6 Boundary Value Problems

In Chapter 5, we examined methods for solving initial value problems. The solution of these equations depends on the nature of the equation and the initial conditions. In this chapter algorithms for solving certain boundary value problems and problems with both boundary and initial values are given. The solution of a boundary value problem in one independent variable must satisfy specified conditions at two points, and the solution of a boundary value problem in two independent variables must satisfy specified conditions along a curve or set of lines enclosing a specified region.

Although not considered in this chapter, a further important boundary value problem is one with three independent variables—for example, Laplace's equation in three dimensions. In this case the solution must satisfy specified conditions over a surface enclosing a specified volume. Note that in a mixed boundary and initial value problem one independent variable, usually time, will be associated with one or more initial values and the remaining independent variables will depend on boundary values.

6.1 Classification of Second-Order Partial Differential Equations

In this chapter we restrict the discussion to second-order differential equations in one or two independent variables; Figure 6.1 shows how these equations may be classified. The general form of these equations for one and two independent variables is given by (6.1) and (6.2), respectively.

$$A(x)\frac{d^2z}{dx^2}+f\left(x,z,\frac{dz}{dx}\right)=0 \tag{6.1}$$

$$A(x,y)\frac{\partial^2z}{\partial x^2}+B(x,y)\frac{\partial^2z}{\partial x\partial y}+C(x,y)\frac{\partial^2z}{\partial y^2}+f\left(x,y,z,\frac{\partial z}{\partial x},\frac{\partial z}{\partial y}\right)=0 \tag{6.2}$$

These equations are linear in the second-order terms, but the terms

$$f\left(x,z,\frac{dz}{dx}\right)\quad\text{and}\quad f\left(x,y,z,\frac{\partial z}{\partial x},\frac{\partial z}{\partial y}\right)$$

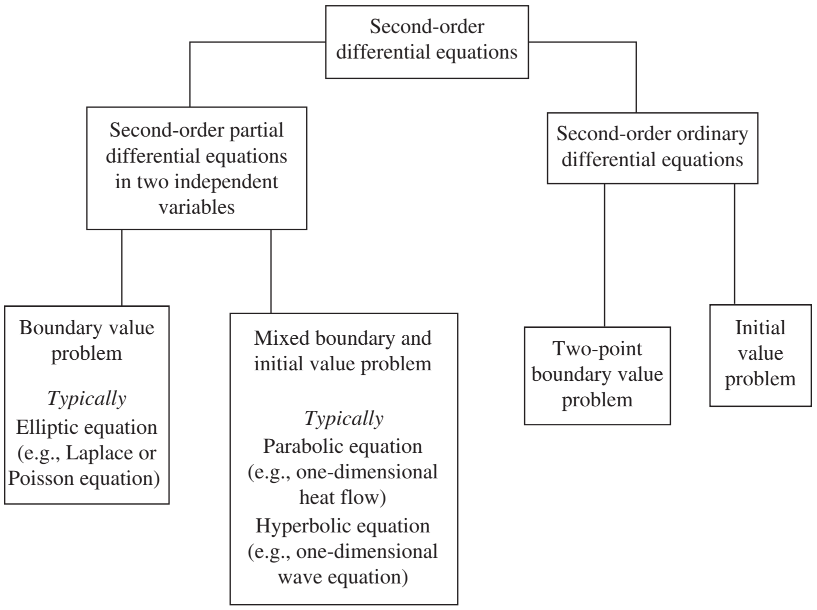
Second-order differential equations
Second-order partial differential equations in two independent variables
Second-order ordinary differential equations
Boundary value problem
Typically
Elliptic equation (e.g., Laplace or Poisson equation)
Mixed boundary and initial value problem
Typically
Parabolic equation (e.g., one-dimensional heat flow)
Hyperbolic equation (e.g., one-dimensional wave equation)
Two-point boundary value problem
Initial value problem

FIGURE 6.1 Second-order differential equations with one or two independent variables and their solutions.

may be linear or nonlinear. In particular, (6.2) is classified as an elliptic, parabolic, or hyperbolic partial differential equation as follows:

If $B^2 - 4AC < 0$, the equation is elliptic.

If $B^2 - 4AC = 0$, the equation is parabolic.

If $B^2 - 4AC > 0$, the equation is hyperbolic.

Since the coefficients A, B, and C are, in general, functions of the independent variables, the classification of (6.2) may vary in different regions of the domain in which the problem is defined. We will commence with a study of (6.1).

6.2 The Shooting Method

An initial value problem and a two-point boundary value problem derived from the same differential equation may have the same solution. For example, consider the differential equation

$$\frac{d^2y}{dx^2} + y = \cos 2x \tag{6.3}$$

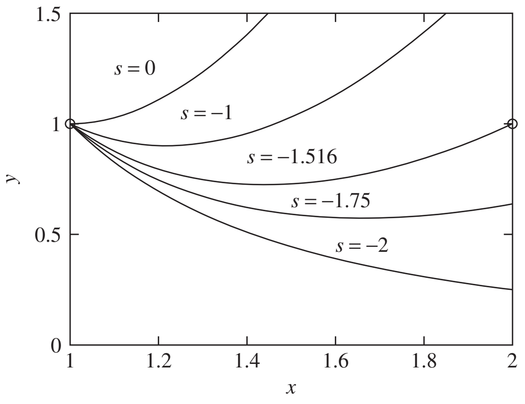

FIGURE 6.2 Solutions for $x^2(d^2y/dx^2) - 6y = 0$ with $y = 1$ and $dy/dx = s$ when $x = 1$, for trial values of s.

Given the initial conditions that when $x = 0$, then $y = 0$ and $dy/dx = 1$, the solution of (6.3) is

$$y = (\cos x - \cos 2x)\,/3 + \sin x$$

However, this solution also satisfies (6.3) with the two boundary conditions $x = 0$, $y = 0$ and $x = \pi/2$, $y = 4/3$.

This observation provides a useful method of solving two-point boundary value problems called "the shooting method." As an example, consider the equation

$$x^2 \frac{d^2y}{dx^2} - 6y = 0 \tag{6.4}$$

with boundary conditions $y = 1$ when $x = 1$ and $y = 1$ when $x = 2$. We will treat this problem as an initial value problem where $y = 1$ when $x = 1$ and assume trial values for dy/dx when $x = 1$, denoted by s. Figure 6.2 shows the solution for various trial values of s. When $s = -1.516$, the solution satisfies the required boundary condition that $y = 1$ when $x = 2$. The solution for (6.4) can be found by changing it into a pair of first-order differential equations and using any appropriate numerical method described in Chapter 5. Equation (6.4) is equivalent to

$$\begin{aligned} dy/dx &= z \\ dz/dx &= 6y/x^2 \end{aligned} \tag{6.5}$$

We must determine the slope dy/dx that gives the correct boundary condition. This could be achieved by trial and error, but this is tedious and in practice we can use interpolation. The following script solves (6.5) for four trial slopes using the MATLAB function `ode45`. Vector `s` contains trial values of the slope dy/dx at $x = 1$. Vector `b` contains the corresponding values of y when $x = 2$, computed by `ode45`. From these values of y we can interpolate to determine the value of s required to give $y = 1$ when $x = 2$. The interpolation is carried out using the function `aitken` (described in Chapter 7). Finally, this interpolated value of slope, `s0`, is used in `ode45` to determine the correct solution to (6.5).

```
% e3s601.m
f = @(x,y)[y(2); 6*y(1)/x^2];
option = odeset('RelTol',0.0005);
s = -1.25:-0.25:-2; s0 = [ ];
ncase = length(s); b = zeros(1,ncase);
for i = 1:ncase
    [x,y] = ode45(f,[1 2],[1 s(i)],option);
    [m,n] = size(y);
    b(1,i) = y(m,1);
end
s0 = aitken(b,s,1)
[x,y] = ode45(f,[1 2],[1 s0],option);
[x y(:,1)]
```

The right sides of the differential equations (6.5) are defined in the first line of this script. Running this script gives

```
s0 =
   -1.5161


ans =
    1.0000    1.0000
    1.0111    0.9836
    1.0221    0.9679
    1.0332    0.9529
    1.0442    0.9386
    1.0692    0.9084
    1.0942    0.8812
    1.1192
```

This output is very lengthy and part of it has therefore been deleted. The final stages are as follows:

```
              0.9293
    1.9442    0.9501
    1.9582    0.9622
    1.9721    0.9745
    1.9861    0.9871
    2.0000    1.0000
```

The interpolated value of the slope is -1.5161. The first column of `ans` gives the values of x and the second gives the corresponding values of y.

