Chapter 19. Partial Differential Equations

19.0 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, and so on. 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 [1-4] 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

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} \tag{19.0.1}$$

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

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial u}{\partial x}\right) \tag{19.0.2}$$

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

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \rho(x, y) \tag{19.0.3}$$

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 (19.0.1) and (19.0.2) both define *initial value* or *Cauchy* problems: If information on u (perhaps including time derivative information) is

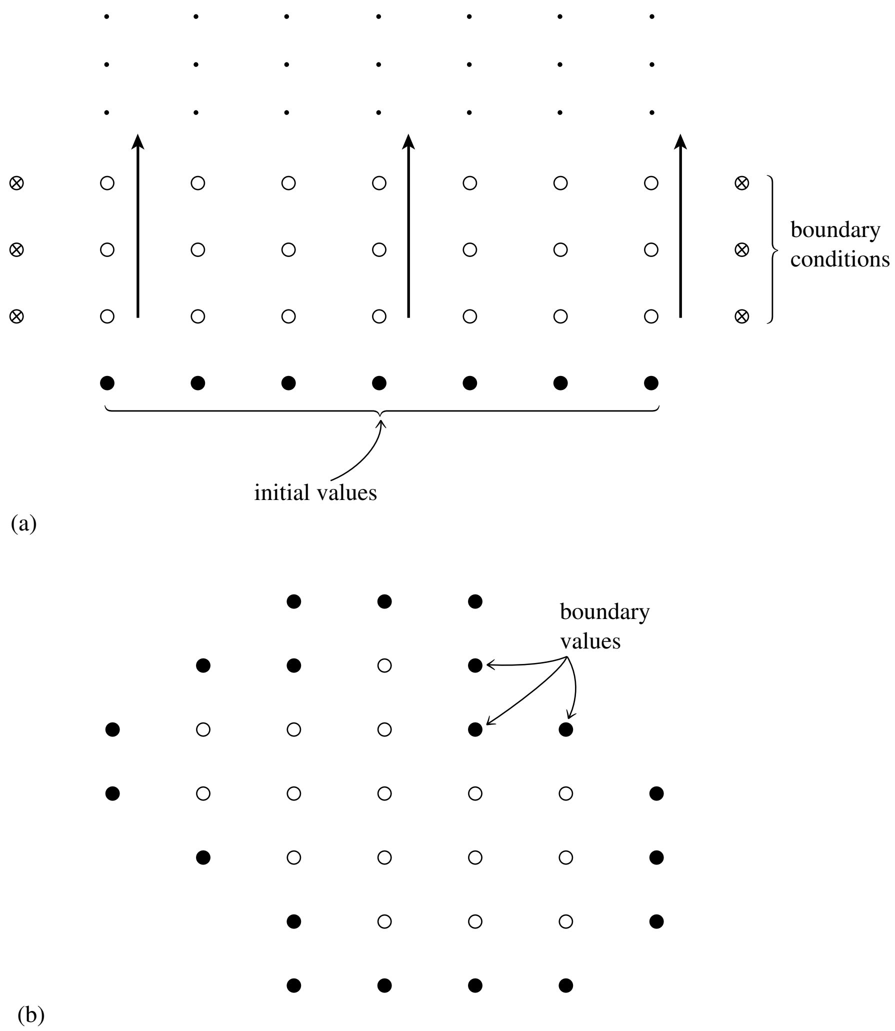

Figure 19.0.1. Initial value problem (a) and boundary value problem (b) are contrasted. In (a) initial values are given on one "time slice," and it is desired to advance the solution in time, computing successive rows of open dots in the direction shown by the arrows. Boundary conditions at the left and right edges of each row ($\otimes$) 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 grid points must be maintained in memory.

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 (19.0.1) and (19.0.2) describe time evolution. The goal of a numerical code should be to track that time evolution with some desired accuracy.

