Contents

This page intentionally left blank.

Chapter 1

Hyperbolic and Parabolic Partial Differential Equations

1.1. EXAMPLES AND CONCEPTS OF HYPERBOLIC PDE'S

In the last chapter, we discussed in some detail the heat and Laplace's equations, which are prototypes for parabolic and elliptic PDEs, respectively. We would like now to introduce some concepts and theory for the wave equation, which is the prototype for hyperbolic equations. The wave equation models many natural phenomena, including gas dynamics (in particular, acoustics), vibrating solids and electromagnetism. It was first studied in the eighteenth century to model vibrations of strings and columns of air in organ pipes. Several mathematicians contributed to these initial studies, including Taylor, Euler, and Jean D'Alembert, about whom we will say more shortly. Subsequently in the nineteenth century, the wave equation was used to model elasticity as well as sound and light waves, and in the twentieth century, it has been used in quantum mechanics and relativity and most recently in such fields as superconductivity and string theory. In general, the wave equation has a time variable t and any number of space variables JC, y, z,... and takes the form

$$u_u = c^2 \triangle u = c^2(u_{xx} + u_{yy} + ...) \tag{1.1}$$

where c is a positive constant and the Laplace operator on the right is with respect to all of the space variables. Modifications of this equation have been successfully used to model numerous physical waves and wavelike phenomena. In two space variables, for example, allowing for a variable wave speed due to depth differences in an ocean, the PDE: $u_u = \nabla \cdot [H(x,y,t)\nabla u] + H_u$ has been used to model large destructive ocean waves. [1] In such an application, the function H is the depth of the ocean at space coordinates (longitude and latitude) (x, y) and at time /. The latter term corresponds to the changes in depth due to underwater landslides. For more on this and other applications of this variable media wave equation, we mention the text [Lan-99].

Much of the general theory of hyperbolic PDEs is well represented by that for the **one-dimensional wave equation** ($u = u(x,t)$ depends on time / and one space variable x so we proceed now to introduce it through its historical model of a vibrating string and present some of the theory. At the end of the section we indicate some differences and similarities of higher-dimensional waves to onedimensional waves.

We consider a small segment of taut string having length As and uniform tension T that is acted on by a vertical force q, as shown in Figure 12.1.

We assume that the string is displaced only in the vertical (transverse) direction, and let $u(x,t)$ denote the y-coordinate of the string at horizontal coordinate x at the time t. If we let ρ denote the mass density (mass per unit length) of the string (assumed constant), then Newton's second law (F = ma) gives us that

$$-T sin\Theta + T sin(\Theta + \triangle \Theta) + q \triangle s\rho \triangle s u_u(x,t)$$

, where the first two terms represent the vertical component of the internal elastic forces acting on the segment of string.

[1] The symbol ∇, read as "nabla" or "del," is used to represent the gradient operator, which is the vector of all partial derivatives of a function. Thus for a function of two variables $f(x,y)$, $\nabla f = \nabla f(x,y) = (f_x(x,y), f_y(x,y))$. The large dot represents the vector dot product, so in long form: $\nabla \cdot [H(x,y,t)\nabla u] = (\partial_x, \partial_y) \cdot (Hu_x, Hu_y) = \partial_x(Hu_x) + \partial(Hu_y)$. In particular, when $H \equiv 1$ we have $\nabla \cdot [\nabla u] = \partial_x(u_x) + \partial_y(u_y) = u_{xx} + u_{yy} = \triangle u$, another way to write the Laplacian of u. Such notations are very common in the literature for partial differential equations involving several space variables.

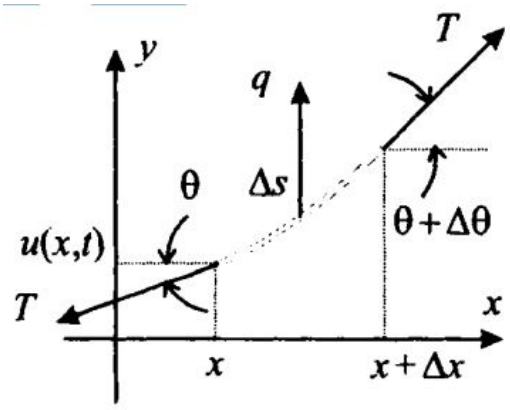

Figure 1.1: : A segment of a uniformly taut string having tension Tand external load g. The string is displaced vertically only, and u(x,t) is the vertical level of the string at time / and horizontal position x .

For small deflections in the string, we have $\triangle s \approx \triangle x$ and also $sin(\theta) \approx \theta \approx u_x(x,t)$ In the limit as As -ż 0, this brings us to

$$Tu_{xx} + q = \rho u_u, u = u()x,t \tag{1.2}$$

which is the **one-dimensional wave equation with external load term** q. In case q = 0, this reduces to the one-dimensional wave equation (1) with $c = (T/\rho)^{1/2}$. It turns out that this parameter c is the speed at which the wave (i.e., any solution of the equation) propagates. This will be made clear shortly. Intuitively, it makes sense that the speed of any disturbance on a string should increase along with the tension and decrease for heavier strings. For a derivation of wave equations for strings under more general hypotheses we refer to the article by S. Antman [Ant80] or Chapter 3 of the textbook by Kevorkian [Kev-00].

Figure 1.2: Jean Le Rond D'Alembert (1717-1783), French mathematician.

The general solution of the one-dimensional wave equation was first derived by the French mathematician Jean D'Alembert. [2] D'Alemberfs derivation is simple and elegant and the form of the solution will give many insights into qualitative aspects of wave equations. It begins by introducing the new variables:

$$\xi = x - ct, \eta = x + ct \tag{1.3}$$

We may now think of u as either a function of (x,t) or of (ξ, η). When we use the chain rule to translate the wave equation (1) into a PDE with respect to the new variables (ξ, η) something very nice will happen. The resulting PDE will be extremely easy to solve for the general solution. Applied using (3), the chain rule gives the following:

$$\begin{aligned} u_x &= u_\xi \xi_x + u_\eta \eta_x = u_\xi u_\eta \\ u_t &= u_\xi \xi_t + u_\eta \eta_t = -cu_\xi + cu_\eta \end{aligned} \tag{1.4}$$

In the same fashion, if we differentiate once again, we arrive at

$$u_{xx} = u_{\xi\xi} + 2u_{\xi\eta} + u_{\eta\eta}, u_u = c^2(u\xi\xi - 2u_{\xi\eta} + u_{\eta\eta}) \tag{1.5}$$

When we substitute equations (5) into the one-dimensional wave equation (1), we obtain the following version of the wave equation in the new variables (ξ, η) :

$$u_{\xi\eta} = 0. \tag{1.6}$$

[2]Jean D'Alembert was born in Paris as an illegitimate child of a former nun while the father was out of the country. Unable to support her son, his mother left him on the steps of a church. The infant was quickly found and taken to an orphanage. He was baptized as Jean Le Rond, after the name of the church where he was found. When the infant's father returned to Paris, he arranged for Jean to be adopted by a married couple, who were friends of his. His adoptive parents brought him up well. He studied law and earned a law degree. He soon decided that mathematics was his true passion and studied it on his own. Although mostly self-taught, D'Alembert became an eminent mathematician and scholar in the same league with the likes of Euler, Laplace, and Lagrange. He made significant contributions to partial differential equations and his elegant methods, including his solution to the wave equation, very much impressed Euler. Frederick II (King of Prussia) offered D'Alembert the presidency of the prestigious Berlin Academy, a position which he declined. He was quite an eloquent and well-rounded scholar and he made significant contributions to Diderot's famous encyclopedia. Apparently, D'Alembert was prone to argumentation and his disputes with other contemporary mathematicians caused him some professional difficulties on several occasions.

This PDE is very easy to solve, by "integrating" twice. Since it says that $\partial/\partial\eta(u_\xi) = 0$, we can integrate with respect to η to get $u_\xi = (F\xi)$,where

$F(\xi)$ is an arbitrary function of ξ Next we integrate again, this time with respect to ξ , to conclude that

$$u(\xi,\eta) = f(\xi) + g(\eta), \tag{1.7}$$

where $f(\xi)$ and $g(\eta)$ are arbitrary functions of the indicated variables. (Note $f(\xi)$ is an antiderivative of $F(\xi)$) Translating back to the original variables using (3) gives us the following general solution of the wave equation:

$$u(x,t) = f(x-ct) + g(x+ct), \tag{1.8}$$

where f and g are arbitrary functions (with continuous second derivatives). We point out that each term in (8) represents a wave propagating along the x-axis with speed c. For example, $f(x-ct)$ is constant on lines of the form $x = ct$. As time t advances, values of x must also increase to maintain the same value of f (disturbance). Thus the first term represents a wave that propagates in the positive jc-direction with speed c (right traveling wave). Similarly, the term $g(x+ct)$ represents a left-traveling wave Both waves travel without distortion (i.e., the profile of either one of them / units of time later will be the exact same profile, but shifted to the left or right ct units along the x-axis.)

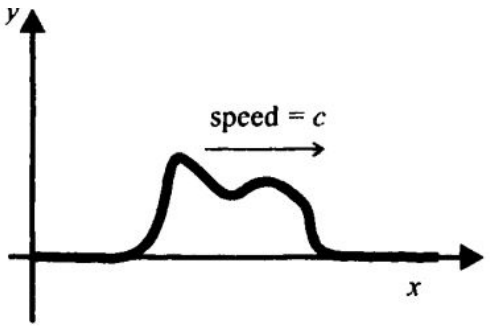

Figure 1.3: : A right-propagating pulse $f(x-ct)$. The general solution (8) of the onedimensional wave equation $u_\eta = c^2 u_{xx}$ also includes a left-propagating pulse. Both wavefronts propagate without distortion.

D'Alembert went on further with his general solution (8), formulating and solving a well-posed problem for the one-dimensional wave equation. We consider a very long string and so consider the one-dimensional wave equation on the space range $-\infty < x < \infty$ o, and the time range $0 \leqslant t < \infty$ Unlike with the heat equation, it is quite clear from (8) that merely specifying the wave profile W(JC,0) at time / = 0 is not sufficient to determine a unique solution. Indeed, the initial wave could come from a single left-moving wave, a single right-moving wave, or more generally could be made up as a superposition of two waves each moving in different directions.If we specify both the initial wave profile $u(x,0)$ and its initial velocity $u_t(x,0)$) , then this together with the wave equation will give a well-

posed problem. These initial boundary conditions are often referred to as **Cauchy boundary conditions** (or **Cauchy boundary data**) Thus the **Cauchy problem for the wave equation** is summarized as follows:

$$\begin{cases} (PDE)u_\eta = c^2 u_{xx}, -\infty < x < \infty, 0 < t < \infty, u = u(x,t) \\ (BC's)u(x,0) = \phi(x), u_t(x,0) = v(x) - \infty < x < \infty, = \leqslant \infty \end{cases} \tag{1.9}$$

This highlights an important general difference between elliptic PDEs versus hyperbolic PDEs. Recall from the last chapter that for elliptic PDEs, simply specifying the value of the solution on the boundary of the domain (Dirichlet boundary conditions) resulted in a well-posed problem. For hyperbolic PDEs, more information is needed for the problem to be well posed. We now state d'Alembert's solution of this Cauchy problem:

THEOREM 12.1:

(*D 'Alembert 's Solution of the Cauchy Problem?*[3] Suppose that the function $\phi(x)$ has a continuous second derivative and $v(x)$ has a continuous first derivative on the whole real line. Then the Cauchy problem (9) for the onedimensional wave equation has the unique solution given by

[3]In applications, it is convenient to allow functions $\phi(x)$ and $v(x)$ for initial data which may violate the technical assumptions of having the required derivatives at all values of x. Often there are a finite set of values (singularities) of x at which either $\phi(x)$ or $v(x)$ may not even be defined or their derivatives may not exist. Such singularities do not pose any serious problems for d'Alembert's solution, but they will give rise to corresponding singularities in the solution at all future time values. See, for example, the initial profile of Figure 12.4 $(\phi(x))$ for a wave problem. This function has singularities at the three points where there are sharp corners in the graph $x = -1, 0, 1$. Future profiles in the solution shown in Figure 12.5 show also the presence of such singularities. Recall that for solutions of heat equations that were seen in the last chapter, singularities arising from discontinuities in an initial temperature distribution or its derivative immediately got smoothed out as time advanced. This is one of the major distinguishing features between hyperbolic versus parabolic PDE's. In the former, singularities are preserved and propagate, while in parabolic PDE's, initial singularities disappear as soon as time becomes positive.

$$u(x,t) = \frac{1}{2}\phi(x+ct) + \phi(x-ct) + \frac{1}{2c}\int_{x-ct}^{x+ct} v(s)ds \tag{1.10}$$

Proof: Substitution of the general solution (8) into the BCs of (9) produces (put / = 0):

$$\phi(x) = f(x) + g(x), and v(x) = -cf'(x) + cg'(x)$$

Integrating the second equation and dividing by c gives: $(1/c)\int_0^x V(s)ds = g(x) - f(x)$ (Since $f(x)$ and $g(x)$ are arbitrary functions we can assume that the constant of integration is zero.) This last equation together with the first of the original pair are easily solved to give:

$$f(x) = \frac{1}{2}[\phi(x) - \frac{1}{c}]\int_0^x V(s)ds], g(x) = \frac{1}{2}[\phi(x) + \frac{1}{c}]\int_0^x V(s)ds]$$

Substituting these formulas into (8) now lets us write the solution as:

$$f(x-ct) + g(x+ct) = \frac{1}{2}[\phi(x+ct) + \phi(x-ct)] + \frac{1}{2c}[-\int_0^{x-ct} v(s)ds + \int_0^{x+ct} v(s)ds]'$$

which equals the expression in (10).

We emphasize that the foregoing analysis was only for one-dimensional waves on an infinite string. Of course, infinite strings do not exist, but for long strings, or for modeling disturbances on finite strings for limited time intervals, the above analysis can lead to useful insights. It is rare to have such an explicit analytical general solution. Soon we will consider boundary conditions that will require nonanalytical numerical methods, and finite-difference methods will be employed as in the last chapter. For now, let us get some hands-on experience with traveling waves. In the following example, we will get MATLAB to create a series of snapshots of a solution of a natural wave problem.

EXAMPLE 12.1:

(A Plucked Infinite String) Consider what happens to a long string that is plucked with three fingers as shown in Figure 12.4 and then released (at time $t = 0$). Assume that the units are chosen so that wave speed $c = (T/\phi)^1/2$ equals 1. Using d'Alembert's solution, get MATLAB to create a series of snapshots of the wave profiles for each of the seven times starting with time $t = 0$ and advancing to $t = 3$ in increments of 0.5.

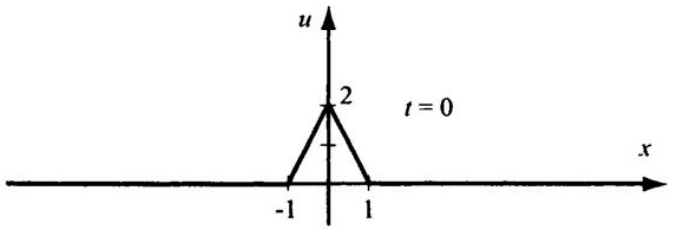

Figure 1.4: Initial profile for the plucked string of Example 12.1.

SOLUTION: In the Cauchy problem (9), we put $c = 1$, and $v(x) = 0$ (since at time $t = 0$, the three-finger plucked string is released with no initial velocity). From Figure 12.4, we can write the initial profile of the string as

$$\phi(x) = \begin{cases} 2-2|x|, & for\ |x| \leqslant 1 \\ 0, for & |x| \geqslant 1 \end{cases}$$ It is not too difficult to analyze the resulting wave [0, for |JC| > 1 propagation analytically using Theorem 12.1, but a MATLAB code can be easily written to produce snapshots and/or movies of this and more complicated waves. Since an inline function construction is not appropriate for functions whose formulas change, we first construct an M-file for the function $\phi(x)$:

```
function y = EX121(x)
if abs(x)<l, y=2-2*abs(x);
else y=0;
end
```

Using this M-file in the following code, we create relevant vectors to produce the snapshots, and we use the *subplot* command to conveniently collect all of the profiles in a single figure. The resulting MATLAB plot window is reproduced in Figure 12.5.

```
>> x=-5:.01:5;
>> counter =1;
>> for t=0:.5:3;
xl=x+t; x2=x-t;
  for i=l:1001
    u(i)=.5*(EX12_l(xl(i))+EX12_l(x2(i)));
  end
  subplot(7,1,counter)
  plot(x,u)
  hold on
  axis ([-5 5 -1 3]) rsWo fix a good axis range.
  counter=counter+l;
end
```

EXERCISE FOR THE READER 12.1: Following the procedure for making amovie in Section 7.2, get MATLAB to create a movie of the solution of the waveproblem of Example 12.1 for the time range $0 < / < 4$. Play it back at varyingspeeds (and perhaps with varying repetitions).

EXERCISE FOR THE READER : (a) Write a function M-file:

```
function [] = dalembert(c,step, finaltime, phi, nu, range),
```

for creating a series of snapshots for the solution of the one-dimensional waveproblem (9). The inputs should be: a positive number c for the wave speed, apositive number `step` for the time steps of the snapshots, and another positivenumber `finaltime` for the time limit of the snapshots. Also, the initial data ofthe problem will be inputted as two inline or M-file functions **phi** and `nu` Thelast input variable is a 4x1 vector `range` for the xy-axis range to use in thesnapshots. There will be no output variables, but the program will produce agraphic of snapshots of the Cauchy problem (9)starting at time t = 0 andcontinuing in increments of `step` until `finaltime` e is exceeded.
(b) Run your program using the data of Example 12.1.
(c) Run your program on the "hammer blow" problem that consists of the Cauchyproblem (9) (for the wave equation) with c = 1, <p(x) = 0, and $V(x)\begin{cases}1, for|x| \leqslant 1\\0, for|x| \geqslant 1\end{cases}$ Create a series of snapshots of the solution from / = 0 to t = 5 in increments of / = 0.5.
(d) Use your program to help you estimate the length ofn time it takes for the of the waves of both parts (b) and (c) above to reach an observer atposition x = 10. How do your answers fit in with the previously mentioned factthat the waves in making up d'Alembert's general solution of the wave equationtravel at speed c (here c = 1)?

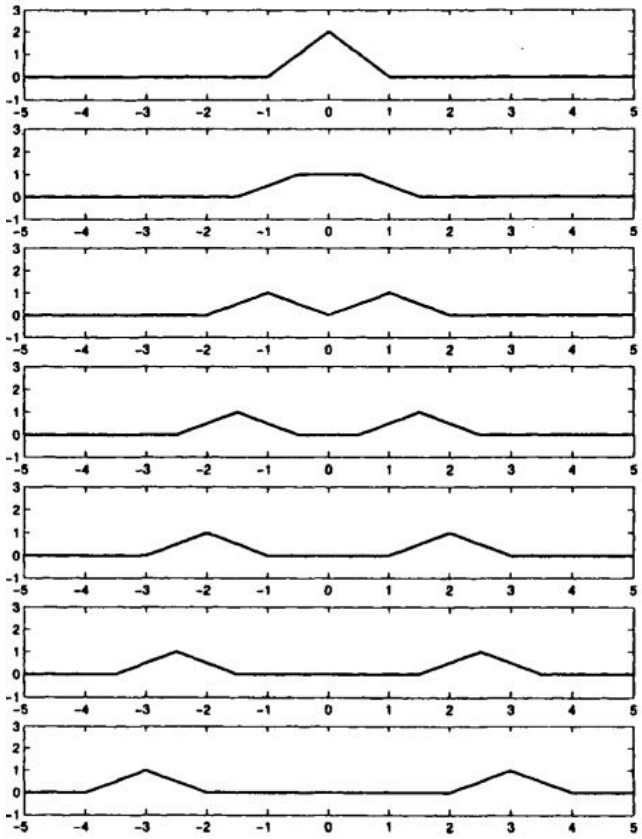

Figure 1.5: Progressive snapshots of the solution of the Cauchy problem for the plucked string of Example 12.1, at times t = 0, t = 0.5, t = 1, ..., t = 3. Note that the initial disturbance separates into two disturbances that eventually take on the same shape but each having half the size of the original. The function u(x,t) could also be graphed in three dimensions as a function of two variables. The snapshots, which are merely "slices** of the three-dimensional graphs, are often more useful than the latter

We now introduce a concept that will help us to highlight another important difference between parabolic and hyperbolic PDEs. Note that from d'Alembert's solution of the wave initial value problem (9), the solution is made up of two waves propagating at speed c and traveling in opposite directions. The actual disturbances can travel at speeds less than but not exceeding c (see part (d) of Exercise for the Reader 12.2). It also follows from d'Alembert's solution that the value of the solution u of (9) at a certain point (x,t), i.e., the vertical disturbance of the string at location x and at time t, can only be affected by the initial data 0?,vover the interval $[x-ct, x+ct]$. This interval is called the interval of dependence of the "space-time" point (jt,f); and the corresponding triangle (see Figure 12.6) in the space-time plane is called the **domain of dependence** of (x, t).

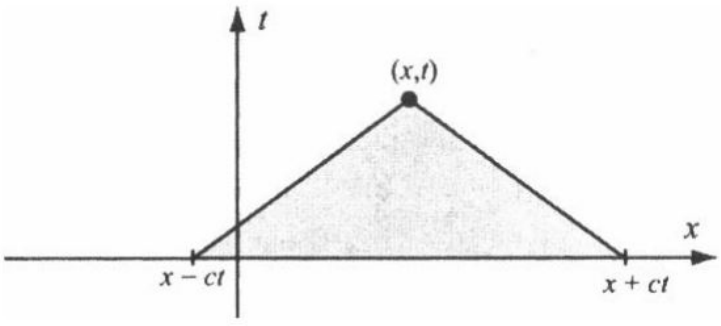

Figure 1.6: Illustration of the interval of dependence $[x-ct, x+ct]$] (on the x-axis) for the wave equation on a line. The shaded triangle in the space-time plane (jc/-plane) is called the domain of dependence. The values of the initial condition functions $\phi(x) and v(x)$ outside of the interval of dependence for (x,t) are irrelevant to the determination of $u(x,t)$.

Although d'Alembert's solution of the wave equation on the infinite string makes it possible to analyze analytically most of the properties of the solution, the next variation of a Cauchy problem for the one-dimensional wave equation that we consider will give rise to analytical formulas that are extremely complicated and intractable. We now study the wave equation on a string of finite length, which is fixed at both ends. The precise Cauchy problem that we work with is

as follows:

$$V(x)\begin{cases}(PDE)u_n = c^2u_{xx}, & ,0<x<L,0<t<\infty,u=u(x,t)\\ (BC's)\begin{cases}u(x,0)=\phi(x),u_t(x,0)=V(x)\\ u(0,t)=u(L,t)=0\end{cases} & ,0<x<L,0\leqslant t<\infty\end{cases} \tag{1.11}$$

A model to help visualize this Cauchy problem would be the motion of a guitar string of length L that is fixed at both ends. What makes a nice analytical formula impossible here is the fact that once the disturbances reach the ends of the string, they will bounce back, and things will continue to get more complicated as time goes on.

Theoretically, we can solve (11) by using d'Alembert's solution for the infinite string in a clever way. The useful artifice that will be used is called the **method of reflections** We first extend the functions $\phi(x) and V(0)$ to be functions on the whole real line, based on their values in the interval $0<x<L$ Labeling these extensions as $\hat{\phi}(x) and \hat{V}(x)$), respectively, they will be created so that they are odd functions across both of the boundary values $x=0 and x=L$. Analytically, this means that

$$\hat{\phi}(-x)=-\hat{\phi}(x) \quad and \quad \hat{\phi}(2L-x)=-\hat{\phi}(x), \quad -\infty<x<\infty \tag{1.12}$$

and the corresponding identities for $\hat{V}$ It can be easily verified (Exercise 14) that the following formula gives such an extension $-\hat{\phi}(x)$ of $\hat{\phi}(x)$. [4]

$$\hat{\phi}(x)\begin{cases}\phi(x) \quad if \quad 0<x<L\\ -\phi(-x) \quad if \quad -L<x<0\\ \text{Extend to be periodic of period 2L}\end{cases} \qquad ,0<x<L,0\leqslant t<\infty \tag{1.13}$$

See Figure 12.7 for a graphical depiction of this construction. An analogous formula is used to construct $\hat{V}(x)$.

EXERCISE FOR THE READER 12.3: (Constructing an M-file for a Periodic Function) (a) For the function $\phi(x)=1-|1-x|$ on the interval [0, 2] (L = 2). Write an M-file, called `y=phihat(x)` that extends the given ftinction to $-\infty<x<\infty$ by the rule of (13). Try to write your M-file so that it does not use any loops.
(b) Get MATLAB to plot the graph of your `phihat(x)` on the interval $-6\leqslant x\leqslant 6$

If we solve the corresponding Cauchy problem (9) on the whole real line using as data the extended functions $\hat{\phi}(x)$ and $\hat{V}(x)$ d v(x) for boundary data, the ftinction $\hat{u}(x,t)$ that arises will, in fact, extend the solution of (11).

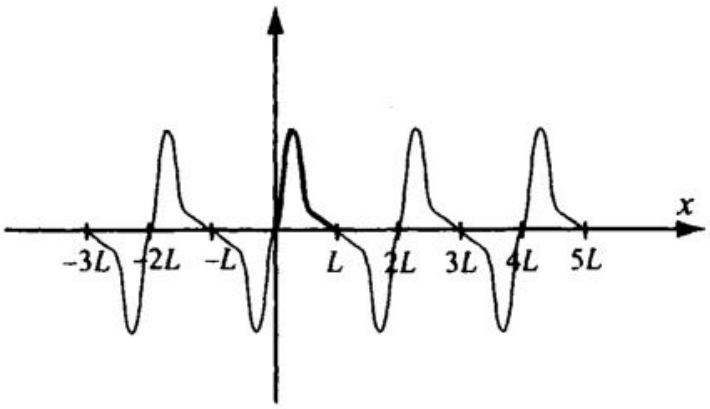

Figure 1.7: Illustration of the extension (13) of a function $\phi(x)$) defined from x = 0 to x = L (heavy graph portion) to a ftinction $\hat{\phi}(x)$. that is odd about each of the endpoints x = 0 and x - L

Some parts of this assertion are clear. Defining $u(x,t)=\hat{u}(u,t)$ for $0\leqslant\leqslant L$ and $t\geq 0$ (i.e., take u to be the function u restricted to the domain of the problem (11), it is clear that $u(x,t)$ satisfies the wave equation and the first two (initial) boundary conditions since $\hat{u}$ does. Because of the odd extension properties of $\hat{\phi}$ and $\hat{v}(x)$

EXAMPLE 12.2:

(A Plucked Guitar String) Consider what happens to a guitar string of length 4 units that is plucked with one finger as shown in Figure 12.8 and then released (at time $t=0$). Assume that the units are chosen so that wave speed

[4] Technically, this definition does not define $\hat{\phi}(x)$ for $x=0,\pm L,\pm L,\cdots$. The original function $\phi(x)$) was also not defined at the endpoints x = 0, and x = L. This was only for notational convenience in the boundary conditions of (11). The boundary conditions corresponding to the ends of the string being fixed would force $\phi(0)=\phi(L)$ = 0 so we extend the definition $\hat{\phi}(x)$ to all real numbers by specifying $\hat{\phi}$ for $\hat{\phi}(0)=\hat{\phi}(\pm L)=\hat{\phi}(\pm L)=\cdots=0$. The resulting function will be continuous (otherwise the string would be broken).

$c = (1/\rho)^1/2$ equals 1. By using the method of reflections, get MATLAB to create a series of snapshots of the wave profiles for each of the 12 times starting with time t = 0 and advancing to t = 6 in increments of 0.5.

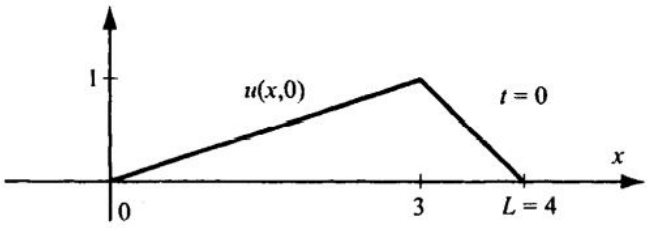

Figure 1.8: The initial profile of the plucked guitar string of Example 12.2.

SOLUTION: Looking at Figure 12.8, we can write:

$$\phi(x)(\equiv u(x,0)) = \begin{cases} x/3, & for\ 0 \leqslant x \leqslant 3 \\ 4-x, & for\ 0 \leqslant x \leqslant 3 \end{cases} .$$

Also $V(x)(\equiv W,(JC,0)) = 0$ ince the string is released without velocity. We first create an M-file for $\hat{\phi}(x)$ using a similar construction as was done in the solution of Exercise for the Reader 12.3.

```
function y = EX12_2phihat(x)
if (0 <= x)&(x <= 3), y=x/3;
elseif (x >=3)&(x<=4), y=4-x;
elseif (x<0)&(x>=-4), y = -EX12_2(-x);
else q=floor(<x+4)/8); y=EX12_2phihat(x-8*q);
end
```

We can now use MATLAB to create the desired snapshots. To make for a convenient single graphic of all 41 plots, we use the subplo t command to partition the plot window into smaller pieces.

```
function y = EX12_2phihat(x)
>> counter=l;
>> x=0:.01:4;
>> z=zeros (size (x) ) ; will be used to add axes to plots
>> elf Itreshon up the plot window
  for t=0:.2:8
  xl=x+t; x2=x-t;
    for i=l:401
    u(i)=.5*(EX12_2phihat(xl(i))+EX12_2phihat(x2(i)));
  end
  subplot(7,6,counter), plot(x,u), hold on
  plot (x, z, ' k ) adds a central axis to each   plot
  axis((0 4 -1 1])
  counter=counter+l;
  hold off
end
```

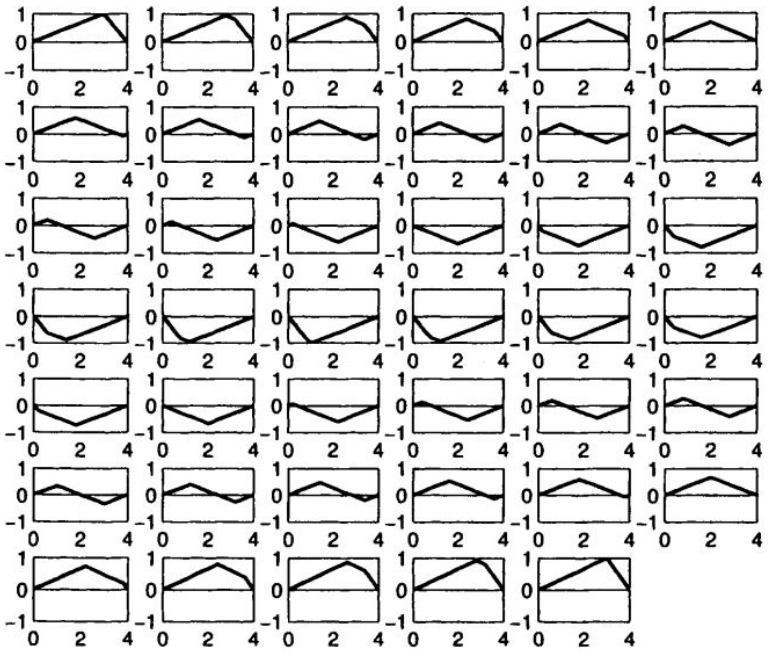

Figure 1.9: The initial profile of the plucked guitar string of Example 12.2.

FIGURE 12.9: Snapshots of the plucked guitar string of Example 12.2. (To be read from left to right, and then top to bottom.) The speed of the wave is taken to be one unit length per unit time. Each successive square represents an increment of 0.2 units of time. Notice that the last frame corresponds to eight units of time and is exactly the initial profile.