While the shooting method is not particularly efficient, it does have the advantage of being able to solve nonlinear boundary value problems. We now examine an alternative method for solving boundary problems: the finite difference method.

6.3 The Finite Difference Method

Chapter 4 shows how derivatives can be approximated by the use of finite differences. We can use the same approach for the solution of certain types of differential equations. The method effectively replaces the differential equation by a set of approximate difference equations. The central difference approximations for the first and second derivatives of z with respect to x are given by (6.6) and (6.7), which follow. In these and subsequent equations the operator D_x represents d/dx, $\mathrm{D}_x^2 = d^2/dx^2$, and so on. The subscript x is omitted where there is no danger of confusion. Thus at z_i,

$$\mathrm{D}z_i \approx (-z_{i-1} + z_{i+1})/\left(2h\right) \tag{6.6}$$

$$\mathrm{D}^2 z_i \approx (z_{i-1} - 2z_i + z_{i+1})/h^2 \tag{6.7}$$

In (6.6) and (6.7), h is the distance between the nodal points (see Figure 6.3) and these approximating formulae have errors of order h^2. Higher-order approximations can be generated that have errors of order h^4, but we do not require them. To achieve the same degree of accuracy we can make h smaller.

We can also determine the approximations for unevenly spaced nodal points. For example, it can be shown that (6.6) and (6.7) become

$$\mathrm{D}z_i \approx \frac{1}{h\beta(\beta+1)}\left\{-\beta^2 z_{i-1} - \left(1-\beta^2\right)z_i + z_{i+1}\right\} \tag{6.8}$$

$$\mathrm{D}^2 z_i \approx \frac{2}{h^2\beta(\beta+1)}\left\{\beta z_{i-1} - (1+\beta)z_i + z_{i+1}\right\} \tag{6.9}$$

where $h = x_i - x_{i-1}$ and $\beta h = x_{i+1} - x_i$. Note that when $\beta = 1$, (6.8) and (6.9) simplify to (6.6) and (6.7), respectively. Approximation (6.8) has an error of order h^2, regardless of the value of β, and (6.9) has an error of order h for $\beta \neq 1$ and h^2 for $\beta = 1$.

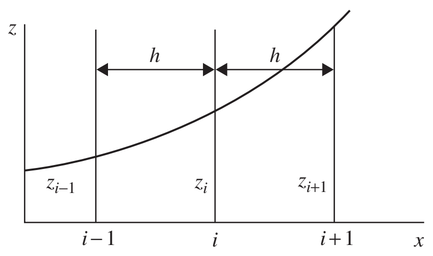

FIGURE 6.3 Equispaced nodal points.

Equations (6.6) through (6.9) are central difference approximations; that is, the approximation for a derivative uses values of the function on either side of the point at which the derivative is to be determined. These are generally the most accurate approximations, but in some situations it is necessary to use forward or backward difference approximations. For example, the forward difference approximation for $\mathrm{D}z_i$ is

$$\mathrm{D}z_i \approx \left(-z_i + z_{i+1}\right)/h \text{ with an error of order } h \tag{6.10}$$

The backward difference approximation for $\mathrm{D}z_i$ is

$$\mathrm{D}z_i \approx \left(-z_{i-1} + z_i\right)/h \text{ with an error of order } h \tag{6.11}$$

To determine solutions for partial differential equations we require the finite difference approximation for various partial derivatives in two or more variables. These approximations can be derived by combining some of the preceding equations. For example, we can determine the finite difference approximation for $\partial^2 z/\partial x^2 + \partial^2 z/\partial y^2$ (i.e., $\nabla^2 z$) from the approximation (6.7) or (6.9). To avoid double subscripts we use the notation applied to the mesh shown in Figure 6.4. Thus, from (6.7),

$$\nabla^2 z_i \approx (z_l - 2z_i + z_r)/h^2 + (z_a - 2z_i + z_b)/k^2 \approx \left\{r^2 z_l + r^2 z_r + z_a + z_b - 2\left(1 + r^2\right) z_i\right\} \Big/ \left(r^2 h^2\right) \tag{6.12}$$

where $r = k/h$. If $r = 1$, then (6.12) becomes

$$\nabla^2 z_i \approx (z_l + z_r + z_a + z_b - 4z_i)/h^2 \tag{6.13}$$

These central difference approximations for $\nabla^2 z_i$ have an error of $O(h^2)$.

The finite difference approximation for the second-order mixed derivative of z with respect to x and y, $\partial^2 z/\partial x \partial y$ or D_{xy}, is determined by applying (6.6) in the x direction to

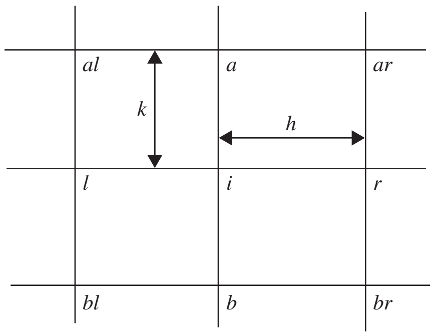

FIGURE 6.4 Grid mesh in rectangular coordinates.

each term of (6.6) in the y direction:

$$\mathrm{D}_{xy}z_i \approx \left[(z_r - z_l)_a/(2h) - (z_r - z_l)_b/(2h)\right]/(2k) \approx (z_{ar} - z_{al} - z_{br} + z_{bl})/(4hk) \tag{6.14}$$

We can develop the finite difference approximations in other coordinate systems such as skew and polar coordinates, and we can have uneven spacing of the node points in any direction; see Salvadori and Baron (1961).

6.4 Two-Point Boundary Value Problems

Before considering the application of finite difference methods to solve a differential equation, we first consider the nature of the solution. We begin by considering the following second-order inhomogeneous differential equation in one independent variable:

$$\left(1+x^2\right)\frac{d^2z}{dx^2} + x\frac{dz}{dx} - z = x^2 \tag{6.15}$$

subject to the boundary conditions $x = 0$, $z = 1$ and $x = 2$, $z = 2$. The solution of this equation is

$$z = -\frac{\sqrt{5}}{6}x + \frac{1}{3}\left(1+x^2\right)^{1/2} + \frac{1}{3}\left(2+x^2\right) \tag{6.16}$$

This is the only solution that satisfies both the equation and its boundary conditions. In contrast to this, consider the solution of the second-order homogeneous equation

$$x\frac{d^2z}{dx^2} + \frac{dz}{dx} + \lambda x^{-1}z = 0 \tag{6.17}$$

subject to the conditions that $z = 0$ at $x = 1$ and $dz/dx = 0$ at $x = e$ (where $e = 2.7183\ldots$). If λ is a given constant, this homogeneous equation has the trivial solution $z = 0$. However, if λ is an unknown, then we can determine values of λ to give nontrivial solutions for z. Equation (6.17) is then a characteristic value or eigenvalue problem. Solving (6.17) gives an infinite number of solutions for λ and z as follows:

$$z_n = \sin\left\{(2n+1)\frac{\pi}{2}\log_e|x|\right\},\ \lambda_n = \{(2n+1)\pi/2\}^2 \quad \text{where} \quad n = 0, 1, 2, \ldots \tag{6.18}$$

The values of λ that satisfy (6.18) are called characteristic values or eigenvalues, and the corresponding values of z are called characteristic functions or eigenfunctions. This particular type of boundary value problem is called a characteristic value or eigenvalue problem. It has arisen because both the differential equation and the specified boundary conditions are homogeneous.

The application of finite differences to the solution of boundary value problems is now illustrated by Examples 6.1 and 6.2.

