n\right) - \frac{\Delta^2}{4}\rho_{j,l} \tag{17.5.5}$$

Thus the algorithm consists of using the average of u at its four nearest-neighbor points on the grid (plus the contribution from the source). This procedure is then iterated until convergence.

This method is in fact a classical method with origins dating back to the last century, called *Jacobi's method* (not to be confused with the Jacobi method for eigenvalues). The method is not practical because it converges too slowly. However, it is the basis for understanding the modern methods, which are always compared with it.

Another classical method is the *Gauss-Seidel* method. Here we make use of updated values of u on the right-hand side of (17.5.5) as soon as they become available. In other words, the averaging is done "in place" instead of being "copied" from an earlier timestep to a later one. If we are proceeding along the rows, incrementing j for fixed l, we have

$$u_{j,l}^{n+1} = \frac{1}{4}\left(u_{j+1,l}^n + u_{j-1,l}^{n+1} + u_{j,l+1}^n + u_{j,l-1}^{n+1}\right) - \frac{\Delta^2}{4}\rho_{j,l} \tag{17.5.6}$$

This method is also slowly converging and only of theoretical interest, but some analysis of it will be instructive.

Let us look at the Jacobi and Gauss-Seidel methods in terms of the matrix splitting concept. We change notation and call u "x", to conform to standard matrix notation. To solve

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{17.5.7}$$

we can consider splitting $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U} \tag{17.5.8}$$

where $\mathbf{D}$ is the diagonal part of $\mathbf{A}$, $\mathbf{L}$ is the lower triangle of $\mathbf{A}$ with zeros on the diagonal, and $\mathbf{U}$ is the upper triangle of $\mathbf{A}$ with zeros on the diagonal.

In the Jacobi method we write for the r'th step of iteration

$$\mathbf{D} \cdot \mathbf{x}^{(r)} = -(\mathbf{L} + \mathbf{U}) \cdot \mathbf{x}^{(r-1)} + \mathbf{b} \tag{17.5.9}$$

For our model problem (17.5.5), $\mathbf{D}$ is simply the identity matrix. The Jacobi method converges for matrices $\mathbf{A}$ that are "diagonally dominant" in a sense that can be made mathematically precise. For matrices arising from finite differencing, this condition is usually met.

What is the rate of convergence of the Jacobi method? A detailed analysis is beyond our scope, but here is some of the flavor: The matrix $-\mathbf{D}^{-1} \cdot (\mathbf{L}+\mathbf{U})$ is the *iteration matrix* which, apart from an additive term, maps one set of $\mathbf{x}$'s into the next. The iteration matrix has eigenvalues, each one of which reflects the factor by which the amplitude of a particular eigenmode of undesired residual is suppressed during one iteration. Evidently those factors had better all have modulus < 1 for the relaxation to work at all! The rate of convergence of the method is set by the rate for the slowest-decaying eigenmode, i.e., the factor with largest modulus. The modulus of this largest factor, therefore lying between 0 and 1, is called the *spectral radius* of the relaxation operator, denoted ρ_s.

The number of iterations r required to reduce the overall error by a factor 10^{-p} is thus estimated by

$$r \approx \frac{p \ln 10}{(-\ln \rho_s)} \tag{17.5.10}$$

In general, the spectral radius ρ_s goes asymptotically to the value 1 as the grid size J is increased, so that more iterations are required. For any given equation, grid geometry, *and boundary condition*, the spectral radius can, in principle, be computed analytically. For example, for equation (17.5.5) on a $J \times J$ grid with Dirichlet boundary conditions on all four sides, the asymptotic formula for large J turns out to be

$$\rho_s \simeq 1 - \frac{\pi^2}{2J^2} \tag{17.5.11}$$

The number of iterations r required to reduce the error by a factor of 10^{-p} is thus

$$r \simeq \frac{2pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{2} pJ^2 \tag{17.5.12}$$

In other words, the number of iterations is proportional to the number of mesh points, J^2. Since 100×100 and larger problems are common, it is clear that the Jacobi method is only of academic interest.