By contrast, equation (19.0.3) 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 19.0.1 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 19.1–19.3 of this chapter deal with initial value problems of several different forms. We make no pretence of completeness, but rather hope 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 §§19.4–19.6 as our "model problem," let us consider the solution of equation (19.0.3) by the *finite-difference method*. We represent the function $u(x, y)$ by its values at the discrete set of points

$$\begin{aligned} x_j &= x_0 + j\Delta, \qquad & j &= 0, 1, ..., J \\ y_l &= y_0 + l\Delta, \qquad & l &= 0, 1, ..., L \end{aligned} \tag{19.0.4}$$

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 (19.0.3) we substitute a finite-difference representation (see Figure 19.0.2),

$$\frac{u_{j+1,l} - 2u_{j,l} + u_{j-1,l}}{\Delta^2} + \frac{u_{j,l+1} - 2u_{j,l} + u_{j,l-1}}{\Delta^2} = \rho_{j,l} \tag{19.0.5}$$

or equivalently

$$u_{j+1,l} + u_{j-1,l} + u_{j,l+1} + u_{j,l-1} - 4u_{j,l} = \Delta^2 \rho_{j,l} \tag{19.0.6}$$

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

$$i \equiv j(L+1) + l \qquad \text{for} \qquad j = 0, 1, ..., J, \qquad l = 0, 1, ..., L \tag{19.0.7}$$

In other words, i increases most rapidly along the columns representing y values. Equation (19.0.6) now becomes

$$u_{i+L+1} + u_{i-(L+1)} + u_{i+1} + u_{i-1} - 4u_i = \Delta^2 \rho_i \tag{19.0.8}$$

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

The points where

$$\begin{aligned} j &= 0 \qquad & &[\text{i.e., } i = 0, ..., L] \\ j &= J \qquad & &[\text{i.e., } i = J(L+1), ..., J(L+1) + L] \\ l &= 0 \qquad & &[\text{i.e., } i = 0, L+1, ..., J(L+1)] \\ l &= L \qquad & &[\text{i.e., } i = L, L+1+L, ..., J(L+1) + L] \end{aligned} \tag{19.0.9}$$

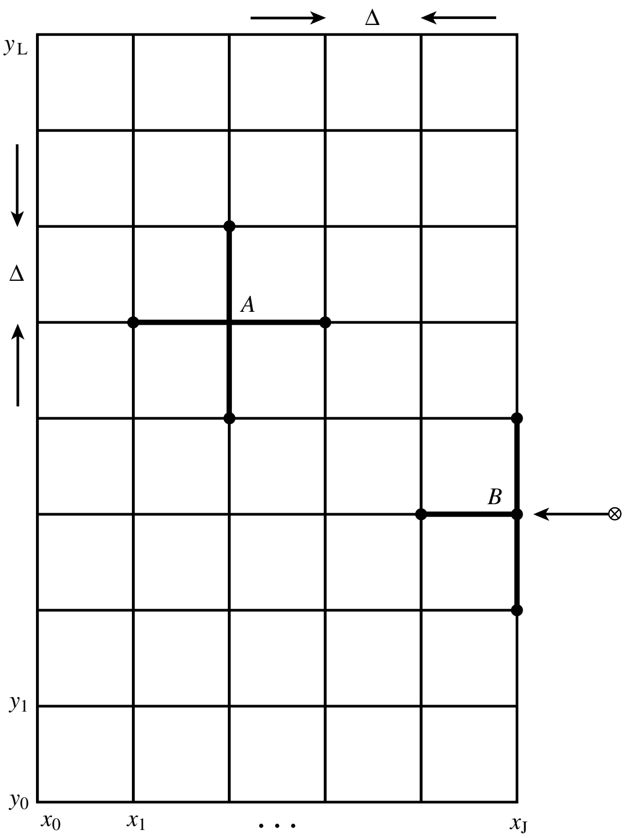

Figure 19.0.2. Finite-difference representation of a second-order elliptic equation on a two-dimensional grid. The second derivatives at the point A are evaluated using the points to which A is shown connected. The second derivatives at point B are evaluated using the connected points and also using "right-hand side" boundary information, shown schematically as $\otimes$.