Example 6.1

Determine an approximate solution for (6.15). We begin by multiplying (6.15) by $2h^2$ and writing d^2z/dx^2 as D^2z, and so on, to give

$$2(1+x^2)(h^2\mathrm{D}^2z)+xh\left(2h\mathrm{D}z\right)-2h^2z=2h^2x^2 \tag{6.19}$$

Using (6.6) and (6.7), we can replace (6.19) by

$$2\left(1+x_i^2\right)(z_{i-1}-2z_i+z_{i+1})+x_ih(-z_{i-1}+z_{i+1})-2h^2z_i=2h^2x_i^2 \tag{6.20}$$

Figure 6.5 shows x divided into four segments ($h=1/2$) with nodes numbered 1 to 5. Applying (6.20) to nodes 2, 3, and 4 gives

At node 2: $2(1+0.5^2)(z_1-2z_2+z_3)+0.25(-z_1+z_3)-0.5z_2=0.5(0.5^2)$

At node 3: $2(1+1.0^2)(z_2-2z_3+z_4)+0.50(-z_2+z_4)-0.5z_3=0.5(1.0^2)$

At node 4: $2(1+1.5^2)(z_3-2z_4+z_5)+0.75(-z_3+z_5)-0.5z_4=0.5(1.5^2)$

The problem boundary conditions are $x=0, z=1$ and $x=2, z=2$. Thus $z_1=1$ and $z_5=2$. Using these values, the preceding equations can be simplified and written in matrix form:

$$\begin{bmatrix}-44 & 22 & 0\\ 28 & -68 & 36\\ 0 & 46 & -108\end{bmatrix}\begin{bmatrix}z_2\\ z_3\\ z_4\end{bmatrix}=\begin{bmatrix}-17\\ 4\\ -107\end{bmatrix}$$

This equation system can easily be solved using MATLAB as follows:

```
>> A = [-44 22 0;28 -68 36;0 46 -108];
>> b = [-17 4 -107].';
>> y = A\b

y =
    0.9357
    1.0987
    1.4587
```

Note that the rows in the preceding matrix equation can always be scaled in order to make the coefficient matrix symmetrical. This is important in a large problem.

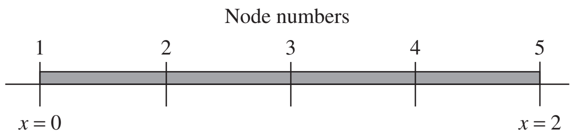

FIGURE 6.5 Node numbering used in the solution of (6.15).

To increase the accuracy of the solution we must increase the number of nodal points that consequently decrease h. However, formulating the finite difference approximation by hand for a large number of nodes is a tedious and error-prone process. The MATLAB function `twopoint` implements the process of solving the second-order boundary problem comprising differential equation (6.21) together with appropriate boundary conditions.

$$C(x)\frac{d^2z}{dx^2}+D(x)\frac{dz}{dx}+E(x)z=F(x) \tag{6.21}$$

The user must supply a vector listing the values of nodal points chosen. These do not have to be equispaced. The user must also supply vectors listing the values of $C(x)$, $D(x)$, $E(x)$, and $F(x)$ for the nodal points. Finally, the user must provide the boundary conditions, which can be in terms of either z or dz/dx.

```
function y = twopoint(x,C,D,E,F,flag1,flag2,p1,p2)
% Solves 2nd order boundary value problem
% Example call: y = twopoint(x,C,D,E,F,flag1,flag2,p1,p2)
% x is a row vector of n+1 nodal points.
% C, D, E and F are row vectors
% specifying C(x), D(x), E(x) and F(x).
% If y is specified at node 1, flag1 must equal 1.
% If y' is specified at node 1, flag1 must equal 0.
% If y is specified at node n+1, flag2 must equal 1.
% If y' is specified at node n+1, flag2 must equal 0.
% p1 & p2 are boundary values (y or y') at nodes 1 and n+1.
n = length(x)-1;
h(2:n+1) = x(2:n+1)-x(1:n);
h(1) = h(2); h(n+2) = h(n+1);
r(1:n+1) = h(2:n+2)./h(1:n+1);
s = 1+r;
if flag1==1
    y(1) = p1;
else
    slope0 = p1;
end
if flag2==1
    y(n+1) = p2;
else
    slopen = p2;
end
W = zeros(n+1,n+1);
if flag1==1
    c0 = 3;
    W(2,2) = E(2)-2*C(2)/(h(2)^2*r(2));
    W(2,3) = 2*C(2)/(h(2)^2*r(2)*s(2))+D(2)/(h(2)*s(2));
    b(2) = F(2)-y(1)*(2*C(2)/(h(2)^2*s(2))-D(2)/(h(2)*s(2)));
```

```
else
    c0=2;
    W(1,1) = E(1)-2*C(1)/(h(1)^2*r(1));
    W(1,2) = 2*C(1)*(1+1/r(1))/(h(1)^2*s(1));
    b(1) = F(1)+slope0*(2*C(1)/h(1)-D(1));
end
if flag2==1
    c1 = n-1;
    W(n,n) = E(n)-2*C(n)/(h(n)^2*r(n));
    W(n,n-1) = 2*C(n)/(h(n)^2*s(n))-D(n)/(h(n)*s(n));
    b(n) = F(n)-y(n+1)*(2*C(n)/(h(n)^2*s(n))+D(n)/(h(n)*s(n)));
else
    c1 = n;
    W(n+1,n+1) = E(n+1)-2*C(n+1)/(h(n+1)^2*r(n+1));
    W(n+1,n) = 2*C(n+1)*(1+1/r(n+1))/(h(n+1)^2*s(n+1));
    b(n+1) = F(n+1)-slopen*(2*C(n+1)/h(n+1)+D(n+1));
end
for i = c0:c1
    W(i,i) = E(i)-2*C(i)/(h(i)^2*r(i));
    W(i,i-1) = 2*C(i)/(h(i)^2*s(i))-D(i)/(h(i)*s(i));
    W(i,i+1) = 2*C(i)/(h(i)^2*r(i)*s(i))+D(i)/(h(i)*s(i));
    b(i) = F(i);
end
z = W(flag1+1:n+1-flag2,flag1+1:n+1-flag2)\b(flag1+1:n+1-flag2)';
if flag1==1 & flag2==1, y = [y(1); z; y(n+1)]; end
if flag1==1 & flag2==0, y = [y(1); z]; end
if flag1==0 & flag2==1, y = [z; y(n+1)]; end
if flag1==0 & flag2==0, y = z; end
```

We can use this function to solve (6.15) for nine nodes using the following script:

```
% e3s602.m
x = 0:.2:2;
C = 1+x.^2; D = x; E = -ones(1,11); F = x.^2;
flag1 = 1; p1 = 1; flag2 = 1; p2 = 2;
z = twopoint(x,C,D,E,F,flag1,flag2,p1,p2);
B = 1/3; A = -sqrt(5)*B/2;
xx = 0:.01:2;
zz = A*xx+B*sqrt(1+xx.^2)+B*(2+xx.^2);
plot(x,z,'o',xx,zz)
xlabel('x'); ylabel('z')
```

This script outputs the graph of Figure 6.6.

The results from the finite difference analysis are very accurate. This is because the solution of the boundary problem, given by (6.16), is well approximated by a low-order polynomial.

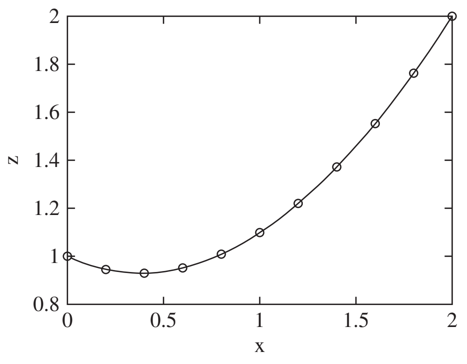

FIGURE 6.6 Finite difference solution of $(1+x^2)(d^2z/dx^2)+xdz/dx-z=x^2$. The $\circ$ indicates the finite difference estimate; the *continuous line* is the exact solution.

Example 6.2

Determine the approximate solution of (6.17) subject to the boundary conditions that $z=0$ at $x=1$ and $dz/dx=0$ at $x=e$. The exact eigensolutions are given by $\lambda_n=\{(2n+1)\pi/2\}^2$ and $z_n(x)=\sin\{(2n+1)(\pi/2)\log_e|x|\}$, where $n=0,1,\ldots,\infty$. We will use the node-numbering scheme shown in Figure 6.7. To apply the boundary condition at $x=e$ we must consider the finite difference approximation for Dz at node 5 (i.e., at $x=e$) and make D$z_5=0$. Applying (6.6), we have

$$2h\mathrm{D}z_5=-z_4+z_6=0 \tag{6.22}$$

Note that we have been forced to introduce a fictitious node, node 6. However, from (6.22), $z_6=z_4$.

Multiplying (6.17) by $2h^2$ gives

$$2x(h^2\mathrm{D}^2z)+h\left(2h\mathrm{D}z\right)=-\lambda 2x^{-1}h^2z$$

Thus,

$$2x_i\left(z_{i-1}-2z_i+z_{i+1}\right)+h\left(-z_{i-1}+z_{i+1}\right)=-\lambda 2x_i^{-1}h^2z_i$$

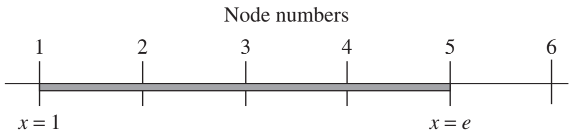

FIGURE 6.7 Node numbering used in the solution of (6.17).