The Gauss-Seidel method, equation (17.5.6), corresponds to the matrix decomposition

$$(\mathbf{L} + \mathbf{D}) \cdot \mathbf{x}^{(r)} = -\mathbf{U} \cdot \mathbf{x}^{(r-1)} + \mathbf{b} \tag{17.5.13}$$

The fact that $\mathbf{L}$ is on the left-hand side of the equation follows from the updating in place, as you can easily check if you write out (17.5.13) in components. One can show (Varga; Young; Stoer and Bulirsch) that the spectral radius is just the square of the spectral radius of the Jacobi method. For our model problem, therefore,

$$\rho_s \simeq 1 - \frac{\pi^2}{J^2} \tag{17.5.14}$$

$$r \simeq \frac{pJ^2 \ln 10}{\pi^2} \simeq \frac{1}{4} pJ^2 \tag{17.5.15}$$

The factor of two improvement in the number of iterations over the Jacobi method still leaves the method impractical.

Simultaneous Over-Relaxation (SOR)

We get a practical algorithm if we make an *overcorrection* to the value of $\mathbf{x}^{(r)}$ at the r'th stage of Gauss-Seidel iteration, thus anticipating future corrections. Add and subtract $\mathbf{x}^{(r-1)}$ on the right-hand side of equation (17.5.13), and write the Gauss-Seidel method as

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - (\mathbf{L} + \mathbf{D})^{-1} \cdot [(\mathbf{L} + \mathbf{D} + \mathbf{U}) \cdot \mathbf{x}^{(r-1)} - \mathbf{b}] \tag{17.5.16}$$

The term in square brackets is just the residual vector $\xi^{(r-1)}$, so

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - (\mathbf{L} + \mathbf{D})^{-1} \cdot \xi^{(r-1)} \tag{17.5.17}$$

Now *overcorrect*, defining

$$\mathbf{x}^{(r)} = \mathbf{x}^{(r-1)} - \omega(\mathbf{L} + \mathbf{D})^{-1} \cdot \xi^{(r-1)} \tag{17.5.18}$$

Here ω is called the *overrelaxation parameter*, and the method is called *simultaneous overrelaxation* (SOR).

The following theorems can be proved (Varga; Young; Stoer and Bulirsch):

- The method is convergent only for $0 < \omega < 2$. If $0 < \omega < 1$, we speak of *underrelaxation*.
- Under certain mathematical restrictions generally satisfied by matrices arising from finite differencing, only overrelaxation ($1 < \omega < 2$) can give faster convergence than the Gauss-Seidel method.
- If ρ_{Jacobi} is the spectral radius of the Jacobi iteration (so that the square of it is the spectral radius of the Gauss-Seidel iteration), then the *optimal* choice for ω is given by

$$\omega = \frac{2}{1 + \sqrt{1 - \rho_{Jacobi}^2}} \tag{17.5.19}$$

- For this optimal choice, the spectral radius for SOR is

$$\rho_{SOR} = \left(\frac{\rho_{Jacobi}}{1 + \sqrt{1 - \rho_{Jacobi}^2}} \right)^2 \tag{17.5.20}$$

As an application of the above results, consider our model problem for which ρ_{Jacobi} is given by equation (17.5.11). Then equations (17.5.19) and (17.5.20) give

$$\omega \simeq \frac{2}{1 + \pi/J} \tag{17.5.21}$$

$$\rho_{SOR} \simeq 1 - \frac{2\pi}{J} \qquad \text{for large} \quad J \tag{17.5.22}$$

Equation (17.5.10) gives for the number of iterations to reduce the initial error by a factor of 10^{-p},

$$r \simeq \frac{pJ \ln 10}{2\pi} \simeq \frac{1}{3} pJ \tag{17.5.23}$$

Comparing with equations (17.5.12) or (17.5.15), we see that optimal SOR requires of order J iterations, as opposed to of order J^2. Since J is typically 100 or even larger, this makes a tremendous difference! Equation (17.5.23) leads to the mnemonic that 3-figure accuracy ($p = 3$) requires a number of iterations equal to the number of mesh points along a side of the grid. For 6-figure accuracy, we require about twice as many iterations.

How do we choose ω for a problem for which the answer is not known analytically? That is just the weak point of SOR! The advantages of SOR obtain only in a fairly narrow window around the correct value of ω. It is better to take ω slightly too large, rather than slightly too small, but best to get it right.