Analytically, the waves that result on such finite strings are quite messy to describe. Physically, what is happening is that two waves are still moving in opposite directions at speeds equal to c. Each is constantly bouncing off the ends, reflecting and superimposing with the other. To get a better idea of the properties of the solution, it is a good idea to create a MATLAB movie for it (Exercise 4). Further details in this area can be found in Section 3.2 of [Str-92].

EXERCISE FOR THE READER 12.4: Prove that the solution of the wave problem on the finite string (11) is always periodic in the time variable with period Lie.

Suggestion: Use the solution arising from the method of reflections.
EXERCISE FOR THE READER 12.5: (Single Pulse Wave on a Finite String) Consider the wave problem (11) with c = 2, and initial profile $\phi(x)$ given as in Figure 12.10. Obtain a series of snapshots from time / = 0 through t = 10 in increments of 0.5 of the solution of the problem (11) under the hypotheses that:
(a) The impulse is moving to the right initially with speed 2 units per unit of time.
(b) The impulse is moving to the right initially with speed 1 unit per unit of time.
(c) The impulse is moving to the right initially with speed 4 units per unit of time.
You need not worry about finding an extremely accurate analytical formula to model the initial profile $\phi(x)$(x); you can simply use polynomial interpolation (as in Section 7.4) with or without derivative conditions.

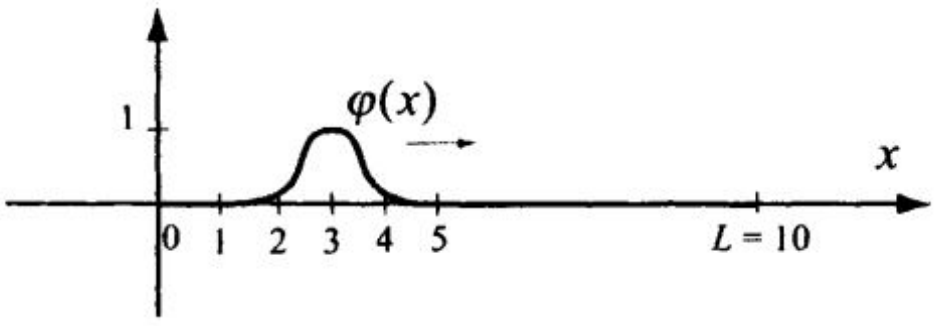

Figure 1.10: Initial profile for the impulse wave of Exercise for the Reader 12.5. The impulse is moving to the right.

Waves (i.e., solutions of the wave equation) satisfy a conservation of energy principle that is very important in physics. We demonstrate this principle for the one-dimensional wave equation written in physical form: $\rho u_u = T u_{ss}$, where, we recall, ρ is the mass density of the string and T is the tension. From physics, the **kinetic energy** of a mass m,

which is moving at a velocity v, is defined to be $\frac{1}{2}mv^2$. Breaking the wave into infinitesimal segments, this gives rise to the definition:

$$KE(t) = \frac{1}{2}\rho \int_a^b u_t(x,t)^2 dx \qquad t \geq 0, \tag{1.14}$$

for the kinetic energy of the string at time /. This improper integral will converge under most reasonable physical assumptions. For example, if both of the initial condition functions $\rho(x), V(x)$) vanish outside a finite interval, so will the integrand (but with a larger interval determined by the intervals of dependence). If we differentiate this kinetic energy function with respect to /, we may differentiate under the integral sign to obtain: Using the PDE to substitute $Tu_{xx}$for ρu_u in the above integral, and then integrating by parts, we obtain: [5]

$$\frac{d}{dt}KE(t) = \rho \int_{-\infty}^{\infty} u_t u_u dx \qquad t \geq 0 \tag{1.15}$$

Using the PDE to substitute $Tu_{xx} for \rho u_u$ in the above integral, and then integrating by parts, we obtain:

$$\frac{d}{dt}KE(t) = T\int_{-\infty}^{\infty} u_t u_{xx} dx = Tu_t u_x \Big|_{-\infty}^{\infty} - T\int_{-\infty}^{\infty} u_{tx} u_x dx = -T\int_{-\infty}^{\infty} u_{tx} u_x dx,$$

the last equation being valid since the integrated term vanishes off a finite interval. Since $u_{tx}u_x = \partial/\partial t \frac{1}{2}u_x^2$ we may write (again using the differentiation under the integral sign rule):

$$\frac{d}{dt}KE(t) = -\frac{d}{dt}\frac{1}{2}Tu_x^2 dx, \qquad t \geq 0. \tag{1.16}$$

In basic physics, the **potential energy** of an object of mass m located at height h is defined to be mgh, where g is the gravitational constant. The basic conservation of energy principle in elementary mechanical physics states that if no external forces other than gravity are present, then the total energy = kinetic energy + potential energy remains constant. (Think of when an object falls, its velocity increases so its kinetic energy increases and its height decreases so its potential energy decreases.) The analogue for the potential energy for the string is the following integral:

$$PE(t) = \frac{1}{2}T\int_{-\infty}^{\infty} u_x(x,t)^2 dx \qquad t \geq 0 \tag{1.17}$$

and, correspondingly, the **total energy** is defined to be

$$E(t) = KE(t) + PE(t) = \frac{1}{2}\int_{-\infty}^{\infty} [\rho u_t^2 + Tu_x^2] dx \qquad t \geq 0 \tag{1.18}$$

The identity (16) states that $\frac{d}{dt}KE(t) = -\frac{d}{dt}KE(t)$, and it follows from (18) that E'(t) = 0 (i.e., the total energy in the wave remains constant). This is the conservation of energy. It is extremely important and noteworthy! Regardless of how long we let the string propagate, the total energy E of the configuration will remain unchanged.

EXAMPLE 12.3:

(a) Compute the total energy of the plucked infinite string of Example 12.1, and (b) of the plucked guitar string of Example 12.2.

SOLUTION: In light of the conservation of energy, we may simply use the initial conditions to evaluate E(0) in each case. In both cases, $u_t(x,0) = v(x) = 0$ and $u_t(x,0) = \phi'(x)$.

Part (a): $E = E(0) = \frac{T}{2}\int_{-\infty}^{\infty} u_x^2 dx = \frac{T}{2}\int_{-\infty}^{\infty} [\phi'(x)]^2 dx = \frac{T}{2}2^2 \cdot 2 = 4T$. The tension T is not specified in the example, so this is as far as we can take this answer Part (b): Here, since the string is finite, we similarly obtain:

$$E = E(0) = \frac{T}{2}\int_0^4 u_x^2 dx = \frac{T}{2}\phi'(x)dx = \frac{T}{2}[(1/3)^2 \cdot 3 + 1^2 \cdot 1] = 2T/3$$

There are some interesting similarities and differences of waves in one, two, three, and higher dimensions. We first point out that future profiles of one-dimensional waves will inherit symmetries in the initial conditions. Such results can be obtained from d'Alembert's formula (see Exercise 10). The analogue in higher dimensions of such symmetry would be **radially symmetric** waves. In n space dimensions such a wave would be a solution of the wave equation (1):

$$u_n = c^2 \delta u = c^2(u_{x_1x_1} + \cdots + u_{x_2x_2}), \quad u = u(x_1, x_2, \cdots, x_n, t)^6$$

[5] Such differentiations are permissible under general circumstances. Here is a relevant theorem: Suppose that $f(x,t)$ is a continuous function of two variables in some rectangular region in the xt-plane:$a \leqslant x \leqslant b, c \leqslant t \leqslant d$ Suppose also that the partial derivative $f,(x,t)$ is continuous in this same . b b region. Then the following identity is valid for any t, $c \leqslant t \leqslant d : \frac{d}{dt}\int_a^b f(x,t)dx = \int_a^b f_t(x,t)dx$. Note that although the integral in (14) is over the whole real line, if $\phi(x), V(x)$ vanish outside a finite interval, the integral can be evaluated over a finite interval and the theorem can be applied. The theorem can even be extended to certain improper integral settings and in cases where the continuity assumptions break down at isolated singularities. See any good book on advanced calculus for details on this theorem and related results, for example, [Rud-64], [Ros-96], or [Apo-74].

[6] Most interesting applications of the wave equation occur in one, two or three space dimensions in which cases the customary choices x, y, and z are used in place of x_1, x_2 and x_3 .

which is expressible in the form w(r,t), where $\sqrt{x_1^2+x_2^2+\cdots+x_n^2}$ is the distance to the origin. Thus, a radially symmetric w-dimensional wave is not really a function of n + 1 variables (as a general such wave might be) but actually just a function of two variables. There are analytical techniques for finding formulas for radially symmetric waves, but they involve special mathematical functions (such as Bessel functions) and the analysis can get a bit complicated. See, for example, [Str-92] for a nice treatment on radially symmetric waves. In two dimensions, water ripples provide a nice and telling example of radially symmetric waves. In three dimensions, sound waves and electromagnetic (e.g., radio) waves provide prototypical examples. If a pebble is dropped in water, the water ripples continue to propagate and reproduce themselves. In general, disturbances resulting from two-dimensional waves continue to propagate at a given point of space, once they have reached this point. In one and three dimensions, once the disturbance of a wave passes by a certain point, the wave is finished there and moves on. In three dimensions, however, there is an important difference from one-dimensional waves. The intensity of the wave decreases as we move away from the source. This can be proved from the conservation of energy. (Once a disturbance from a three-dimensional radially symmetric wave reaches a distance R from the source, it must cover an entire sphere with the same amount of energy that the wave packed on much smaller spheres, and the intensity will be decreased at each point on these larger spheres. This argument can be made into a rigorous proof.) In higher than three dimensions, radially symmetric waves turn out to have the same distorted properties of two-dimensional waves. These facts make it clear that we are very fortunate to live in a three-dimensional world. Indeed, if the dimension of our world were two or higher than three, than anytime someone spoke, we would never stop hearing them. In a one-dimensional world, anytime anyone spoke or a noise was made, everyone would hear it and with the same intensity regardless of how far away from the source they were! For a rigorous proof that radially symmetric distortion-free waves are only possible in one and three dimensions, and that only in one dimension are radially symmetric waves possible without loss of intensity, we refer the reader to the article (with a rather presumptuous title) by Morley [Mor-85] and [Mor-86].

EXERCISES 12.1:

1. (Making Snapshots of Vibrating Strings) For each of the following initial data sets, create a series of snapshots of the solution of the wave problem (9):

$$\begin{cases} (PDE)u_u = u_{xx}, & -\infty < x < \infty, 0 < t < \infty, u = u(x,t) \\ (BCs)u(x,0) = \phi(x), u_t(x,0) = v(x) & -\infty < x < \infty, 0 \leqslant t < \infty' \end{cases}$$

with c (wave speed) = 1

(a) $\phi(x) = \begin{cases} sin(x), for\ \ 0 \leqslant x \leqslant 2\pi \\ 0, \quad otherwise \end{cases}$, v(x)=0

(b) $\phi(x) = \begin{cases} sin(2x), for\ \ 0 \leqslant\leqslant x \leqslant 2\pi \\ 0, \quad otherwise \end{cases}$, v(x)=0

(c) $\phi(x) = 0,\ \ v(x) \begin{cases} 1, \quad for\ \ 6\pi \leqslant 2\pi \leqslant 8\pi \\ 0, \quad otherwise \end{cases}$.

(d) $\phi(x) = 0 \begin{cases} sin(x), for\ \ 0 \leqslant x \leqslant 2\pi \\ 0, \quad otherwise \end{cases} . \begin{cases} 1, \quad for\ \ 6\pi \leqslant 2\pi \leqslant 8\pi \\ 0, \quad otherwise \end{cases}$.

Obtain snapshots for the time range $0 \leqslant t \leqslant 14$ in increments of $\triangle t = 2$, and choose the axes range so that the plots show all disturbances of the wave in an informative fashion.

2. (More snapshots of vibrating strings) For each of the following initial data sets, create a series of snapshots of the solution of the wave problem (9):

$$\begin{cases} (PDE)u_u = u_{xx}, & -\infty < x < \infty, 0 < t < \infty, u = u(x,t) \\ (BCs)u(x,0) = \phi(x), u_t(x,0) = v(x) & -\infty < x < \infty, 0 \leqslant t < \infty' \end{cases}$$

with c (wave speed) = 1.

(a) $\phi(x) = 0 \begin{cases} sin(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}, v(x) \begin{cases} -cos(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}$,

(b) $\phi(x) = 0 \begin{cases} sin(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}, v(x) \begin{cases} -2cos(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}$,

(c) $\phi(x) = 0 \begin{cases} sin(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}, v(x) \begin{cases} -0.5cos(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}$,

(d) $\phi(x) = 0 \begin{cases} sin(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}, v(x) \begin{cases} -4cos(x), \ \ for\ \ 0 \leqslant x \leqslant \pi \\ 0, \quad otherwise \end{cases}$,

Obtain snapshots for the time range $0 \leqslant t \leqslant 20$ in unit increments. Physically, explain how the four sets of initial conditions are related.

3. *(Making Movies of Vibrating Strings)* For each of the vibrating string problems ((a) through (d)) of Exercise 1, create a MATLAB movie of the vibrating string on the time range $0 \leqslant t \leqslant 14$. View each at various speeds and repetitions.

4. *(More Movies of Vibrating Strings)* For each of the vibrating string problems ((a) through (d)) of Exercise 2, create a MATLAB movie of the vibrating string on the time range $0 \leqslant t \leqslant 20$. View each at various speeds and repetitions

5. (a) Create a MATLAB movie for the guitar string wave of Example 12.2 from time t = 0 till time / = 24. View it at various speeds and repetitions. (b) Create a MATLAB movie for the single impulse wave of Exercise for the Reader 12.5 from time / = 0 till time t = 40. View it at various speeds and repetitions.

6. Snapshots of Vibrating Finite Strings) For each of the following initial data sets, create a series of snapshots of the solution of the wave problem (11):

$$\begin{cases} (PDE) u_u = u_{xx}, \qquad 0 < x < L, 0 < t\infty, u = u(x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), u) t(x,0) = v(x) \\ u(x,t) = u(L,t) = 0 \end{cases} \quad , 0 < x < L, 0 \leqslant t < \infty' \end{cases}$$

with c (wave speed) = 1.

(a) $\phi(x) = \begin{cases} sin(x), for \;\; 0 \leqslant x \leqslant 2\pi \\ 0, \qquad otherwise \end{cases}$, $v(x) = 0, L = 4\pi.$

(b) $\phi(x) = \begin{cases} sin(x), for \;\; 0 \leqslant x \leqslant 2\pi \\ 0, \qquad otherwise \end{cases}$, $v(x) = 0, L = 3\pi.$

(c) $\phi(x) = \begin{cases} sin(x), for \;\; 0 \leqslant x \leqslant \pi \\ 0, \qquad otherwise \end{cases}$, $\begin{cases} -2cos(x), \;\; for \;\; 0 \leqslant x \leqslant \pi \\ 0, \qquad\qquad otherwise \end{cases}$ $L = 3\pi.$

(d) $\phi(x) = \begin{cases} sin(x), for \;\; 0 \leqslant x \leqslant 2\pi \\ 0, \qquad otherwise \end{cases}$, $\begin{cases} 1, \;\; for \;\; 6\pi \leqslant x \leqslant 8\pi \\ 0, \qquad\qquad otherwise \end{cases}$ $L = 10\pi.$

Obtain snapshots for the time range $\triangle \leqslant t \leqslant 40$ in increments of $\triangle t = 2$.

7. *(Making movies of Vibrating Finite Strings)* For each of the vibrating string problems ((a) through (d)) of Exercise 6, create a MATLAB movie of the vibrating string on the time range $0 \leqslant t \leqslant 60$. View each at various speeds and repetitions.

8. Compute the total energies of each of the vibrating infinite strings in Exercise 1.

9. Compute the total energies of each of the vibrating finite strings in Exercise 6.

10. *(Symmetry of Waves on an Infinite String)* Consider the solution of the wave problem (9):

$$\begin{cases} (PDE) u_u = c^2 u_{uu}, \qquad -\infty < x < \infty, 0 < t\infty, u = u(x,t) \\ (BCs) u(x,0) = \phi(x), u_t(x,0) = v(x) \qquad -\infty < x < \infty, 0 \leqslant t < \infty \end{cases}$$

Use d'AIembert's formula to prove the following symmetry inheritance results.
(a) If both of the initial data are even functions of $x (i.e., \phi(-x) = \phi(x) and v(-x) - v(x)$ for all x) then so will be the wave profile at any future time: $u(-x > t) = u(x,t)$ for all x and $t \geq 0$.
(b) If both of the initial data are odd functions of x $(i.e., \phi(-x) = -\phi(x) and V(-x) = -V(x)$ for all x), then so will be the wave profile at any future time: $u(-x,t) = -w(x,t)$ for all x and $t \geq 0$.

11. *(Waves on a Semi-infinite String)* Consider the solution of the following wave problem similar to the finite-string problem (11) except that only one end of the string is held fixed.

$$\begin{cases} (PDE) u_u = u_{xx}, \qquad 0 < x < \infty, 0 < t\infty, u = u(x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), u_t(x,0) = v(x) \\ u(x,t) = 0 \end{cases} \quad 0 < x < \infty, 0 \leqslant t < \infty \end{cases}$$

(a)Making use of d'AIembert's formula and an appropriate "method of reflections" technique similar to that used in the text for the finite string, develop a program for solving this problem. We point out that such a method will not be a numerical method, per se, since it will simply use the computer to perform analytical computations (and the only errors are due to roundoff).
(b) Obtain snapshots of profiles of the solution to the above problem using the following initial conditions: $\phi(x) =$

$\begin{cases} sin(x) for \ \ 0 \leqslant x \leqslant 2\pi \\ 0, \qquad otherwise \end{cases}$,$v(x) = 0, c = 1$ (c) Obtain snapshots of profiles of the solution to the above problem using the following initial conditions:

$$\phi(x) = \begin{cases} sin(x) for \ \ 0 \leqslant x \leqslant 2\pi \\ 0, \qquad otherwise \end{cases} \quad ,v(x) = 0 \begin{cases} 1, \ \ for \ \ \ 6\pi \leqslant x 8\pi 0, \qquad otherwise \quad L = 10\pi \end{cases}$$

(d) Create a MATLAB movie of the propagation of the wave in part (b).
(e) Create a MATLAB movie of the propagation of the wave in part (c).

12. *(A Maximum Principle for the Wave Equation)* (a) Suppose that the hypotheses of d'AIembert's theorem are satisfied for the Cauchy problem (9):

$$\begin{cases} (PDE) u_u = c^2 u_{uu}, \qquad -\infty < x < \infty, 0 < t \infty, u = u(x,t) \\ (BCs) u(x,0) = \phi(x), u_t(x,0) = v(x) \qquad -\infty < x < \infty, 0 \leqslant t < \infty \end{cases}$$

and that $|\phi(x)|$ for all x and that $|\int_a^b v(s)ds| \leqslant L$ for all numbers a and b. Show that the solution $u(x,t)$ of the Cauchy problem satisfies the inequality: $|u)x,t)| \leqslant M + L/2c$ for all x and t in the domain.
(b) Under what general circumstances can you conclude that the maximum amplitude of the wave is attained at time zero $(i.e., |u(x,t)| \leqslant max|K(x,0)| : \infty < x < \infty)$?

1.2. FINITE DIFFERENCE METHODS FOR HYPERBOLIC PDE'S

We begin this section by developing finite difference schemes for the numerical solution of the one-dimensional wave problem (11):

$$\begin{cases} (PDE) u_u = c^2 u_{xx}, \qquad 0 < x < L, 0 < t \infty, u = u(x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), u_t(x,0) = v(x) \\ u(x,t) = u(L,t) = 0 \end{cases} \qquad 0 \leqslant x \leqslant \infty, 0 \leqslant t < \infty \end{cases}$$

on a finite string. Our development works in general if we allow c to be a function of t and/or $x : c = c(x,t)$. Physically, this corresponds to modeling a vibrating string where its characteristics can change depending on time and space. We have already shown that in case c is a constant, d'Alembert's Theorem 12.1 coupled with the method of reflections can lead to a practical numerical method for solving this problem. Since the method simply evaluates the theoretical solution, it is relatively error free and so completely adequate for solving (11) with any sets of data. This will allow us to compute the errors of the numerical solutions we obtain from the finite difference methods. D'Alembert's solution, however, is specific to the wave equation, while the finite difference methods that we introduce can be easily adapted to work for more general hyperbolic PDE problems.

At first glance, the similarity of the wave and Laplace's equation would make it seem quite plausible that the same general finite difference discretization would work nicely, as we witnessed in the case for elliptic boundary value problems. The boundary conditions in (11), however, are different in two major ways: (i) The region $0 \leqslant x < L, 0 \leqslant t < \infty >$ is no longer a bounded rectangle, but rather a half strip extending to infinity in the positive /-direction, (ii) There are two boundary conditions on the lower side of the strip rather than one. We will indeed discretize the PDE in the analogous fashion to what was done to Laplace's equation (replace each second derivative with its central difference approximation), but because of (i), we will not be able to set up the problem as a finite linear system (there are infinitely many nodes). Instead, we will do what is called a marching scheme, where the nodal approximations are computed one time level at a time, moving up from $t = 0$.At first glance, this may seem like a better situation, since the number of variables and the size of the linear systems will be much smaller than if we were to do it all at once, as with the elliptic method. For the most part it is true that the computations will generally move faster, but one new issue that we will need to confront with such marching schemes is the issue of stability. At each step, the local truncation errors will still be very small, but they can compound quite quickly to make the numerical solutions meaningless. Fortunately, there are some stability criteria that give easy ways to arrange the relative step sizes so that the schemes will be stable.

All finite difference methods require that the variables be restricted to finite intervals, [7] so we will need to restrict time to some specified range, $0 \leqslant t \leqslant T$. As in Chapter 11 (see Figure 11.11), we begin by introducing a grid of equally spaced x- and /-coordinates for the rectangular region $0 \leqslant x \leqslant L, 0 \leqslant t \leqslant T$:

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N+1} \quad \triangle x_i \equiv x_i - x_{i-1} = h$$

[7] Thus, in terms of the original variables of the PDE, the regions on which finite difference methods can be used to solve problems must be rectangular (if there are two variables, or n-dimensional box shapes if there are n variables). Coordinate transforms (such as polar coordinates) can allow for other sorts of shapes. One of the key advantages of the finite element methods that we will introduce in the next chapter is that they allow the solution of PDE problems on more complicated geometrical configurations.

$$0 = t_0 < t_1 < t_2 < \cdots < t_{M+1} \quad \triangle x_i \equiv t_i - t_{i-1} = k \tag{1.19}$$

By using the central difference formulas (see Lemma 10.3) in the wave equation, we get the following discretization of it:

$$\frac{u(x_i,t_{j+1}) - 2u(x_i,t_j) + u(x_i,t_{j-1})}{k^2} = c^2 \frac{u(x_i,t_{j+1}) - 2u(x_i,t_j) + u(x_i,t_{j-1})}{h^2} \tag{1.20}$$

We recall that the truncation errors here are $0(k^2)$ and $0(h^2)$, respectively. Using the notation:

$$u_{i,j} = u(x_i,t_j),$$

and introducing the parameter

$$\mu = ck/h, \tag{1.21}$$

we may express (20) in the following simplified form:

$$u_{i,j+1} - 2u_{i,j} + u_{i,j-1} - \mu^2[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}] = 0 \tag{1.22}$$

We next solve this equation for the unique term corresponding to the highest time value to obtain:

$$u_{i,j+1} = 2(l-\mu^2)u_{i,j} + \mu^2[u_{i+l,j} + u_{i-l,j}] - u_{u,j-1} \tag{1.23}$$

for $i = 1,2,...,N$, and $y = 1,2,...,M$. The endpoint boundary conditions tell us that:

$$.u_{0.j} = 0 = u_{N,j}, \quad for\ all\ j \tag{1.24}$$

It follows from (22) and (23) that we may represent the time level j + 1 functional values in terms of the previous two time level functional values by means of the following tridiagonal linear system:

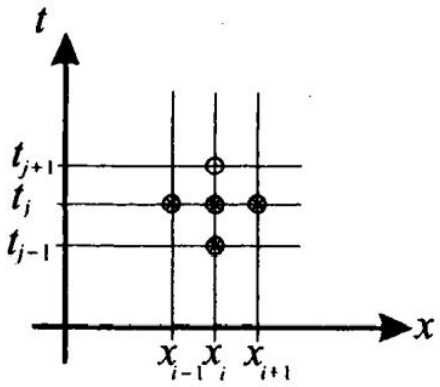

Figure 1.11: Illustration of the computational stencil for the discretization (20), (21) of the wave equation. The single point with largest time coordinate is emphasized, since the finite difference method will solve for it using the values of the solution at the previously found lower time grid points.

$$\begin{bmatrix} u_{1,j+1} \\ u_{2,j+1} \\ \vdots \\ u_{N-1,j+1} \\ u_{N,j+1} \end{bmatrix} = \begin{bmatrix} 2(l-\mu^2) & \mu^2 & 0 & \cdots & 0 \\ \mu^2 & 2(l-\mu^2) & \mu^2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & & & \mu^2 \\ 0 & \cdots & 0 & \mu^2 & 2(l-\mu^2) \end{bmatrix} \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{N-1,j} \\ u_{N,j} \end{bmatrix} - \begin{bmatrix} u_{1,j-1} \\ u_{2,j-1} \\ \vdots \\ u_{N-1,j-1} \\ u_{N,j-1} \end{bmatrix} \cdots \tag{1.25}$$

Such a scheme is referred to as an explicit three-level scheme, explicit since the highest $(t = (j+1)l)$ level values are explicitly solved in terms of the lower level values; three-level simply means that the nodal values involved in the scheme span over three time levels $(t = (J-1)k, jk, (j+1)k)$. This scheme will progress by iterating as we march upward in time. In order to start this recursion, we will need the functional values at the first two time levels $t = 0$ and $t = k(= t_1)$. These are the two column vectors on the right when j = 1. At time / = 0, these values are specified by the initial condition $u(x,0) = \phi(x)$ (initial wave profile) of (11) and this gives us:

$$u_{i,0} = \phi(x_1) \ \ for\ \ i = 1,2,\cdots,N. \tag{1.26}$$

In order to get the required next time level functional values we will need to make use of the initial wave velocity condition of (11): $u_t(x,0) = V(x)$. The fact that this extra information is actually required (unlike in the elliptic case)

is consistent with the fact that the wave problem (11) is well posed. To use this initial velocity to approximate the time level $t = k$ functional values, we will need another difference formula for approximation of derivatives; either the forward or backward difference formulas (Lemma 11.5) will give us what we need. For reasons that will soon be apparent, we choose to use the forward difference formula here.

For a fixed value of JC, and treating u(x>t) as a function of/, the forward difference formula implies that:

$$v(x) = u_t(x,0) \approx (u(x,k) - u(x,0))/k \Rightarrow u(x,k) \approx u(x,0) + kv(x)$$

(this is nothing more than the usual tangent line approximation). In terms of our grid functional values this translates to:

$$u_{i,t} \approx ui,0 + kv(x_i) \;\; for \;\; i = 1,2,\cdots,N. \tag{1.27}$$

Note that (viz. Lemma 11.5) the error of this approximation is $0(k)$, which is of lower order and hence potentially much greater than the $0(h^2+k^2)$ local truncation error for (23). Thus, this lower quality estimate for the / = k time level values (needed to start (23)) could contaminate the overall quality of (23). This problem can be avoided since the approximation can be improved to have error $0(k^2)$ (thus matching those in the foregoing development) if we furthermore assume that the wave equation is valid on the initial line and is sufficiently differentiate. Indeed, based on the differentiability assumption, (the onevariable) Taylor's theorem from Chapter 2 allows us to write:

$$u(x,k) = u(x_1,0) + ku_t(x_1,0) + \frac{k^2}{6}u_u(x_i,0) + \frac{k^3}{6}u_{ttt}(x_i,\hat{k},$$

where $\hat{k}$ is a number between 0 and k. The assumption that the wave equation is valid on the initial line tells us that$u_u(x,0) = c^2u_{xx}(x,0) = c^2\phi^n(x)$and we are led to the following approximation

$$u_{i,l} \approx u_{i,0} + kv(x_i) + \frac{c^2k^2}{2}\phi^2(x_i) \;\; for \;\; i = 1,2,\cdots,N, \tag{1.28}$$

with error bound $0(k^2)$. [8] To avoid computation of derivatives, we may approximate $\phi(x_i)$ using the central difference formula: $\phi^n(x_i)[\phi(x_{j+l}) - 2\phi(x_i) + \phi(x_{i-1})]/h^2$ (with error $0(h^)$). Installing this approximation into (28) produces the following practical approximating formula for the time level $t = 1$ functional values:

$$u_{i,l} = (l-\mu^2)\phi(x_1) + \frac{\mu^2}{2}[\phi(x_{i-1})] + kv(x_1) \;\; for \;\; i = 1,2,\cdots,N, \tag{1.29}$$

which has local truncation error $0(h^2+k^2)$. The next exercise for the reader gives us another way to arrive at the above $0(k^2)$ approximations for $u_{j,l}$ and show it to be valid under slightly different assumptions.

EXERCISE FOR THE READER 12.6: (a) Use Taylor's theorem to establish the following **centered difference approximation**: Suppose that $f(x)$ is a function having a continuous third derivative in the interval$a-h \leqslant x \leqslant a+h$ then the following approximation for $f'(x)$ is valid:

$$f'(x)\frac{f(x+h) - f(x-h)}{2h} + O(h^2). \tag{1.30}$$

Note that this is a second-order approximation to $f'(x)$, whereas the forward and backward difference approximation are only first-order approximations. (b) Using the artifice of ghost nodes (introduced in Section 11.4), obtain estimate (29) (with local truncation error $0(h^2+k^2)$) under the assumption that the solution $u(x,t)$ of the Cauchy problem (11)extends to have a continuous third order time derivative for $t \geq -k$.
Suggestion: For part (b), introduce a line of nodes at level $t = -k$ and denote the ghost values of u on these nodes by $u_{i,l}$. The centered difference approximationgives the estimate $u_{i,l} - u_{i,-l} \approx 2kv(x_i)$ that has error $0(k^2)$.

Even with all of the above attention to detail in developing a finite difference method with $0(h^2+k^2)$ local truncation error, stability issues can seriously corrupt the method. The next example will give good evidence of how badly things can go.

EXAMPLE 12.4:

(Illustration of Instability) Consider the following Cauchy problem of a long plucked string:

$$\begin{cases} (PDE)u_u = 4u_{xx}, & -\infty < x < \infty, 0 < t\infty, u = u(x,t) \\ (BCs)u(x,0) = \phi(x), u_t(x,0) = v(x) & -\infty < x < \infty, 0 \leqslant t < \infty' \end{cases}$$

[8] This is the reason we choose to adopt the forward over the backward difference method. We would not have been able to make such a local error truncation if we had used the backward difference method.

where $\phi(x)$ is as in Example 12.1 (Figure 12.4). Note this problem is identical to the problem of Example 12.1 except that the wave speed has changed from $c = 1$ to $c = 2$. By D'Alembert's solution (Theorem 12.1), we know the exact solution of this problem is given by (from (10)):

$$u(x,t) = \frac{1}{2}[\phi(x+2t) + \phi(x-2t)]$$

and the solution will look just like the one shown in Figure 12.5, except now the speed is doubled. Thus, for any specified range of time values, we can view this problem as taking place on a finite string centered at $x = 0$ of sufficiently large length.
(a) Apply the above finite difference scheme (22) with $h = k = 1$ up to time level $t = 5k(= 5)$, and $-12 \leqslant x \leqslant 2$. To isolate just the effectiveness of (22), use the exact values for the time level $t = k$ values, as determined by D'Alembert's solution, in place of (28). Examine the u-values and compare with those of the exact solution (cf. Figure 12.5).
(b) Repeat Part (a) with $= k = 0.1$ up to time level $t = 50k(= 5)$.
(c) Repeat Part (a) with $h = 1, k = .5$ up to time level $t = 10k(= 5)$.