Now $L = e - 1 = 1.7183$; thus $h = L/4 = 0.4296$. Applying (6.19) to nodes 2 through 5, we have

At node 2: $2(1.4296)(z_1 - 2z_2 + z_3) + 0.4296(-z_1 + z_3) = -2\lambda(1.4296)^{-1}(0.4296)^2 z_2$

At node 3: $2(1.8591)(z_2 - 2z_3 + z_4) + 0.4296(-z_2 + z_4) = -2\lambda(1.8591)^{-1}(0.4296)^2 z_3$

At node 4: $2(2.2887)(z_3 - 2z_4 + z_5) + 0.4296(-z_3 + z_5) = -2\lambda(2.2887)^{-1}(0.4296)^2 z_4$

At node 5: $2(2.7183)(z_4 - 2z_5 + z_6) + 0.4296(-z_4 + z_6) = -2\lambda(2.7183)^{-1}(0.4296)^2 z_5$

Letting $z_1 = 0$ and $z_6 = z_4$ leads to

$$\begin{bmatrix} -5.7184 & 3.2887 & 0 & 0 \\ 3.2887 & -7.4364 & 4.1478 & 0 \\ 0 & 4.1478 & -9.1548 & 5.0070 \\ 0 & 0 & 10.8731 & -10.8731 \end{bmatrix} \begin{bmatrix} z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}$$

$$= \lambda \begin{bmatrix} -0.2582 & 0 & 0 & 0 \\ 0 & -0.1985 & 0 & 0 \\ 0 & 0 & -0.1613 & 0 \\ 0 & 0 & 0 & -0.1358 \end{bmatrix} \begin{bmatrix} z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}$$

We can solve these equations using MATLAB as follows:

```
>> A = [-5.7184 3.2887 0 0;3.2887 -7.4364 4.1478 0;
   0 4.1478 -9.1548 5.0070; 0 0 10.8731 -10.8731];
>> B = diag([-0.2582 -0.1985 -0.1613 -0.1358]);
>> [u lambda] = eig(A,B)

u =
   -0.5424    1.0000   -0.4365    0.0169
   -0.8362    0.1389    1.0000   -0.1331
   -0.9686   -0.6793   -0.3173    0.5265
   -1.0000   -0.9112   -0.8839   -1.0000

lambda =
    2.5110         0         0         0
         0   20.3774         0         0
         0         0   51.3254         0
         0         0         0  122.2197
```

The exact values for the lowest four eigenvalues are 2.4674, 22.2066, 61.6850, and 120.9027. The graph of Figure 6.8 shows the first two eigenfunctions $z_0(x)$ and $z_1(x)$ and the estimates derived from the first and second columns of the preceding array u. Note that the values of u have been scaled to make those corresponding to the node z_5 either 1 or -1. The following script evaluates and plots the exact eigenfunctions $z_0(x)$ and $z_1(x)$ and plots the scaled sample points that estimate these functions.

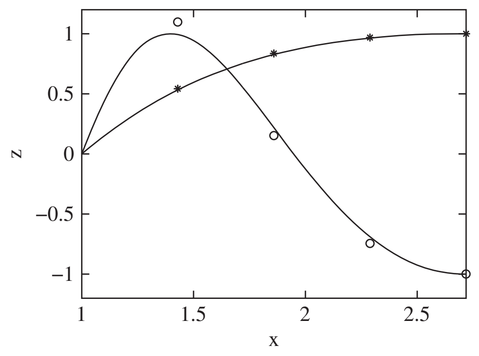

FIGURE 6.8 The finite difference estimates for the first (∗) and second (○) eigenfunctions of $x(d^2z/dx^2)+dz/dx+\lambda z/x=0$. *Solid lines* show the exact eigenfunctions $z_0(x)$ and $z_1(x)$.

```
% e3s603.m
x = 1:.01:exp(1);
% compute eigenfunction values scaled to 1 or -1.
z0 = sin((1*pi/2)*log(abs(x)));
z1 = sin((3*pi/2)*log(abs(x)));
% plot eigenfuctions
plot(x,z0,x,z1),  hold on
% Discrete approximations to eigenfunctions
% Scaled to 1 or -1.
u0 = [0.5424 0.8362 0.9686 1];
u1 = (1/0.9112)*[1 0.1389 -0.6793 -0.9112];
% determine x values for plotting
r = (exp(1)-1)/4;
xx = [1+r 1+2*r 1+3*r 1+4*r];
plot(xx,u0,'*',xx,u1,'o'),  hold off
axis([1 exp(1) -1.2 1.2])
xlabel('x'), ylabel('z')
```

6.5 Parabolic Partial Differential Equations

The general second-order partial differential equation in terms of the independent variables x and y is given by (6.2). The equation is repeated here, except that y has been replaced by t.

$$A(x,t)\frac{\partial^2 z}{\partial x^2}+B(x,t)\frac{\partial^2 z}{\partial x\partial t}+C(x,t)\frac{\partial^2 z}{\partial t^2}+f\left(x,t,z,\frac{\partial z}{\partial x},\frac{\partial z}{\partial t}\right)=0 \tag{6.23}$$

This equation will be a parabolic equation if $B^2 - 4AC = 0$. Parabolic equations are not defined in a closed domain, but propagate in an open domain. For example, the one-dimensional heat-flow equation, which describes heat flow assuming no energy generation, is

$$K\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}, \quad 0 < x < L \quad \text{and} \quad t > 0 \tag{6.24}$$

where K is the thermal diffusivity and u is the temperature of the material. Comparing (6.24) with (6.23), we see that A, B, and C of (6.23) are K, 0, and 0, respectively, so that the term $B^2 - 4AC$ is zero and the equation is parabolic.

To solve this equation, boundary conditions must be specified at $x = 0$ and $x = L$ and initial conditions when $t = 0$ must be given. To develop a finite difference solution we divide the spatial domain into n sections, each of length h, so that $h = L/n$, and consider as many time steps as required, each time step of duration k. A finite difference approximation for (6.24) at node (i,j) is obtained by replacing $\partial^2 u/\partial x^2$ by the central difference approximation (6.7) and $\partial u/\partial t$ by the forward difference approximation (6.10) to give

$$K\left(\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}\right) = \left(\frac{-u_{i,j} + u_{i,j+1}}{k}\right) \tag{6.25}$$

or

$$u_{i,j+1} = u_{i,j} + \alpha(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}), \quad i = 0, 1, \ldots, n; \quad j = 0, 1, \ldots \tag{6.26}$$

In (6.26), $\alpha = Kk/h^2$. Node (i,j) is the point $x = ih$ at time jk. Equation (6.26) allows us to determine $u_{i,\,j+1}$, that is, u at time $j+1$ from values of u at time j. Values of $u_{i,0}$ are provided by the initial conditions; values of $u_{0,\,j}$ and $u_{n,\,j}$ are obtained from the boundary conditions. This method of solution is called the explicit method.

In the numeric solution of parabolic partial differential equations, solution stability and convergence are important. It can be proved that when using the explicit method we must make $\alpha \le 0.5$ to ensure a steady decay of the entire solution. This requirement means that the grid separation in time must sometimes be very small, necessitating a very large number of time steps.

An alternative finite difference approximation for (6.24) is obtained by considering node $(i, j+1)$. We again approximate $\partial^2 u/\partial x^2$ by the central difference approximation (6.7), but we approximate $\partial u/\partial t$ by the backward difference approximation (6.11) to give

$$K\left(\frac{u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}}{h^2}\right) = \left(\frac{-u_{i,j} + u_{i,j+1}}{k}\right) \tag{6.27}$$

This equation is identical to (6.25) except that approximation is made at the $(j+1)$th time step instead of at the jth time step. Rearranging (6.27) with $\alpha = Kk/h^2$ gives

$$(1 + 2\alpha)u_{i,j+1} - \alpha(u_{i+1,j+1} + u_{i-1,j+1}) = u_{i,j} \tag{6.28}$$

where $i = 0, 1, \ldots, n$; $j = 0, 1, \ldots$. The three variables on the left side of this equation are unknown. However, if we have a grid of $n+1$ spatial points, then at time $j+1$ there are $n-1$ unknown nodal values and two known boundary values. We can assemble the set of $n-1$ equations of the form of (6.28) as follows:

$$\begin{bmatrix} \gamma & -\alpha & 0 & \cdots & 0 \\ -\alpha & \gamma & -\alpha & \cdots & 0 \\ 0 & -\alpha & \gamma & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & -\alpha \\ 0 & 0 & 0 & \cdots & \gamma \end{bmatrix} \begin{bmatrix} u_{1,j+1} \\ u_{2,j+1} \\ u_{3,j+1} \\ \vdots \\ u_{n-2,j+1} \\ u_{n-1,j+1} \end{bmatrix} = \begin{bmatrix} u_{1,j} + \alpha u_0 \\ u_{2,j} \\ u_{3,j} \\ \vdots \\ u_{n-2,j} \\ u_{n-1,j} + \alpha u_n \end{bmatrix}$$

where $\gamma = 1 + 2\alpha$. Note that u_0 and u_n are the known boundary conditions, assumed to be independent of time. By solving the preceding equation system, we determine $u_1, u_2, \ldots, u_{n-1}$ at time step $j+1$ from $u_1, u_2, \ldots, u_{n-1}$ at time step j. This approach is called the implicit method. Compared with the explicit method, each time step requires more computation; however, the method has the significant advantage that it is unconditionally stable. However, although stability does not place any restriction on α, h and k must be chosen to keep the discretization error small to maintain accuracy.