One way to choose ω is to map your problem approximately onto a known problem, replacing the coefficients in the equation by average values. Note, however, that the known problem must have the same grid size and boundary conditions as the actual problem. We give for reference purposes the value of ρ_{Jacobi} for our model problem on a rectangular $J \times L$ grid, allowing for the possibility that $\Delta x \neq \Delta y$:

$$\rho_{Jacobi} = \frac{\cos\frac{\pi}{J} + \left(\frac{\Delta x}{\Delta y}\right)^2 \cos\frac{\pi}{L}}{1 + \left(\frac{\Delta x}{\Delta y}\right)^2} \tag{17.5.24}$$

Equation (17.5.24) holds for homogeneous Dirichlet or Neumann boundary conditions. For periodic boundary conditions, make the replacement $\pi \to 2\pi$.

A second way, which is especially useful if you plan to solve many similar elliptic equations each time with slightly different coefficients, is to determine the optimum value ω empirically on the first equation and then use that value for the remaining equations. Various automated schemes for doing this and for "seeking out" the best values of ω are described in the literature.

While the matrix notation introduced earlier is useful for theoretical analyses, for practical implementation of the SOR algorithm we need explicit formulas. Consider a general second-order elliptic equation in x and y, finite differenced on a square as for our model equation. Corresponding to each row of the matrix $\mathbf{A}$ is an equation of the form

$$a_{j,l}u_{j+1,l} + b_{j,l}u_{j-1,l} + c_{j,l}u_{j,l+1} + d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} = f_{j,l} \tag{17.5.25}$$

For our model equation, we had $a = b = c = d = 1, e = -4$. The quantity f is proportional to the source term. The iterative procedure is defined by solving (17.5.25) for $u_{j,l}$:

$$u^*_{j,l} = \frac{1}{e_{j,l}} \left(f_{j,l} - a_{j,l}u_{j+1,l} - b_{j,l}u_{j-1,l} - c_{j,l}u_{j,l+1} - d_{j,l}u_{j,l-1} \right) \tag{17.5.26}$$

Then $u^{\text{new}}_{j,l}$ is a weighted average

$$u^{\text{new}}_{j,l} = \omega u_{j,l}{}^* + (1 - \omega) u^{\text{old}}_{j,l} \tag{17.5.27}$$

We calculate it as follows: The residual at any stage is

$$\xi_{j,l} = a_{j,l}u_{j+1,l} + b_{j,l}u_{j-1,l} + c_{j,l}u_{j,l+1} + d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} - f_{j,l} \tag{17.5.28}$$

and the SOR algorithm (17.5.18) or (17.5.27) is

$$u_{j,l}^{\text{new}} = u_{j,l}^{\text{old}} - \omega \frac{\xi_{j,l}}{c_{j,l}} \tag{17.5.29}$$

This formulation is very easy to program, and the norm of the residual vector $\xi_{j,l}$ can be used as a criterion for terminating the iteration.

Another practical point concerns the order in which mesh points are processed. The obvious strategy is simply to proceed in order down the rows (or columns). Alternatively, suppose we divide the mesh into "odd" and "even" meshes, like the black and white squares of a chessboard. Then equation (17.5.26) shows that the odd points depend only on the even mesh values and vice versa. Accordingly, we can carry out one half-sweep updating the odd points, say, and then another half-sweep updating the even points with the new odd values. For the version of SOR implemented below, we shall adopt odd-even ordering.

The last practical point is that in practice the asymptotic rate of convergence in SOR is not attained until of order J iterations. The error often grows by a factor of 20 before convergence sets in. A trivial modification to SOR resolves this problem. It is based on the observation that, while ω is the optimum *asymptotic* relaxation parameter, it is not necessarily a good initial choice. In SOR with *Chebyshev acceleration*, one uses odd-even ordering and changes ω at each half-sweep according to the following prescription:

$$\begin{aligned} \omega^{(0)} &= 1 \\ \omega^{(1/2)} &= 1/(1-\rho_{Jacobi}^2/2) \\ \omega^{(n+1/2)} &= 1/(1-\rho_{Jacobi}^2\omega^{(n)}/4), \qquad n = 1/2, 1, \ldots, \infty \\ \omega^{(\infty)} &\to \omega_{optimal} \end{aligned} \tag{17.5.30}$$

The beauty of Chebyshev acceleration is that the norm of the error always decreases with each iteration. (This is the norm of the actual error in $u_{j,l}$. The norm of the residual $\xi_{j,l}$ need not decrease monotonically.) While the asymptotic rate of convergence is the same as ordinary SOR, there is never any excuse for not using Chebyshev acceleration to reduce the total number of iterations required.

We should also mention the existence of so-called block methods of SOR. These methods can improve convergence by factors like $\sqrt{2}$, but are more complicated to program.