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 (19.0.8), then the equation takes the form

$$\mathbf{A} \cdot \mathbf{u} = \mathbf{b} \tag{19.0.10}$$

where **A** has the form shown in Figure 19.0.3. The matrix **A** is called "tridiagonal with fringes." A general linear second-order elliptic equation

$$\begin{aligned} a(x, y)\frac{\partial^2 u}{\partial x^2} + b(x, y)\frac{\partial u}{\partial x} + c(x, y)\frac{\partial^2 u}{\partial y^2} + d(x, y)\frac{\partial u}{\partial y} \\ + e(x, y)\frac{\partial^2 u}{\partial x \partial y} + f(x, y)u = g(x, y) \end{aligned} \tag{19.0.11}$$

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 (19.0.10), not all applicable in all cases: relaxation methods, "rapid" methods (e.g., Fourier methods), and direct matrix methods.

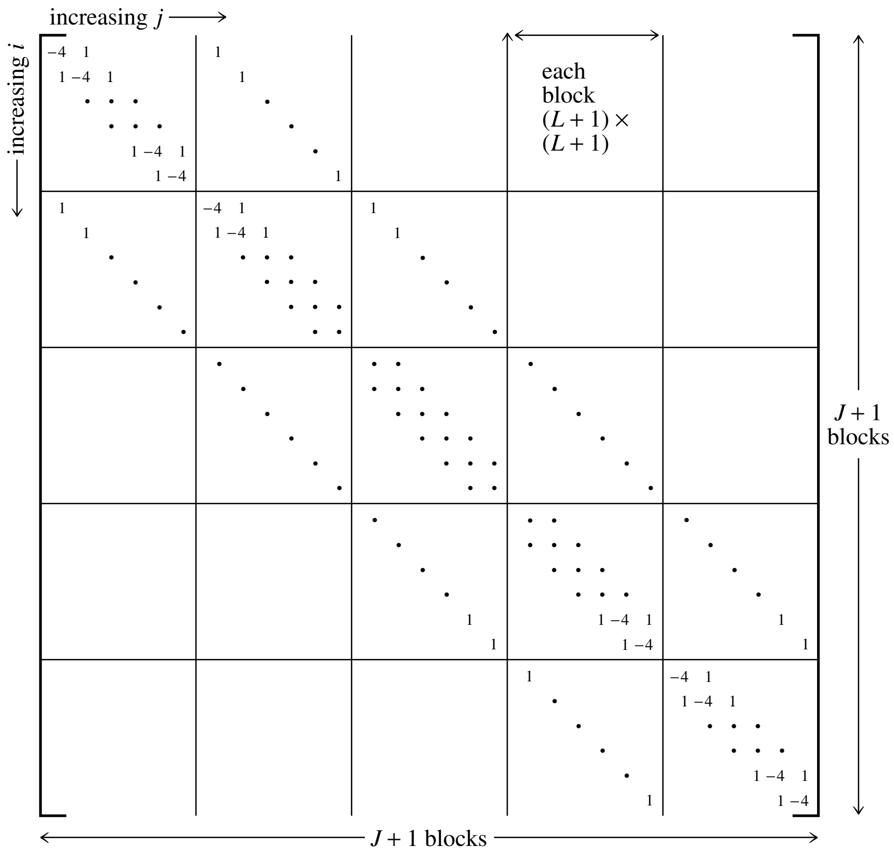

Figure 19.0.3. Matrix structure derived from a second-order elliptic equation (here equation 19.0.6). All elements not shown are zero. The matrix has diagonal blocks that are themselves tridiagonal, and sub- and super-diagonal blocks that are diagonal. This form is called “tridiagonal with fringes.” A matrix this sparse would never be stored in its full form as shown here.