The following function `heat` implements an implicit finite difference solution for the parabolic differential equation (6.24).

```
function [u alpha] = heat(nx,hx,nt,ht,init,lowb,hib,K)
% Solves parabolic equ'n.
% e.g. heat flow equation.
% Example call: [u alpha] = heat(nx,hx,nt,ht,init,lowb,hib,K)
% nx, hx are number and size of x panels
% nt, ht are number and size of t panels
% init is a row vector of nx+1 initial values of the function.
% lowb & hib are boundaries at low and hi values of x.
% Note that lowb and hib are scalar values.
% K is a constant in the parabolic equation.
alpha = K*ht/hx^2;
A = zeros(nx-1,nx-1); u = zeros(nt+1,nx+1);
u(:,1) = lowb*ones(nt+1,1);
u(:,nx+1) = hib*ones(nt+1,1);
u(1,:) = init;
A(1,1) = 1+2*alpha; A(1,2) = -alpha;
for i = 2:nx-2
    A(i,i) = 1+2*alpha;
    A(i,i-1) = -alpha; A(i,i+1) = -alpha;
end
```

```
A(nx-1,nx-2) = -alpha; A(nx-1,nx-1) = 1+2*alpha;
b(1,1) = init(2)+init(1)*alpha;
for i = 2:nx-2, b(i,1) = init(i+1); end
b(nx-1,1) = init(nx)+init(nx+1)*alpha;
[L,U] = lu(A);
for j = 2:nt+1
    y = L\b; x = U\y;
    u(j,2:nx) = x'; b = x;
    b(1,1) = b(1,1)+lowb*alpha;
    b(nx-1,1) = b(nx-1,1)+hib*alpha;
end
```

We now use the function `heat` to study how the temperature distribution in a brick wall varies with time. The wall is 0.3 m thick and is initially at a uniform temperature of 100°C. For the brickwork, $K = 5 \times 10^{-7}\,\mathrm{m/s^2}$. If the temperature of both surfaces is suddenly lowered to 20°C and kept at this temperature, we wish to plot the subsequent variation of temperature through the wall at 440 s (7.33 min) intervals for 22,000 s (366.67 min).

To study this problem we will use a mesh with 15 subdivisions of x and 50 subdivisions of t.

```
% e3s604.m
K = 5e-7; thick = 0.3; tfinal = 22000;
nx = 15; hx = thick/nx;
nt = 50; ht = tfinal/nt;
init = 100*ones(1,nx+1); lowb = 20; hib = 20;
[u al] = heat(nx,hx,nt,ht,init,lowb,hib,K);
alpha = al, surfl(u)
axis([0 nx+1 0 nt+1 0 120])
view([-217 30]), xlabel('x - node nos.')
ylabel('Time - node nos.'), zlabel('Temperature')
```

Running this script gives

```
alpha =
    0.5500
```

together with the plot shown in Figure 6.9.

The plot shows how the temperature across the wall decreases with time. Figure 6.10 shows the variation of temperature with time at the center of the wall, calculated by both the implicit method (using the MATLAB function `heat`) and the explicit method using the same mesh size. In the latter case a MATLAB function is not provided. The solution determined using the explicit method becomes unstable with increasing time. We expect this because the mesh size has been chosen to make $\alpha = 0.55$.

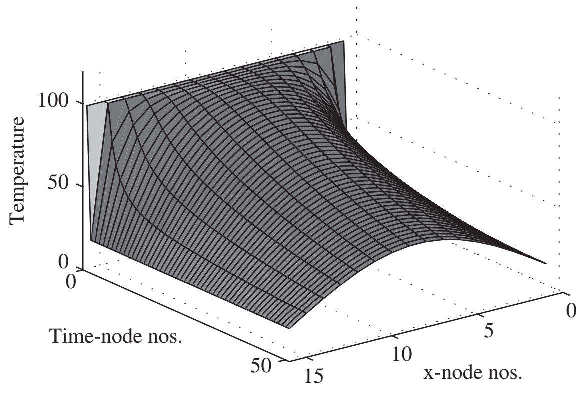

FIGURE 6.9 Plot showing how the distribution of temperature through a wall varies with time.

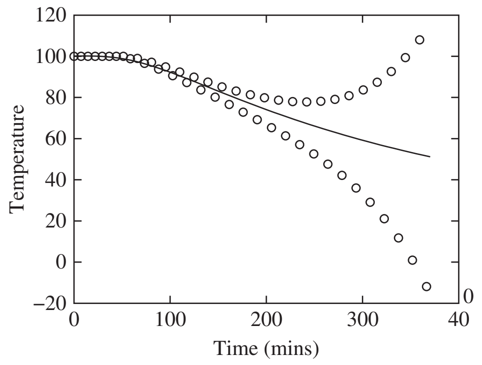

FIGURE 6.10 Variation in temperature in the center of a wall. The steadily decaying solution was generated using the implicit method of solution; the oscillating solution was generated using the explicit method of solution.

6.6 Hyperbolic Partial Differential Equations

Consider the following equation:

$$c^2\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2}, \quad 0 < x < L \quad \text{and} \quad t > 0 \tag{6.29}$$

This is the one-dimensional wave equation, and like the heat-flow problem of Section 6.5, its solution usually propagates in an open domain. Equation (6.29) describes the wave in a taut string where c is the velocity of propagation of the waves in the string. Comparing (6.29) with (6.23), we see that $B^2 - 4AC = -4c^2(-1)$. Since c^2 is positive, $B^2 - 4AC > 0$ and the equation is hyperbolic. Equation (6.29) is subject to boundary conditions at $x = 0$ and $x = L$ and also subject to initial conditions when $t = 0$.

We now develop equivalent finite difference approximations for these equations. Divide L into n sections so that $h = L/n$ and consider time steps of duration k. Approximating (6.29) by central finite difference approximations based on (6.7) at node (i,j), we have

$$c^2\left(\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}\right) = \left(\frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2}\right)$$

or

$$\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right) - (1/\alpha^2)\left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right) = 0$$

where $\alpha^2 = c^2k^2/h^2$, $i = 0, 1, \ldots, n$, and $j = 0, 1, \ldots$. Node (i,j) is the point $x = ih$ at time $t = jk$. Rearranging the preceding equation gives

$$u_{i,j+1} = \alpha^2\left(u_{i-1,j} + u_{i+1,j}\right) + 2(1-\alpha^2)u_{i,j} - u_{i,j-1} \tag{6.30}$$

When $j = 0$, equation (6.30) becomes

$$u_{i,\,1} = \alpha^2\left(u_{i-1,\,0} + u_{i+1,\,0}\right) + 2(1-\alpha^2)u_{i,\,0} - u_{i,\,-1} \tag{6.31}$$

To solve a hyperbolic partial differential equation, initial values of $u(x)$ and $\partial u/\partial t$ must be specified. Let these values be U_i and V_i, respectively, where $i = 0, 1, \ldots, n$. We can replace $\partial u/\partial t$ by its central finite difference approximation based on (6.6) as follows:

$$V_i = (-u_{i,-1} + u_{i,1})/(2k)$$

Thus,

$$-u_{i,-1} = 2kV_i - u_{i,1} \tag{6.32}$$

In (6.31) we replace $u_{i,0}$ by U_i and $u_{i,-1}$ by using (6.32) to give

$$u_{i,1} = \alpha^2\left(U_{i-1} + U_{i+1}\right) + 2(1-\alpha^2)U_i + 2kV_i - u_{i,1}$$

so that

$$u_{i,1} = \alpha^2\left(U_{i-1} + U_{i+1}\right)/2 + (1-\alpha^2)U_i + kV_i \tag{6.33}$$

Equation (6.33) is the starting equation and allows us to determine the values of u at time step $j = 1$. Once we obtain these values, we can use (6.30) to provide an explicit method of solution. In order to ensure stability, the parameter α should be equal to or less than one. However, if α is less than one, the solution becomes less accurate.