Here we give a routine for SOR with Chebyshev acceleration.

```
      SUBROUTINE SOR(A,B,C,D,E,F,U,JMAX,RJAC)
C     Simultaneous overrelaxation solution of equation (17.5.25) with Chebyshev acceleration.
C     A, B, C, D, E and F are input as the coefficients of the equation, each dimensioned to the
C     grid size JMAX x JMAX. U is input as the initial guess to the solution, usually zero, and
C     returns with the final value. RJAC is input as the spectral radius of the Jacobi iteration,
C     or an estimate of it.
      IMPLICIT REAL*8(A-H,O-Z)      Double precision is a good idea for JMAX bigger than about 25.
      DIMENSION A(JMAX,JMAX),B(JMAX,JMAX),C(JMAX,JMAX),
     *    D(JMAX,JMAX),E(JMAX,JMAX),F(JMAX,JMAX),U(JMAX,JMAX)
      PARAMETER(MAXITS=1000,EPS=1.D-5,ZERO=0.D0,HALF=.5D0,QTR=.25D0,ONE=1.D0)
      ANORMF=ZERO                   Compute initial norm of residual and terminate iteration when norm
      DO 12 J=2,JMAX-1                  has been reduced by a factor EPS.
        DO 11 L=2,JMAX-1
          ANORMF=ANORMF+ABS(F(J,L))     Assumes initial U is zero.
11      CONTINUE
12    CONTINUE
      OMEGA=ONE
      DO 15 N=1,MAXITS
        ANORM=ZERO
        DO 14 J=2,JMAX-1
          DO 13 L=2,JMAX-1
            IF(MOD(J+L,2).EQ.MOD(N,2))THEN      Odd-even ordering.
              RESID=A(J,L)*U(J+1,L)+B(J,L)*U(J-1,L)+
     *              C(J,L)*U(J,L+1)+D(J,L)*U(J,L-1)+
     *              E(J,L)*U(J,L)-F(J,L)
              ANORM=ANORM+ABS(RESID)
              U(J,L)=U(J,L)-OMEGA*RESID/E(J,L)
            ENDIF
13        CONTINUE
14      CONTINUE
        IF(N.EQ.1) THEN
          OMEGA=ONE/(ONE-HALF*RJAC**2)
        ELSE
          OMEGA=ONE/(ONE-QTR*RJAC**2*OMEGA)
        ENDIF
        IF((N.GT.1).AND.(ANORM.LT.EPS*ANORMF))RETURN
15    CONTINUE
      PAUSE 'MAXITS exceeded'
      END
```


REFERENCES AND FURTHER READING:

Hockney, R.W., and Eastwood, J.W. 1981, *Computer Simulation Using Particles* (New York: McGraw-Hill) Chapter 6.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §8.3–§8.5.

Varga, R.S. 1962, *Matrix Iterative Analysis* (Englewood Cliffs, N.J.: Prentice-Hall).

Young, D.M. 1971, *Iterative Solution of Large Linear Systems* (New York: Academic Press).

Rigal, A. 1979, *Journal of Computational Physics*, vol. 32, pp. 10–23.

17.6 Operator Splitting Methods and ADI

The basic idea of operator splitting, which is also called *time splitting* or *the method of fractional steps*, is this: Suppose you have an initial value equation of the form

$$\frac{\partial u}{\partial t} = \mathcal{L}u \tag{17.6.1}$$

where $\mathcal{L}$ is some operator. While $\mathcal{L}$ is not necessarily linear, suppose that it can at least be written as a linear sum of m pieces, which act additively on u,

$$\mathcal{L}u = \mathcal{L}_1 u + \mathcal{L}_2 u + \cdots + \mathcal{L}_m u \tag{17.6.1}$$

Finally, suppose that for *each* of the pieces, you already know a differencing scheme for updating the variable u from timestep n to timestep $n+1$, valid if that piece of the operator were the *only* one on the right-hand side. We will write these updatings symbolically as

$$\begin{aligned} u^{n+1} &= \mathcal{U}_1(u^n, \Delta t) \\ u^{n+1} &= \mathcal{U}_2(u^n, \Delta t) \\ &\cdots \\ u^{n+1} &= \mathcal{U}_m(u^n, \Delta t) \end{aligned} \tag{17.6.3}$$