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

$$\mathbf{A} = \mathbf{E} - \mathbf{F} \tag{19.0.12}$$

where **E** is easily invertible and **F** is the remainder. Then (19.0.10) becomes

$$\mathbf{E} \cdot \mathbf{u} = \mathbf{F} \cdot \mathbf{u} + \mathbf{b} \tag{19.0.13}$$

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

$$\mathbf{E} \cdot \mathbf{u}^{(r)} = \mathbf{F} \cdot \mathbf{u}^{(r-1)} + \mathbf{b} \tag{19.0.14}$$

Since **E** is chosen to be easily invertible, each iteration is fast. We will discuss relaxation methods in some detail in §19.5 and §19.6.

So-called rapid methods [5] 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 §19.4. Note, however, that the multigrid relaxation methods discussed in §19.6 can be faster than "rapid" methods.

Matrix methods attempt to solve the equation

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{19.0.15}$$

directly. The degree to which this is practical depends very strongly on the exact structure of the matrix $\mathbf{A}$ 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 $\mathbf{A}$, containing 10^8 elements!

As we discussed at the end of §2.7, if $\mathbf{A}$ 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 $\mathbf{A}$ is transformed to a matrix $\mathbf{A}'$ 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. In §2.7, in the routine `linbcg`, an analogous *preconditioner* was exploited for non-positive definite problems with the more general biconjugate gradient method. For the positive definite case that arises in PDEs, an example of a successful implementation is the *incomplete Cholesky conjugate gradient method (ICCG)* (see [6-8]).

Another method that relies on a transformation approach is the *strongly implicit procedure* of Stone [9]. A program called SIPSOL that implements this routine has been published [10].

A third class of matrix methods is the Analyze-Factorize-Operate approach as described in §2.7.

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. Only multigrid relaxation methods (§19.6) are competitive with the best matrix methods. For grids larger than, say, 300×300, 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. Finite element methods [11-12] 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 [13-15] are preferred for very regular geometries and smooth functions; they converge more rapidly than finite-difference methods (cf. §19.4), but they do not work well for problems with discontinuities.

CITED REFERENCES AND FURTHER READING:

Ames, W.F. 1977, *Numerical Methods for Partial Differential Equations*, 2nd ed. (New York: Academic Press). [1]

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

Roache, P.J. 1976, *Computational Fluid Dynamics* (Albuquerque: Hermosa). [3]

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]. [4]

Dorr, F.W. 1970, *SIAM Review*, vol. 12, pp. 248–263. [5]

Meijerink, J.A., and van der Vorst, H.A. 1977, *Mathematics of Computation*, vol. 31, pp. 148–162. [6]

van der Vorst, H.A. 1981, *Journal of Computational Physics*, vol. 44, pp. 1–19 [review of sparse iterative methods]. [7]

Kershaw, D.S. 1970, *Journal of Computational Physics*, vol. 26, pp. 43–65. [8]

Stone, H.J. 1968, *SIAM Journal on Numerical Analysis*, vol. 5, pp. 530–558. [9]

Jesshope, C.R. 1979, *Computer Physics Communications*, vol. 17, pp. 383–391. [10]

Strang, G., and Fix, G. 1973, *An Analysis of the Finite Element Method* (Englewood Cliffs, NJ: Prentice-Hall). [11]

Burnett, D.S. 1987, *Finite Element Analysis: From Concepts to Applications* (Reading, MA: Addison-Wesley). [12]

Gottlieb, D. and Orszag, S.A. 1977, *Numerical Analysis of Spectral Methods: Theory and Applications* (Philadelphia: S.I.A.M.). [13]

Canuto, C., Hussaini, M.Y., Quarteroni, A., and Zang, T.A. 1988, *Spectral Methods in Fluid Dynamics* (New York: Springer-Verlag). [14]

Boyd, J.P. 1989, *Chebyshev and Fourier Spectral Methods* (New York: Springer-Verlag). [15]

19.1 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*,

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\partial \mathbf{F}(\mathbf{u})}{\partial x} \tag{19.1.1}$$

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

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

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} \tag{19.1.2}$$