The following function, `fwave`, implements an explicit finite difference solution for (6.29).

```
function [u alpha] = fwave(nx,hx,nt,ht,init,initslope,lowb,hib,c)
% Solves hyperbolic equ'n, e.g. wave equation.
% Example: [u alpha] = fwave(nx,hx,nt,ht,init,initslope,lowb,hib,c)
% nx, hx are number and size of x panels
% nt, ht are number and size of t panels
% init is a row vector of nx+1 initial values of the function.
% initslope is a row vector of nx+1 initial derivatives of
% the function.
% lowb is a column vector of nt+1 boundary values at the
% low value of x.
% hib is a column vector of nt+1 boundary values at hi value of x.
% c is a constant in the hyperbolic equation.
alpha = c*ht/hx;
u = zeros(nt+1,nx+1);
u(:,1) = lowb; u(:,nx+1) = hib; u(1,:) = init;
for i = 2:nx
    u(2,i) = alpha^2*(init(i+1)+init(i-1))/2+(1-alpha^2)*init(i) ...
    +ht*initslope(i);
end
for j = 2:nt
    for i = 2:nx
        u(j+1,i)=alpha^2*(u(j,i+1)+u(j,i-1))+(2-2*alpha^2)*u(j,i) ...
        -u(j-1,i);
    end
end
```

We now use the `fwave` function to examine the effect of displacing the boundary at one end of a taut string by 10 units in a positive direction for the time period $t = 0.1$ to $t = 4$ units.

```
% e3s605.m
T = 4; L = 1.6;
nx = 16; nt = 40; hx = L/nx; ht = T/nt;
c = 1; t = 0:nt;
hib = zeros(nt+1,1); lowb = zeros(nt+1,1);
lowb(2:5,1) = 10;
init = zeros(1,nx+1); initslope = zeros(1,nx+1);
[u al] = fwave(nx,hx,nt,ht,init,initslope,lowb,hib,c);
alpha = al, surfl(u)
axis([0 16 0 40 -10 10])
xlabel('Position along string')
ylabel('Time'), zlabel('Vertical displacement')
```

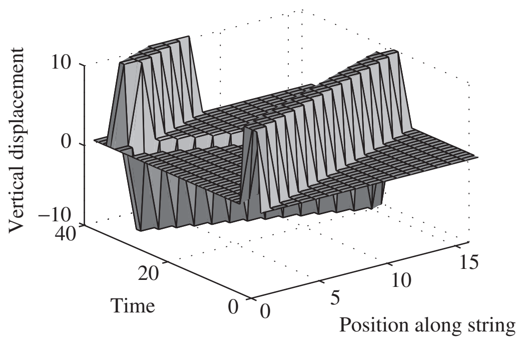

FIGURE 6.11 Solution of (6.29) subject to specific boundary and initial conditions.

Running this script produces the following output, together with Figure 6.11.

```
alpha =
     1
```

The figure shows that the disturbance at the boundary travels along the string. At the other boundary, it is reflected and becomes a negative disturbance. This process of reflection and reversal continues at each boundary. The disturbance travels at a velocity c and its shape does not change. Similarly, pressure fluctuations do not change as they travel along a speaking tube; if pressure fluctuations representing the sound "HELLO" enter the tube, the sound "HELLO" is detected at the other end. In practice, energy loss, which is not included in this model, would cause the amplitude of the disturbance to decay to zero over a period of time.

6.7 Elliptic Partial Differential Equations

The solution of a second-order elliptic partial differential equation is determined over a closed region, and the shape of the boundary and its condition at every point must be specified. Some important second-order elliptic partial differential equations, which arise naturally in the description of physical systems, are

$$\text{Laplace's equation: } \nabla^2 z = 0 \tag{6.34}$$

$$\text{Poisson's equation: } \nabla^2 z = F(x,y) \tag{6.35}$$

$$\text{Helmholtz's equation: } \nabla^2 z + G(x,y)z = F(x,y) \tag{6.36}$$

where $\nabla^2 z = \partial^2 z/\partial x^2 + \partial^2 z/\partial y^2$ and $z(x,y)$ is an unknown function. Note that the Laplace and Poisson equations are special cases of Helmholtz's equation. In general, these equations must satisfy boundary conditions that are specified in terms of either the function value or the derivative of the function normal to the boundary. Furthermore, a problem can have mixed boundary conditions. If we compare (6.34), (6.35), and (6.36) to the standard second-order partial differential equation in two variables, that is,

$$A(x,y)\frac{\partial^2 z}{\partial x^2} + B(x,y)\frac{\partial^2 z}{\partial x \partial y} + C(x,y)\frac{\partial^2 z}{\partial y^2} + f\left(x, y, z, \frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}\right) = 0$$

we see that in each case $A = C = 1$ and $B = 0$, so that $B^2 - 4AC < 0$, confirming that the equations are elliptic.

The Laplace equation is homogeneous, and if a problem has boundary conditions that are also homogeneous then the solution, $z = 0$, will be trivial. Similarly in (6.35), if $F(x,y) = 0$ and the problem boundary conditions are homogeneous, then $z = 0$. However, in (6.36) we can scale $G(x,y)$ by a factor λ, so that (6.36) becomes

$$\nabla^2 z + \lambda G(x,y) z = 0 \qquad (6.37)$$

This is a characteristic or eigenvalue problem, and we can determine values of λ and corresponding nontrivial values of $z(x,y)$.

The elliptic equations (6.34) through (6.37) can only be solved in a closed form for a limited number of situations. For most problems, it is necessary to use a numerical approximation. Finite difference methods are relatively simple to apply, particularly for rectangular regions. We will now use the finite difference approximation for $\nabla^2 z$, given by (6.12) or (6.13), to solve some elliptic partial differential equations over a rectangular domain.

Example 6.3

Laplace's equation. Determine the distribution of temperature in a rectangular plane section, subject to a temperature distribution around its edges as follows:

$$x = 0, T = 100y; \quad x = 3, T = 250y; \quad y = 0, T = 0; \quad \text{and} \quad y = 2, T = 200 + (100/3)x^2$$

The section shape, the boundary temperature distribution section and two chosen nodes are shown in Figure 6.12.

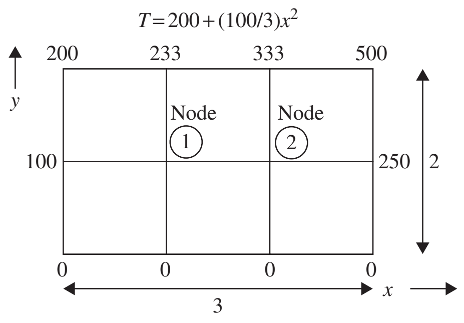

FIGURE 6.12 Temperature distribution around a plane section. Locations of nodes 1 and 2 are shown.

The temperature distribution is described by Laplace's equation. Solving this equation by the finite difference method, we apply (6.13) to nodes 1 and 2 of the mesh shown in Figure 6.12. This gives

$$(233.33 + T_2 + 0 + 100 - 4T_1)/h^2 = 0$$
$$(333.33 + 250 + 0 + T_1 - 4T_2)/h^2 = 0$$

where T_1 and T_2 are the unknown temperatures at nodes 1 and 2, respectively, and $h = 1$. Rearranging these equations gives

$$\begin{bmatrix} -4 & 1 \\ 1 & -4 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} -333.33 \\ -583.33 \end{bmatrix}$$

Solving this equation, we have $T_1 = 127.78$ and $T_2 = 177.78$.