Now, one form of operator splitting would be to get from n to $n+1$ by the following sequence of updatings:

$$\begin{aligned} u^{n+(1/m)} &= \mathcal{U}_1(u^n, \Delta t) \\ u^{n+(2/m)} &= \mathcal{U}_2(u^{n+(1/m)}, \Delta t) \\ &\cdots \\ u^{n+1} &= \mathcal{U}_m(u^{n+(m-1)/m}, \Delta t) \end{aligned} \tag{17.6.4}$$

For example, a combined advective-diffusion equation, such as

$$\frac{\partial u}{\partial t} = -v\frac{\partial u}{\partial x} + D\frac{\partial^2 u}{\partial x^2} \tag{17.6.5}$$

might profitably use an explicit scheme for the advective term combined with a Crank-Nicholson or other implicit scheme for the diffusion term.

We already introduced the *alternating direction implicit (ADI)* method in §17.3. That is a slightly different kind of operator splitting. Let us reinterpret (17.6.3) to have a different meaning: Let $\mathcal{U}_1$ now denote an updating method that includes algebraically *all* the pieces of the total operator $\mathcal{L}$, but which is desirably *stable* only for the $\mathcal{L}_1$ piece; likewise $\mathcal{U}_2, \ldots \mathcal{U}_m$. Then a method of getting from u^n to u^{n+1} is

$$\begin{aligned} u^{n+1/m} &= \mathcal{U}_1(u^n, \Delta t/m) \\ u^{n+2/m} &= \mathcal{U}_2(u^{n+1/m}, \Delta t/m) \\ &\cdots \\ u^{n+1} &= \mathcal{U}_m(u^{n+(m-1)/m}, \Delta t/m) \end{aligned} \tag{17.6.6}$$

The timestep for each fractional step in (17.6.6) is now only $1/m$ of the full timestep, because each partial operation acts with all the terms of the original operator.

Equation (17.6.6) is usually, though not always, stable as a differencing scheme for the operator $\mathcal{L}$. In fact, as a rule of thumb, it is often sufficient to have stable $\mathcal{U}_i$'s only for the operator pieces having the highest number of spatial derivatives – the other $\mathcal{U}_i$'s can be *unstable* – to make the overall scheme stable!

We previously discussed ADI as a method for solving the time-dependent heat-flow equation

$$\frac{\partial u}{\partial t} = \nabla^2 u - \rho \tag{17.6.7}$$

By letting $t \to \infty$ one also gets an iterative method for solving the elliptic equation

$$\nabla^2 u = \rho \tag{17.6.8}$$

In either case, the operator splitting is of the form

$$\mathcal{L} = \mathcal{L}_x + \mathcal{L}_y \tag{17.6.9}$$

where $\mathcal{L}_x$ represents the differencing in x and $\mathcal{L}_y$ that in y.

For example, in our model problem (17.0.6) with $\Delta x = \Delta y = \Delta$, we have

$$\begin{aligned} \mathcal{L}_x u &= 2u_{j,l} - u_{j+1,l} - u_{j-1,l} \\ \mathcal{L}_y u &= 2u_{j,l} - u_{j,l+1} - u_{j,l-1} \end{aligned} \tag{17.6.10}$$

More complicated operators may be similarly split, but there is some art involved. A bad choice of splitting can lead to an algorithm that fails to converge. Usually one tries to base the splitting on the physical nature of the problem. We know for our model problem that an initial transient diffuses away, and we set up the x and y splitting to mimic diffusion in each dimension.

Having chosen a splitting, we difference the time-dependent equation (17.6.7) implicitly in two half-steps:

$$\begin{aligned}\frac{u^{n+1/2}-u^n}{\Delta t/2} &= -\frac{\mathcal{L}_x u^{n+1/2}+\mathcal{L}_y u^n}{\Delta^2}-\rho \\ \frac{u^{n+1}-u^{n+1/2}}{\Delta t/2} &= -\frac{\mathcal{L}_x u^{n+1/2}+\mathcal{L}_y u^{n+1}}{\Delta^2}-\rho\end{aligned} \tag{17.6.11}$$