SOLUTION: Part (a): With $c = 2$, we have, by (21), $\mu = ck/h = 2$, so that (23) becomes:

$$u_{i,j+1} = -6u_{i,j} + 4[u_{i+l,j} + u_{i-l,j}] - u_{i,j-1}$$

Since $h = 1$, we get $x_0 = -12, x_{12} = 0, x_{25} = 12$ so by (10) (exact solution), we may write
$u_{i,0} \begin{cases} 2, & i = 12 \\ 0, & otherwise \end{cases}$ and $u_{i,0} \begin{cases} 1, & i = 12 \pm 2 \\ 0, & otherwise \end{cases}$. Note that by (22), the of indices with nonzero w-values can advance only one index to the left/right with each new time level. The following MATLAB loop will produce the needed nonzero w-values up to time level $t = 5k$. The instability is so severe that it is convenient to view the matrix of values. In creating the 6×25 matrix of nodal values, we let the bottom row correspond to the time level zero values and so the top row corresponds to the $t = 5$ values. Note this requires us to modify the of (23) accordingly in our MATLAB code below:

```
>> U=zeros(6,25); U(6,12)=2 ; U(5,[1 0 14])=1 ;
>> for j=5:-1: 2
     for i=2:2 4
        U(j-1,i)=-6*U+4*[U(j,i+1)+U<j,i-1) ]-U(j+1,i);
end
end
```

The nonzero matrix values are shown below. Note that the actual solution has two pulses of height 1 moving from left to right at speed two. The numerical solution below is totally off and unstable, it oscillates out of control. Also, the disturbances only propagate at speed one.

256	-1536	4432	-8048	10373	-10688	10424	-10688	10373	-8048	4432	-1536	256
0	64	-288	616	-812	776	-710	776	-812	616	-288	64	0
0	0	16	-48	67	-56	44	-56	67	-48	16	0	0
0	0	0	4	-6	4	-2	4	-6	4	0	0	0
0	0	0	0	1	0	0	0	1	0	0	0	0
0	0	0	0	0	0	2	0	0	0	0	0	0

Part (b): Since c and μ are still 2, (22) takes the same form as in part (a), but since $x_0 = -12, x_{120} = 0, x_{241} = 12$, and we have

$$u_{i,0} \begin{cases} 2, & |120 - i|/5, \ \ 110 \leqslant i \leqslant 130 \\ 0, & otherwise \end{cases}$$

To get uiX, we note that (10) and the initial values give us (since h = k = 0.1) that $u_{i,t} = u_{i+2,0} + u_{i-2,0}$ It is most simple to use a MATLAB loop to compute these values before entering into the main loop based on (23). Using the matrix conventions of part (a), the construction of the matrix of values can be accomplished in MATLAB with the following commands:

```
>>U=zeros(51,251);
for i=110:130
  U(51,i)=2-abs(i-120)/5;
end
  for i=108:132
  U(50,i)=U(51,i+2)+U(51,i-2);
end
for j=50:-1:2
  for i=2:250
    U(j-1,i)=-6*U(j,i)+4MU(j,i + l)+U<j,i-l)]-U(j + l,i);
  end
end
```

-45.2	416.8	-842.8	1091.2	-1008.8	920	-1008.8	1091.2	-842.8	416.8	-45.2
5	-19.2	71	-84.8	77.8	-84.8	71	-19.6	5	616	-288
4	4.8	-0/8	12.8	-0.4	7.2	-0.4	12.8	-0.8	4.8	4
3	3.6	4.2	3.2	4.6	4.4	4.6	3.2	4.2	3.6	3
2	2.4	2.8	3.2	3.2	3.2	3.2	3.2	2.8	2.4	2
1	1.2	1.4	1.6	1.8	2	1.8	1.6	1.4	1.2	1

To see that the numerical solution is still badly unstable, we need only look at the middle portion of the last six rows of the matrix (corresponding to the time range: $0 \leqslant t \leqslant .5$):

Indeed, this shows that even at time level t = 0.5, the profile oscillates rapidly between ± 1000.

Part (c): Since k is now half of h, we have $\mu = 1$, so that (23) takes the following form:

$$u_{i,j+1} = u_{i,j+1} + u_{i,j-1} - u_{i,j-1}$$

Since $h = 1$, we get $x_0 = -12, x_{-l2} = 0, x_{25} = 12$ as in part (a), and from (10) (exact solution), we may write $u_{i,0} \begin{cases} 2, & i = 12 \\ 0, & otherwise \end{cases}$

and $u_{i,0} \begin{cases} 1, & i = 12 \pm 1 \\ 0, & otherwise \end{cases}$ Note that by (23), the set of indices with nonzero w-values can advance only one index to the left/right with each new time level. The construction of the 11×25 matrix of u-values is done as before and the relevant entries are displayed below.

```
U=zeros(ll,25);U(11,12)=2; U(10,[11 13])=1;
for j=10:-1:2
  for i=2:24
    U(j-l,i)=U(j,i+l)+U(jf i-l)-U(j+l,i);
  end
end
```

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0

Note the rather surprising results! The result of part (c) quite well represents the actual solution (up to the resolution on the jc-grid). The results of parts (a) and (b) were totally unstable, and despite the fact that the grid of part (b) was much finer (in both variables) than that for part (c), the grid for part (c) turned out to give a much more stable method. It turns out that the relative ratio of h and k, not their actual sizes, is what will make or break stability. Such remarkable phenomena did not occur when we applied finite difference methods to elliptic problems in the last chapter.

The finite difference methods shown above can be proved to converge to the exact solution of the wave problem (11) (as the partitions become more and more refined) provided that, in addition to the required differentiability assumptions, the following Courant-Friedrichs-Levy (CFL) condition holds:

$$\mu \equiv ck/h \leqslant 1. \tag{1.31}$$

If this condition is violated (i.e., if $\mu > 1$), examples can be constructed where (as in Example 12.4), although all other differentiability assumptions are satisfied, the finite difference approximations will not converge to the exact solution, even as the mesh sizes of both variables tend to zero! In fact when $\mu > 1$, the method is unstable in the sense that errors made at each time stage of the process can significantly affect the subsequent time numerical values. For complete details and proofs on these matters we refer to Section 9.3.1 of [IsKe-66]. The exercises will include an outline of the proof and in the next section we will give some details of the analogous theory for the heat equation.

We give here a nontechnical explanation of why such instability can arise. From (23), the numerical values at a new time level at x. depend on those of previous two levels, which lie at most one horizontal step to the left and right of x_i. From this we can determine the "numerical interval of dependence" of a grid point $(x_i > t_{j+1})$, analogously to

how we defined the interval of dependence of the exact solution (see Figure 12.12). From Figure 12.12, we can see how violation of the condition can lead to instability of the numerical method. What this amounts to is that the size of the t-steps (= k) is too large relative to the size of the x-steps h). This means that the numerical interval of dependence (shown by the black double arrowed segment) for (x_i, y_{j+1}) is smaller than the theoretical interval of dependence (green arrowed segment). We know from the theory in thelast section that the numerical interval of dependence thus does not take enough information into account to properly formulate the approximations. This can lead to catastrophic results, as we have seen. The diagonally upward black arrows indicate flows of information in the finite difference scheme.

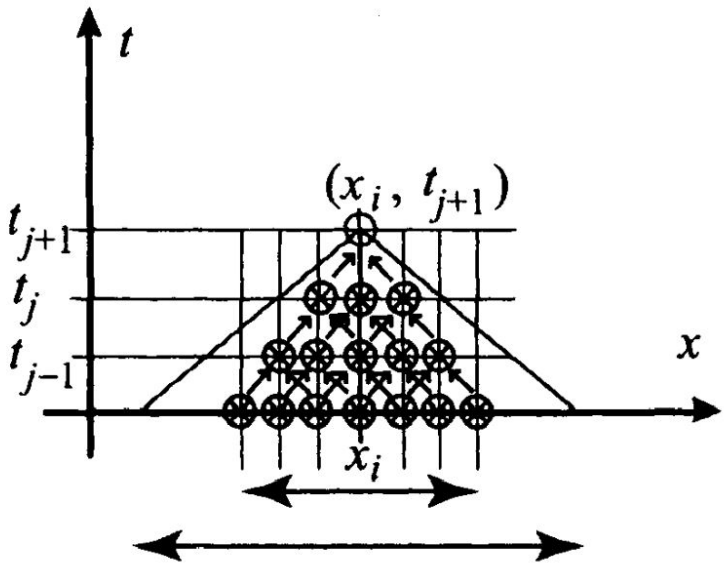

Figure 1.12: Illustration of the problem when the Courant-Friedrichs-Levy (CFL) condition is violated

In both parts (a) and (b) of Example 12.4, we had $h = k$, so that (since $c = 2$), $\mu =$ and the Courant-Friedrichs-Levy conditionis violated. We can see that the numerical wave profiles propagate at only h units (to left and right) for each k unit time level increase. Thus, the numerical profiles cannot keep up with the actual wave propagation (two A units left and right of space for each k unit of time) and the scheme goes haywire. In part (c), however,$k = h/2$ and now the numerical scheme can keep up, and it does so quite well in that example. It turns out that when the problem is a smooth one, taking step sizes so that $\mu = 1$ can greatly enhance the accuracy of the scheme. It seems quite surprising that for a given (stable) choice of A and k, fixing h and decreasing k can sometimes have a detrimental effect on the numerical solution. Shortly, we will introduce an implicit scheme that has better stability properties.

Recall that the finite difference scheme for elliptic PDEs that we used in the last chapter was implicit and very stable; also, in Part II, we saw that implicit schemes for ODE problems, although more difficult to work with, had better stability properties than explicit schemes. This is a general rule: Implicit schemes are more stable than explicit schemes in numerical differential equations. One advantage of explicit schemes, however, is that many are easily adapted to effectively solve nonlinear problems (provided stability requirements are met). Although we will not enter into any detailed discussion of stability issues for nonlinear PDEs, we will occasionally try to adapt some of our linear schemes to solving nonlinear problems. Often this is what is done in practice. Indeed, a nonlinear problem, when looked at locally (in a small portion of the domain), can be approximated by a linear problem and the latter one dealt with according to linear schemes. For more on nonlinear PDEs, we cite the reference [Log-94]. A more advanced treatment is given in [Smo-83].

Numerical methods for nonlinear PDEs is an extremely active area of mathematical research. We caution the reader that many reasonable-looking finite difference schemes may do poorly for a given nonlinear problem. In general those that are based on conservation laws (physical principles) are the most successful. This seems to imply that a purely mathematical approach to the numerical solution of nonlinear PDEs is not sufficient; an additional requirement is a certain knowledge of the physical principles governing the phenomena that are modeled by the PDEs. For a detailed investigation of such issues, we refer to the two nvolume set [Tho-95a],[Tho-95b]. The book [Dur-99] gives a detailed treatment of various numerical methods for wave (hyperbolic) problems. A particular nonlinear one-dimensional wave equation with a conservation law based finite difference method is nicely examined in [StVa-78].

We now proceed to write a function M-file that will apply the above finite difference scheme to solve a more general version of the wave problem (11) which allows a certain nonlinearity in the PDEs. Specifically, we allow the ends of the wave to have time-dependent variable heights and we allow the coefficient c (wave speed) to depend on /, JC, and/or u. The former conditions mean that we allow forced control on each of the string ends; the more general assumption on c corresponds physically to having a string whose characteristics are not uniform in x (e.g., it could be thicker in some

places than in others), are time dependent (e.g., it could weaken or strengthen with time), and even depend on the current position and slope of the string (e.g., the properties of the string may weaken, depending on its composition, in areas where there is a steep slope stretch.)

In Program 12.1, the main change will be that when we use (29), we need take note of the fact that $\mu - ck/h$ is now no longer (necessarily) constant: $\mu\mu_{i,j} = c(t_j, x_i, u_{ij}, (u_x)_{ij})k/h$. In the fourth argument of c we use the centered difference approximation: $(u_x)_{ij} \approx [u_{i,j} - u_{i-1,j}]/2h$. The resulting (23) will still be an explicit one.

PROGRAM 12.1: Function M-file for solution of the following wave problem by the finite difference method, [9]

$$\begin{cases} (PDE) u_n = c(t,x,u,u_x)^2 u_{xx}, \ 0 < x < L, \ 0 < t < \infty, \ u = u(x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), u_t(x,0) = v(x) \\ u(x,t) = A(t), \ u(L,t) = B(t) \end{cases} \quad 0 < x < L, 0 \leqslant t < \infty \end{cases} \tag{1.32}$$

This program uses the improved approximation (29) for the level-one time values that work better under greater differentiability hypotheses on the initial data. A more basic program, which uses (27) in place of (29), is left to the following exercise for the reader; it is recommended over this one in case the initial conditions possess singularities. The program assumes that the Courant-Friedrichs-Levy condition has been checked to be satisfied in the region under consideration.

```
function [x, t, U] = onedimwave(phi, nu, L, A, B, T, N, M, c)
solves the one-dimensional wave problem u tt - c(t,x,u,u x)N2*u
Input variables: phi-phi(x) - initial wave profilo function
nu=nu(x) = initial wave velocity function, L = length of string,
= Alt) height function of left end of .string u(0,t)=A(t), B=E(t)
he i gnt function for right end of string u(L, t)-B, T~ final time
which solution will be computed, N - number of internal x-grid
values, M - number of internal t-grid values, c -c(t,x,u,u x) -
speed of wave. Functions of the indicated variables must be
stored as (either inline or M-file) functions with the same
variables, in the same order.
Output variables: t - time grid row vector (starts at t-0, ends
t^T, has M+2 equally spaced values) , x - space grid row vector,
(M*2) by (M*2) matrix of solution approximations at corresponding
grid points; y grid will correspond to first (row) indices of U,
grid values to second (column) indices of U.
CAUTION: For stability of the method, the Couiant-Friedrichs-Levy
condition should hold: c(x,t,u,u x)(T/L)(N+l)/(M+l)<1

h = L/(N+1); k = T/(M+1);
```

```
U=zeros(M+2,N+2); x=0:h:L; t=0:k:T;
   Recall matrix indices must start at 1. Thus the
   matrix will always be one more than the corrospcn
   were used in theoretical development.

  Assign 1 ef t. and r ight i;i r i.oh 1 et boundar y va 1 ues .
U(:,l)=feval(A,t)f ; U(:,N+2)=feval (B,t) ';

  Assign initial time t=0 values and next step t = k v
for i=2:(N+l)
  U(l,i)=feval(phi,x(i));
  mu(i)=k*feval(c,0,x(i), U(l,i),(feval(phi,x(i+1))
l)))/2/h)/h;
  U(2,i) = (l-mu(i) A2)*feval(phi,x(i) ) +mu (i) A 2/2* (feval(phi,x(i-1)
   )+
...
  feval(phi,x(i+l))) + k*feval(nu,x(i));
end

  Assign values at interior grid points
for j=3:(M+2)
for i=2:(N+l)
  mu(i)=k*feval(c,t(j),x(i),U(j-l,i), (U (j-1, i + 1) -U (
  First form needed tridiagonai matri
  Tri = diag(2*(l-mu(2:N+l).A 2)) + diag(mu(3:N+1).A
diag(mu(2:N). A 2 , 1) ;
  Now perform the matrix multiplications to iterat
  solution values for increasing time levels.
  U(j,2: (N+l))*(Tri*(U(j-l,2:(N+l)") ')) '-U(j-2,2: (N+
  U(j,2)=U(j,2)+mu(2) A2*feval(A,t(j-1)) ;
  U(j,N+l)=U(j,N+l)+mu(N+l)A 2*feval(B,t(j-l));
end
end
```

[9] Although we have not yet made explicit mention of the incorporation of boundary conditions into finite difference schemes for the wave equation, this is a rather obvious extension of ideas presented in the previous chapter. Program 12.1 provides an example of such a feature.

As was implicit in Program 12.1, we point out that the Courant-Friedrichs-Levy (CFL) condition can be expressed using the input parameters in the above M-flle in the following way:

$$\mu = c(t,x,u,u_x)\frac{T}{L}(\frac{N+1}{M+1}) \leqslant 1 \tag{1.33}$$

The actual M-file is quite short. In the next example we will test both the accuracy runtime of this program with a wave problem with nicely smooth input data and whose exact solution is available to compute errors. It will also demonstrate some interesting pathologies when played against the Courant-Friedrichs-Levy condition.

EXAMPLE 12.5:

Use Program 12.1 to solve the following wave problem on the time interval $0 \leqslant t \leqslant 2$:

$$\begin{cases} (PDE)U_{tt} = u_{xx}, & 0 < x\pi, 0 < t < \infty, u = u(x,t) \\ (BCs)\begin{cases} u(x,0) = sinx, u_t(x,0) = 0 \\ u(x,t)0, \;\; u(\pi,0) = 0 \end{cases} & 0 < x < \pi, 0 \leqslant t < \infty \end{cases}$$

using the following grid sizes. In each case, compare the results with the actual solution $u(x,t) = costsinx$ on the indicated time levels. If the graphs are too close to discern differences, compute the maximum error numerically.
(a) $N = 10, M = 15$. Note that this set of parameters slightly violates the Courant-Friedrichs-Levy condition. Compare the numerical solution with the exact solution at time levels $t = 0.5, t = 1, t = 1.5, t = 2$.
(b) $N = 10, M = 29$. Note that this set of parameters satisfies the Courant-Friedrichs-Levy condition. Compare numerical solution with exact solution at time levels $t = 4$ and $t = 8$. (c) $N = 100, M - 15$. Note that this set of parameters strongly violates the Courant-Friedrichs-Levy condition (31) ($\mu = 16.0746$).

SOLUTION: We create three sets of data for each of the three sets of parameters and label them differently for future use.

We first create inline functions for the initial data of this wave problem:

```
>> phi = inline('sin(x)');
>> nu= inline ('0'); A=nu; B=A; c=inline('1', 't,'x','u', 'ux');
-> nu = Inline function:
    nu(x) = 0
```

We now create the numerical solutions for each of the three parts:

```
>> (xl, tl, Ul] = onedimwave(phi, nu, pi, A, B, 8, 10, 15, c) ;
>> [x2, t2, U2] = onedimwave(phi, nu, pi, A, B, 8, 10, 29, c);
>> fx3, t3, U3] = onedimwave(phi, nu, pi, A, B, 8, 100, 15, c) ;
```

Part (a): To produce the desired snapshots, we take note of the general relationships between t and j: $k = \frac{2}{M+1}, t_j = jk = \frac{2j}{M+1}$ so $j = \frac{(m+1)t_j}{2}$. Thus, when M = 15, we have y = 8ry, so the values f = 0.5, 1.0, 1.5, and 2.0 correspond respectively to the indices $y = 4, y = 8, y = 12$ and $j = 16$. Since the MATLAB indices are one greater than these actual indices, we may create and plot the desired numerical snapshots as follows:

```
>> subplot(1,4,1)
>> plot(xl, Ul(5, :)), axis([0 pi -1 l]),hold on
>> plot(xl, cos(2)*sin(xl),'r')
>> subplot(l,4,2)
>> plot(xl, Ul(9, :)), axis([0 pi -1 l]),hold on
>> plot(xl, cos(4)*sin(xl),'r')
>> subplot(l,4,3)
>> plot(xl, Ul(13, :)), axis([0 pi -1 l]),hold on
>> plot(xl, cos(6)*sin(xl),'r')
>> subplot(1,4,4)
>> plot(xl, Ul(17, :)), axis([0 pi -1 l]),hold on
>> plot(xl, cos(8)*sin(xl),'r')
```

We have set the axes to an appropriate setting for comparisons and used the horizontal stacking of the subplot so as to make the vertical errors more detectable. The resulting graphic is shown in Figure 12.13.

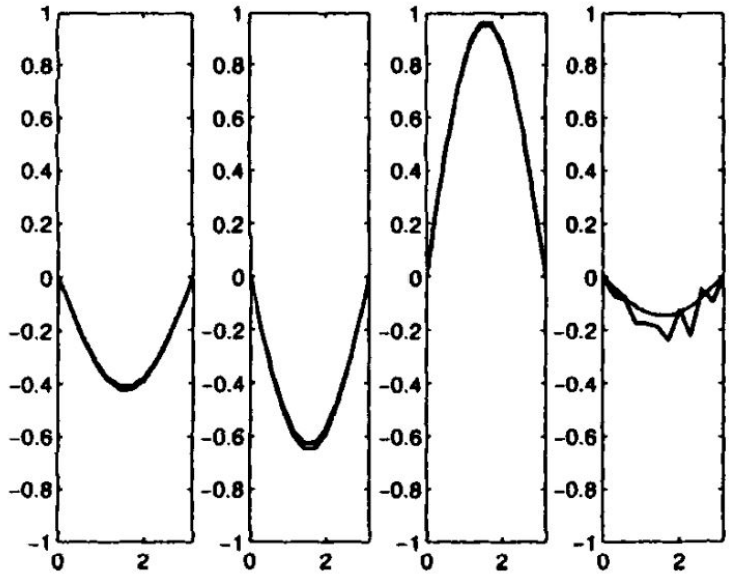

Figure 1.13: Comparison of the computed finite difference solution's snapshots jagged) with the actual solution's snapshots (smooth) for the wave problem of Example 12.5. The four plots correspond to snapshots at levels $t = 0.5, t = 1, t = 1.5, t = 2$, respectively. The numerical solution was obtained using $N = 10$ interior grid points for x and M- 15 interior grid points for x, which resulted in a violation of the Courant-Friedrichs-Levy condition (31) with $\mu = 1.75... > 1$. All except the last profile show the numerical snapshots to be reasonably decent with only small errors that are visible to the naked eye. At $t = 2$, however, the numerical graph starts to break its pattern and relative errors reach orders of magnitude of 100%. Time levels (from left to right) are $t = 0.5, t = 1, t = 1.5$, and $t = 2$

Part (b): In this case, if we plot (as in part (a)) and compare the numerical solution with the actual solution, the results are indistinguishable at both time levels / = 1 and t = 2. To compute the maximum absolute values of the differences, using again the index relation $j = \dfrac{(m+1)t_j}{2}$ (and adding one to j to get MATLAB's indices) we enter the following commands:

```
>> max(max(abs(U2(:,16)'-cos(4)*sin(x2))))
->ans = 0.00131359012313
>>max(max(abs(U2(:,31)'-cos(8)*sin(x2))))
->ans = 0.00343796574347
```

The results are rather accurate considering the somewhat large step sizes (especially in t).
Part (c): Plotting the numerical solution's snapshot at time level / = 2 is accomplished with the command:

```
»plot(x3 , U3(17, :))
```

The rather surprising result is shown in Figure 12.14.

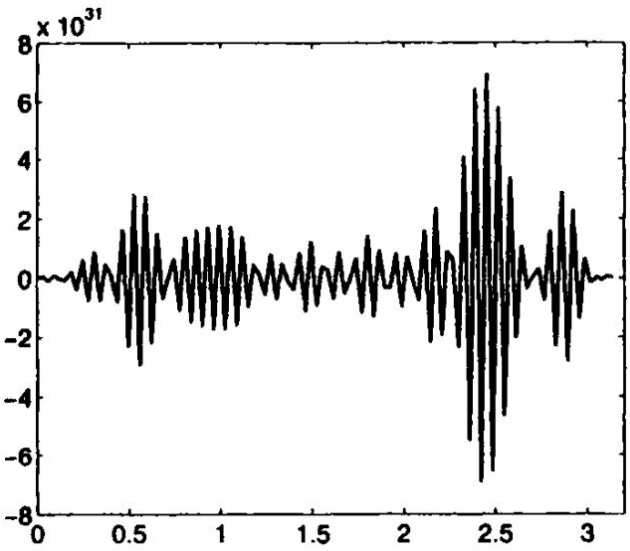

Figure 1.14: Plot of the snapshot of the numerical solution for part (c) of the wave problem of Example 12.5, using $N = 100$ interior grid points for x and $M = 15$ interior grid points for t, which resulted in a serious violation of the Courant-Friedrichs-Levy condition (31) with $\mu = 16.07... > 1$. Note the amplitude of the graph is 31 orders of magnitude greater than the actual solution, so the result is quite meaningless. Note also that the grid used was actually finer than that used in part (a) (which gave much better results). Thus, blindly refining grids can lead to disastrous results that use more computing time, unless the Courant-Friedrichs-Levy condition is respected.

The next exercise for the reader will show how, with a bit finer of a grid on the time axis (and keeping the same grid on the JC- axis) we can arrive at numerical with the above program that are numerically indistinguishable from the actual solution. We also point out that the above program is able to handle grids for both x and t with over 1000 points in a reasonable amount of time (a few minutes). This is quite different from the situation for the finite difference methods for elliptic PDEs discussed in the previous chapter. Recall that in the algorithm for elliptic PDEs, it was required to solve a linear system of order roughly $N \cdot M$ to simultaneously solve for the numerical solution at all interior grid values. Numerically solving parabolic equations will also take far fewer computations than do elliptic equations with similar grids, and this is another reason we have grouped hyperbolic and parabolic PDEs together in this chapter.

EXERCISE FOR THE READER 12.7:
(a) Modify Program 12.1 into one that uses (27) in place of (29) for the approximation of the function on the level t = k time line. Call this modified function `onedimwavebasic` .
(b) Starting with N = 10 interior jt-grid points, begin with Af= 30 interior /-grid points and re-solve the wave problem of Example 12.5, with both the `onedimwave` program and your newly constructed `onedimwavebasic` program. Compare with the exact solution at / = 8. Continue to double M (keeping N fixed) until you have completed nine doublings of M Collect the graphs of the resulting errors (at $t = 8$) in a separate 5×2 partitioned (by `subplot`) window. Repeat with N = 40. Compare and contrast the graphical results, and comment on any observed instability.

Physically, both hyperbolic and parabolic problems model time-dependent . The main difference between them is that while parabolic phenomena are dissipative, solutions to hyperbolic PDEs are conservative. In particular, initial singularities are smoothed out and lost with time under a parabolic PDE and are preserved and propagated under a hyperbolic PDE. Since finite difference schemes tend to average things out (we saw a good example of this in the previous chapter when we showed maximum principles hold for elliptic finite difference schemes), this suggests that finite difference schemes may run into problems for hyperbolic problems with discontinuous data. This is indeed the case, and for this reason, there are other methods that are more suitable for hyperbolic problems with discontinuous data. Examples of alternative methods suitable for hyperbolic problems with singularities include the method of characteristics (the D' Alembert method of the previous section is a special case), and the method of lines. More can be found on such methods in [Abb-66], [Dur-99], and [Ame-77]. Discontinuous data is very natural in hyperbolic problems modeling events such as shocks, explosions, or earthquakes. Our next example will show some typical pathologies that can occur when the finite difference method is applied to a discontinuous problem.

EXAMPLE 12.6:

Consider the following wave problem:

$$\begin{cases} (PDE) u_u = c^2 u_{xx}, & 0 < x < 5, 0 < t\infty, u = u(x,t) \\ (BCs) \begin{cases} u(x,0) = 0(x), u_t(x,0) = v(x) \\ u(0,t) = A(t), \ u(L,t) = 0 \end{cases} & ,0 < x < 5, \ 0 \leqslant t < \infty \end{cases}$$

where $c(x) \begin{cases} 2, & if\ 0 \leqslant x \leqslant 3 \\ 1, & if\ x \geq 3 \end{cases}$ and$A(t) \begin{cases} sin(5t), & if\ 0 \leqslant t \leqslant \pi/5 \\ 0, & if\ t > \pi/5 \end{cases}$. Physically this wave problem can be thought of as that of a string of length 5 that is made up of two materials which are glued together at x = 3. The right portion is more dense than the left (recall c is inversely proportional to $1/\sqrt{\rho}, \rho$ mass density of string). The string is initially at rest, and for a short period of time the left end is "moved" upward and then back down to the original position and held there. The right end of the string remains pinned down. Use the above program onedimwave, respecting the Courant-Friedrichs-Levy condition, to numerically solve this problem on the time range $0 \leqslant t \leqslant 5$ using finer and finer grids until the solutions become visually consistent Plot a series of snapshots of the wave profiles.

SOLUTION: In cases where the wave speed c is nonconstant, we must replace c with its maximum value (worst-case scenario) in the CFL condition (33). This tells us that we should have

$$M + 1 \geq 2(N + 1)$$

We create M-files for $c(x)$ and $A(t)$, and inline functions for the remaining data:

```
function y =c_EG12_5(t,x,u,ux)
  for i = 1:length(x)
  if (0<=x(i))&(x(i)<=3)
    y(i)=2;
  elseif (3<x(i))&(x(i)<=5)
    y(i)=1;
  else
    y(i)=0;
  end
end
```

```
function y = Apulse_EG12_5 (t) ,
  for i * 1:length(t)
  if (0<=t(i))&(t(i)<=pi/5)
    y(i)=sin(5*t(i));
  else
    y(i)=0;
  end
end
```

```
» phi=inline('0'); nu=phi; B=nu;
```

After some experimentation with increasing resolution, the patterns of the solutions become clear. The code below produced the series of snapshots shown in Figure 12.15.

```
>> [x, t, U] = onedimwaveiphi, nu, 5, @Apulse_EG12_5, B, 5, 350,...
   800, @c_EG12_5);
>> count=l;
>> for k=l:80:800
  subplot(10,1,count)
  plot(x,U(k,:)), hold on, axis([0 5 -1.1 1.1]);
  text(5.1,0, [*t = ',num2str((k-l)/800*5),])
  count=count+l;
end
```

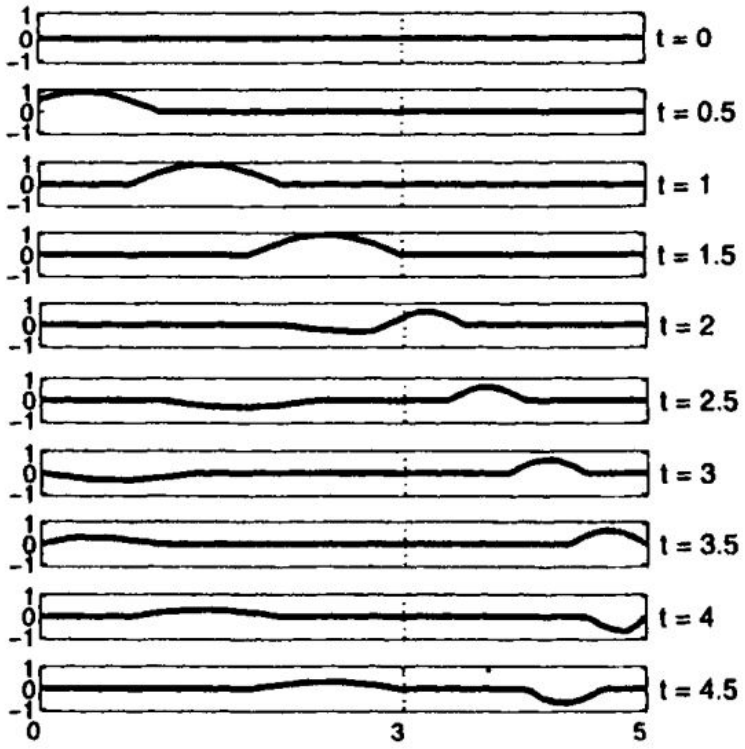

Figure 1.15: Series of snapshots for the solution of the "glued string" problem of Example 12.6. The interface at $x = 3$ (where the heavier string on the right meets the lighter one on the left) is emphasized with a grid. Note that when the wave reaches the heavier portion of the string, a secondary reflection takes place, and the original wavefront gets smaller and slows down.

We point out the very large numbers ofN= 350 and M= 500 that were used. The discontinuities in the data necessitated this. Indeed, Figure 12.5 shows a single plot of a snapshot of the above numerical solution at a late time `plot (x,U(780,:))`. This is in sharp contrast to the results of part (b) of the Example 12.5 (a wave problem with smooth data), where we got excellent accuracy with a very rough grid on the finite difference method. The general rule is that for hyperbolic problems with discontinuous data, finite difference methods will require a lot of work to get decent results. Even (many) nonlinear hyperbolic problems take less work if the data is smooth. The exercises will further explore these issues.