If we require a more accurate solution of Laplace's equation, then we must use more nodes and the computation burden increases rapidly. The following MATLAB function `ellipgen` uses the finite difference approximation (6.12) to solve the general elliptic partial differential equations (6.34) through (6.37) for a rectangular domain only. The function is also limited to problems in which the boundary value is specified by values of the function $z(x,y)$, not its derivative. If the user calls the function with 10 arguments, the function solves (6.34) through (6.36); see Examples 6.4 and 6.5. Calling it with six arguments causes it to solve (6.37); see Example 6.6.

```
function [a,om] = ellipgen(nx,hx,ny,hy,G,F,bx0,bxn,by0,byn)
% Function either solves:
% nabla^2(z)+G(x,y)*z = F(x,y) over a rectangular region.
% Function call: [a,om]=ellipgen(nx,hx,ny,hy,G,F,bx0,bxn,by0,byn)
% hx, hy are panel sizes in x and y directions,
% nx, ny are number of panels in x and y directions.
% F and G are (nx+1,ny+1) arrays representing F(x,y), G(x,y).
% bx0 and bxn are row vectors of boundary conditions at x0 and xn
% each beginning at y0. Each is (ny+1) elements.
% by0 and byn are row vectors of boundary conditions at y0 and yn
% each beginning at x0. Each is (nx+1) elements.
% a is an (nx+1,ny+1) array of sol'ns, inc the boundary values.
% om has no interpretation in this case.
% or the function solves
% (nabla^2)z+lambda*G(x,y)*z = 0 over a rectangular region.
% Function call: [a,om]=ellipgen(nx,hx,ny,hy,G,F)
% hx, hy are panel sizes in x and y directions,
% nx, ny are number of panels in x and y directions.
% G are (ny+1,nx+1) arrays representing G(x,y).
% In this case F is a scalar and specifies the
% eigenvector to be returned in array a.
% Array a is an (ny+1,nx+1) array giving an eigenvector,
% including the boundary values.
% The vector om lists all the eigenvalues lambda.
nmax = (nx-1)*(ny-1); r = hy/hx;
a = zeros(ny+1,nx+1); p = zeros(ny+1,nx+1);
if nargin==6
    ncase = 0; mode = F;
end
if nargin==10
    test = 0;
    if F==zeros(nx+1,ny+1), test = 1; end
    if bx0==zeros(1,ny+1), test = test+1; end
    if bxn==zeros(1,ny+1), test = test+1; end
    if by0==zeros(1,nx+1), test = test+1; end
    if byn==zeros(1,nx+1), test = test+1; end
    if test==5
        disp('WARNING - problem has trivial solution, z = 0.')
        disp('To obtain eigensolution use 6 parameters only.')
        return
    end
    bx0 = bx0(1,ny+1:-1:1); bxn = bxn(1,ny+1:-1:1);
    a(1,:) = byn; a(ny+1,:) = by0;
    a(:,1) = bx0'; a(:,nx+1) = bxn'; ncase = 1;
end
for i = 2:ny
    for j = 2:nx
        nn = (i-2)*(nx-1)+(j-1);
        q(nn,1) = i; q(nn,2) = j; p(i,j) = nn;
    end
end
C = zeros(nmax,nmax); e = zeros(nmax,1); om = zeros(nmax,1);
if ncase==1, g = zeros(nmax,1); end
for i = 2:ny
    for j = 2:nx
        nn = p(i,j); C(nn,nn) = -(2+2*r^2); e(nn) = hy^2*G(j,i);
        if ncase==1, g(nn) = g(nn)+hy^2*F(j,i); end
        if p(i+1,j)~=0
            np = p(i+1,j); C(nn,np) = 1;
        else
            if ncase==1, g(nn) = g(nn)-by0(j); end
        end
        if p(i-1,j)~=0
            np = p(i-1,j); C(nn,np) = 1;
        else
            if ncase==1, g(nn) = g(nn)-byn(j); end
        end
            if p(i,j+1)~=0
                np = p(i,j+1); C(nn,np) = r^2;
            else
            if ncase==1, g(nn) = g(nn)-r^2*bxn(i); end
        end
        if p(i,j-1)~=0
            np = p(i,j-1); C(nn,np) = r^2;
        else
            if ncase==1, g(nn) = g(nn)-r^2*bx0(i); end
        end
    end
end
if ncase==1
    C = C+diag(e); z = C\g;
    for nn = 1:nmax
        i = q(nn,1); j = q(nn,2); a(i,j) = z(nn);
    end
else
    [u,lam] = eig(C,-diag(e));
    [om,k] = sort(diag(lam)); u = u(:,k);
    for nn = 1:nmax
        i = q(nn,1); j = q(nn,2);
        a(i,j) = u(nn,mode);
    end
end
```

We now give examples of the application of the `ellipgen` function.

Example 6.4

Use the function `ellipgen` to solve Laplace's equation over a rectangular region subject to the boundary conditions shown in Figure 6.12. The following script calls the function to solve this problem using a 12 $\times$ 12 mesh. The example is the same as Example 6.3, but a finer mesh is used in the solution.

```
% e3s606.m
Lx = 3; Ly = 2;
nx = 12; ny = 12; hx = Lx/nx; hy = Ly/ny;
```

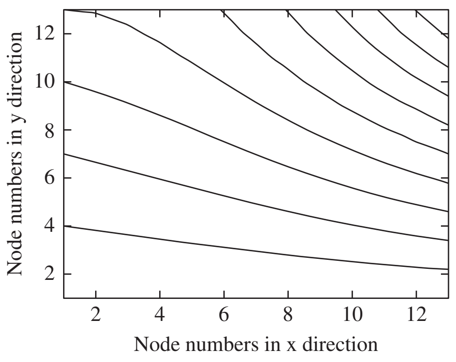

FIGURE 6.13 Finite difference estimate for the temperature distribution for the problem defined in Figure 6.12.

```
byO = 0*[0:hx:Lx];
byn = 200+(100/3)*[0:hx:Lx].^2;
bx0 = 100*[0:hy:Ly];
bxn = 250*[0:hy:Ly];
F = zeros(nx+1,ny+1); G = F;
a = ellipgen(nx,hx,ny,hy,G,F,bx0,bxn,by0,byn);
aa = flipud(a); contour(aa,'k')
xlabel('Node numbers in x direction');
ylabel('Node numbers in y direction');
```

The output from this script is the contour plot shown in Figure 6.13. The temperature is not shown on the contour plot; if required, it can be obtained from `aa`.

Example 6.5

Poisson's equation. Determine the deflection of a uniform square membrane, held at its edges and subject to a distributed load, which can be approximated to a unit load at each node. This problem is described by Poisson's equation, (6.35), where $F(x, y)$ specifies the load on the membrane. We use the following script to determine the deflection of this membrane using the MATLAB function `ellipgen`.

```
% e3s607.m
Lx = 1; Ly = 1;
nx = 18; ny = 18; hx = Lx/nx; hy = Ly/ny;
by0 = zeros(1,nx+1); byn = zeros(1,nx+1);
bx0 = zeros(1,ny+1); bxn = zeros(1,ny+1);
F = -ones(nx+1,ny+1); G = zeros(nx+1,ny+1);
a = ellipgen(nx,hx,ny,hy,G,F,bx0,bxn,by0,byn);
surfl(a)
axis([1 nx+1 1 ny+1 0 0.1])
```

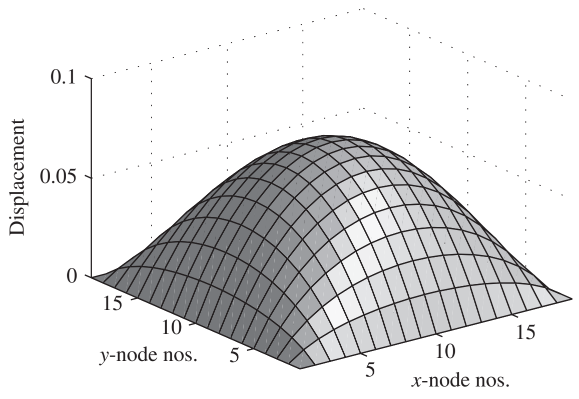

FIGURE 6.14 Deflection of a square membrane subject to a distributed load.

```
xlabel('x-node nos.'), ylabel('y-node nos.')
zlabel('Displacement')
max_disp = max(max(a))
```

Running this script gives the output shown in Figure 6.14 together with

```
max_disp =
    0.0735
```

This compares with the exact value of 0.0737.