(cf. equation 17.3.16). Here we have suppressed the spatial indices (j,l). In matrix notation, equations (17.6.11) are

$$(\mathbf{L}_x + r\mathbf{1})\cdot \mathbf{u}^{n+1/2} = (r\mathbf{1}-\mathbf{L}_y)\cdot \mathbf{u}^n - \Delta^2\rho \tag{17.6.12}$$

$$(\mathbf{L}_y + r\mathbf{1})\cdot \mathbf{u}^{n+1} = (r\mathbf{1}-\mathbf{L}_x)\cdot \mathbf{u}^{n+1/2} - \Delta^2\rho \tag{17.6.13}$$

where

$$r \equiv \frac{2\Delta^2}{\Delta t} \tag{17.6.14}$$

The matrices on the left-hand sides of equations (17.6.12) and (17.6.13) are tridiagonal (and usually positive definite), so the equations can be solved by the standard tridiagonal algorithm. Given $\mathbf{u}^n$, one solves (17.6.12) for $\mathbf{u}^{n+1/2}$, substitutes on the right-hand side of (17.6.13), and then solves for $\mathbf{u}^{n+1}$. The key question is how to choose the iteration parameter r, the analog of a choice of timestep for an initial value problem.

As usual, we want to minimize the spectral radius of the iteration matrix. Although it is beyond our scope to go into details here, it turns out that, for the optimal choice of r, the ADI method has the same rate of convergence as does SOR (§17.5). Since the individual iteration steps in the ADI method are much more complicated than in SOR, the ADI method would appear to be inferior. This is certainly true if we choose the same parameter r for every iteration step. The trick is to choose a *different* r for each step.

The determination of an optimal sequence of r's for an arbitrary elliptic problem is an unsolved problem. The answer is known for a set of N iterations if the operators $\mathcal{L}_x$ and $\mathcal{L}_y$ have a common set of eigenvectors, and if the eigenvalues can be bounded by some $\lambda_{\min}$ and $\lambda_{\max}$. Even though this is a restrictive set of assumptions, in practice the same prescription often works satisfactorily on more general problems, so we present it here. Although there are general formulas (in terms of elliptic functions) for arbitrary N, usually we don't know N in advance. We therefore choose a sequence of N r_n's, and then repeat the cycle after N iterations. The formulas for r_n are simplest when N is a power of 2, $N = 2^k$ say, so we might as well choose N in this way.

Suppose the eigenvalues of $\mathcal{L}_x$ and $\mathcal{L}_y$ are bounded by the interval $[\alpha,\beta]$, where $0<\alpha<\beta$ since the operators are positive definite. The prescription is: Define

$$\alpha_0 = \alpha \quad , \quad \beta_0 = \beta \tag{17.6.15}$$

and recursively compute

$$\alpha_{j+1} = \sqrt{\alpha_j \beta_j}, \qquad \beta_{j+1} = \frac{\alpha_j+\beta_j}{2}, \qquad j = 0,1,\ldots,k-1 \tag{17.6.16}$$

Now set

$$s_1^{(0)} = \sqrt{\alpha_k \beta_k} \tag{17.6.17}$$

For each $j = 0,1,\ldots,k-1$, recursively compute $s_n^{(j+1)}$ for $n = 1,2,\ldots,2^{j+1}$ as the solutions of the following quadratic equation in x:

$$s_n^{(j)} = \frac{1}{2}\left(x + \frac{\alpha_{k-1-j}\beta_{k-1-j}}{x}\right), \qquad n = 1,2,\ldots,2^j \tag{17.6.18}$$

Here n labels the 2^{j+1} solutions x of equation (17.6.18) for each j. Finally, put

$$r_n = s_n^{(k)}, \qquad n = 1,2,\ldots,N = 2^k \tag{17.6.19}$$

The astute reader will recognize that this procedure is related to the arithmetic-geometric mean method for computing elliptic functions.

For the model problem (Poisson equation on a square with Dirichlet boundary conditions), bounds on the eigenvalues turn out to be

$$\begin{aligned}\alpha &= 2\left(1-\cos\frac{\pi}{J}\right), \\ \beta &= 2\left(1-\cos\frac{(J-1)\pi}{J}\right)\end{aligned} \tag{17.6.20}$$

One can show using induction that the above prescription leads to a spectral radius of the iteration matrix for a cycle of N iterations

$$\rho^{(N)}(ADI) \simeq 1 - 4\left(\frac{\pi}{4J}\right)^{1/k} \quad , \quad J\to\infty \tag{17.6.21}$$

This should be contrasted with equation (17.5.22). The k'th root in equation (17.6.21) leads to a dramatic increase in the rate of convergence.