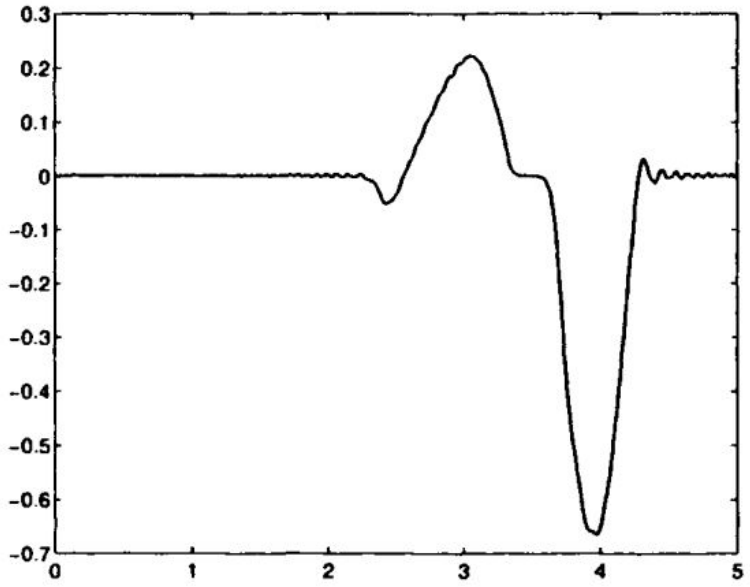

Figure 1.16: Wave profile near the end of the time interval $0 \leqslant t \leqslant 5$ of the numerical solution of Example 12.6. Note the small (secondary) oscillations. This type of numerical noise is not part of the exact solution; it arises since the finite difference method tries to smooth out certain discontinuities in the data of the problem.

EXERCISE FOR THE READER 12.8: The reader may observe that becauseof the discontinuities in the data, it

would have been more appropriate to use the less restrictive version of the finite difference method `onedimwavebasic` of Exercise for the Reader 12.7. Check that with this M-file, the results will be quite identical to those of the above example.

We turn now to briefly describe implicit finite difference methods. For simplicity, we describe them only for the basic wave equation $u_u = c^2 u_{xx}$ of (11). Their main advantage of allowing a more flexible choice of time step sizes is not so crucial here since the Courant-Friedrichs-Levy condition does not pose too stringent (small) a step size requirement on t. When we move on to parabolic equations in the next section, however, we will see that the stability of explicit methods requires a much smaller time step and so implicit methods will be a more attractive alternative. A family of such methods can be obtained by approximating utl with the centered difference:$u_u(x,t) \approx [u_{i,j} + 2u_{i,j} + u_{i,j-1}]/k^2$ but for u_{xx} we 558 use a weighted average of the centered difference approximations at the three levels: $t = t_{j-1}, t_j, t_{j+1}$

$$u_{xx}(x_i,t_j) \approx \omega[u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}]/h^2 + (1-2\omega)[u_{i+1,j} - 2u_{i,j} + u_{i-1,j}]/h^2 + \omega[u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}]/h^2 \tag{1.34}$$

where the parameter ω satisfies $0 < \omega < 1$. The choices $\omega = 1/2$ and $\omega = 1/4$ are the most popular since they lead to symmetric stencils. It can be shown that this scheme is unconditionally stable as long as $\omega > 1/4$. We caution the reader that the unconditional stability, the scheme may produce poor results if we use too large a time step size (relative to the space step size). For brevity, it is helpful to use the following notations for centered difference approximations:

$$\delta_x^2 u_{i,j} \equiv \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}, \delta_l^2 u_{i,j} \equiv \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2}, \tag{1.35}$$

Thus with these notations, the general **implicit (three-level) finite difference scheme** for the wave equation $u_u = c^2 u_{xx}$ can be expressed as:

$$\delta_x^2 u_{i,j} = c^2[\omega\delta_x^2 u_{i,j-1} + (1-2\omega)\delta_x^2 u_{i,j} + \omega\delta_x^2 u_{i,j+1}] \tag{1.36}$$

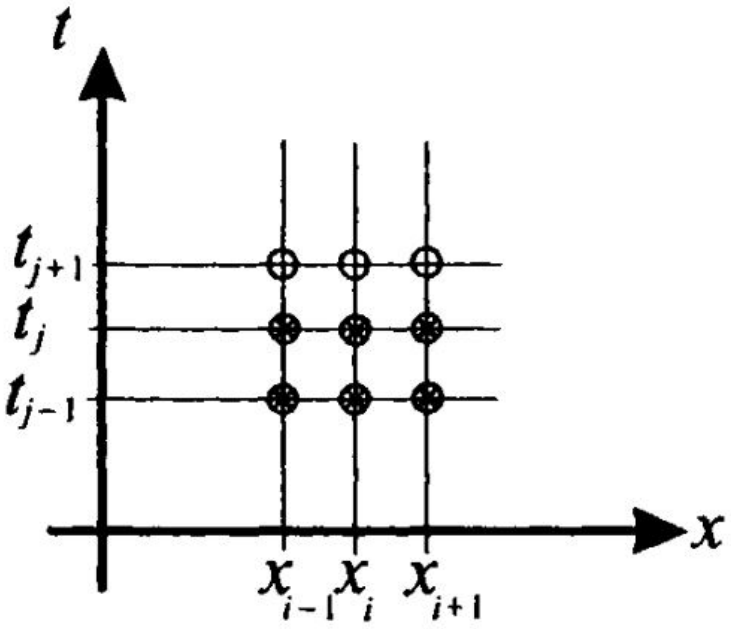

Figure 1.17: FIGURE 12.17: Stencil the implicit finite difference method (36) for the wave equation. At each iteration in the upward marching scheme, the hollow nodal values need to be determined. In the case where the parameter ω is $1/2$, the central row of nodes (time level t_j) is not present in the stencil.

When translated into a linear system, at each iteration, the scheme (36) leads to a tridiagonal system in the variables $u_{i,j+1}(1 \leqslant i \leqslant M)$ so the Thomas method can be used.

EXERCISE FOR THE READER 12.9: (a) Write a function M-file for applying the implicit finite difference method (36) with parameter $\omega = 1/4$. The syntax should be as follows:

`[x, t, U] =onedimawaveimpl(phi, nu, L, A, B, T, N, M, c)` where the variables and functionality should be just as with Program 12.1.
(b) Do some experiments on the problem of Example 12.6 comparing the performance of the implicit finite difference method of part (a) with the explicit method of Program 12.1 using values of N and M that would make the latter unstable.
(c) How fine a resolution is needed in the implicit finite difference method of part (a) to obtain graphical results of the

quality of the numerical solution created in Example 12.6? Do the results seem better when the CFL condition is satisfied? Do some experiments.

A wave problem, or for that matter, any PDE problem with two space variables (and the time variable) would require four dimensions to represent the graph of the solution. Snapshots, however, obtained by fixing the time at a certain level, can be graphed in three dimensions, and from these movies of wave propagation can be put together and viewed. Since such graphical investigations are particularly useful for understanding wave propagation, we proceed now to develop a finite difference method for solving wave equations in two space variables.

We consider the following two-dimensional wave problem on a rectangular domain:

$$\begin{cases} (PDE)u_u = c^2(u_{xx}+u_{yy}), \quad 0<x<a, 0<y<b, 0<t<\infty, u=(x,y,t) \\ (BC's)\begin{cases} u(x,y,0)=\phi(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty \\ u_t(x,y,0)=v(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty \\ u(x,y,t)=0, \text{for all (x, y) on theboundary of the rectangle:} \\ \qquad\qquad R=\{0\leqslant x\leqslant a, 0\leqslant y\leqslant b\} \end{cases} \end{cases} \tag{1.37}$$

For simplicity we have kept the edges of the wave fixed at height zero. We may visualize (37) as a problem for the vibrations of a flexible membrane of elastic material that has been stretched over the edges of the rectangular frame R, much like a drumhead. The initial conditions will produce a vibration of the drumhead, which will be governed by the wave PDE in (37). With this interpretation, it can be shown using physical principles that $= T/\rho$, where T is the tension of the membrane [10] and ρ is the mass per unit area of the membrane.

We restrict time to be bounded on some fixed interval: $Q\leqslant t\leqslant T$ (apologies for using the letter T for two different purposes; the distinction should be clear from the context). We introduce a grid for each variable under thefollowing notation:

$$\begin{gathered} 0=x_0<x_1<x_2<\cdots<x_{N_x+1}=a, \quad \triangle x_i - x_{i-1}=h \\ 0=yx_0<y_1<y_2<\cdots<y_{N_x+1}=b, \quad \triangle y_i - y_{i-1}=h \\ 0=t_0<t_1<t_2<t<_{M+1}=T, \triangle t_i \equiv t_i - t_{i-1}=k \end{gathered} \tag{1.38}$$

We have specialized to using the same step size h for $x-$ and y-coordinates. This, of course, is not always possible, depending on the dimensions of the rectangle R, but it will keep the notation more manageable. To keep the notation somewhat streamlined, we use superscripts to denote indices for time levels, and subscripts to indicate indices of space variables:

$$U^t_{i,j}=u(x_i,y_j,t_t).$$

Using this notation and applying the central difference formulas for approximating the PDE in (37), as we did in the one-dimensional case, we arrive at the following discretization of the two-dimensional wave equation:

$$u^{t+l}_{i,j}=2(l-2\mu^2)u^t_{i,j}+\mu^2[u^2_{i+1,j}+u^2_{i-1,j}+u^2_{i,j+1}+u^2_{i+1,j-1}]-u^{t-1}_{i,j}, \tag{1.39}$$

We are now faced with a new difficulty. Namely, in order to store all of the functional values of our numerical solution, standard matrices are no longer feasible. Fortunately, MATLAB can handle higher-dimensional arrays or matrices. We digress momentarily to introduce them. A three-dimensional matrix will have three coordinates that specify each of its entries. For example, to create a three-dimensional matrix of size $2\times2\times2$ all of whose entries are zeros, we could enter:

```
» A=zeros(2,2,2 )
```

This object can be thought of as two 2×2 matrices stacked on top of one another. When displaying such a matrix, as above, MATLAB will display each "layer" matrix in order starting from the lowest (final) index and moving on. All of the matrix manipulation tricks that we have learned, as well as all of the MATLAB functions that make sense for such higher-dimensional matrices, can be used for these general arrays. There is no limit to the dimension of the matrices that MATLAB can handle. For example, we could create a $3\times2\times2\times2$ all of whose entries are 5's as follows:

```
» A=5*ones(3,2 , 2,2) )

->A(:,:,1,1) =  5 5 A(:,:,1,2) = 5  5
                5 5               5  5
                5 5               5  5
   A(:,:,1,2) = 5 5 A(:,:,2,2) = 5  5
                5 5               5  5
                5 5               5  5
```

[10] The stretching of the membrane results from the boundary forces. It is assumed that the membrane is stretched uniformly in all directions.

Geometrically, we can visualize a three-dimensional matrix as simply a three-dimensional array of numbers, as, for example, a discretization of the heat distribution of a three-dimensional object. Algebraically, it is helpful to think of higher-dimensional matrices as simply being indexed sets of ordinary (two-dimensional) matrices. This is how MATLAB treats them; the first two-indices will always denote the indices of the displayed two-dimensional constituent matrices of a higher-dimensional matrix. Higher-dimensional matrices are ideal for storing numerical values for functions of more than two variables.

As in the one-dimensional case, the Courant-Friedrichs-Levy condition now takes on the form $\mu^2 \equiv (ck/h)^2 \leqslant 1/2$. [11] Under the sufficient differentiability assumptions, Taylor's theorem (Chapter 2) can be used, once again, to produce the following $0(h^2+k^2)$ approximation for the time level $t=k$ values:

$$u_{i,j} \equiv (1-2\mu^2)\phi(x_i,y_j) + \frac{\mu^2}{2}[\phi(x_{i+1},y_j)+\phi(x_{i-1},y_j)+\phi(x_i,y_j+1)] \tag{1.40}$$

$$+\phi(x_i,y_{j-1})+kv(x_i,y_j) \quad for\ \ i=1,2,\cdots,N_x \ \ and \ \ j=1,2,\cdots,N_y.$$

EXERCISE FOR THE READER 12.10: Justify the OQi2 +A2) error quality of the approximation (41).

In order to turn the above formulas into an M-file, it is not feasible to incorporate (or even try to make sense of) higher-dimensional matrix multiplication. MATLAB's versatile features for manipulating arrays, however, will allow us to again write a rather short (and efficient) M-file for theabove finite difference scheme. For smooth problems, it will no longer be true that taking μ to be its maximal value for stability will yield zero truncation errors (see Exercises 19 and 22 and the note preceding Exercise 19), but nonetheless, for smooth problems, this choice is usually a good one and further it simplifies the formulas. In our M-file below, we thus take $\mu = 1/2$.

PROGRAM 12.2:

Function M-file for solution of the two-dimensional wave problem (37) by the finite difference method:

$$\begin{cases} (PDE)u_u = c^2(u_{xx}+u_{yy}), \quad 0<x<a, 0<y<b, 0<t<\infty, u=(x,y,t) \\ (BC's)\begin{cases} u(x,y,0)=\phi(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty \\ u_t(x,y,0)=v(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty \\ u(x,y,t)=0, \text{for all (x, y) on theboundary of the rectangle:} \\ \qquad\qquad R=\{0\leqslant x\leqslant a, 0\leqslant y\leqslant b\} \end{cases} \end{cases} \tag{1.41}$$

```
function [x, y, t, U]=twodimwavedirbc(phi, nu ,a ,b, T,h ,c)
 solves the two-dimensional wave probl;em u_tt = c^2(u_xx, u_yy)
 on the rectangle (0<= x <= a, 0<= y <= b), with u(x,y)=0
 on the boundary of the rectangle.
 input variables: pYti ^initial wave profile function
 nu initial wave velocity function, both should be functions of
 (x,y). a= right endpoint of x, b = upper endpoint of y,
  T = final time solution will be computed. h=common gap on
  x, y-grids, c = speed of wave.
  Output variables: x - row vector for first space variable, y .- y
  row vector for second space variable, t - time grid row vector
  (starts at t=0, ends at t=T, has Nt equally spaced values),
  U = (Nx)by(Ny)by(Nt) matrix of solution approximations at
  corresponding grid points (where Ny - number of y-grid points)
  x grid will correspond to first entries of U, y grid
  values to second entries of U, and t grid to third entries of U.
  CAUTION: This method will only work if h is chosen so that the x
  and y grids can have a common gap size, i.e., if h = a/h and
   b / h mu s t be i n tege i s .
  The time gtid gap is chosen so that mu'2 - 1/2; this guarantees the
  Courant-fc'riedrichs-Levy condition holds and simplifies the
  main finite d i f f c- r e n c e for mu 1 a

k=h/sqrt (2) /c; f* k is determined from muA 2 = 1/2
MaxRatio=max([b/h a/h 1]);
if ((abs(b/h-round(b/h))>MaxRatio*eps)I(abs(a/h-
round(a/h))>MaxRatio*eps)
fprintf('Space grid gap h must divide evenly into both a and b \r')
fprintf ( Either try another input or modify the algorithm')
error('M-file will exit')
end
Nx - round(a/h)+1; number of points on x-grid
Ny - round(b/h)+1; number of points on y-grid
```

[11] In general, if we wanted to allow different step sizes h_x, h_y for the x- and y-grids, the Courant-Friedrichs-Levy stability condition takes the form: $\mu^2 \equiv c^2k^2(h_x^{-2}+h_y^{-2}) \leqslant t$, see [IsKe-6].

```
Nt - f loor (T/k) +1; number of points on t-grid
U=zeros(Nx, Ny, Nt) ; x=0:h:a; y=0:h:b; t=0:k:T;
  Recall matrix indices must start at 1. Thus the indices of the
  matrix will always be one more than the corresponding indices that
  were used in theoretical development.
  Note that by default, zero boundary values have been assigned to
  all grid points on the edges of the rectangle (and, for the time
  being, at all. grid points

  Assign initial time t-^ values and next step t^k values.
for i=2:(Nx-1)
    for j=2:(Ny-1)
  U(i,j,1)=feval(phi,x(i),y(j));
  U(i,j,2)=.25*(feval(phi,x(i-1),y(j))+...
    feval(phi/x(i + 1)/y(j))+feval(phi,x(i),y(j-1))+.. .
    feval(phi,x(i),y(j + 1))) + k*feval(nu,x(i), y(j)) ;
end
end

  Assign values at interior grid points
for ell=3:Nt ^letter ell looks too much like number one

  U(2:(Nx-1),2:(Ny-1), ell) = ...
   + .5*(U(3:Nx,2: (Ny-1), ell-1)+ U(1:(Nx-2),2:(Ny-1), ell-1)...
   + U(2:(Nx-1),3:Ny, ell-1) + U(2:(Nx-1),1:(Ny-2), ell-1))...
   - U(2:(Nx-1),2:(Ny-1), ell-2);
end
```

EXAMPLE 12.7:

(a) Make a (color) movie of the wave that solves the following problem:

$$\begin{cases} (PDE) u_u = c^2 u_{xx} + u_{yy}, \quad 0 < x < \pi, 0 < y < \pi, 0 < t < \infty, u = (x,y,t) \\ (BC's) \begin{cases} u(x,y,0) = sin2xsin2y, \quad 0 \leqslant x \leqslant \pi, 0 \leqslant y \leqslant \pi, 0 \leqslant t < \infty \\ u_t(x,y,0) = v(x,y), \quad 0 \leqslant x \leqslant \pi, 0 \leqslant y \leqslant \pi, 0 \leqslant t < \infty \\ u(x,y,t) = 0, \text{for all (x, y) on theboundary of the rectangle:} \\ \qquad\qquad R = \{0 \leqslant x \leqslant \pi, 0 \leqslant y \leqslant \pi\} \end{cases} \end{cases}$$

for the time interval $0 \leqslant T \leqslant 4$. Divide the JC- and v-ranges into 25 equally spaced intervals for the grid. (b) The exact solution of this problem is $u(x,y,t) = cos(2\sqrt{2}t)sin(2x)sin(2y)$, as is easily verified. Measure the maximum value of the errors of the computed solution above versus the exact solution at four time values that are close to the values $t = 1,2,3,4$.

SOLUTION: We first construct inline functions for the boundary conditions. It is important that they be made functions of x and y (in this order).

```
>>phi=inline('sin(2*x ) *sin(2*y) ' , 'x' , 'y')
->phi =lnline function:
  phi(x.y) = sin(2*x)*sin(2*y)

>>nu=inline('0','x' , 'y')
->nu = Inline function:
  nu(x.y) = 0
```

The remaining input parameters for the above M-file are as follows: $a = b = \pi, T = 4, h = \pi/25$, and $c = 1$

```
 >>[x, y, t, U] = twodimwavedirbc(phi, nu, pi, pi, 4, pi/25, 1)
>> size(U)
->ans = 26 26 46

>> for ell=l:26
surf (x,y,U(:, :,ell)) ;
axis([0 pi 0 pi -1.5 1.5]);
M{:,:,ell)=getframe; %see footnote below
end
>> movie(M, 2, 4)
```

[12] Several representative snapshots of the movie are displayed in Figure 12.18; the reader is urged to run the code on his or her machine and view the actual movie.

Part (b): We first create an inline function for the exact solution:

```
>> exact=inline('cos(2*sqrt(2)*t).*sin(2*x).*sin(2*y)','x' , 'y', 't; )
  ->exact = Inline function:
      exact(x,y,t) = cos(2*sqrt(2)*t).*sin(2*x).*sin(2*y)
```

[12] Note that the movie matrix M for three-dimensional graphics is set up as a three-dimensional array. For versions earlier than Version 7 in MATLAB, the usual (two-dimensional) syntax *M(:, ell)* =get frame; should be used.

By viewing the vector t, we see that a good set of representative times would be:

```
>> t([1 3 24 35 46])
->ans =1.0663 2.0437 3.0212 3.9986

>> for i=l:2 6
f or j=l:2 6
UexactMi , j)=exact( (i-l)*pi/25 , (j-1)*pi/25,t(13)) ;
end
end
>> max(max (U(:,:,13)-Uexactl) )
->ans=1.3323e-015 y)
```

Repeating this for the other three listed time levels gives the following errors (in order): 1.7764e-015, 1.6653e-015, 1.8319e-015! The results are astoundingly pleasing; not only is the numerical solution graphically indistinguishable from the exact solution, but they are still identical up to MATLAB's working precision! In general, whenever the solution of the wave problem (11) or (37) has a "smooth" solution, then the finite difference solution (with appropriate value of μ) coincides with the exact solution! See Exercise 19 for a precise statement and outline of the proof in the case of (11).

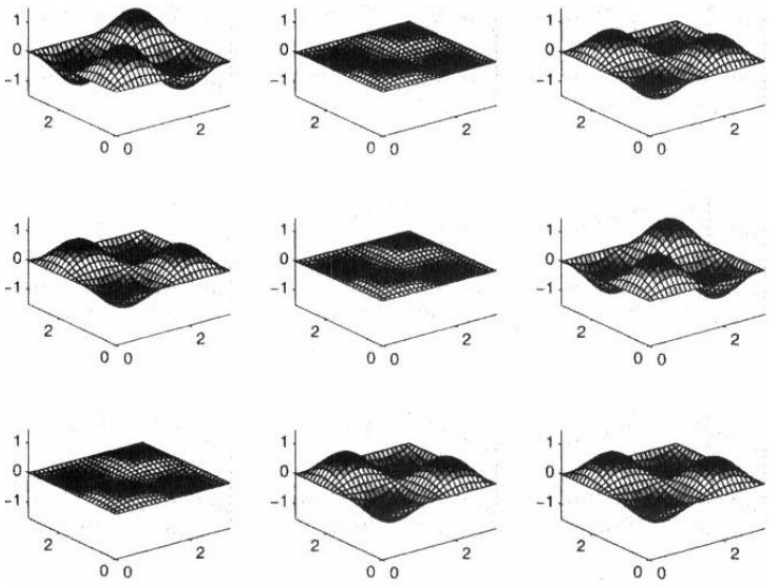

Figure 1.18: Slides from the movie of Example 12.7 of a wave on a square membrane. MATLAB's default colormap is set up so that highest values of the graph are colored red (hot) and lowest values are colored blue (cold). The movie must be seen!

EXERCISES 12.2:

1. a) Use finite difference methods to create a series of snapshots of the solution of the following vibrating string problem:

$$\begin{cases} (PDE)u_u = 2u_{xx}, \quad 0 < x < \pi, 0 < t < \infty, u = (x,t) \\ (BC's)\begin{cases} u(x,0) = sinx, u_t(x,0) = 0, \\ u_t(0,t) = u(\pi,t), \end{cases} \quad 0 < x < \pi, 0 \leqslant t < \infty \end{cases}$$

   Have your snapshots range from $t = 0$ through $t = 6$.
   (b) Compute the maximum errors of the snapshots at six time levels close to $t = 1,2,3,4,5,6$ by comparing with the exact solution $u(x,t) = cos(\sqrt{2t})sinx$. (c) Are there moderate values of TV (- number of interior jc-grid points) and N(= number of interior t-grid points) for which the finite difference solution would be accurate essentially to machine precision? Take moderate to mean less than 100, and machine precision to be approximately 10^{15}.
   (d) Do the answers to these questions change significantly depending on whether we use Program onedimwave as the solver or the program onedimwavebasi c of Exercise for the Reader 12.7?

2. Make a movie, as distortion-free as possible, of the wave in Exercise 1 in the range $t = 0$ through $t = 6$.

3. Repeat all parts of Exercise 1 for the following vibrating string problems:

$$(a)\begin{cases}(PDE)u_u = u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=sinx, u_t(x,0)=sin(x),\\ u_t(0,t)=u(\pi,t)=0,\end{cases} \quad 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

The exact solution is $u(x,t) = sin(x)(cos(xt)+sin(t))$

$$(a)\begin{cases}(PDE)u_u = 9u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=0, u_t(x,0)=sin^3(x),\\ u_t(0,t)=u(\pi,t)=0,\end{cases} \quad 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

The exact solution is $u(x,t) = [9sinxsin3t - sin3xsin3at]/36$.

$$(a)\begin{cases}(PDE)u_u = 4u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=x(x-1), u_t(x,0)=0,\\ u_t(0,t)=u(\pi,t)=0,\end{cases} \quad 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

The exact solution $u?(x,t) = \dfrac{8}{n^3}\sum_{n=1}^{\infty}\dfrac{1}{(2n-1)^2}sin[(2n-1)\pi x]cos[(4n-2)\pi t]$.

Suggestion: For part (c), when computing the exact solution, use a finite sum that approximates the infinite sum to within machine precision. See Section 5.3 for related approximations.

4. Make movies, as distortion-free as possible, for each of the waves in parts (a) through (c) of Exercise 3. Use the time range $0 \leqslant t \leqslant 6$.

5. Consider the moving wave problem modeled by the basic wave equation $u_u = u_{xx}$on the string $0 \leqslant x \leqslant 10$ with initial profi $u(x,0) = \phi(x)\begin{cases}x/2, & \text{if } 0\leqslant x\leqslant 2\\ (x-3)^2, & \text{if } 2\leqslant x\leqslant 3\\ 0, & \text{otherwise}\end{cases}$ and moving to the right at speed one; see Figure 12.17. (a) Use the finite difference method (with sufficiently fine grids) to solve this problem from $t=0$ through $t=20$, and plot some snapshots of the numerical solution. Do the results change significantly depending on whether we use Program 12.1 `onedimwave` as the solver or the program `onedimwavebasic` of for the Reader 12.7? (b) Create a MATLAB movie of this wave motion, (c) Use the implicit finite difference method (of Exercise for the Reader 12.9) so re-solve part (a) with grids comparable to those that were used in part (a). How do the results compare? (d) Make a movie of the motion of the wave, from $t=0$ through $t=12$.
Suggestion: To get the initial velocity $u_t(x,0)$, use D'Alembert's theorem to get the analytical solution (for t less than 7) and differentiate with respect to t). Note that this initial velocity will be discontinuous.

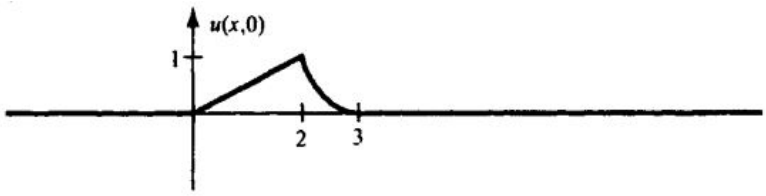

Figure 1.19: Initial wave profile for Exercise 5.

6. The wave problem of Exercise 5 involved a nonsmooth wave. Repeat all parts of Exercise 5 for the corresponding wave problem with initial pulse $u(x,0) = BS(x-2)$, where $BS(x)$ is the cubic spline given by equation (51) of Chapter 10. The initial pulse is still moving to the right at speed one. Because of the smoothness of the data, the finite difference methods should perform much better than for Exercise 5.

7. Use finite difference methods to solve the following vibrating string problem where one end is in sustained motion:

$$(a)\begin{cases}(PDE)u_u = u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=0, u_t(x,0)=0,\\ u(0,t)=sint, u(\pi,t)=' \end{cases} \quad 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

(a) Display the results graphically as a series of snapshots.
(b) Create a MATLAB movie of this vibrating string.
(c) According to your data, does the solution eventually become a periodic function? If it seems so, can this be precisely confirmed?
(d) Compare the performance of explicit finite difference methods versus the implicit method of Exercise for the Reader 12.9 (using grids that obey the CFL stability criterion).

8. For each wave problem below, do the following: (i) Use an explicit finite difference method to create a series of snapshots of the wave propagation from time $t = 0$ up through (at least) $t = 10$. Try successively refining the resolutions until the numerical results stabilize (graphically at least), (ii) Create a MATLAB movie of the solution of (i). (iii) Repeat (i) using the implicit finite difference method of Exercise for the Reader 12.9.

$$(a)\begin{cases}(PDE)u_u = (1+x/2)u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=sinx, u_t(x,0)=sin(x),\\ u(0,t)=u(\pi,t)=0,\end{cases} \quad 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

$$(b)\begin{cases}(PDE)u_u = (1+x^2)^{1/2}u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=0, u_t(x,0)=0,\\ u(0,t)=sin(4x), u(\pi,t)=0'\end{cases} \quad 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

$$(c)\begin{cases}(PDE)u_u = (1+x^2)^{1/2}u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=0, u_t(x,0)=0,\\ u(0,t)=sin(4x), u(\pi,t)=(1/4)sin(10x)'\end{cases} \quad 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

9. (a) Write a function M-file for applying the implicit finite difference method (36) with parameter $\omega = 1/2$. The syntax should be as follows:

```
[x ,t, U] = onedimwaveimpl_2(phi, nu, L, A, B, T, N, M, c)
```

where the variables and functionality should be just as in the program `onedimwaveimpl_4` of Exercise for the Reader 12.9.
(b) Do some experiments on the problem of Example 12.6 comparing the performance of the implicit finite difference method of part (a) with the explicit method of Program 12.1, using values of N and M that would make the latter unstable.Compare with the performance of `onedimwaveimpl_4` as was seen in Exercise for the Reader 12.9.
(c) How fine a resolution is needed to get the implicit finite difference method of part (a) to obtain graphical results of the quality of the numerical solution created in Example 12.6? Do the results seem better when the CFL condition is satisfied? Do some experiments. Compare with the performance of `onedimwaveimpl_4` as was seen in Exercise for the Reader 12.9.

10. *(Another M-file for Two-Dimensional Waves)* (a) Write another M-file with the following syntax:

```
( x, y, t , UJ = twodimwavedirbcV2(phi, nu, a, b , T, h, k, c)
```

that is designed to solve the two-dimensional wave problem (34). The variables and functionality are similar to the `twodimwavedirbc` M-file of Program 12.2 but there is one new input variable, k, which is the time step size. Thus this new program gives more flexibility in choosing time step in that it is no longer determined by forcing $\mu^2 2 = 1/2$ (the maximum value allowed in the stability condition).
(a) Run the program on the problem of Example 12.7 to reproduce the results ofthat example. In other words, choose k to be the value that was internally computed = $t(1)$ by the previous M-file, and check if the U matrix is identical, modulo roundoff, to the one in obtained in Example 12.7. (They should be if your program is correctly written.)
(b) Keeping h the same as in part (c), successively halve the it-step size from its value there, rerun the program, and compute the errors (at the same four /-values) of the numerical solution of your program with that in part (a). Do things get better, worse, or stay about the same? Comment on your findings.
(c) Can you find a two-dimensional wave BVP where the `twodimwavedirbcV2` seems (with appropriate choices of k) to do a better job than `twodimwavedirbcV2` ? This may take a good deal of numerical experimentation.