Example 6.6

Characteristic value problem. Determine the natural frequencies and mode shapes of a freely vibrating square membrane held at its edges. This problem is described by the eigenvalue problem (6.37). The natural frequencies are related to the eigenvalues, and the mode shapes are the eigenvectors. The following MATLAB script determines the eigenvalues and vectors. It calls the function `ellipgen` and outputs a list of eigenvalues and provides Figure 6.15, showing the second mode shape of the membrane.

```
% e3s608.m
Lx = 1; Ly = 1.5;
nx = 20; ny = 30; hx = Lx/nx; hy = Ly/ny;
G = ones(nx+1,ny+1); mode = 2;
[a,om] = ellipgen(nx,hx,ny,hy,G,mode);
eigenvalues = om(1:5), surf(a)
view(140,30)
axis([1 nx+1 1 ny+1 -1.2 1.2])
xlabel('x - node nos.'), ylabel('y - node nos.')
zlabel('Relative displacement')
```

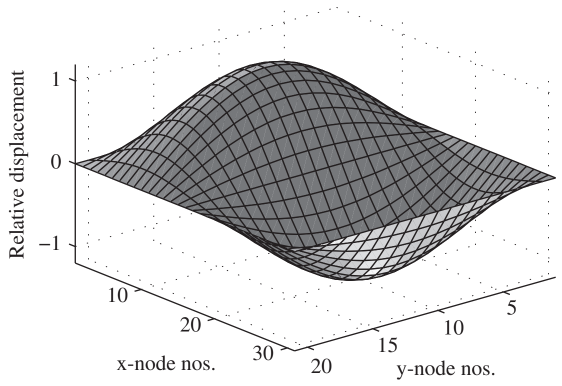

FIGURE 6.15 Finite difference approximation of the second mode of vibration of a uniform rectangular membrane.

Table 6.1 Finite Difference Approximations versus Exact Eigenvalues for Uniform Rectangular Membrane

FD Approximation	Exact	Error (%)
14.2318	14.2561	0.17
27.3312	27.4156	0.31
43.5373	43.8649	0.75
49.0041	49.3480	0.70
56.6367	57.0244	0.70

Running this script gives

```
eigenvalues =
   14.2318
   27.3312
   43.5373
   49.0041
   56.6367
```

These eigenvalues compare with the exact values given in Table 6.1.

6.8 Summary

In this chapter, we examined the application of finite difference methods to a broad range of second-order ordinary and partial differential equations. A major problem in the development of scripts is the difficulty of accounting for the wide variety of boundary conditions and boundary shapes that can occur. Software packages have been developed to solve partial differential equations that arise in computational fluid dynamics and continuum mechanics, using either finite difference or finite element methods; however, they are both complex and expensive because they allow the user total freedom to define boundary shapes and conditions.

Problems

6.1. Classify the following second-order partial differential equations:

$$\frac{\partial^2 y}{\partial t^2} + a\frac{\partial^2 y}{\partial x \partial t} + \frac{1}{4}(a^2 - 4)\frac{\partial^2 y}{\partial x^2} = 0$$

$$\frac{\partial u}{\partial t} - \frac{\partial}{\partial x}\left(A(x,t)\frac{\partial u}{\partial x}\right) = 0$$

$$\frac{\partial^2 \varphi}{\partial x^2} = k\frac{\partial^2(\varphi^2)}{\partial y^2} \quad \text{where} \quad k > 0$$

6.2. Use the shooting method to solve $y'' + y' - 6y = 0$, where the prime denotes differentiation with respect to x, given the boundary conditions $y(0) = 1$ and $y(1) = 2$. Note that an illustrative script for the shooting method is given in Section 6.2. Use trial slopes in the range $-3 : 0.5 : 2$. Compare your results with those you obtain using the finite difference method with 10 divisions. The finite difference method is implemented by the function `twopoint`. Note that the exact solution is

$$y = 0.2657\exp(2x) + 0.7343\exp(-3x)$$

6.3. **(a)** Use the shooting method to solve $y'' - 62y' + 120y = 0$, where the prime denotes differentiation with respect to x, given the boundary conditions $y(0) = 0$ and $y(1) = 2$. Solve this equation by applying the shooting method, using trial slopes in the range $-0.5 : 0.1 : 0.5$. Note that the exact solution is

$$y = 1.751302152539304 \times 10^{-26}\{\exp(60x) - \exp(2x)\}$$

(b) By substituting $x = 1 - p$ in the original differential equation, show that $y'' + 62y' + 120y = 0$, where the prime denotes differentiation with respect to p. Note that the boundary conditions of this problem are $y(0) = 2$ and $y(1) = 0$. Solve this equation by applying the shooting method, using trial slopes in the range 0 to -150 in steps of -30 at $p = 0$. Note that a very good approximation to the solution is $y = 2\exp(-60p)$.

Compare the two answers you obtain for (a) and (b). Note that an illustrative script for the shooting method is given in Section 6.2. Also solve (a) and (b) using the finite difference method, implemented in `twopoint`. Use 10 divisions and repeat with 50 divisions. You should plot your answers and compare with a plot of the exact solution.

6.4. Solve the boundary value problem $xy'' + 2y' - xy = e^x$ given that $y(0) = 0.5$ and $y(2) = 3.694528$, using the finite difference method implemented by the function `twopoint`. Use 10 divisions in the finite difference solution and plot the results, together with the exact solution, $y = \exp(x)/2$.

6.5. Determine the finite difference equivalence of the characteristic value problem defined by $y'' + \lambda y = 0$, where $y(0) = 0$ and $y(2) = 0$. Use 20 divisions in the finite difference method. Then solve the finite difference equations using the MATLAB function `eig` to determine the lowest value of λ, that is, the lowest eigenvalue.

6.6. Solve the parabolic equation (6.24) with $K = 1$, subject to the following boundary conditions: $u(0,t) = 0$, $u(1,t) = 10$, and $u(x,0) = 0$ for all x except $x = 1$. When $x = 1$, $u(1,0) = 10$. Use the function `heat` to determine the solution for $t = 0$ to 0.5 in steps of 0.01 with 20 divisions of x. You should plot the solution for ease of visualization.

6.7. Solve the wave equation (6.29) with $c = 1$, subject to the following boundary and initial conditions: $u(t,0) = u(t,1) = 0$, $u(0,x) = \sin(\pi x) + 2\sin(2\pi x)$, and $u_t(0,x) = 0$, where the subscript t denotes partial differentiation with respect to t. Use the function `fwave` to determine the solution for $t = 0$ to 4.5 in steps of 0.05, and use 20 divisions of x. Plot your results and compare with a plot of the exact solution, which is given by

$$u = \sin(\pi x)\cos(\pi t) + 2\sin(2\pi x)\cos(2\pi t)$$

6.8. Solve the equation

$$\nabla^2 V + 4\pi^2(x^2 + y^2)V = 4\pi\cos\{\pi(x^2 + y^2)\}$$

over the square region $0 \le x \le 0.5$ and $0 \le y \le 0.5$. The boundary conditions are

$$V(x,0) = \sin(\pi x^2),\quad V(x,0.5) = \sin\{\pi(x^2 + 0.25)\}$$
$$V(0,y) = \sin(\pi y^2),\quad V(0.5,y) = \sin\{\pi(y^2 + 0.25)\}$$

Use the function `ellipgen` to solve this equation with 15 divisions of x and y. Plot your results and compare with a plot of the exact solution, which is given by $V = \sin\{\pi(x^2 + y^2)\}$.

6.9. Solve the eigenvalue problem $\nabla^2 z + \lambda G(x,y)z = 0$ over a rectangular region bounded by $0 \le y \le 1$ and $0 \le x \le 1.5$, $z = 0$ at all boundaries. Use the function `ellipgen` with six divisions in y and nine in x. The function $G(x,y)$ over this grid is given by the MATLAB statements `G = ones(10,7); G(4:7,3:5) = 3*ones(4,3);`.

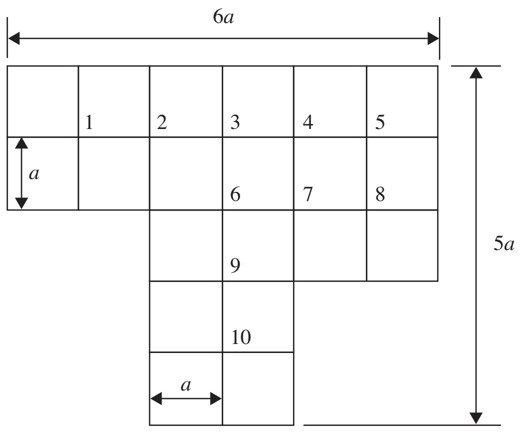

FIGURE 6.16 Region for Problem 6.10.

This represents a membrane with a central area thicker than its periphery. The eigenvalues are related to the natural frequencies of this membrane.

6.10. Solve Poisson's equation $\nabla^2\phi + 2 = 0$ over the region in Figure 6.16 with $a = 1$ at the boundary $\phi = 0$. You will have to assemble the finite difference equation by hand, applying (6.13) to the 10 nodes, and then use MATLAB to solve the resulting linear equation system.