For a more complicated problem, one can usually estimate bounds α and β by mapping the problem onto a known model problem, although as before the mapping must be onto a problem with the same types of boundary conditions. The result (17.6.20) holds for homogeneous Dirichlet or Neumann boundary conditions. For periodic boundary conditions, make the replacement $\pi \to 2\pi$. For an example where the eigenvalues of $\mathcal{L}_x$ are not the same as the eigenvalues of $\mathcal{L}_y$, see Spanier.

To minimize the total number of iterations, one should choose N to be a power of 2 close to $\approx \ln(4J/\pi)$, but the exact value is not crucial. Very roughly, the ADI method requires about $J/(8\log_{10} J)$ less operations than optimal SOR. Reductions in execution time of 6 are common for $J = 100$, while for $J = 1000$ a factor of 40 is possible.

In summary, the method of choice among relaxation methods is ADI. Occasionally there are problems for which ADI does not converge, in which case SOR with Chebyshev acceleration should be tried. SOR is much easier to program than ADI, and this accounts in part for its continued popularity. In an attempt to persuade you that ADI is not that much more complicated, we give below an ADI routine with $\mathcal{L}_x$ and $\mathcal{L}_y$ defined as the following slight generalization of equation (17.6.10):

$$\begin{aligned}(\mathcal{L}_x u)_{jl} &= a_{j,l}u_{j-1,l} + b_{j,l}u_{j,l} + c_{j,l}u_{j+1,l} \\ (\mathcal{L}_y u)_{jl} &= d_{j,l}u_{j,l-1} + e_{j,l}u_{j,l} + f_{j,l}u_{j,l+1}\end{aligned} \tag{17.6.22}$$

To cut down on the number of arithmetic operations per iteration, we organize the computation as follows:

- Initialize a vector ψ by

$$\psi = (r\mathbf{1} - \mathbf{L}_y)\cdot \mathbf{u}^0 \tag{17.6.23}$$

- For $n = 0, 1, \ldots$, solve

$$(\mathbf{L}_x + r\mathbf{1})\cdot \mathbf{u}^{n+1/2} = \psi - \rho\Delta^2 \tag{17.6.24}$$

  for $\mathbf{u}^{n+1/2}$.
- Set

$$\phi = -\psi + 2r\mathbf{u}^{n+1/2} \tag{17.6.25}$$

  (The quantity ϕ can overwrite ψ in storage.)
- Solve

$$(\mathbf{L}_y + r\mathbf{1})\cdot \mathbf{u}^{n+1} = \phi \tag{17.6.26}$$

  for $\mathbf{u}^{n+1}$
- If you have not yet converged, set

$$\psi = -\phi + 2r\mathbf{u}^{n+1} \tag{17.6.27}$$

  and continue from (17.6.24) again.