11. For each of the following two-dimensional wave problems do the following: (i) Create a series of snapshots of the wave motion, (ii) Create a movie of the wave motion, (iii) If an exact solution is given, compute the (maximum) errors of the finite difference solutions at several different time values. The problems all fall under the umbrella of the following Dirichlet BVP:

$$\begin{cases}(PDE)u_u = c^2(u_{xx}+u_{yy}), \quad 0<x<a, 0<y<b, 0<t<\infty, u=(x,y,t)\\ (BC's)\begin{cases}u(x,y,0)=\phi(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty\\ u_t(x,y,0)=v(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty\\ u(x,y,t)=0, \text{for all (x, y) on theboundary of the rectangle:}\\ \qquad\qquad R=\{0\leqslant x\leqslant a, 0\leqslant y\leqslant b\}\end{cases}\end{cases}$$

(a) *(Two-Dimensional Hammer Blow)* Take $a=b=3, c=1$, $\phi(x,y)=0$ and $c(x,y)=\begin{cases}5, 1\leqslant x,y\leqslant 2\\ 0, otherwise\end{cases}$ Run your computations on the time interval $t=0$ through $t=5$.
(b) Take $a=b=1, c=2$, $\phi(x,y)=x(1-x), y(1-y)$ and $x(x,y)=0$ Exact solution

$$u(x,y,t)=\frac{64}{\pi^6}\sum_{N=0}^{\infty}\sum_{N=0}^{\infty}\frac{1}{(2N+1)^3(2M+1)^3}\cdot sin((2n+1)\pi x)sin((2m+1)\pi y)cos(\pi\sqrt{(2n+1)^2+(2m+1)^2}$$

(c) Data as in part (b) but $v(x,y)=2sin(2\pi x)sin(\pi y$. Exact solution: Solution of part (b) plus $(2\sqrt{5})sin(2\pi x)sin(\pi y)sin(\sqrt{5}\pi t)$. Suggestion: The exact solutions of parts (b) and (c), so-called double Fourier series, can be estimated by finite sums of the form $u(x,y,t)=\frac{64}{\pi^6}\sum_{n=0}^{K}\sum_{n=0}^{K}\cdots$ where the integer K is chosen sufficiently large so that the series is accurate to MATLAB precision (i.e., maximum error $<10^{15}$). Try to estimate mathematically how large K should be for this accuracy. See Section 5.3 for similar estimates.

12. (a) *Pinch-Gripped Membrane*) Suppose a membrane that occupies the square $0\leqslant x,y\leqslant 4$ has initial position given by a pyramid with height 1 at $(xy)=(2,2)$, and initial velocity equal to zero; see Figure 12.20(a). Create a movie of the resulting motion of the membrane if the wave speed is $c=1$.
(b) (*Smooth Bumped Membrane*) Suppose a membrane that occupies the square $-2\leqslant x,y\leqslant 2$ has initial position given by the graph z = BS(r); where $r=\sqrt{x^2+y^2}$ is the distance to the origin and BS is the cubic spline function (51) of Chapter 10, see Figure 12.20(b). The initial velocity is equal to zero. Create a movie of the resulting motion of the membrane if the wave speed is $c=1$.

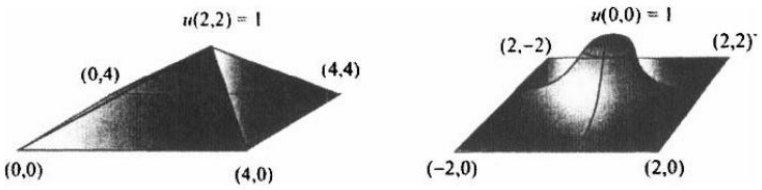

Figure 1.20: Initial membrane profiles for the wave problems of Exercise 12 (a) (left) and (b) (right).

13. Write a function M-file for solution of the following two-dimensional wave problem by the finite difference method.

$$\begin{cases}(PDE)u_u=c^2u_{xx}+u_{yy}, \quad 0<x<a, 0, 0<y<b, 0<t<\infty, u=(x,y,t)\\ (BCs)\begin{cases}u(x,y,0)=\phi(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty\\ u_t(x,y,0)=v(x,y), \quad 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty\\ u(0,y,t)=L(x,t), \;\; u/(a,y,t)=R(y,t),\\ u(0,y,t)=B(x,t), \;\; u/(b,x,t)=R(x,t),\end{cases}\end{cases}$$

The syntax should be as follows:

```
[x,y,t,U] = twodimwavedirbc2(phi, nuf a, b, T, h, c, Lr R, B, T)
```

where the variables (all but the last four input variables) and functionality are as in Program 12.2. The last four input variables are the boundary value functions.
(a) Use the program to solve the problem above with the following data:

$$x=1, a=b=\pi, \phi=v=L=T=R=0, B(x,t)=sintsin(x)$$

Create both snapshots and a movie of the wave propagation.

b) Repeat part (a) but change $T(x,t/)=(1/4)sin(5t)sin(3x)$.

NOTE: The next exercise will outline a proof of the necessity of the Courant-Friedrichs-Levy condition for the stability of the finite difference method in solving the wave equation. The proof will rely on finding exact solutions of the difference equation. As was the case with difference schemes for ODE in Part II, the theory of finding a solution of a finite difference scheme for a PDE closely parallels the theory of analytical solutions. Since we do not discuss this analytical theory, we will only start with a suitable form for a solution of the difference equation without discussing the motivation for this choice. The proofs will run more smoothly if we use complex numbers and, in particular, Euler's identity: $e^{i\theta}-cos(\theta)+/sin(\theta)$, where $i=\sqrt{-1}$ is the complex unit

14. Use Euler's identity to show that $cos(\theta)=(e^{i\theta}+e^{-i\theta})/2 and sin(\theta)=(e^{i\theta}+e^{-i\theta})/2i$.

15. (*Stability Analysis*) This exercise will outline a proof that the Courant-Friedrichs-Levy condition (31) $\mu = ck/h \leqslant 1$ is necessary for the stability of the finite difference scheme (23)

$$u_{n,j+1} = 2(l-\mu^2)u_{n,j} + \mu^2[u_{n+1,j} + u_{n-1,j}] - u_{n,j-1}$$

for the wave equation $u_u = c^2 u_{xx}$. Since the proof will invoke complex number notation, we have changed the index i to n in (23) to avoid any confusion.
(a) We begin by looking at functions (of n and j) of the form: $U_{n,j} = a^j e^{i\beta nh}$ (here α and β are parameters and h is the grid spacing on the x-axis). Substitute this function into (23) and show that it solves it if and only if $\alpha + 1/\alpha - 2 = \mu(2(\eta - 2 + 1/\eta)$ [13]
(b) Show that the equation obtained in part (a) can be rewritten in the form $\alpha^2 + 2(2\mu^2 sin^2(\beta h/2) - l)\alpha + 1 = 0$. The reason that we wrote the equation as a quadratic in the variable α (rather than emphasizing β) is that in the solution form $U_{n,j} = \alpha^j e^{i\beta nh}$, the factor depending on β always has bounded absolute value, in fact $|e^{i\beta nh}| = 1$ regardless of the values of β , n, and h. The a-dependent factor α^j can blow up (cause instability) if $|\alpha| > 1$, but will remain stable if | al< 1.
(c) Introduce the parameter $\rho = 2\mu^2 sin^2(\beta h/2) - 1$ so the equation in part (b) can be expressed as $a^2 + \sqrt{\rho^2} + 1 = 0$. Use the quadratic formula to show the roots are $\alpha = -\rho \pm \sqrt{\rho}^2 - 1$. Show that both roots will have absolute values less than one if $\rho \leqslant 1$.
(d) Show that if $\mu \leqslant 1$, then both roots of the equation in part (b) have absolute values less than or equal to one, and hence conclude the stability of the finite difference method, i.e., conclude that the solutions of part (a) will remain bounded independent ofj and n.
Suggestions: For part (b) use a half angle formula from trig.

NOTE: If we impose Neumann and Robin boundary conditions at the ends of a wave problem (for a finite string):

$$u_u = u_{xx}, a < x < b, 0 < t < \infty, u = u(x,t),$$

the result will be a well-posed problem. The standard Neumann boundary condition (at the right end $x = b$) $u_x(b,t) = 0$ corresponds to the end of the string being a **free end**, perhaps being glued to a ring that is free to move up and down on a frictionless vertical rod. The Robin boundary condition ($\alpha u(b,t) + \beta u_x(b,t) = \gamma$) can be viewed physically as (Hooke's Law) having a spring attached to the end of the string (with one end of the spring fixed). In general, the wave problem on a finite string will be well posed if there is specified any combination of Dirichlet, Neumann, or Robin boundary conditions at the two ends. This is physically quite reasonable; for a mathematical proof, see [Wei-65].

16. (*M-filefor a Dirichlet-Neumann mixed Wave Problem*) (a) Write a MATLAB function M-file that will employ the finite difference method to solve the following wave problem:

$$\begin{cases} (PDE) u_u = c(t,x,u,u_x)^2 u_{xx}), \quad 0 < x < L, 0 < t < \infty, u = (x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), \ u_t(x,0) = v(x), \\ u(0,t) = A(t), \ u_x(L,t) = B(t) \end{cases} \quad ,0 < x < L, 0 \leqslant t < \infty \end{cases}$$

The syntax of the M-file should be similar to that of Program 12.1:

```
[x, t, Ü] = onedimwavedirneu(phi, nu, L, A, B, T, N, M, c)
```

(b) Use the program of part (a) to solve the BVP above with the following data: $c = 1, L = 5, \phi = v = B = 0$, and A(t) as in Example 12.6. Display the solution as a series of snapshots.
(c) Repeat part (b), but change c to be as in Example 12.6..

17. (*Conservation Laws*) A one-dimensional PDE that models numerous physical conservation phenomena is the following first-order PDE: $u_t + c(t,x,u)u_x = 0, u = u(x,t)$. For applications, such as to highway traffic flow and fluid flow, we refer to [DuCZa-89] or [Log-94]. (a) Write a function M-file that will use finite difference methods to solve the above PDE on a finite segment $0 \leqslant x \leqslant L$ with initial profile u(xy0) $u(x,0) = \phi(x)$ and Dirichlet boundary conditions $u/(0,t) = A(t), u(L,t) = B(t)$ (because the equation is only first-order we do not need to specify the initial wave velocity to make the problem well-posed). The syntax and functionality should be as in Program 12.1. Use centered difference (second-order) schemes on both the time and space derivative discretizations.
(b) Use your M-file of part (a) to solve the above BVP with data: $c(t,x,u) = x, L = 6, \phi(x) = exp(-2(x-2)^2)$ and $A = \phi(0), B = \phi(L)$. Plot a series of snapshots of the solution over the time range $0 \leqslant t \leqslant 10$. Experiment with different resolution parameters until the numerical solutions tend to stabilize.
c) In case $c(t,x,u) = c$ is constant, the resulting PDE $u_t + cu_x = 0, u = u(x,t)$ is often known as the **one-way wave**

[13] For readers who are familiar with the method of separation of variables for finding solutions of PDEs, this form of the discrete problem can be derived by a similar discrete approach, i.e., we assume that the solution of the discrete equation can beseparated as a product A(j)B(n), substitute it into the difference equation, and then determine the form of A and B. For more details on this interesting analogy (as well as a full account of the relevant analytical theory) we refer the reader to [DuCZa-89],

equation. Show that the solution of the resulting Dirichlet problem is given by $u(x,t) = \phi(x-ct)$, and interpret in terms of moving waves.
(d) Apply the program of part (a) to solve the one-way wave problem with data $c = 1, \phi(x)$ as in Figure 12.19, L - 10. How fine a resolution is needed so as to get a decent approximation to the exact solution as specified in part (c)?

18. (Some Nonlinear Shock Wave Problems) For each nonlinear wave problem do the following: (i) Use explicit finite difference methods to create a series of snapshots of the wave propagation from time $t = 0$ up through (if possible) $t = 10$. Try successively refining the resolutions until the numerical results stabilize (graphically at least), (ii) Create a MATLAB movie of the solution of (i). (iii) Repeat (i) using the implicit finite difference method of Exercise for the Reader 12.9.

$$(a)\begin{cases}(PDE)u_u = (1+u^2)^{1/2}u_{xx}, & 0<x<\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0) = sin(x), u_t(x,0) = 0,\\ u(0,t) = u(\pi,t) = 0,\end{cases} & 0<x<\pi, 0\leqslant t<\infty\end{cases}$$

$$(b)\begin{cases}(PDE)u_u = (1+u^2)^{1/2}u_{xx}, & -10<x<\pi, 10<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0) = f(x), u_t(x,0) = 0,\\ u(0,t) = 0, u(\pi,t) = 0,\end{cases} & -10<x<\pi, -10\leqslant t<\infty\end{cases}$$

$$\text{where} f(x) = \begin{cases}1-|x|, & if |x|<1\\ 0, & therwise\end{cases}$$

$$(a)\begin{cases}(PDE)u_u = (1+u^2)^{1/2}u_{xx}, & 0<x<6\pi, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0) = 0, u_t(x,0) = 0,\\ u(0,t) = sin(4t), u(\pi,t) = 0,\end{cases} & 0<x<6\pi, 0\leqslant t<\infty\end{cases}$$

Note: The PDE here represents a wave equation where the speed of the wave depends on the amplitude, with higher (in absolute value) portions moving faster than lower portions. It would thus seem that high parts of the wave would catch up and overpass the lower parts that are ahead. This would seem to indicate that eventually the profiles would no longer be functions of x. (Think of the surface of an ocean wave as it begins to break.) See Figure 12.21 for an illustration. Such nonlinear BVPs thus do not have ordinary single-valued functions as solutions but rather what are called multivalued waveforms. For more on this interesting phenomenon see Chapter 3 of [Log-94]. Of course, the finite difference methods we developed are not set up to deal with such multivalued wave forms. In this exercise you should simply carry out the finite difference methods for a time interval stretching until the results no longer seem meaningful. Do your numerical results allow you to detect such shocking phenomena?

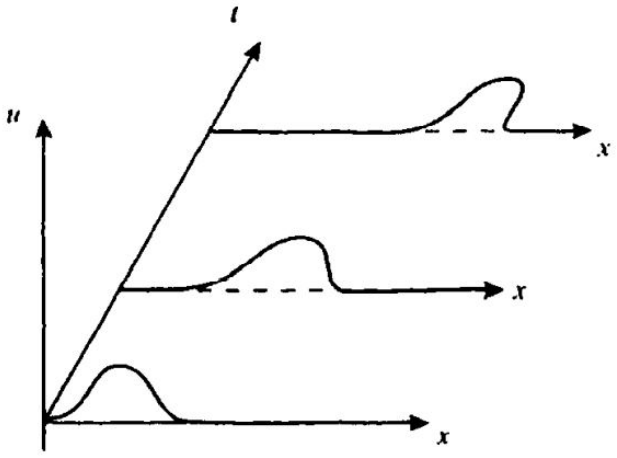

Figure 1.21: The shock-wave phenomena of the nonlinear wave equation of Exercise 18. Higher parts of the wave propagate faster than lower parts, eventually causing the wave profile to "break" from being a function of x.

NOTE: A general mathematical principle of partial differential equations roughly states that if the "data" of a boundary value problem has a certain amount of "smoothness," then the solution of the problem will enjoy the same amount of "smoothness." [14] Often, numerical methods require certain smoothness assumptions on the solution that is not usually known, so such smoothness results can be of great practical value in deciding in advance on a numerical method and predicting its success. The next exercise gives a wonderful yet rare situation where the numerical method gives the exact result.

[14] The language here is admittedly quite vague, since it is not feasible to rigorously state a general result. To clarify things a bit: "Data" simply refers to the known functions in the problem, i.e., functions appearing as coefficients in the PDE or in the boundary conditions. "Smoothness" has to do with the order of differentiability of a given function. Such theorems are referred to as **regularity theorems**, and often require quite advanced mathematical methods to prove (and even to precisely state).

19. In this exercise we will show that if the solution to the wave problem (11)

$$\begin{cases} (PDE) u_u = c^2 u_{xx}, \quad 0 < x < L, 0, 0 < t < \infty, u = (x,y,t) \\ (BCs) \begin{cases} u(x,0) = u(x,0) = \phi(x), u_t(x,0) = v(x), \\ u(0,t) = u(0,t),\ u(L,t) = 0, \end{cases} \quad ,0 \leqslant xL, 0 \leqslant t < \infty \end{cases}$$

is infinitely differentiable, then the finite difference scheme (23):

$$u_{i,j+1} = 2(l-\mu^2)u_{i,j} + \mu^2[u_{i+1,j} + u_{i-1,j}] - u_{i,j-1}$$

will be exact provided that we take $\mu = 1$ (the maximum value allowed by the Courant-Friedrichs-Levy stability condition). Thus in applying this explicit finite difference scheme to the wave problem (11), the only errors that will arise will be either roundoff errors or errors in the (needed) level $t = k$ values w_i, l (perhaps obtained from (29)). Proceed through the following outline to establish this result. The method being used here is the so-called **bootstrapping technique**.
(a) Use Taylor's theorem to obtain the following expansions of finite difference quotients:

$$\triangle_t \equiv \frac{u(x_i,t_{j+1} - 2u(x_i,t_j) + y(x_i,t_{j-1})}{k^2} = u_u(x_i,t_j) + 2[\frac{k^2}{4!}\partial_t^4 u(x_i,t_j) + \frac{k^4}{6!}\partial_t^6 u(x_i,t_j) + \cdots],$$

and

$$\triangle_t \equiv \frac{u(x_{i+1},t_j) - 2u(x_i,t_j) + y(x_{i-1},t_j)}{k^2} = u_u(x_i,t_j) + 2[\frac{k^2}{4!}\partial_t^4 u(x_i,t_j) + \frac{k^4}{6!}\partial_t^6 u(x_i,t_j) + \cdots],$$

(The symbols $\triangle_x, \triangle_y$ have been introduced as shorthand for what remains.)
(b) Use part (a) and the fact that u satisfies the PDE of (11) to show that:

$$\triangle_t - c^2\triangle_x = \frac{2}{4!}[k^2\partial_t^2 u(x_i,y_j) - c^2h^4\partial_x^6 u(x_i,y_j)] = \frac{2}{6!}[k^4\partial_t^6 u(x_i,y_j) - c^2h^2\partial_x^4 u(x_i,y_j)] + \cdots.$$

(c) In the expansion of part (b), use the PDE and to show that

$$k^2\partial_i^4 u(x_i,y_j) = k^2\partial_t^2[\partial_t^2 u(x_i,y_j)] = k^2\partial_t^2[c^2\partial_t^2 u(x_i,y_j)] =$$

$$c^2k^2\partial_x^2[\partial_t^2 u(x_i,y_j)] = c^2k^2\partial_x^2[c^2\partial_x^2 u(x_i,y_j)] = k^2c^4\partial_x^4 u(x_i,y_j).$$

Use this to show that under the assumption $\mu = 1$, the first term on the right side of the expansion in part (b) is zero.
(d) Repeating the argument in part (c), show that the second term on the right side of the expansion in part (b) is zero.
(e) Go on to show that all of the remaining (infinitely many) terms on the right side of the expansion in part (b) are zero, and hence the local truncation error of the finite difference scheme (23) is zero.

20. To what extent (if any) does result of Exercise 19 still continue to hold if we allow the wave speed in the PDE to depend on any of the variables t, x, u? Assume (if you need to) that the function c is infinitely differentiate.

21. (a) Explain why the argument of Exercise 19 does not hold for the corresponding wave problem in two variables (34) with $\mu = 1/2$ being the maximal allowable value for stability.
(b) Choose from this section (or one of the previous exercises) a particular instance of the problem (34) where an exact solution is known. Use the exact solution to determine the seed values $\mu_{i,j}^l$, and do some numerical experiments with the finite difference scheme given in the text (with $\mu = 1/2$) to demonstrate that the scheme is not exact

1.3. FINITE DIFFERENCE METHODS FOR PARABOLIC PDE'S

As a prototypical problem for this section, we will use the one-dimensional heat equation on a finite interval with (possible) internal heat source and (possibly time-dependent) Dirichlet boundary conditions at both ends: [15]

$$\begin{cases} (PDE) u_t = \alpha u_{xx} + q(x,t), \quad 0 < x < L, 0, 0 < t < \infty, u = (x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x) \\ u(0,t) = A(t),\ u(L,t) = B(t), \end{cases} \quad ,0 < xL, 0 < l, 0 \leqslant t < \infty \end{cases} \tag{1.42}$$

As was explained in Section 11.2, this boundary value problem models the heat distribution $u(x,t)$ on a thin rod of length L whose ends are maintained at the specified temperatures $A(t)$ and $B(t)$ whose initial temperature distribution is specified by $\phi(x)$, and with an internal heat source $q(x,t)$. Note that for simplicity we initially assume the difrusivity α is

[15] Notice that we have changed the notation a bit from Chapter 11. The diffusivity constant used to be labeled as "č", but we have changed this parameter to " α " so that we may continue to use "k" as the time step for finite difference methods.

constant, but this restriction can be easily lifted later. We will numerically solve this problem for all time values up to some predetermined value T. With our experience of finite difference methods for elliptic and hyperbolic problems behind us, we have developed essentially all of the methods required for parabolic problems. As with hyperbolic problems, stability will be a serious issue here. For explicit methods, stability puts a rather severe restriction on the size of the time steps, so implicit methods tend to be more practical for parabolic problems. Because of their dissipative nature, parabolic problems do tend to be more amenable to finite difference schemes than were hyperbolic problems, especially when discontinuous data is concerned.

Following the notation of our previous finite difference schemes, we introduce grids of equally spaced x- and t-coordinates for the rectangular region $0 \leqslant x \leqslant L, 0 \leqslant t \leqslant T$:

$$\begin{aligned} 0 = x_0 < x_1 < x_2 < \cdots < x_{N+1} = L, \quad \triangle x_i \equiv x_i - x_{i-1} = h, \\ 0 = t_0 < t_1 < t_2 < \cdots < t_{N+1} = T, \quad \triangle t_i \equiv t_i - t_{i-1} = k. \end{aligned} \tag{1.43}$$

If we discretize the PDE of (42) by using the forward difference formula (Lemma 11.5) for ut and the central difference formulas (Lemma 10.3) for $_{xx}$, we get the **forward-time central-space scheme**:

$$\frac{u(x_i,t_{j+1}) - u(x_i,t_j)}{k} = \alpha \frac{u(x_{i+1},t_j - 2u(x_i,t_j) + u(x_{i-1},t_j)}{h^2} + q(x,t_j) \tag{1.44}$$

the stencil for which is shown in Figure 12.23(a). We recall that the truncation here are $0(k)$ and $0(h^2)$, respectively, and so the local truncation error of this method is $0(k+h^2)$. Using the notation:

$$u_{i,j} = u(x_i,t_j), \;\; q_{i,j} = q(x_i,t_j)$$

and introducing the parameter

$$\mu = \alpha k/h^2$$

allows us to express (44) in the following simplified form:

$$u_{i,j+1} = (1/2\mu)u_{i,j} + \mu[u_{i+1,j} + u_{i-1,j}] + kq_{i,j} \tag{1.45}$$

For $i = 1,2,...,N$, and $j = 1,2,...M$ The stencil for (45) is shown in Figure 12.23(a). It can be shown that the forward-time central-space method is stable, provided that the following **stability condition** is met:

$$\mu = \alpha \frac{k}{h^2} \leqslant \frac{1}{2}. \tag{1.46}$$

Recall that stability means that any errors introduced in any particular stage of (44) (e.g., truncation errors) remain under control in all future iterations and that if the initial data is sufficiently differentiable, then the global error of the method will have the same quality bound as the local truncation error: $0(k+h^2)$. A complete proof of this stability result can be found in Section 9.4 of [KeIs-66]; some elements will be given in the exercises. To see that (46) often makes the method impractical, for example, with $\alpha = 1$, if we wanted to use a space step size of $h = 0.05$, the stability condition (46) would force us to take a much smaller time step size $k \leqslant 0.00125$.

By using instead the backward-time discretization (Lemma 11.5), we get the following **backward-time central-space scheme** (see Figure 12.23(b)), which is implicit but unconditionally stable:

$$\frac{u(x_i,t_{j+1}) - u(x_i,t_j)}{k} = \alpha \frac{u(x_{i+1},t_j - 2u(x_i,t_j) + u(x_{i-1},t_j)}{h^2} + q(x,t_j) \tag{1.47}$$

Using the parameter $\mu \alpha k.h^2$ this scheme can be expressed as:

$$-\mu u_{i-1,j+1} + (1+2\mu)u_{i,j+1} - \mu u_{u+1,j+1} = u_{i,j} + kq_{i,j} \tag{1.48}$$

By the underlying finite difference approxim-ations, both of the above schemes have local truncation errors of order $0(k+h^2)$ so unless k is $0(h^2)$ (i.e., small enough so that the stability condition (46) holds), then the local truncation error be first-order ($\approx O(k)$). Experience might lead us to believe using a centered difference (second-order) discretization of w, would lead to a more effective scheme. The resulting finite difference method, known as **Richardson's method** (see Exercises 12 and 13), indeed has local truncation order $0(k^2+h^2)$ but unfortunately has serious stability problems.
An interesting second order implicit scheme can be obtained by averaging the forward-time centralspace scheme and the backward-time centralspace scheme (at corresponding time levels). This scheme is known as the **Crank-Nicolson method** [16] and runs follows:

[16]The English school was quite involved with finite difference methods for evolution equations. Back in 1910, Lewis Richardson (1881-1953) had introduced his method andused it and other finite difference methods in numerous applications ranging from meteorology to eddy-diffusion in the atmosphere, and even (at the start of the Cold War) foreign policy and arms control. The stability problems with Richardson's method were not really recognized until the 1940s, when Crank, Nicolson, and others, with the aid of simple mechanical desk computing machines, performed lengthy computations. Crank and Nicolson's work culminated in a 1947 paper [CrNi-47] in which they introduced their unconditionally stable second-order method. This history brings up an important point regarding finite difference methods. It is easy to invent finite difference methods for any PDE. Getting efficient methods which actually converge with good stability and small truncation errors often requires a much more detailed investigation.

$$\frac{u_{i,j+1}-u_{i,j}}{k}\frac{\alpha}{2}[\frac{u_{i,j+1}-2u_{i,j}-u_{i-1,j}}{h^2}]+[\frac{u_{+1i,j+1}-2u_{i,j+1}-u_{i-1,j+1}}{h^2}]+\frac{1}{2}[q(x_i,t_j)+q(x_i,t_{j+1})] \tag{1.49}$$

EXERCISE FOR THE READER 12.11: Show that the local truncation error for the Crank-Nicolson method is $0(k^2+h^2)$ provided that the solution is sufficiently differentiable. [17]

The stencil for the method is shown in Figure 12.23(c). As before, we let $\mu - ak/h^2$,, which allows us to rewrite (51) in the form:

$$-\mu u_{i-1,j+1}+2(1+\mu)u_{i,j+1}-\mu u_{i+1,j+1}=\mu u_{i-1,j}+2(1+\mu)u_{i,j}+\mu u_{u+1,j}+k[q_{i,j+1}] \tag{1.50}$$

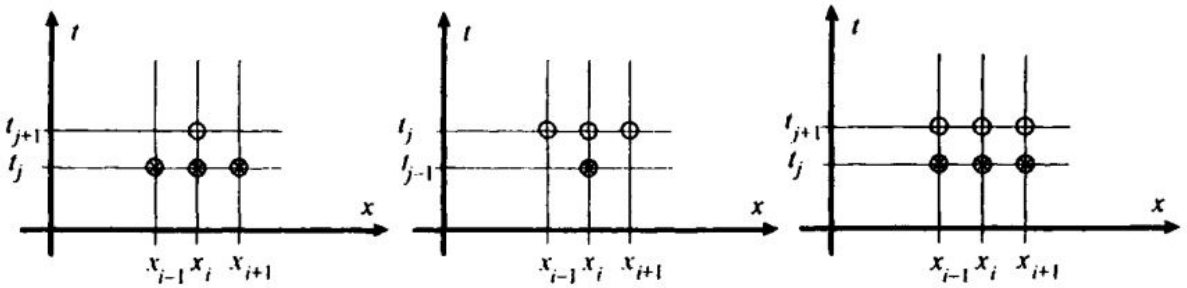

Figure 1.22: Stencils for three finite difference methods for the parabolic BVP (42): (a) (left) The forward-time central-space scheme, an explicit method, (b) (middle) The backward-time central-space scheme, an implicit unconditionally stable method, (c) (right) The Crank-Nicolson scheme. .

We will revisit stability issues in more detail later in this section as well as in the exercises; for now we pass to the task of developing M-files for the above methods and then illustrate and compare them with some examples. The M-file constructions are similar to those in the last section; we outline now the construction only for the Crank-Nicolson method and leave the other two as exercises.

We can convert (50) into matrix form quite naturally. Before we do this, let us first observe what happens in (50) when one of the terms involves a boundary value (i.e., either when $i = 1$ or $i = N$. In case $i = 1$, we then have $u_{i-1,j(+1)} = u(0,t_{j(+1)}) = A(t_{j(+1)})$, and similarly in case $i = N$, $u_{i-1,j(+1)} = u(L,t_{j(+1)}) = B(t_{j(+1)})$. This, for example, in case $i = 1$, (50) should be rewritten in the form:

$$2(l+\mu)u_{j,j+1}-\mu u_{j+1,j+1}=2(l+\mu)u_{j,j}-\mu u_{j+1,j}+\mu(A(t_j)+A(t_{t+1}))+k[q_{i,j}+q_{i,j+1}]$$

Borrowing from MATLAB's notation by letting U/$(:,j)$ denote the y'th column vector of $U : [u_{1,j}\ u_{2,j}\ \cdots\ u_{N,j}]'$ the equation (50) can be written in the form:

$$TU(:,j+1)=SU(:,j)+\mu V_j+Qk_j, \tag{1.51}$$

where the Tand 5 are the following tridiagonal matrices:

$$T=\begin{bmatrix} 2(l+\mu) & -\mu & 0 & \cdots & 0 \\ -\mu & 2(l+\mu) & -\mu & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & & & -\mu \\ 0 & \cdots & 0 & -\mu & 2(l+\mu) \end{bmatrix},$$

$$S=\begin{bmatrix} 2(l+\mu) & -\mu & 0 & \cdots & 0 \\ -\mu & 2(l+\mu) & -\mu & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & & & \mu \\ 0 & \cdots & 0 & \mu & 2(l+\mu) \end{bmatrix},$$

and the vectors V_j and Q_j are: $V_j = [A(t_j)+A(t_{j+1}),0\cdots,0,B(t_j)+B(t_{j+1})]'$ and $Q_j = [q_{1,j}+q_{1,j+1},\cdots,q_{N,j}+q_{N,j+1}]'$; Making use of MATLAB's matrix handling capabilities will now make coding the Crank-Nicolson method a simple task. The following program does this on a slightly more general version of the BVP (42).

[17] The proof is rather technical, so this exercise for the reader may well be skipped by less theoretically oriented readers.

PROGRAM 12.3:

Function M-file for solution of the following parabolic BVP by the Crank-Nicolson method [18]

$$\begin{cases} (PDE)u_t = \alpha(x,t,u)u_{xx} + q(x,t), \quad 0 < x < L, 0 < t < \infty, u = (x,t) \\ (BCs)\begin{cases} u(x,0) = \phi(x), & 0 < x < L, 0 \leqslant t < \infty \\ u(0,t) = A(t),\ u(L,t) = B(t), \end{cases} \end{cases} \tag{1.52}$$

The Thomas algorithm is used to solve the tridiagonal systems which arise.

```
function [x, t, U] = cranknicolson(phi, L, A, B, T, N, M, alpha, q)
  solves the one-dimens heat problem
  u_t = alpha(t,x,u)*u_xx+q(x,t)
  using the Crank-Nicolson method.
  Input variables: phi-phi(x) - initial wave profits function
  L - lennth of rod, A =A(t) -temperature of left end of rod
  u(0, t)-A(t) , B-B(t) - temperature of right end of rod u (L,t) - B(t)
  T- final time for which solution will be
  computed, N = number of internal x-grid values, M = number
  of internal t-grid values, alpha = alpha(x,t,u) = diffusivity of rod.
  q - q(x,t) - internal heat source function
  Output vaiiabl.es: t. - time grid row vector (starts at t=0, ends at
  t-T, has M+2 equally spacced values), x - space grid row vector,
  U = (M+2) by (N>+12) matrix of solution approximations at
  corresponding grid points. x grid will correspond to second
  column)entries of U, y grid values to first (row) entries of U.
  Row 1 of U correspond s to t - 0.

h = L/(N+1); k = T/(M+1);
1 U=zeros(M+2,N+2); x=0:h :L; t=0:k:T;

  Recall matrix indices must start at 1. Thus the indices of the
  matrix will always be one more than the corresponding indices that
  were used in theoretical development.

VAssign left and right Dirichlet boundary values.
U(:,l)=feval (A,t) '; U(:,N+1)=feval(B,t)';

  Assign initial time t-0 values.
  for i=2:(N+1)
    U(l,i)=feval(phi,x( i));
end
  Assign values at interior grid points
  for j=2:(M+2)
    for i=2:(N+l)
      mu(i)=k*feval(alpha,t(j-l),x(i),U(j-1, i))/hA
2;
      mu2(i)=k*feval(alpha,t(j),x(i),U(j-l,i))/h ^ 2;
      ql(i)=feval(q,x(i),t(j-l)); q2 (i) =feval (q, x (i) , t (j ) );
    end
  First form needed vectors and matrices, because we will be using
  the thonvas M-file, we do not need to construct, the coefficient
  matrix T.

  S = diag(2*(l-mu2(2:N+1))) + diag(mu2(3:N+1), -1) + diag(mu(2:N),
1);
  V = zeros(N,l); V(1)=mu(2)*U(j-1,1)+mu2(2)*U(j, 1);
V(N)=mu(N+l)*U(j-l,N+2)+mu2(N+l)*U(j,N+2);
Q = k*(ql(2:N+l)+q2(2:N+l) ) ';

  Now perform the matrix multiplications to iteratively obtain
  solution values for increasing time levels.
    c=S*((U(j-1,2:(N+l)))')+V+Q;
      a=-mu2(2:N+l); b=a; a(N)=0; b(l)=0;
      U(j,2:N+l)=thomas(a,2*(l+mu2(2:N+l)),b,c);
    end
```

EXERCISE FOR THE READER 12.12: (a) Write a corresponding M-file for the problem (52) that applies the forward-time central-space scheme. The syntax should be as follows:

```
[x, t, U] = fwdtimecentspace(phi, L, A, B, T, N, M, alpha,q)
```

where the input variables, output variables, and functionality are just as in Program 12.3. If possible, avoid any square matrix multiplication in your program.
(b) Write a corresponding M-file for the problem (52) that applies the backwardtime central-space scheme. The syntax

[18] Since we are now allowing a to be variable, (49) takes on the following slightly more general form: $\frac{u_{i,j+1}-u_{i,j}}{k} = [\frac{\alpha_{i,j}}{2}\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2} + \frac{\alpha_{i,j+1}}{2}\frac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}}{h^2}] + \frac{1}{2}[q(x_i,y_j)+q(x_i,t_{j+1})]$, where $\alpha_{i,j} = \alpha(x_j,t_j,u(x_it_j))$. The obvious problem, of course, is that we will not be able to evaluate the coefficients $\alpha i, j+$, in cases where α depends on u (or u_x). We circumvent this problem in such cases by setting $\alpha_{i,j+1} \approx \alpha(x_j,t_{j+1},u(x_it_j))$. The other formulas are modified accordingly, and this is how the M-file is constructed; see Exercise 16.

should be as follows:

```
[ x, t, Ul = backwdtimecentspace(phi, L, A, B, T, N, M, alpha,q)
```

where the input variables, output variables, and functionality are just as in Program 12.3.

The reader is encouraged to compare the performances of the above two programs with the Crank-Nicolson method results of the next example. Another such comparison will be called for in Exercise for the Reader 12.13.

EXAMPLE 12.8:

Consider the heat problem:

$$\begin{cases} (PDE) u_t = \alpha(x,t,u)u_{xx} + q(x,t), \quad 0 < x < L, 0 < t < \infty, u = (x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), \\ u(0,t) = A(t), \ u(L,t) = B(t), \end{cases} \quad 0 < x < L, 0 \leqslant t < \infty \end{cases}$$

where $\phi(x) \begin{cases} x, & \text{if } x < \pi/2 \\ \pi - x, & \text{if } x > \pi/2 \end{cases}$ (a) Use the Crank-Nicolson method to create a mesh plot of the solution for $0 \leqslant t \leqslant 3$ with 25 interior space grid points and 92 interior time grid points. Then obtain a simultaneous two-dimensional plot of several temperature profiles of this numerical solution for various times in this range.

(b) The exact solution is given by $u(x,0) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{\pi(2n+1)^2} sin[(2n+1)x]e^{-(2n+1)^2 t}$(see, e.g., [Asm-00]). How large a value of N can be used so that the corresponding partial sum

$$u_N(x,0) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{\pi(2n+1)^2} sin[(2n+1)x]e^{-(2n+1)^2 t}$$

approximates the exact solution with error less than 10^15, i.e., $|u(x,t) - u_N(x,t)| < 10^15$ for all values of x and t
(c) Use the "exact solution" of part (b) to plot a surface graph of the error of the Crank-Nicolson solution of part (a).
(d) Using the same number of space grid points, use two different numbers of interior time grid points to illustrate the stability condition (46) for this example with the forward-time central-space method.

SOLUTION: Part (a): We first create an M-file for the initial temperature distribution and then use Program 12.3.

```
function y = phi_EG12_8(x)
 for i = 1:length(x);
  if (0<=x(i))&(x(i)<=pi/2)
    y(i)=x(i);
  else
    y (i)=pi-x(i) ;
  end
end
>> alpha=inline('I','x', t','u') ;
>> A=inline('0') ; B=A; q=inline ( 0 ', 'x','t') ;
>> [x, t, UCN) = cranknicolson(@phi_EG12_8, pi , A, B, 3, 25, 92,
alpha,q);
```

To create the surface plot we simply enter:

```
>> surf(x,t,UCN)
>> xlabel('space'), ylabel('time'), zlabel('temperature')
```

The result is shown in Figure 12.24(a). The temperature profile plot shown in Figure 12.24(b) can be obtained by continuing to enter commands similar to the following:

```
>> plot(x,UCR(l,:)), hold on
>> xlabel('space'), ylabel('temperature')
>> gtext('t=0') use nouse to plcice text
>> plot(x,UCR(5,:)), gtext ('t=.323') %continue
```

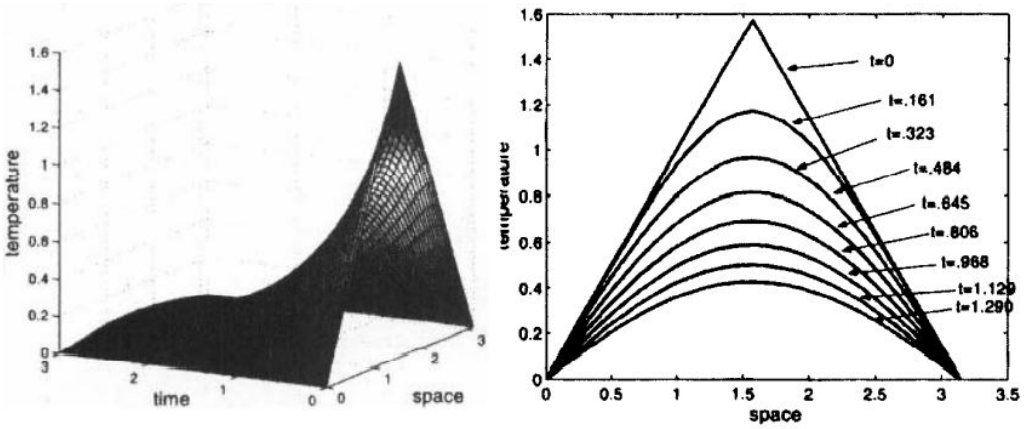

Figure 1.23: (a) (left) Surface graph of the (Crank-Nicolson) solution to the heat problem of Example 12.8. (b) (right) Individual temperature profiles of this solution for some particular values of /. The individual curves in (b) are simply "slices" of the surface in (a) with planes perpendicular to the time axis

(b) We estimate the error by using a standard idea from calculus (see Chapter 5):

$$|u(x,t)-u_N(x,t)| < |\sum_{n=N+1}^{\infty} \frac{(-1)^{n+1}}{\pi(2n+1)^2} sin[(2n+1)x]e^{-2+1^2t}| < \frac{1}{\pi(2n+1)^2} \leqslant \frac{1}{pi}\int_{2N+1}^{\infty} \frac{du}{u^2} = \frac{1}{8\pi(N+1)}$$

Thus, the error will always be less than 1e-15 (regardless of x and t) provided that $1/(8\pi(N+1)^3)) < 1e-15$. Solving for N gives N > 34,139. Thus, we may take the finite sum $u_N(x,t)$ as the exact solution provided $N \geq 34,140$.

(c) We may now create a matrix `Uexact`, of exact solution values on the spacetime grid of part (a). Rather than perform the entire sum for each point, the following code takes advantage of the separated nature of the summands:

```
>> Uexact=zeros(92+2,25+2);
>> for n=0:34140
  V=sin((2*n+l)*x); W=exp(-(2*n+l)A
2*t);
  for i=l:length(t )
   Uexac t (i , O^Uexact (i,:) + (-1)^ (n) *4/pi /(2*n+l)A2*V*W (i) ;
  end
end
>> mesh(x,t,Uexact-UCN)
>> xlabel('space'), ylabel('time'), zlabel('temperature')
```

The result is shown in Figure 12.25.

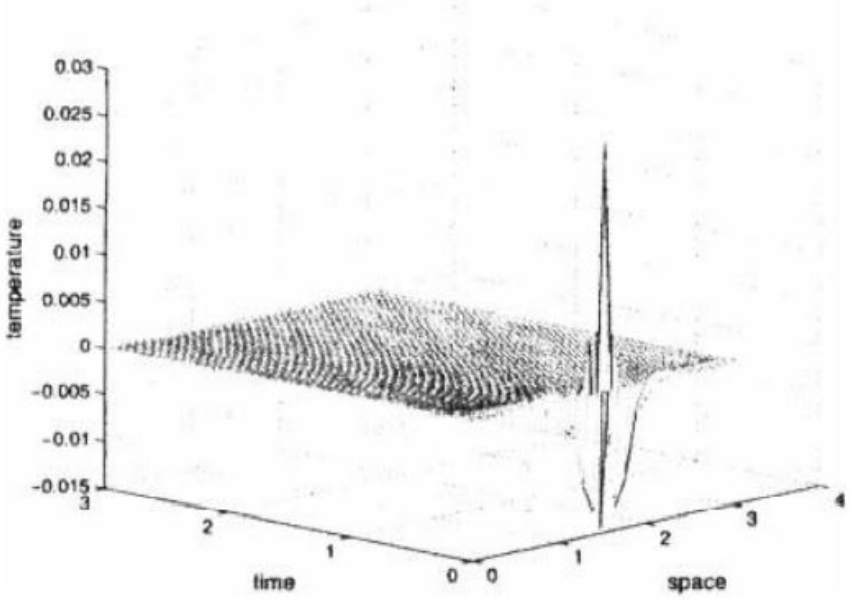

Figure 1.24: : Surface plot of the error for the Crank-Nicolson approximation in part (a) of Example 12.8. Notice how the initially large error (due to the singularity of the boundary data) tends to dissipate very rapidly as time increases.

Part (d): For future reference, we first rewrite the stability condition (46) in terms of the input variables for our M-file solvers:

$$\mu \equiv \frac{T(N+1)^2}{L^2(M+1)} \leq \frac{1}{2}. \tag{1.53}$$

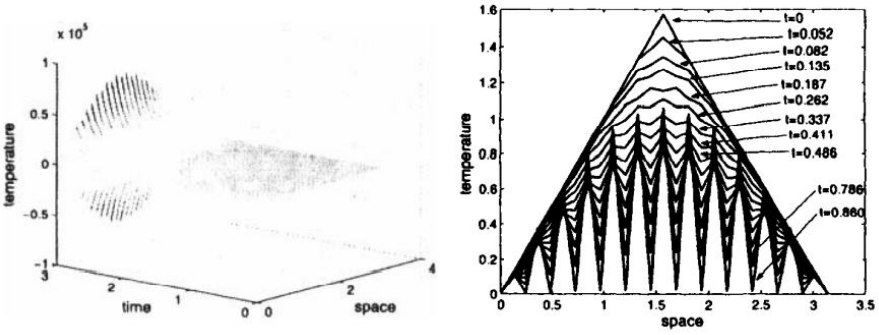

Figure 1.25: Illustration of instability when the forward-time central-space method is used with $\mu \approx .513...$ in the heat problem of Example 12.8. (a) (left) Mesh plot of the forward-time central-space showing the large-scale instability, (b) (right) Simultaneous plots of some heat profiles showing small-scale evolution of instability.

The parameter in part (a) would give a value $\mu \approx 2.23...$ which would result in the forward-time central-space method being quite unstable. We will look at what happens with the forward-time central-space scheme with some values of μ that are much closer to the stability barrier. Using the same number of internal space grid points ($N = 25$), and with $M = 400$ internal time grid points, we would have $\mu \approx .513...$which slightly violates the stability condition. If we run the M-file f `wdtimecentspace` , just as in part (a), we can analogously obtain the mesh and two-dimensional plots shown in Figure 12.26, which illustrate the instability. If instead we used $M = 500$, which results in a stable value of $\mu \approx .410...$, we get a surface graph identical to the one in Figure 12.24(a).

The initial temperature in the last example was continuous but not differentiable, and for such problems, both the backward-time central-space and Crank-Nicolson methods perform admirably well. In general, errors tend to decay with time as they did in the last example. When the initial data is discontinuous, however, the Crank-Nicolson method will sometimes introduce some unwanted oscillations and the backward-time central-space method is usually a better choice for such problems. Of course, the oscillations can be mitigated by using smaller time step sizes, but this might entail significantly more computation than with the backwardtime central-space method. For some theoretical explanations of such pathologies, see Section 9.1 of [Epp-02]. The next exercise for the reader gives an illustration.

EXERCISE FOR THE READER 12.13: Consider the following heat problem:

$$\begin{cases} (PDE) u_t = u_{xx}, \quad 0 < x < \pi, 0 < t < \infty, u = (x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), \\ u(0,t) = u(L,t) = 0, \end{cases} \quad 0 < x < L, 0 \leqslant t < \infty \end{cases}$$

where $\phi(x) \begin{cases} x, & \text{if } x < \pi/2 \\ \pi - x, & \text{if } x > \pi/2 \end{cases}$ (a) Run both the Crank-Nicolson method and the backward-time central-space method with an equivalent set of grids to obtain plots of the resulting numerical solutions as shown in Figure 12.27.
(b) Obtain also mesh plots of these two numerical solutions.

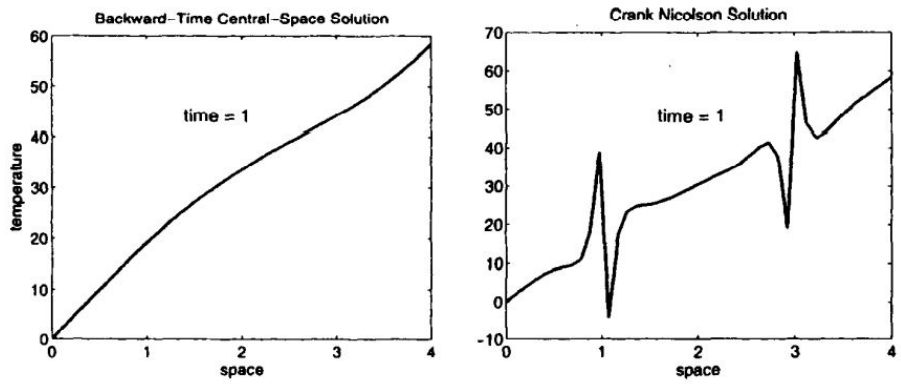

Figure 1.26: With any given grid sizes, the (a) (left) backward-time central-space method usually outperforms the (b) (right) Crank-Nicolson method when the parabolic problem has discontinuous data, as is demonstrated from these two snapshots of the two methods on the problem of Exercise for the Reader 12.13. The oscillations in (b) are not part of the actual solution; using finer time grids will cause them to fade away.

We next move on to boundary conditions involving derivatives. Previously we have separated these sorts of boundary conditions into two kinds: Neumann and Robin. For a physical interpretation in terms of our heated-rod problem, it is helpful to consider both of these conditions as special cases of the following general condition:

$$\partial u/\partial n = -\gamma(u-\eta). \tag{1.54}$$

In general the parameters γ and η are allowed to be functions of other variables, e.g., $\eta = \eta(x,t,W)$, but we have written them as constants for simplicity. In case $\gamma = 0$, we get the basic Neumann boundary condition corresponding to an insulated end of the rod. The general Neumann BC is gotten by taking $\eta = \eta(x) = u(x)+1$ (so (54) becomes $\partial u/\partial n = \gamma$), which corresponds to heat being lost or absorbed by the end of the rod at a specified rate. The general form of (54) with $\gamma \neq 0$ gives Robin BCs that physically correspond to heat being radiated into or out of the rod at a rate proportional to the difference of u and some specified temperature η.

EXAMPLE 12.9:

Consider the heat problem:

$$\begin{cases} (PDE)u_t = u_{xx}, \quad 0<x<\pi, 0<t<\infty, u=(x,t) \\ (BCs)\begin{cases} u(x,0)=100, \\ u(0,t)=u(0,t), u_(1,t), = -u(1,y) \end{cases} \quad 0<x<1, 0 \leqslant t<\infty \end{cases}$$

This corresponds to a laterally insulated rod that is initially uniformly heated to a temperature of 100 and for which heat is being radiated from both ends at a rate equal to the current temperature of the end. Adapt the forward-time central-space method to solve this problem on the time range $0 \leqslant t \leqslant 1$ using a stable grid, and give plots of some temperature profiles.

SOLUTION: Part (a): In order to maintain the 0(h2) quality portion of the local truncation error, we should approximate both derivative boundary conditions using (second-order) central differences. This will require the use of "ghost nodes" at both left and right ends. We modify our indexing scheme for the space variables accordingly as follows:

$$x_i = (i-1)h \ \ (0 \leqslant i \leqslant N+1)$$

Thus the unknown grid values are still indexed as $x_l, \cdots, x_N$ and $x_0 = -h$ and $x_{N+l} = L+h$ correspond to the ghost nodes, but now $L = (N-1)h$. In this notation, we discretize the left boundary condition (at each time level) with the central difference approximation:

$$[u_{2,j} - u_{0,j}/2h] = u_{1,j}$$

where $u_{0,j}$ is the ghost node value. We can eliminate the ghost node in the system (45) by assuming that this discretization is valid at the left end, i.e.,

$$[u_{1,j+1} = (1-2\mu)u_{1,j} + \mu[u_{2,j} + u_{0,j}].$$

Eliminating the ghost node value using the two preceding equations leads us to

$$[u_{1,j+1} = (1-2\mu)u_{1,j} + 2\mu[u_{2,j} + u_{1,j}].$$

In the same fashion, we obtain the following formula for $u_{N,j+1}$ corresponding to the right boundary:

$$[u_{N,j+1} = (1-2\mu)u_{N,j} + 2\mu[u_{N-1,j} + u_{N,j}].$$

To move to the next time level we use these in conjunction with (45) to compute all of the remaining $u_{i,j+1}(1 < i < N)$:

$$[u_{i,j+1} = (1-2\mu)u_{i,j} + 2\mu[u_{i+1,j} + u_{i-1,j}].$$

We can specialize the code of `fwdtimecentspac` e with the above modifications to create a matrix of numerical values for the numerical solution as follows. Below we are using $h = 1/20$ and $k = 1/1850$. This gives a stable value for $\mu = \alpha k/h2 = 1\dot{(}1/1850)/(1/20)^2 = 0.2162... < 1/2$.

```
h=l/20; k=l/1850; mu=k/hA
2;
N=21; M=1851;
U=zeros(M+l,N); x=0:h:l; t=0:k:l;
;A3sign initial time t-0 values and next step t-k values.
for i=l:N, U(l,i)=100; end
Assign values Rt intorior grid points
for j=2:M+l
  U(j,2:N-l) = (l-2*niu) *U (j-1, 2 :N-l) +mu* (U (j-1, 3 :N) +U (j-1, l:
   N-2) ) ;
  U(j,l)= (l-2*mu)*U(j-l,l)+2*mu*(U(j-l,2)-h* U(j-l,l));
  U(j,N)= (l-2*mu)*U(j-l,N)+2*mu*(-h*U(j-l,N)+U(j-1,N-l));
end
```

The simultaneous plots of some temperature profiles, shown in Figure 12.28(a), are now obtained just as in the previous example.

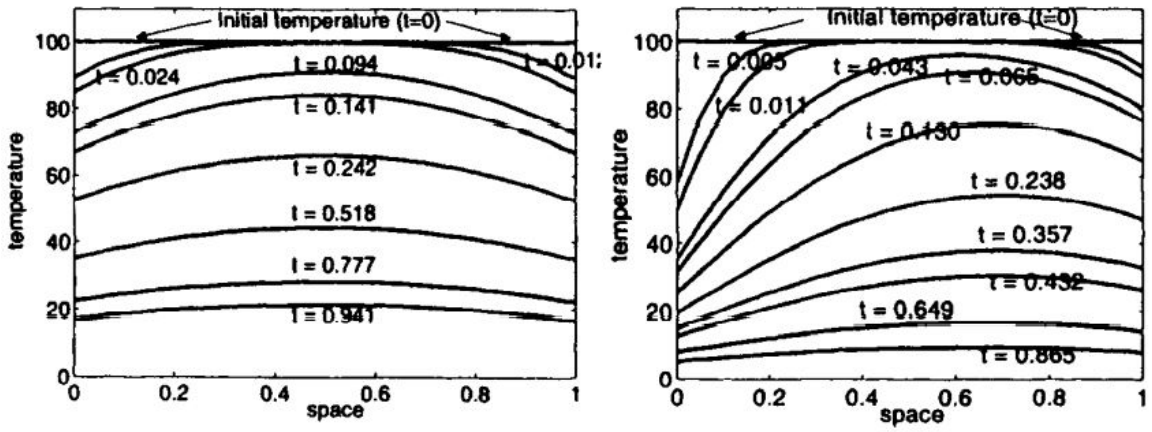

Figure 1.27: (a) (left) Some snapshots of the temperature in the rod of Example 12.9, where heat is lost through equal radiation at both ends, (b) (right) Snapshots for the analogous problem of Exercise for the Reader 12.14. The latter problem has a greater radiation heat loss on the left end rather than on the right, perhaps due to differences in insulation.

EXERCISE FOR THE READER 12.14: (a) Redo the Example 12.9 when the left BC is changed to $u_x(x,t) = u(0,t)^{1.5}$; your plots should look similar to those of Figure 12.28(b).
(b) Will the temperature on the right end eventually be lower than that on the left end? Try to answer this question on physical grounds first, then attempt to use MATLAB to back up your answer. Does the BC of this example have any physical models?
(c) If instead the left BC of the BVP of Example 12.9 was changed to $u_x(0,t) = -u(0,t)$, what do you think would eventually happen to the temperature in the rod? Try to answer this question on physical grounds first, then attempt to use MATLAB to back up your answer.

Similar methods can be developed for any other kinds of boundary conditions, and everything can be done likewise for the Crank-Nicolson or the purely implicit method. There will be some exercises to give more practice with this.
EXERCISE FOR THE READER 12.15: (a) Write an M-file with the following syntax:

```
[x, t, U] = cranknicolsonRobinLR(phi, L, A, B, T, N, M, alpha,q)
```

that will numerically solve the following BVP:

$$\begin{cases} (PDE)u_t = \alpha(x,t,u)u_{xx} + q(x,t), \quad 0 < x < L, 0 < t < \infty, u = (x,t) \\ (BCs)\begin{cases} u(x,0) = \phi(x), \\ u(0,t) = \alpha u_x(0,t) + b, u_x(L,t) = cu(0,t) + d \end{cases} \quad 0 < x < 1, 0 \leqslant t < \infty \end{cases}$$

where a, b, c, and d are constants. The input variables and functionality are similar to those of Program 12.3, except here the input variables A and B represent the vectors [a b] and [b c], respectively.

(b) Apply the program to re-solve the problem of Example 12.9 (using the same values for M and N) and compare plots of the corresponding numerical temperature profiles with thoseof Figure 12.28(a) (obtained using the explicit method).

We end this section with some theoretical developments for a heat problem and finite difference methods. For simplicity, we work with a basic Dirichlet problem for the one-dimensional heat equation of the following theorem. Many of the results, ideas, and concepts generalize to other parabolic BVPs.

THEOREM 12.2:

(*Existence and Uniqueness for a Heat Problem*) Suppose that L and T are positive numbers and $A(t)9B(t)$, and $\phi(x)$ are continuous functions. The following BVP (42) has a unique solution

$$\begin{cases} (PDE) u_t = \alpha u_{xx} + q(x,t), \quad 0 < x < L, 0 < t < \infty, u = (x,t) \\ (BCs) \begin{cases} u(x,0) = \phi(x), \\ u(0,t) = A(t), \;\; u(L,t) = B(t) \end{cases} \quad 0 < x < 1, 0 \leqslant t < \infty \end{cases}$$

Elements of the proof of this result can be found in the exercises; see also [Asm00] (in particular, see Sections 3.5 and 3.10 therein). We now give a formal definition of stability of a finite difference scheme for this (BVP) (42).

Definition: A finite difference scheme for the BVP (42) that satisfies the hypotheses of Theorem 12.2 is **unconditionally stable** if there exists a constant $C > 0$ (depending only on the data α, q, A, B of the BVP) such that for any initial profile function $\phi(x)$, any grid of the space interval $[0, L]$ and any grid of the time interval $[0, T]$, we have

$$\max_{i,j} |u_i^j| \leqslant C \max_{0 \leqslant x \leqslant L} |\phi(x)| \tag{1.55}$$

In words, this roughly states that the numerical values do not get too much larger than the initial data. The method is **conditionally stable** if (54) holds provided that the uniform time step size (k) and space step size (h) satisfy some specified relationship.

Note that if we used a stable finite difference method to solve the problem (42) with perturbed initial heat profile $\bar{\phi}(x)$ with $|\bar{\phi}(x) - \phi(x)| < \varepsilon$ where ε is a very small number, and if $\bar{u}_i^j$ą denotes the resulting numerical values, then $|\bar{u}_i^j - u_i^j| < C\varepsilon$. (Proof: Apply the stability inequality (55) to the BVP (42)with initial heat profile changed to $\bar{\phi}(x) - \phi(x)$.) The advantage of stable schemes is that any roundoff errors propagate only in an additive fashion. In unstable schemes, as we have seen, the errors usually propagate exponentially.

Using the so-called von Neumann approach, we will establish the stability results that have been stated thus far in the section as well as others in the context of the (42). [19] To help make the ideas more transparent, we assume the equation and the boundary conditions are homogeneous: $q(x,t) = 0, A(t), B(t) = 0$. Let us begin with the (homogeneous) explicit method (45):

$$u_{i,j+1} = (1 - 2\mu)u_{i,j} + \mu[u_{i+1,j} + u_{i-1,j}]$$

For the homogeneous problem, in addition to the guaranteed existence and uniqueness from the above theorem, we also have the **maximum principle**: The solution satisfies $|u(x,t)| \leqslant \max|\phi(x)|$. Physically, this simply says that the maximum temperature of the rod is attained initially. [20] (discrete) solution of the finite difference method has the following separated form:

$$u_{i,j} = X_i T_j \tag{1.56}$$

Substitution of (56) into the above finite difference scheme and dividing by $X_i T_j$ produces:

$$\frac{T_{j+1}}{T_j} = 1 - 2\mu + \mu \frac{X_{i+1} T_{j-1}}{X_i} \tag{1.57}$$

Since the left sequence depends only on j while that on the right depends only on i, it follows that both sides must equate to the same constant. Calling this constant ξ, the temporal (time) portion of (57) yields$T_{j+1} = \xi T_j$ and hence

$$T_j = \xi^j T_0 \tag{1.58}$$

The spatial portion of (57): $1 - 2\mu \frac{X_{i+1} T_{j-1}}{X_i} = \xi$, will take a bit more effort Xą to solve. We assume that X_1. takes the form: $X_i = Acosi\theta + sini\theta$, [21] where the parameters A and B and θ are to be determined. The left (homogeneous) boundary condition forces $A = 0$. The right (homogeneous) boundary condition now implies that $(N+1)\theta = n\pi$ for some positive

[19] This approach was used in the exercises of the last section. There we used complex notation which afforded a more efficient analysis; the approach in the text above will use only real numbers, but the exercises will revisit the complex number approach.

[20] The maximum principle holds more generally in the presence of time-dependent Dirichlet boundary conditions $u(0,t) = A(t), \;\; u(L,t) = B(t)$, in which case the inequality changes to $|u(x,t)|M$ where M is the largest of max $| |\phi(x)|$, max $|A(t)|$, and max $|B(i)|$ (if thesemaxima exist). For a proof, se Section 3.10 of [Asm-00]. The maximum principle for the heat equation ceases to be valid in case of heat sources; see Exercise 21.

[21] Here i is an index (not a complex number). For readers who are familiar with the analytic theory of ODE, the choice of the form of X_i was motivated by viewing the spatial equation $\frac{X_{i+1} - X_{j-1}}{X_i} + 1 - 2\mu = \xi$ as a discretization of the second-order ODE: $\mu X^n(s) + 1 - 2\mu = \xi$.

integer n. Combining this with the relation $(N+1)h = L$ (h = spatial step size), produces $\theta = n\pi h/L$. The(nonzero) value of the parameter B may now be arbitrarily specified (since it cancels out of the spatial equation and the boundary conditions are already guaranteed, note that we have not yet dealt with the boundary condition). Using $B = 1$ gives the following candidate solution to the spatial equation:

$$X_i = sin(in\pi h/L). \tag{1.59}$$

Substituting this into the spatial equation gives:

$$1 - 2\mu + \mu \frac{sin((i+1)n\pi h/L) + sin((i-1)n\pi h.L)}{sin(in\pi h/2)}$$

If we apply the trig addition formulas (e.g., $sin((i+1)n\pi h/L) = sin(in\pi h/L)cos(n\pi h/L) + sin(in\pi h/L)cos(n\pi h/L)$ we can convert the above equation to the following form:

$$\xi = \xi(n) = 1 - 2\mu(1 - cos(n\pi h/L)). \tag{1.60}$$

Since the resulting separated solutions $\xi^j T_0 sin(in\pi h/L)$ of the finite difference scheme exist for any integer n, and since we know (from the maximum principle) the actual solution to the BVP is bounded, it follows that we must have $|\xi(n)| \leqslant 1$ because otherwise this finite difference solution will grow exponentially. Since n can be arbitrary (because for stability h can be arbitrary) the $cos(n\pi h/L)$ can get arbitrarily close to -1 , so (59) shows us that in order for $|\xi(n)| \leqslant 1$, we must have $2\mu \leqslant 1$ or $\mu \leqslant 1/2$, which is the stability condition (46)

This von Neumann method can be applied to obtain stability results for many finite difference schemes for a wide range of BVPs. For example, the following finite difference scheme (for the same homogeneous problem considered above):

$$\frac{u_{i,j+1} - u_{j,k}}{k} = \frac{\alpha}{2}[(1-\sigma)\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \sigma\frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{h^2}] \tag{1.61}$$

where $0 \leqslant \sigma \leqslant 1$, includes as special cases the explicit method ($\sigma = 0$), the backward-time central-space method (49) ($\sigma = 1$), and the Crank-Nicolson ($\sigma = l/2$). Note that this method is always implicit when $\sigma > 0$. The von Neumann method can be used to show this method is unconditionally stable whenever $1/2 \leqslant sigma \leqslant 1$ and when $0 \leqslant \sigma \leqslant 1/2$ the method will be stable if the following condition holds:

$$\mu = \alpha k/h^2 < 1/(2 - 5\sigma). \tag{1.62}$$

There are other approaches to stability theory of finite difference methods. One of these, known as the spectral method, looks at eigenvalues of matricesassociated with the finite difference methods. The exercises will delve deeper into stability theory. Some references include [Smi-85], [RiMo-67], and [IsKe-66] and the more elementary [DuCZa-89].

EXERCISES 12.3

NOTE: For convenience in these exercises, we will refer to the forward-time central-space method (44) simply as the "explicit method" and the backward-time central-space method as the "implicit method."