The ADI routine given below could be made even more efficient, but at the expense of readability. For typical compilers, the SOR routine given earlier is about the same speed as the ADI routine on the model problem for $J = 10$, a factor of 4 slower for $J = 40$, and a factor of 6.5 slower for $J = 100$. These numbers are in agreement with the theoretical estimates quoted earlier.

```
      SUBROUTINE ADI(A,B,C,D,E,F,G,U,JMAX,K,ALPHA,BETA,EPS)
        ADI solution of equations (17.6.12) and (17.6.13), with the operators as defined in equa-
        tion (17.6.22).  On input, A, B, C, D, E and F contain the coefficients of the equation.
        G contains the right-hand side, while U is input as the initial guess, usually zero.  All
        these arrays are dimensioned to the grid size, JMAX x JMAX.  The routine carries out 2**K
        iterations with different values of r, and then repeats.  ALPHA and BETA are user-supplied
        bounds for the eigenvalues of Lx and Ly, while EPS is the desired reduction in the norm of
        the residual.  Note that the routine as given requires a double precision version of TRIDAG
        from §2.6.
      IMPLICIT REAL*8(A-H,O-Z)      Double precision is a good idea for JMAX bigger than about 25.
      PARAMETER(JJ=100,KK=6,NRR=2**(KK-1),MAXITS=100,ZERO=0.D0,TWO=2.D0,HALF=.5D0)
      DIMENSION A(JMAX,JMAX),B(JMAX,JMAX),C(JMAX,JMAX),D(JMAX,JMAX),
     *    E(JMAX,JMAX),F(JMAX,JMAX),G(JMAX,JMAX),U(JMAX,JMAX),
     *    AA(JJ),BB(JJ),CC(JJ),RR(JJ),UU(JJ),PSI(JJ,JJ),
     *    ALPH(KK),BET(KK),R(NRR),S(NRR,KK)
      IF(JMAX.GT.JJ)PAUSE 'Increase JJ'
      IF(K.GT.KK-1)PAUSE 'Increase KK'
      K1=K+1
      NR=2**K
      ALPH(1)=ALPHA                 Determine r's from (17.6.15)-(17.6.19).
      BET(1)=BETA
      DO 11 J=1,K
        ALPH(J+1)=SQRT(ALPH(J)*BET(J))
        BET(J+1)=HALF*(ALPH(J)+BET(J))
11    CONTINUE
      S(1,1)=SQRT(ALPH(K1)*BET(K1))
      DO 13 J=1,K
        AB=ALPH(K1-J)*BET(K1-J)
        DO 12 N=1,2**(J-1)
          DISC=SQRT(S(N,J)**2-AB)
          S(2*N,J+1)=S(N,J)+DISC
          S(2*N-1,J+1)=AB/S(2*N,J+1)
12      CONTINUE
13    CONTINUE
      DO 14 N=1,NR
        R(N)=S(N,K1)
14    CONTINUE
      ANORMG=ZERO                   Compute initial residual, assuming U is zero.
      DO 16 J=2,JMAX-1
        DO 15 L=2,JMAX-1
```

```
            ANORMG=ANORMG+ABS(G(J,L))
            PSI(J,L)=-D(J,L)*U(J,L-1)+(R(1)-E(J,L))*U(J,L)
     *          -F(J,L)*U(J,L+1)          Equation (17.6.23).
          15 CONTINUE
        16 CONTINUE
      NITS=MAXITS/NR
      DO 27 KITS=1,NITS
        DO 24 N=1,NR                  Start cycle of 2**K iterations.
          IF(N.EQ.NR)THEN
            NEXT=1
          ELSE
            NEXT=N+1
          ENDIF
          RFACT=R(N)+R(NEXT)    This is "2r" in (17.6.27).
          DO 19 L=2,JMAX-1
            DO 17 J=2,JMAX-1  Solve (17.6.24).
              AA(J-1)=A(J,L)
              BB(J-1)=B(J,L)+R(N)
              CC(J-1)=C(J,L)
              RR(J-1)=PSI(J,L)-G(J,L)
            17 CONTINUE
            CALL TRIDAG(AA,BB,CC,RR,UU,JMAX-2)
            DO 18 J=2,JMAX-1
              PSI(J,L)=-PSI(J,L)+TWO*R(N)*UU(J-1)  Equation (17.6.25).
            18 CONTINUE
          19 CONTINUE
          DO 23 J=2,JMAX-1
            DO 21 L=2,JMAX-1  Solve (17.6.26).
              AA(L-1)=D(J,L)
              BB(L-1)=E(J,L)+R(N)
              CC(L-1)=F(J,L)
              RR(L-1)=PSI(J,L)
            21 CONTINUE
            CALL TRIDAG(AA,BB,CC,RR,UU,JMAX-2)
            DO 22 L=2,JMAX-1
              U(J,L)=UU(L-1)          Store current value of solution.
              PSI(J,L)=-PSI(J,L)+RFACT*UU(L-1)      Equation (17.6.27).
            22 CONTINUE
          23 CONTINUE
        24 CONTINUE
        ANORM=ZERO                    Check residual for convergence every 2**K iterations.
        DO 26 J=2,JMAX-1
          DO 25 L=2,JMAX-1
            RESID=A(J,L)*U(J-1,L)+(B(J,L)+E(J,L))*U(J,L)
     *            +C(J,L)*U(J+1,L)+D(J,L)*U(J,L-1)
     *            +F(J,L)*U(J,L+1)+G(J,L)
            ANORM=ANORM+ABS(RESID)
          25 CONTINUE
        26 CONTINUE
        IF(ANORM.LT.EPS*ANORMG)RETURN
      27 CONTINUE
      PAUSE 'MAXITS exceeded'
      END
```

REFERENCES AND FURTHER READING:

Spanier, J. 1967, in *Mathematical Methods for Digital Computers, Volume 2* (New York: John Wiley), Chapter 11.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §8.6.

Varga, R.S. 1962, *Matrix Iterative Analysis* (Englewood Cliffs, N.J.: Prentice-Hall).

Young, D.M. 1971, *Iterative Solution of Large Linear Systems* (New York: Academic Press).