1. Use the explicit method to solve the following BVP on $0 \leqslant t \leqslant 1$ using the indicated grids:

$$\begin{cases} (PDE) u_t = u_{xx}, \quad 0 < x < L, 0 < t < \infty, u = (x,t) \\ (BCs) \begin{cases} u(x,0) = sin(\pi x)(1 + 2cos(\pi x)), \\ u(0,t) = 0, u(1,t) = 0 \end{cases} \quad 0 < x < 1, 0 \leqslant t < \infty \end{cases}$$

   In each case, compare the results with the actual solution $u(x,t) = sin(\pi x)e^{-\pi^2 t} + sin(2\pi x)e^{-4\pi^2 t}$ at the indicated time levels.
   (a) $N = 19, M = 39$. Note that this set of parameters violates the stability condition, by (P8), $\mu = 1$. Compare the numerical solution with the exact solution at time levels $t = .25, t = .5, t = .75, t = 1$.
   (b) $N = 39, M = 99$. Note that this set of parameters satisfies the stability condition with $mu = 0.4$. Compare numerical solution with exact solution at time levels $t = .5, t = 1, t = 1.5, t = 2$

2. a) Re-solve the BVP of Exercise 1 using the Crank-Nicolson method with $N = 39$ and $M = 99$
   and give a mesh plot of the error of this numerical solution. (b) Repeat part (a) this time using the implicit method.

3. For each of the following BVPs, do the following: (i) Use the explicit method to solve the problem on the indicated time interval. Begin with 10 interior space node values and a corresponding number of interior time node values that results in a stable scheme, (ii) Continue halving the space step size and using a smaller (stable) time step that evenly divides into the previous time step. Compare common temperature profiles of adjacent numerical solutions. Keep track of the maximum discrepancy. Continue until the maximum discrepancy is smaller than 10^{-3} or the numerical computations take more than two minutes on MATLAB. (iii) For your final numerical solution,

plot a (three-dimensional) mesh plot of the solution, (iv) For your final numerical solution, plot (and label) several two-dimensional temperature profiles in the same graph.

$$(a)\begin{cases}(PDE)u_t = u_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=50,\\ u(0,t)=0, u(1,t)=100\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

$$(b)\begin{cases}(PDE)u_t = u_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=\begin{cases}2x, & \text{if } 0\leqslant x\leqslant 1/2\\ 1, & \text{if } 1/2\leqslant x\leqslant 1\end{cases},\\ u(0,t)=0, u(1,t)=0, !0\leqslant t\leqslant 1\end{cases}\end{cases}$$

$$(a)\begin{cases}(PDE)u_t = u_{xx}, \quad 0<x<pi, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=100x(\pi-1)(,\\ u(0,t)=50sin(\pi x), u(\pi,t)=0\end{cases} \quad 0<x<\pi, 0\leqslant t\leqslant 1\end{cases}$$

$$(d)\begin{cases}(PDE)u_t = 2u_{xx}, \quad 0<x<\pi, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=\begin{cases}100\pi x,\\ 100\pi^2-100\pi x\end{cases} \quad 0<x<\pi, 0\leqslant t\leqslant 1,\\ u(0,t)=50sin(\pi x), u(\pi,t)=0\end{cases}\end{cases}$$

4. Parts (a) through (d): For each of the BVPs Exercise 3, repeat the directions of that problem, this time using the Crank-Nicolson method. Try using your temporal step size k to satisfy $h/2\leqslant k\leqslant h$, where h is the spatial step. Try redoing some of these numerical solutions using a much smaller temporal step size (as stipulated by the stability condition (46) for the explicit method). Does this seem like a better strategy than using roughly the same step sizes? In answering this latter question, you should, of course, weigh in the extra work needed for a given spatial step size.

5. Parts (a) through (d): For each of the BVPs of the Exercise 3, repeat the directions of that problem, this time using the implicit method. Try using your temporal step size k to satisfy $h/2\leqslant k\leqslant h$, where h is the spatial step. Try redoing some of these numerical solutions using a much smaller temporal step size (as stipulated by the stability condition (46) for the explicit method). Does this seem like a better strategy than using roughly the same step sizes? In answering this latter question, you should, of course, weigh in the extra work needed for a given spatial step size.

6. Consider the following heat problem:

$$\begin{cases}(PDE)u_t = u_{xx}, \quad 0<x<1, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=sin(\pi x)\\ u(0,t)=t, u(1,t)=0\end{cases} \quad 0<x<1, 0\leqslant t<\infty\end{cases}$$

This can be interpreted as the modeling of a rod of length one with initial heat distribution as specified, right end being maintained at temperature zero, and left end being constantly heated up so has to have a temperature of / at time t.
(a) Modify the explicit method to estimate the time $t*$ it takes for the temperature at the midpoint $t=1/2$ to first reach a value of $u=2$. Run the new program toestimate the time /* by trying out several different grid choices.
(b) Repeat part (a), this time using the implicit method.
(c) Repeat part (a), this time using the Crank-Nicolson method.

7. For each of the following BVPs, do the following: (i) Use the explicit method to solve the problem on the indicated time interval. Begin with 10 interior space node values and a corresponding number of interior time node values that results in a stable scheme, (ii) Continue halving the space step size and using a smaller (stable) time step that evenly divides into the previous time step. Compare common temperature profiles of adjacent numerical solutions. Keep track of the maximum discrepancy. Continue until the maximum discrepancy is smaller than 10^{-3} or the numerical computations take more than two minutes on MATLAB. (iii) For your final numerical solution, plot a (three-dimensional) mesh plot of the solution, (iv) For your final numerical solution, plot (and label) several two-dimensional temperature profiles in the same graph.

$$(a)\begin{cases}(PDE)u_t = \alpha(x)u_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=50,\\ u(0,t)=0, u(1,t)=100\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

where $\alpha(x)\begin{cases}1, & \text{if } x\leqslant 1/2\\ 4, & \text{if } x>1/2\end{cases}$

$$(b)\begin{cases}(PDE) \quad u_t=\alpha(x)u_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=\begin{cases}100, & \text{if } 1/4\leqslant x\leqslant 3/4\\ 0, & otherwise\end{cases}\\ u(0,t)=0, u(1,t)=0, 0\leqslant t\leqslant 1\end{cases}\end{cases}$$

where $\alpha(x)$ is as in part (a).
(c) Same BVP as (a) but change $\alpha(x)$ to $\alpha(u)=(1/25)\sqrt{u^2+1}$.
(d) Same BVP as (b) but change the PDE to $u_t=\alpha(x)u_{xx}+1(x,t)$, where

$$q(x,t)=\begin{cases}200sin(3\pi t/2), & \text{if } 0\leqslant t\leqslant 2/3 \text{and} x\in[1,1/8]\cup[7/8,1]\\ 0, & otherwise\end{cases}$$

8. Parts (a) through (d): For each of the BVPs of Exercise 8, repeat the directions of that problem, this time using the Crank-Nicolson method. Try using your temporal step size k to satisfy $h1/2\leqslant k\leqslant h$, where h is the spatial step. Try redoing some of these numerical solutions using a much smaller temporal step size (as stipulated by the stability condition (46) for the explicit method). Does this seem like a better strategy than using roughly the same step sizes? In answering this latter question, you should, of course, weigh in the extra work needed for a given spatial step size.

9. Parts (a) through (d): For each of the BVPs of Exercise 7, repeat the directions ofthat problem, this time using the implicit method. Try using your temporal step size k to satisfy $h/2\leqslant k\leqslant h$, where h is the spatial step. Try redoing some of these numerical solutions using a much smaller temporal step size (as stipulated by the stability condition (46) for the explicit method). Does this seem like a better strategy than using roughly the same step sizes? In answering this latter question, you should, of course, weigh in the extra work needed for a given spatial step size.

10. Rewrite Program 12.3 (for the Crank-Nicolson method) so that it avoids the creation of any square matrices, but is otherwise the same program; in particular, the input and output variables and functionality of the two programs should be identical. Find a problem and corresponding input parameters where your modified program noticeably outperforms Program 12.3.

11. (*Some Related Neumann Problems*) For each of the following BVPs, set up an appropriate finite difference scheme and numerically solve the problem. Continue to re-solve the problem with a decreasing set of space steps and corresponding decreasing time steps (in a stable way), compare consecutive numerical solutions (at common grid values), and continue until the maximum error becomes less than 0.001 or the computations take more than two minutes. Comment on the stability (or lack thereof) of your method on the problem. In case of stability, plot your final solution, both as a three-dimensional surface plot and as a two-dimensional plot of several superimposed time level profiles.

$$(a)\begin{cases}(PDE)u_t=u_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=sin(x),\\ u(0,t)=0, u(1,t)=0\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

$$(b)\begin{cases}(PDE)u_t=u_{xx}+u, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=sin(x),\\ u_x(0,t)=0, u(1,t)=0\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

$$(c)\begin{cases}(PDE)u_t=u_{xx}+2y, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=sin(x),\\ u_x(0,t)=0, u(1,t)=0\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

$$(d)\begin{cases}(PDE)u_t=u_{xx}+u_x, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=sin(x),\\ u_x(0,t)=0, u(1,t)=0\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

12. For each of the following BVPs, set up an appropriate finite difference scheme and numerically solve the problem. Continue to re-solve the problem with a decreasing set of space steps and corresponding decreasing time steps (in a stable way), compare consecutive numerical solutions (at common grid values), and continue until the maximum error becomes less than 0.001 or the computations take more than two minutes. Comment on the stability (or lack thereof) of your method on the problem. In case of stability, plot your final solution, both as a three-dimensional surface plot and as a two-dimensional plot of several superimposed time level profiles.

$$(a)\begin{cases}(PDE)u_t=u_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=100,\\ u(0,t)=-20, u(1,t)=0\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

$$(b)\begin{cases}(PDE)u_t = u_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=100,\\ u(0,t)=-20, u(1,t)=u-90\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

$$(c)\begin{cases}(PDE)u_t = 2xu_{xx}, \quad 0<x<1, 0<t\leqslant 1, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=100,\\ u(0,t)=-20, u(1,t)=u-90\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

$$(c)\begin{cases}(PDE)u_t = xu_{xx}, \quad 0<x<1, 0<t\leqslant 2, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=100,\\ u(0,t)=-20, u(1,t)=u-90\end{cases} \quad 0<x<1, 0\leqslant t\leqslant 1\end{cases}$$

13. (*Richardson's Method-Experimental Instability*) Recall that Richardson's method for the heat equation $u_t = au_{xx}$ uses centered difference approximations for both the time and space derivative terms. Thus it takes the following form:

$$\frac{u(x_i,t_{j+1})-2(u_i,t_j)-u(x_i,t_{j-1})}{k^2} = \frac{u(x_{i+1},t_{j+1})-2(u_i,t_j)-u(x_i,t_{j-1})}{k^2}$$

Richardson's method turns out to be unstable for any choice of space and time steps (therefore it is an unconditionally unstable method).
(a) Perform Richardson's method on a BVP (of your choice) to demonstrate this instability for values of $mu = \alpha k/h^2 4$.
(b) Write an M-file that will perform Richardson's method on the BVP of Program 12.3; the syntax, and input and output variables should be just as in Program 12.3, i.e.,

```
[ x, t , U] = richardson(phi , L, A, B, T, N, M, alpha , q) .
```

Run your program to reproduce the results of part (a). Also, run your program on the BVP of Example 12.7 (with similar step sizes) and compare the performance with that witnessed for the Crank-Nicolson method in that example.

14. (*Richardson's Method-Theoretical Instability*) (a) Perform a von Neumann stability analysis using real numbers (as in this section) or complex arithmetic to show that Richardson's method (Exercise 13) is always unstable for the BVP:

$$\begin{cases}(PDE)u_t = \alpha u_{xx}, \quad 0<x<L, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=\phi(x),\\ u(0,t)=0, u(L,t)=0\end{cases} \quad 0<x<L, 0\leqslant t<\infty\end{cases}$$

Suggestion: Although the real number notation used in the text will work, complex notation, as used in the exercises of the last section, will yield a more succinct proof

15. In this exercise we will consider the BVP

$$\begin{cases}(PDE)u_t = \alpha u_{xx}, \quad 0<x<L, 0<t<\infty, u=(x,t)\\ (BCs)\begin{cases}u(x,0)=\phi(x),\\ u(0,t)=0, u(L,t)=0\end{cases} \quad 0<x<L, 0\leqslant t<\infty\end{cases}$$

along with the following finite difference scheme:

$$\frac{u(x_i,t_{j+1})-2(u_i,t_j)-u(x_i,t_{j-1})}{k^2} = \alpha\frac{u(x_{i+1,t}+u(x_{i-1,t}-u(x_{i,t+1}-u(x_{i,t-1}),}{h^2}$$

Like Richardson's method (Problem 13), this one used a centered difference approximation for the time derivative, but a different sort of space derivative approximation.
(a) Use this method to solve the BVP of Example 12.8 using the same step sizes that were used in that example (with the Crank-Nicolson method), and compare errors using the "exact" solution given in that example.
(b) Show that the local truncation error of this method is $0(h^2+k^2)$.
(c) Perform a von Neumann stability analysis to show that thismethod is unconditionally stable.

16. (a) Show that with the general formulation of the Crank-Nicolson method given in the footnote for Program 12.3, equation (50) takes on the following form:

$$-\mu_{i,j+1}\mu_{i-1,j+1}+2(+1\mu_{i,j+1})\mu_{i,j+1}-\mu_{i,j+1}\mu_{i+1,j+1}=\mu_{i,j}\mu_{i-1,j}+2(l-\mu_{i,j}\mu_{i,j}+\mu_{i,j}\mu_{i+1,j}+(k/2)[q_{i,j}+q_{i,j+1}]$$

where $\mu_{i,j}=\alpha_{i,j}k/h2$

(b) In cases where a depends on w (and/or wx), the approximation on which our `cranknicolso` n program is based is rather Spartan. Experiment with some particular BVPs where $\alpha = \alpha(u)$ and modify this program to incorporate the first-order (rather than zeroth-order) approximation, until you find one in which the latter method gives improved results.

$$\alpha_{i,j+1} \approx \alpha((x_i,t_{j+1},u(x_i,t_j),[u(x_{i-1},t_j)]/2h) + \partial\alpha/\partial u\alpha((x_i,t_{j+1},u(x_i,t_j)[u(x_{i-1},t_j)]/2h)[u(x_i,t_j)-u(x_i,t_{j-1})]$$

Suggestion: Unless you can find such a problem with a known analytic solution, you should judge the success of the two methods by checking to see how much the numerical results change when both temporal and spatial steps are cut in half.

17. *(*Finite Difference Schemes for Heat Flow in Two Space Dimensions) As was done with hyperbolic equations in Section 12.2, the finite difference methods of this section can be extended to deal with the heat equation (and other parabolic equations) in two-space variables. This exercise outlines the procedure for the following Dirichlet problem on a rectangle:

$$\begin{cases} (PDE) u_t = \alpha(u_{xx}+u_{yy}), & 0<x<a, 0<y<b, 0<t<\infty, u=(x,y,t) \\ (BCs) \begin{cases} u(x,0)=\phi(x,y), & 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty \\ u(0,t)=A(x,y), & \text{for all} (x,y) \text{ on the boundary of the rectangle:} \\ & R=\{0\leqslant x\leqslant a, 0\leqslant y\leqslant b\} \end{cases} \end{cases}$$

(a) For a fixed time interval $0\leqslant t\leqslant T$ introducing a grid as in (40) and letting $u^t_{i,j} = u(x_i,u_j,t_t)$ derive the following forward-time central-space finite difference approximation of the PDE:

$$u^{t+1}_{i,j} = \mu(1-\sigma)[u^t_{i+1,j}+u^t_{i-1,j}u^t_{i,j+1}+u^t_{i,j-1}] + \mu\sigma[u^t_{i+1,j+1}+u^{t+1}_{i-1,j}u^{t+1}_{i,j+1}+u^{t+1}_{i,j-1}] + (1-4\mu)u^t_{i,j}.$$

This scheme is uniformly stable if $i/2\leqslant\sigma\leqslant 1$ and otherwise the stability condition becomes $\mu = ak/h^2 < 1/(4-8\sigma)$. When $\sigma = 1/2$, it is referred to as the Crank-Nicolson method since it naturally generalizes the method in one space variable.
Note: The stability assertions can be established by the von Neumann method; see [RiMo-67] or [IsKe-66].

18. (*Cooling of a Uniformly Heated Slab*) Consider the following heat problem:

$$\begin{cases} (PDE) u_t = \alpha(u_{xx}+u_{yy}), & 0<x<a, 0<y<b, 0<t<\infty, u=(x,y,t) \\ (BCs) \begin{cases} u(x,,y,0)=T_o, & 0\leqslant x\leqslant a, 0\leqslant y\leqslant b, 0\leqslant t<\infty \\ u(x,y,t)=0, & \text{for all} (x,y) \text{ on the boundary of} \{0\leqslant x\leqslant a, 0\leqslant y\leqslant b\} \end{cases} \end{cases}$$

Physically, this problem can be thought of as the cooling of a thin rectangular slab with insulated lateral surfaces and whose edges are maintained at temperature of 0. Initially, the temperature is T_0 . We have left the diffusivity as variable.
(a) Use the forward-time, central-space explicit method of Exercise 17(a) to numerically solve this problem on the time interval $0\leqslant t\leqslant 1$ using 10 interior space grid nodes in both the $x-$ and $y-$directions. Use the following data: $a=b=1, T_0=100, \alpha=1$ and perform your solution on the time interval $0\leqslant t\leqslant 1$. Give three-dimensional snapshots of temperature profiles at the times (in the time grid as close as possible to) $t=0,.2,A,.6,.8,1$.
(b) Repeat part (a), this time using the Crank-Nicolson method of Exercise 17(c). Do it first with the grid that would be suitable for part (b), then repeat with a grid with approximately the same total number of internal nodes (spatial-temporal) but where the step sizes are the same for each of the three variables (x, y, and t).
(c) The exact solution of this BVP can be expressed as:

$$u(x,y,t) = \frac{16T_0}{\pi^2}\left\{\sum_{n=0}^{\infty}\frac{1}{2m+1}sin((2n+1)\pi x/a)exp(-\pi^2(2n+1)^2\alpha t/a^2)\right\}x$$
$$\left\{\sum_{n=0}^{\infty}\frac{1}{2m+1}sin((2m+1)\pi y/b)exp(-\pi^2(2m+1)^2\alpha t/b^2)\right\},$$

see Section 3.7 of [Asm-00]. Find a positive integer N so that the partial sum product:

$$u(x,y,t) = \frac{16T_0}{\pi^2}\left\{\sum_{n=0}^{\infty}\frac{1}{2n+1}\cdots\right\}\ x\ \frac{16T_0}{\pi^2}\left\{\sum_{n=0}^{\infty}\frac{1}{2m+1}\cdots\right\}$$

approximates the exact solution with an error less than 10^6 uniformly for all x, y and t between O and 1.
How much better will the accuracy of this approximation be for each of the temperature profiles corresponding to

the time values (in the time grid as close as possible to) $t = .2, .4., .6, .8, 1$?
(d) Use the "exact" solution of part (c) to obtain three-dimensional mesh plots of the errors of each of the snapshots obtained in part (a), and then for each of those obtained in part (b).
(e) Using the numerical solution of part (a), estimate the time it takes for the maximum temperature on the slab to decrease to 50. Repeat with the numerical solutions of part (b) and finally with the "exact" solution of part (c).
(f) For a particular grid on the x- and y-axes, the average temperature on the plate at time level t0 can be discretely defined by $\frac{1}{n_x N_y}\sum_{i=1}^{N_x}\sum_{j=1}^{N_y} u(x_i, y_j, t)$ Using the numerical solution of part (a), estimate the time it takes for the average temperature on the plate to decrease to 50. Repeat with the numerical solutions of part (b) and finally with the "exact" solution of part (c).
Suggestions: For parts (a) and (b) use some of the MATLAB techniques introduced in Example 12.7, Program 12.2 and the development that precedes it. In part (b), to find (approximately) the correct step size h_h solve the equation $h_b^3 = h_a^2 k_a$, where ha and *e denote the spatial and temporal step sizes, respectively, that were used in part (a). For part (c), use the ideas from Example 12.8(b). For part (d), much computation can be saved by making use of the separated nature of the exact solution.

19. Repeat all parts of Exercise 18, keeping everything the same, except now use the value $\alpha = 2$.

20. Repeat all parts of Exercise 18, keeping everything the same, except now use the value $\alpha = 4$.

21. (*Cooling of a Half-Heated Slab*) Consider the following heat problem:

$$\begin{cases} (PDE) u_t = u_{xx} + u_{yy}, \quad 0 < x < a, 0 < y < b, 0 < t < \infty, u = (x,y,t) \\ (BCs) \begin{cases} u(x,y,0) = \phi(x,y), \quad 0 \leqslant x \leqslant a, 0 \leqslant y \leqslant b, 0 \leqslant t < \infty \\ u(x,y,t) = 0, \qquad \text{for all} (x,y) \text{ on the boundary of} \{0 \leqslant x \leqslant a, 0 \leqslant y \leqslant b\} \end{cases} \end{cases}$$

where $\phi(x,y) \begin{cases} 100, & \text{if } y \leqslant x \\ 0 & otherwise \end{cases}$,
(a) Use the backward-time, central-space purely implicit method of Exercise 17(c) with $\sigma = 1$ to numerically solve this problem on the time interval $0 \leqslant t \leqslant 1$ using 25 interior grid values in each of the $x-$ and $y-$ and t-directions. Give three-dimensional snapshots of temperature profiles at the times (in the time grid as close as possible to) $t = 0, .2, .4., .6, .8, 1$. Repeat using 50 interior grid values for each variable.
(b) Repeat part (a), this time using the Crank-Nicolson method of Exercise 17(c).
(c) The exact solution of this BVP can be expressed as:

$$u(x,y,t) = 100 \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} C_{m,m} sin(n\pi x) sin(m\pi x) exp(-\pi^2(n^2+m^2)t)$$

where $C_{x,m} = \begin{cases} 2((-1)^n((-1)^m - 1)n^2 + ((-1)m^2)/[nm(n^2 - m^2)\pi^2], & \text{if } n \neq m \\ 2((-1)^n - 1)^2/[n^2\pi^2], & \text{if } n = m \end{cases}$
(see Section 3.7 of [Asm-00]). Find a positive integer N so that the partial sum:

$$u(x,y,t) = 100 \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} C_{n,m} \cdots$$

approximates the exact solution with an error less than 10"* uniformly for all x,y, and t between O and 1.
How much better will the accuracy of this approximation be for each of the temperature profiles corresponding to the time values $t = .2, .4., .6, .8, 1$?
(d) Use the "exact" solution of part (c) to obtain three-dimensional mesh plots of the errors of each of the snapshots obtained in part (a), and then for each of those obtained in part (b).
(e) Repeat part (e) of Exercise 18 for the BVP of this problem.
(f) Repeat part (f) of Exercise 18 for the BVP of this problem.

22. Repeat all parts of Exercise 21, keeping everything the same, except now change the PDE $tou_t = 2(u_{xx}u_{yy})$.

23. Repeat all parts of Exercise 21, keeping everything the same, except now change the PDE to u$tou_t = 4(u_{xx}u_{yy})$.

24. (*Failure of the Maximum Principle for Heat Problems with Sources*) Consider the following heat problem:

$$\begin{cases} (PDE) u_t = u_{xx} + 2(t+1) + x(1-x), \quad 0 < x < a, 0 < y < b, 0 < t < \infty, u = (x,y,t) \\ (BCs) \begin{cases} u(x,0) = x(x,0) = x(1-x), \\ u(0,t) = 0, u(1,t) = 0 \end{cases} \quad 0 < x < 1, 0 \leqslant t < \infty \end{cases}$$

(a) Show that $u(x,t) = (t+1)x(1-x)$ solves this BVP and violates the maximum principle stated in the section in that the internal temperature of the rod can exceed the boundary and initial temperature values.
(b) Apply each of the three methods, implicit, explicit, and Crank-Nicolson, to this problem with comparable grids and compare the numerical results with the solution of part (a).

This page intentionally left blank.

Chapter 2

The Finite Element Method

2.0.1. A NONTECHNICAL OVERVIEW OF THE FINITE ELEMENT METHOD

The Finite Element Method (FEM) is actually a large collection of numerical methods for solving PDEs. It was first devised as a numerical tool by mathematician Richard Courant [1]in a 1943 paper on torsion problems [Cou-43], and is based on analogous techniques and principles to those developed in the early twentieth century for one-dimensional boundary value problems, as were presented in Section 10.5. The method was extensively elaborated on during the 1950s and 1960s by engineers as a practical approach for solving various PDEs in structural engineering. In the 1960s and 1970s, mathematicians worked to give the method a solid theoretical basis and extended it as a tool for solving many different PDE problems. Active research on this method continues today and it has become the most commonly used numerical method for partial differential equations. As a very pertinent example, in MATLAB's PDE Toolbox, all of the programs for solving PDEs use FEMs. Writing even somewhat general programs for FEMs is a very complicated task. Our goal in this chapter will be to explain the method and to write some programs to implement it in several specific instances. This should be sufficient for readers needing to delve deeper into FEMs to be able to extend the programs into more general ones. MATLAB's PDE Symbolic Toolbox programs are open to its users to read and modify. So, in principle, after reading this chapter, readers could modify some of the FEM programs in MATLAB's toolbox to suit their exact needs (if notalready met).

Figure 2.1: Richard Courant (1888-1972), German mathematician

The FEM methods basically split up the domain of the problem into small pieces, called **elements**, that have simple structure. There are many different ways to perform such decompositions and the geometry certainly changes with the dimension of the space. A common approach for two-dimensional domains is to **triangulate** the domain into small triangles. The **triangulation** must be done in such a way that whenever two triangles touch, they will have either an entire common edge (and thus two common vertices) or just a common vertex. The reason for this is that the FEM approximate solution for the PDE will be made up of separate "pieces" on the various elements and they need to connect up (interpolate) together in a neat fashion. When a domain has a curved boundary, the sizes of the triangles can be made small enough so that the triangulation approximates the domain reasonably well. An example of such a triangulation is shown in Figure 13.2. In three dimensions, tetrahedra (pyramids) are commonly used; but the process is still known as triangulation.

[1]Richard Courant was an exceptionally influential mathematician. He grew up in Germany and had a rather difficult childhood, having to work to help support his family while going to school. He eventually entered the Univerisity of Breslau (now Wroclaw, Poland) as an undergraduate and was lured to major in mathematics by the exciting lectures in his classes. He went on to Göttingen for his graduate studies where he worked with Hubert and later became a professor there. His education was interrupted by military service for Germany in WWI, where he developed an effective electronic communications system that was implemented for the troops. Despite the important contributions he was making to the University of Göttingen, not to mention his important military service to his country, when the Nazis came to power in the early 1930s, he was forced to resign his professorship. The Nazis had decreed that any "non-Aryan" civil servant was to be terminated and having one Jewish grandparent was sufficient to make someone "non-Aryan." There was supposed to be an exemption for individuals who gave Germany military service in WWI, but despite ardent efforts on the part of the university to keep him, Courant was still "retired." He subsequently accepted an offer at New York University. The transition was very difficult for him. Coming from a world-renowned institute and having been surrounded by top-notch mathematicians, when he got to NYU, he found his colleagues to be very weak and the students likewise poor. He made use, however, of his extensive contacts and was able to hire a large group of strong new faculty. Today, NYU's mathematics department, also known as the *Courant Institute*, is considered by many as the premiere applied mathematics institute in the world.

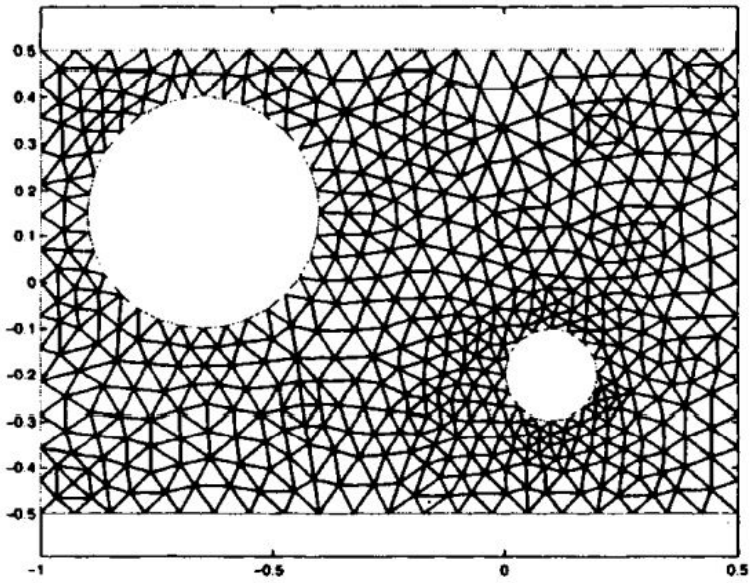

Figure 2.2: A triangulation of a planar domain consisting of a rectangle with two circles deleted. The circular boundary portions are thus approximated by polygons (as shown). This triangulation was created using MATLAB's PDE Toolbox.

Even in two dimensions, of course, there are numerous ways to perform such a triangulation. We point out some important features of the triangulation in Figure 13.2. First, notice that the triangles in the mesh all seem to be roughly the same size. This uniformity is not necessary, but for a general problem on a given domain it is usually the best generic triangulation scheme. Another important property is that none of the sidelengths of any triangle in the mesh is much shorter than the other two sides of the same triangle. Another way to describe this property is that the area of the inscribed circle of any of thesetriangles is not much smaller than the area of the triangle. This feature is essential in order that the finite element method be stable.

Just writing a good program to create such generic triangulations is already an arduous task. It must be thought out how the geometry of the original domain should be inputted (as a matrix) and then the triangles must be created and stored, usually by their vertices. Additionally, the vertices of the triangulation will need to be numbered and it will be helpful for the numbering to be done in such a way that the numbers of the three vertices of any triangle are reasonably close.

Like finite difference methods, finite element methods will discretize the PDE into a linear system. The nature of the discretization, however, is very different for FEMs. Mathematically, the PDE is first converted to a so-called **variational problem**. This is usually done in one of two ways: the **Rayleigh-Ritz method**, where the solution of the PDE is recast as the solution of a certain minimization problem among a large class of functions, or the Galerkin method, where the solution is recast as a certain unique representing function. Although different in philosophy, the two approaches often turn out to be equivalent. With either method, the approximate solution is found by restricting attention to a certain finite-dimensional space of so-called **admissible functions** determined by **basis functions** corresponding to each of the elements. Even with the type of elements being specified, there are numerous choices for the basis functions. The simplest choice would be constant functions, but these do not blend together well. The next simplest choice would be to have linear functions on each element. For two dimensional domains with triangulations, this type of basis function turns out to be quite effective. Over each element, the graph of such a basis function will be the triangular portion of a plane (sitting over the two-dimensional triangle). Since three points determine a plane, these basis functions will be flexible enough to accommodate specifying values at the three vertices of their triangle, and mesh triangles that have common vertices or edges will have their graphs coinciding at common points. The resulting approximating functions will thus be continuous over the original domain, but in general will not be differentiable at common edges or vertices of different triangles. More complicated spline-type basis functions can be used to overcome this differentiability problem, but, of course, the limited benefits of using such more complicated basis functions would have to be weighed against the resulting increase in technical difficulties. For most applications on two-dimensional domains with triangular elements, such linear basis functions are sufficient and most commonly used. MATLAB's PDE Toolbox, for example, uses such basis functions. [2] Focusing only on (linear combinations of) the basis functions, the FEM will solve a linear system to determine the best

[2]The programs in MATLAB's PDE toolbox are designed only to handle PDEs with two space variables, and so, for example, they cannot solve three-dimensional elliptic problems such as steadystate heat distributions and structure of materials. The programs can, however, accommodate a time in hyperbolic or parabolic equations. The reason for this is that FEMs for parabolic and hyperbolic problems invoke finite difference schemes for the time derivatives and thus can accommodate two space variables in addition to the time variable.

candidate to solve the variational problem (Rayleigh-Ritz or Galerkin) and this will be the approximation to the actual solution. This approximation turns out to be simply theprojection of the actual solution onto the finite-dimensional space of admissible functions or, informally, the best admissible function for solving the variational problem. Figure 13.3 shows the FEM solution for the following PDE problem on the domain Ω of Figure 13.2:

$$\begin{cases} \triangle u = 0 \ \ on \ \ \Omega, \quad u = u(x,y) \\ \partial u/\partial n = 0 \ \ \text{on outer rectangle} \\ u = 40, \ \ \text{on small circle}, u = 500, \text{on large circle} \end{cases} \tag{2.1}$$

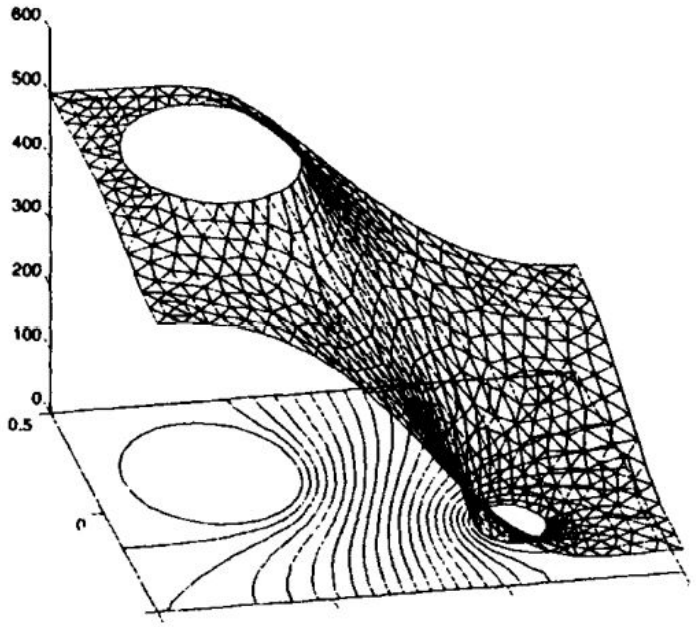

Figure 2.3: A FEM solution of the steady-state heat problem (1) on the domain and triangulation as pictured in Figure 13.2. Contour lines have been added. This solution was created using MATLAB's PDE Toolbox.

The problem can be thought of as the steady-state heat distribution on the rectangular region Ω (from the Laplace equation $\triangle u = 0$). The first boundary condition on the edge of the outer rectangle: $\partial u/\partial n = 0$, is a Neumann boundary condition (n denotes the unit outward normal vector for the domain), stating that heat does not flow out of or into the rectangle (i.e., the boundary is insulated). The two constant temperatures on the interior circular boundaries are Dirichlet boundary conditions specifying certain temperatures that are fixed on each. The problem may be thought of as a basic version of the cooling of a nuclear reactor within some enclosed region (the rectangle); the large very hot circle denoting the reactor and the small circle denoting the cooling source (usuallya stream of fresh water).

In the triangulation process, it is not always efficient to make the triangles be all of essentially the same size. Indeed, at places where the solution varies drastically, smaller triangles should be used and in areas of small variation larger ones can be and should be used. More triangles entail more work so we should use very small ones only where they are needed. Of course, not knowing the solution ahead of time can make it difficult to predict where the solution will be varying wildly. Sharp corners or curves in the domain, as well as areas where the coefficients of the PDE (if variable) rapidly change, are usually problem areas. There are more sophisticated so-called **adaptive methods** of the FEM that iteratively take into account all available information so as to refine the elements accordingly in a way that aims to reach the best possible accuracy with specified constraints (such as operating time, number of triangles, etc.). Figure 13.4 shows such an adaptive triangularon for the boundary value problem (1). Triangulation of domains is an art!

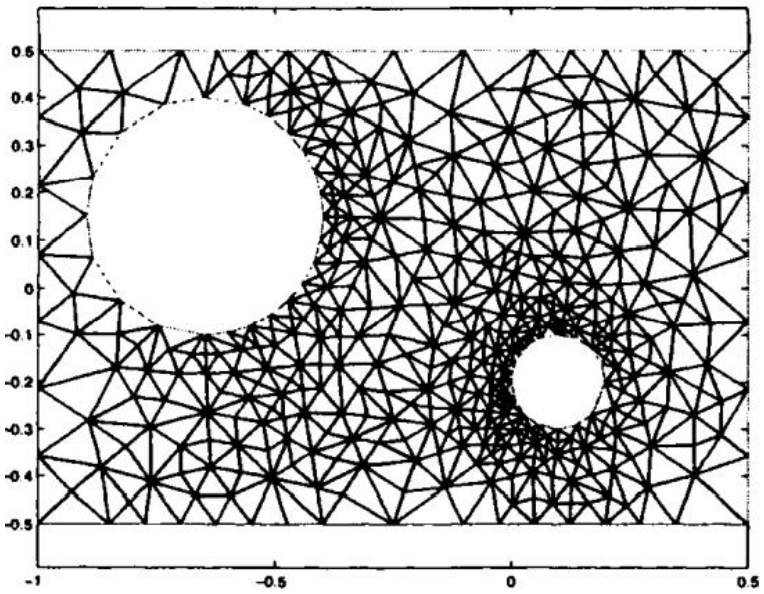

Figure 2.4: A triangularon of the planar domain of Figure 13.2 that was obtained using an adaptive FEM to solve the steady-state heat problem (1). Compare with the triangularon of Figure 13.2 and, in particular, notice how the triangles closer to the boundaries of the circles (facing inward) are much smaller than the farther away triangles. This triangularon was created using MATLAB's PDE Toolbox. .

An outline for the rest of this chapter is as follows: We will be focusing on linear elliptic boundary value problems on planar domains. Our FEM will use piecewise linear basis functions on triangulations of the domains. Section 13.2 will introduce some practical techniques for producing effective triangulations of planar domains and explain how to construct and manipulate basis functions. Section 13.3 will explain the complete program of using a FEM to solve quite general boundary value problems on arbitrary planar domains and boundary conditions. The most time-consuming step of the FEM is the construction of the linear system whose solution will give the values of the approximate solution. This process, known as "the assembly process," is broken up into an element-by-element computation involving the calculation of certain double integrals and (depending on the boundary data) line integrals. We will demonstrate with examples that it is not efficient to use MATLAB's integration tools in the assembly process. Indeed, if the elements are sufficiently small (depending on the coefficients of the problem), it turns out to be perfectly adequate to use some simple numerical quadrature formulas (for triangles and line segments). This will allow us to attain essentially the same accuracy as with the more elaborate integrators but at a very small fraction of the time. In numerical differential equations, much can be learned from experimentation, and this chapter provides numerous opportunities in this area. The committed reader can gain a great deal by exploring some of the more advanced topics that are introduced in the exercises.

2.0.2. TWO-DIMENSIONAL MESH GENERATION AND BASIS FUNCTIONS

Theoretically, the finite element method for two-dimensional problems shares many common threads with the one-dimensional Rayleigh-Ritz methods introduced in Section 10.5. It would behoove the reader to glance over that section periodically as he or she proceeds to work through this and the next section. The major practical difference is in the geometry of the two-dimensional elements and basis functions versus the very simple one-dimensional elements. We will restrict our focus in the text of this chapter to triangular elements and piecewise linear basis ftinctions, although some of the exercises will delve into other sorts of elements and basis functions. In this section we show how triangulations can be created and give some convenient methods for constructing, storing, and manipulating corresponding basis functions. The two main advantages of the FEM over finite difference methods are its ease in dealing with more complicated domains than simply rectangular ones, and its flexibility in dealing with many sorts of boundary conditions. To illustrate construction of the basis functions, we use the simple triangulation of the hexagonal domain shown in Figure 13.5.

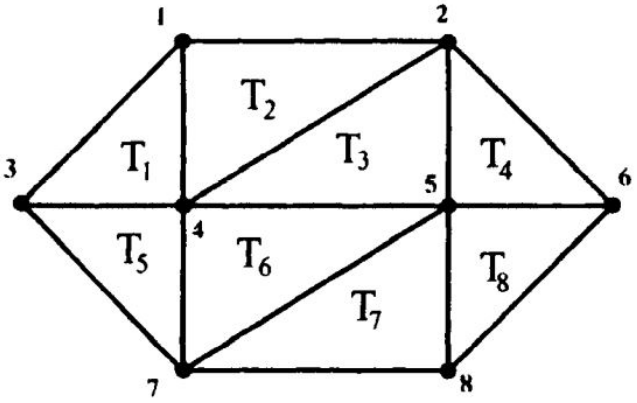

Figure 2.5: A simple triangulation of a hexagonal domain. The eight nodes are labeled in red and the eight triangles are also labeled. The ordering is somewhat arbitrary. It is just a coincidence that the number of nodes coincides with the number of triangles. At this point we left out x- and y-coordinates so as to emphasize the element numbering.

The **nodes** in a triangulation are simply the vertices of the triangles. As in the one-dimensional method, each node gives rise to a basis function that takes on the value 1 at its corresponding node and zero at all other nodes. Piecewise linear functions work well here since a linear function is completely determined on a triangle once its values are specified on the three vertices. Furthermore, two linear functions so determined on triangles that share an edge will agree on the common edge. The resulting basis function will be the unique piecewise linear function on the hexagon having the property that it is linear on each element and takes on the value 1 at its associated node and 0 at all other nodes. In this context the basis functions are sometimes known as **pyramid functions**. The pyramid function for the node #4 of Figure 13.5 is illustrated in Figure 13.6.

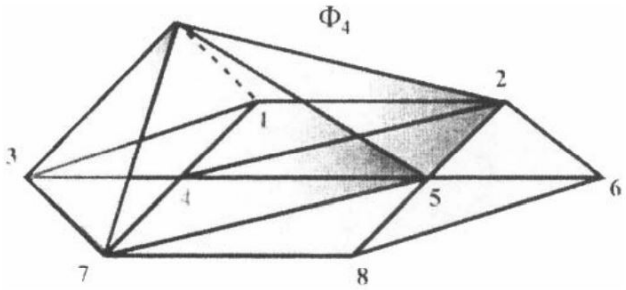

Figure 2.6: A graph of the piecewise linear basis or pyramid function $\Phi_4 = \Phi_4(x,y)$ for node #4 in the triangularon of the hexagonal domain of Figure 13.5. The function takes on the value 1 at node #4, zero at all other nodes, and is linear on each triangle. Thus, on the unshaded triangles, the pyramid function is identically zero.

To get formulas for the pyramid functions, we need to introduce coordinates. For purpose of an example, we assign coordinates as follows: node #3 will be put at the origin $(0,0)$, node #4 will have coordinates $(1,0)$, and the coordinates of nodes #1 and #7 are $(1,1)$, and $(1,-1)$, respectively. The corresponding of nodes #2, #5, and #8 are obtained by adding 3/2 to the first coordinates of the last three, and node #6 has coordinates $(7/2,0)$. Knowing these coordinates, all of the information of the triangulation can be represented by the following two matrices, Abodes) and 71(riangles):

$$N = \begin{bmatrix} 1 & 1 \\ 2.5 & 1 \\ 0 & 0 \\ 1 & 0 \\ 2.5 & 0 \\ 3.5 & 0 \\ 1 & -1 \\ 2.5 & -1 \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 2 & 4 \\ 2 & 4 & 5 \\ 2 & 5 & 6 \\ 3 & 4 & 7 \\ 4 & 5 & 7 \\ 5 & 7 & 8 \\ 5 & 6 & 8 \end{bmatrix}$$

The eight rows of N give the coordinates of the corresponding numbered nodes, the first column entry gives the x-coordinates, and the second column entry the y-coordinates. The eight rows of T give the node numbers of the corresponding numbered triangles, in order (see Figure 13.5). Such matrices will be needed in writing programs for the FEM.

EXAMPLE 13.1:

Write down a formula for the basis function $\Phi_4 = \Phi_4(x,y)$ shown in Figure 13.6.

SOLUTION: From its piecewise linearity, on each of the eight triangles $T_t (1 \leqslant \mathfrak{l} \leqslant 8) \Phi_4(x,y)$, will be a linear function and so can be written as

$$\Phi_4(x,y) = a_t^4 x + b_t^4 y + c_t^4 = a_t x + b_t y + c_t, \;\; (x,y) \in T_t, \tag{2.2}$$

where $a_t^4 x + b_t^4 y + c_t^4 = c_t$ are real constants to be determined. [3] We now fix an index $\mathfrak{l}$ and let the three nodes of T_t be denoted by $(x_r, y_r), (x_s, y_s)>$ and $(x_t > y_t)$ where $t = 4$. The graph of such a linear function $z = \Phi_4(x,y)$ is a plane in three-dimensional space determined by the three nodal values $\Phi_4(x_r, y_r) = 0, \Phi_4(x_s, s_s) = 0$, and $\Phi_4(x_t, y_t) = 1$. These nodal equations may be expressed as the following linear system:

$$\begin{cases} a_t x_r + b_t y_r + c_t = 0 \\ a_t x_s + b_t y_s + c_t = 0 \\ a_t x_t + b_t y_t + c_t = 0 \end{cases} \tag{2.3}$$

Putting (3) in matrix notation gives:

$$MA = Z, \;\; where \;\; M = \begin{bmatrix} x_r & y_r & 1 \\ x_s & y_s & 1 \\ x_t & y_t & 1 \end{bmatrix}, \;\; A = \begin{bmatrix} a_r \\ b_s \\ c_t \end{bmatrix}, \;\; and \;\; Z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.4}$$

Geometrically, since three noncollinear points determine a unique plane, it follows that the linear system (3)/(4) will have a unique solution as long as the three nodes are not collinear. This is certainly the case for any triangulation. We mention one further important point that the system will be well conditioned provided that the area of the triangle T_t is not much greater than that of the inscribed circle. This is a quantitive way of saying that the three nodes of Tt should not be close to being collinear (convince yourself of this!). This can be analytically verified using the following explicit formulas for the determinant of M and M^{-1} :

$$|det(M)| = 2 \cdot area(T_t), \tag{2.5}$$

$$M^{-1} = \frac{1}{det(M)} \begin{bmatrix} (y_s - y_t) & (y_t - y_r) & (y_r - y_s) \\ (x_t - x_s) & (x_r - x_t) & (x_s - x_r) \\ (x_s y_t - x_t y_s) & (x_t y_r - x_r y_t & (x_r y_s - x_s y_r \end{bmatrix} \tag{2.6}$$

For a proof of the interesting equation (5), the reader is referred to Exercise 22. (5), equation (6) can be proved by a direct (albeit tedious) verification. Equations (5) and (6) plainly show that the matrix is well conditioned as long as the triangle is not too long and thin. Apart from this, the explicit formula (6) is useful to build into large-scale FEM programs where such matrices need to be inverted large numbers of times in constructing the basis functions. We continue with this example in a way that will help us later when we need to write general programs. We begin by entering the node matrix N and the triangle matrix T into our MATLAB session:

```
>> N=[l 1/5/2 1;0 0;1 0;5/2 0/7/2 0/1 -1/2.5 -1)/
>> T=[l 3 4;1 2 4/2 4 5/2 5 6/3 4 7/4 5 7/5 7 8/5 6 8]/
```

Since Φ_4 vanishes over triangles #4, #7, and #8, the coefficients a, b, c are all zero for these triangles. The following loop will give us what we need and is easily modified to function in general FEM routines. It will store the neededcoefficients of Φ_4 on the remaining triangles in a four-column matrix A: The first column gives the triangle number; the remaining three give the corresponding coefficients of Φ_4 as in (2). Since the coefficients are all fractions, we display the output in rational format.

```
>> format rat, counter=l/
>> for L=l:8
if ismember(4,T(L,:))==1
 checks to see if 4 is a node of triangle #L
 if yes, next two commands reorder the vector T(L,:) to
 construct a vector Hnv" of length 3
 of nodes of triangle #L with 4 appearing last
  index=find(T(L,:)==4)/
  nv=(T(L/l:2) 4]/ nv(index)=T(L,3)/
  xr=N(nv(l),1)/ yr=N(nv(l),2); xs=N(nv(2),1)/ ys=N(nv(2),2)/
  xt=N(nv(3),1);yt=N(nv(3),2)/
  M=[xr yr 1/xs ys 1/xt yt 1]; %matrix M of (4)
Minv=[ys-yt yt-yr yr-ys/ xt-xs xr-xt xs-xr, xs*yt-xt*ys xt*yr-xr*yt
xr*ys-xs*yr]/det(M)/
  inverse matrix M from (6)
  abccoeff=Minv*[0/0/1]/ %coefficents of basis function on triangletfL
```

[3] The superscipts, although technically necessary, can be omitted in this example since there is only one basis function under consideration.

```
  A(counter,:)=[L abccoeff')/
  counter=counter+1 end end
>> A
->A=        1     1      -1     0
            2     0      -1     1
            3     -2/3   0      5/3
            5     1      1      0
            6     -2/3   1      5/3
```

From this matrix, we can write down the explicit formula for Φ_4:

$$\Phi_4(x,y) = \begin{cases} x-y & \text{if } (x,y) \in T_1 \\ -y+1 & \text{if } (x,y) \in T_2 \\ -\frac{2}{3}x+\frac{5}{3} & \text{if } (x,y) \in T_3 \\ x+y & \text{if } (x,y) \in T_5 \\ -\frac{2}{3}x+y\frac{5}{3} & \text{if } (x,y) \in T_6 \\ 0 & \text{otherwise} \end{cases}$$

The reader should verify that these formulas indeed possess the required values at the nodes and hence (by linearity) on each element.

EXERCISE FOR THE READER 13.1: Find formulas for Φ_3 and Φ_5 analogous to that found for Φ_4 in the above example.

The careftil reader may have realized that we can farther cut our computation time down in the solution of the system (4) if we always agree to set it up so that (x_t, y_t) is the vertex on which the value of the local basis function equals 1. Since the inverse of the coefficient matrix M of (4) is explicitly known (6), the matrix product MZ will simply be the third column of M^{-1} so that in the notation of (4), we have (using (5)):

$$\begin{bmatrix} a_t \\ b_t \\ c_t \end{bmatrix} = \frac{1}{2 \cdot area(T_t)} \begin{bmatrix} y_r - y_s \\ x_s - x_r \\ x_r y_s - -x_s y_r \end{bmatrix}$$

Up to now, most of our plots for functions of two variables have been over rectangular domains. Thus it will be important for us to learn how to get MATLAB to plot functions, such as the above basis functions, that are piecewise linear and continuous on a set of triangular finite elements. Such functions are determined entirely by their nodal values. MATLAB can accommodate us quite nicely for this task with the following command:

`trimesh(T,x,y, z,C) ->`	Given a 3-column matrix T whose rows are node numbers for a triangularon, vectors x and y of the coordinates of the numbered nodes, and corresponding z coordinates of a piecewise linear function on the nodes, this command will produce a plot of the resulting piecewise linear function. The last argument C is an optional rgb vector that can be used to specify color (see Section 7.2). The default edge coloring is proportional to the edge height as in Chapter 11. The vector z can be omitted to produce a two-dimensional plot of the triangularon.

We are nicely set up to have MATLAB construct a plot of Φ_4.

```
>> X=N(:,1) ; y=N(:,2) ;
>> z=zeros(8,1); z(4)=1;
>> trimesh(T,x,y,z)
>> hidden off %allows hidden edges to appear
```

The resulting MATLAB plot is shown in Figure 13.7. Since there are only two heights for the edges, the coloring is not very elaborate. With finer triangulations and more complicated functions, the resulting trimesh plots can be quite useful and attractive, as the one shown in Figure 13.2.

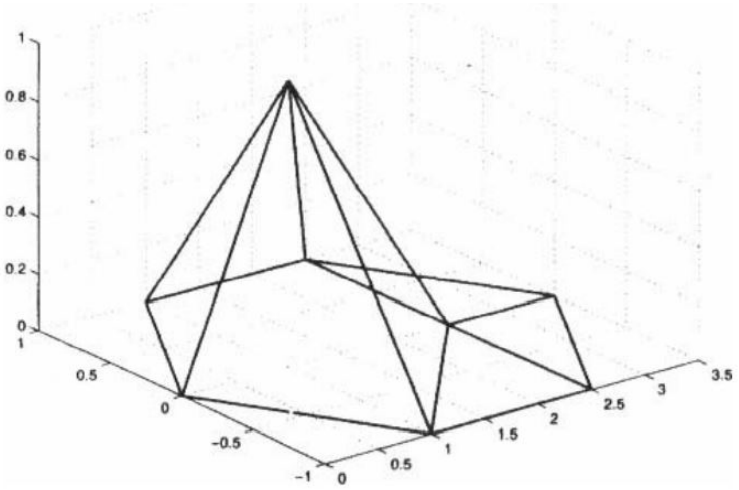

Figure 2.7: MATLAB's graphical rendition of the basis function $\Phi_4(x,y)$ of Example 13.1.

Given a triangulation of a domain and any function or data / defined on the $N_1, N_2, \cdots, N_m$, the **finite element interpolan**t of this function/data is given by:

$$\sum_{j=1}^{n} f(N_j)\Phi_{N_j}(x,y)$$

We point out that the graph of this interpolant is most easily obtained by simply using the trimesh command directly on the triangle matrix and corresponding values of f ; the calculations for the basis functions are not necessary. This will not be the case for more general elements (see Exercise 26).

We stress that in the determination of the hat functions Φ_j , we really split up the problem into determining Φ_j, on each element (triangle). On each element $T_t, \Phi_j(x,y) = a_t^j + b_t^j + c_t^j$ is a linear combination of the three functions x, y, and 1. These three functions are a basis for the set of all linear functions on T_t.We refer to them as a **local basis**, to distinguish them from the (global) basis functions Φ_j . Although these local basis functions are quite natural and have simple formulas, there is another local basis that often has theoretical advantages. To simplify notation, we fix an element $T = T_n$ and denote its three vertices by v_1, v_2, and v_3 (the exact ordering is unimportant, but let's assume they are numbered in counterclockwise order). The corresponding three standard **local basis functions** Φ_1, Φ_2, and Φ_3 are the linear functions determined (exactly) by the following conditions:

$$\Phi_i(V_j) = \delta_{ij} \equiv \begin{cases} 1 & if\ i = j. \\ 0 & if\ i \neq j. \end{cases} \tag{2.7}$$

(The symbol δ_{ij} is called the **Kronecker delta symbol**.) It was described earlier how each of these functions can be expressed using the original local basis functions. In terms of these local basis functions, any linear function $\Phi(x,y)$ can be conveniently expressed as:

$$\Phi(x,y) = \Phi(V_1)\Phi(x,y) + \Phi(V_2) + \Phi_2(x,y) + \Phi(V_3)\Phi(x,y) \tag{2.8}$$

(*Proof*: Both sides are linear functions that agree at the three noncollinear points V_1, v_2 and v_3 and so must be identical.) Each basis function Φ is simply made up of pieces of corresponding elements containing the node associated with Φ, and each of these pieces is a linear combination (8) of the above local basis functions for its element. The previous example thus gave efficient strategies for computing all of the local basis functions as well as the corresponding basis functions.

EXERCISE FOR THE READER 13.2: (a) Explain why piecewise linear basis could not be used if rectangular elements (with sides parallel to the axes) were used in place of triangular ones.
(b) Give an example of a simple type of basis function that could be used for such rectangular elements. Make sure that your construction will ensure that anygiven basis function will be continuous across element edges.

As mentioned Section 13.1, triangulation is an art, and as such there has been a notable amount of research in the development of efficient and effective triangulation and mesh generation schemes. One particularly successful and often used method in this area is that of the **Delaunay triangulation**, relative to a given finite set of points in the plane. This triangulation will result in a set of triangles whose vertices coincide with the given finite set (of nodes) and with the further property that the circumcircle of each triangle in the collection contains only nodes that are vertices ofthat triangle. This condition favors well-rounded triangles over thin ones, which are better for the FEM.

Definitions: Suppose that we have a set of distinct points $P = \{p_1, p_2, \cdots, p_n\}$ in the plane $\mathbb{R}^2$. The Delaunay triangulation relative to P consists of all triangles connecting three noncollinear points $p_i, p_j, p_k \in P$ with the property that there exists a point $a \in \mathbb{R}^2$ which lies equidistant to each of the points p_i, p_j, p_k and closer to these three than to any other point $p_t \in P, \mathfrak{l} \neq i, jk$.

Figure 13.8 illustrates the Delaunay triangulation for a very small set of four points. It can be shown that the Delaunay triangulation of a finite set of points will always be a triangulation for the **convex hull** [4]of this set. The Delaunay triangulation has the important property that the minimum angle of any of its triangles is as large as possible for any triangulation of the same set of points (see Section 1.2 of [Ede-01]). This makes the Delaunay triangulation very suitable for the FEM. There is a dual notion of the Delaunay triangulation which leads to an equivalent formulation. We give the relevant definitions:

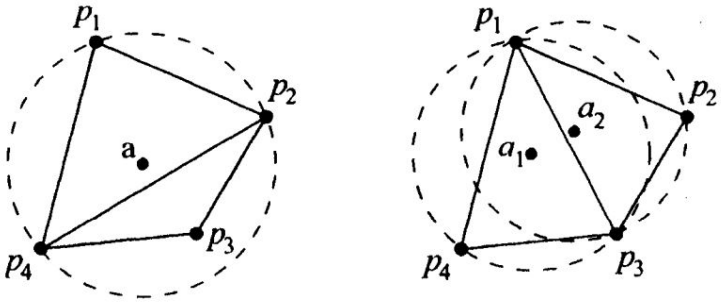

Figure 2.8: Two triangulations are shown for the same 4-point set $\{p_1, p_2, p_3, p_4\}$. (a) (left) The first one violates the Delaunay condition since p_3 lies in the circumcircle of the larger triangle, (b) (right) The second gives the Delaunay triangulation. Circles and centers are drawn in to show the validity of the condition.

Definitions: Suppose that we have a set of distinct points $P = \{p_1, p_2, \cdots, p_n\}$ in the plane $\mathbb{R}^2$ (write $p_i = (x_i, y_i)$). Relative to this set, for each $p_i (\leqslant i \leqslant n)$ we define the **Voronoi region** $V(p_i)$ as:

$$V(p_i) = \left\{ p \in \mathbb{R}^2 : |p - p_i| \leqslant |p - p_t| \;\; for \;\; each \;\; p_t \in P, \mathfrak{l} \neq i \right\} \tag{2.9}$$

Here absolute values denote the Euclidean distance (this coincides with the 2-norm introduced in Chapter 7). The **Voronoi diagram** for P is simply an illustration of the totality of all of the Voronoi regions.

It is not difficult to show that each Voronoi region is a convex set which, if bounded, is a polygon (Exercise 27). In words, the Voronoi region $V(p_i)$ is simply the set of all points in the plane whose closest element of the set P is p_i. If a school district wished to minimize bussing times and costs, and if the points /?. represented locations of schools, the Voronoi region of a given school would roughly include all households whose children would be sent to that school.

The duality result states that two points $p_i, p_j \in P$ are joined by an edge in the Delaunay triangulation if and only if their Voronoi regions $V(p_i)9V(p_j)$ share a common edge. The Ukrainian mathematician Georges Voronoi was the first to introduce his concept in 1908 [Vor-08]. Subsequently, Russian mathematician Boris Delaunay introduced his triangulation in a 1934 paper [Del-34] that he dedicated to Voronoi. These concepts have numerous applications; details of the rich and interesting history can be found in the book [OkBoSu-92], which contains over 600 references. Construction of the Delaunay triangulation for a given set of points in the plane has been the focus of much research. The first algorithms that were discovered worked in $0(n^4)$ time, but modern refined algorithms perform in$O(nlogn)$ time. Some survey articles of this area are: [SuDr-95] and [BeEp-92], see also Chapter 13 of [Ros-00]. We will make use of MATLAB's built-in functions that will perform both of the Delaunay triangulation and the Voronoi diagrams, so the task of triangulations will thus be reduced to the more simple problem of node deployment.
We proceed now to introduce the relevant MATLAB functions:

[4]The convex hull of a set of points is the smallest convex set which contains each of the points. There is a degenerate case in which some four of the points lie on a common circle (with no other points inside this circle). Here no three of the points will lie any closer to the center of the circle than the fourth. In algorithms for the Delaunay triangulation what is usually done in degenerate cases is that one of the points is slightly perturbed (moved). Since the area of a circle is zero, the probability that a fourth point will lie on the circle determined by three points is zero. In degenerate cases "the" Delaunay triangulation is not unique. The whole subject of triangulation and more general mesh generation has become quite an important discipline in itself. Good references are Chapter 13 of [Ros-00] and [Ede-01]

`tr i = delaunay(x,y) ->`	If *x* and *y* are vectors of the same length n giving the coordinates of n (noncollinear) points in the plane, this command will output an $n \times 3$ matrix `tri` whose rows contain the indices (rel. to the x and y vectors) of the triangles in the Delaunay triangulation.
`voronoi (x,y) ->`	If *x* and *y* are as above, this command will result in a graphic of the Voronoi diagram for the set of points corresponding to x and y

5

Once created, the Delaunay triangulation can be used, just like any other triangulation for the FEM. To view the Delaunay triangulation, we could use the above `trimes h` command. We illustrate by having MATLAB compute both objects for the set of node values that we used for the diagram in Figure 13.5. The following commands result in the plot shown in Figure 13.9(a).

```
>> N=[ l l;5/ 2 1; 0 0 ; 1 0;5/ 2 0;7/ 2 0; 1 -1;2. 5 -1] ;
>> x=N(:,1) ; y=N(:,2) ;
>> tri=delaunay(x,y), trimesh(tri,x,y)
>> axis(l-l 4.5 -1.5 1.5]), hold on
>> plot(x,y,'ro')
```

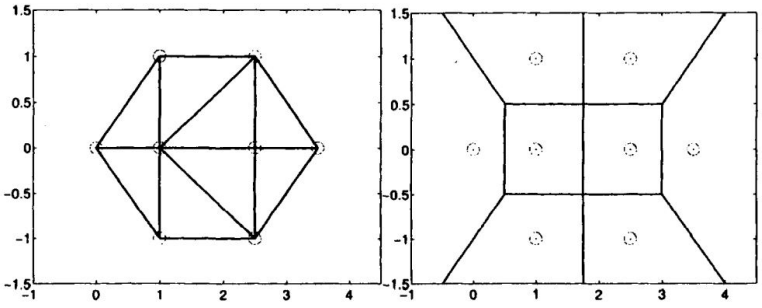

Figure 2.9: (a) (left) MATLAB plot of the Delaunay triangulation of the set of data points indicated by circles, (b) (right) MATLAB plot of the dual Voronoi diagram for the same set of data points

The Voronoi diagram in Figure 13.9(b) was obtained using the minor modification of the MATLAB's `voronoi` program that appears in the following exercise for the reader.

EXERCISE FOR THE READER 13.3: (a) Write an M-file called `voronoial l (x,y )` that will function just like MATLAB's `voronoi`, except that it will show the unbounded Voronoi regions (not all of them, of course) with a reasonable axis view, (b) Use your program to recreate the plot of Figure 13.9(b).

Some comments are in order. First notice that the Delaunay triangulation that MATLAB gave us coincides with the one we used previously. Also notice that this example demonstrates that the Delaunay triangulation is not unique (so it really should not have been called "the" Delaunay triangulation). Indeed, the two diagonal edges in the center could have been reversed (i.e., reflect the triangulation horizontally) to result in another triangulation that also meets the Delaunay criteria, or the duality theorem's criterion. (The reader should convince himself or herself of these assertions.) The Voronoi regions are of course uniquely defined and so the Voronoi diagram is unique.

Having the delaunay function to work with makes it a lot easier to do a triangulation; we need only specify the node points. This should be done in a way that will give rise to a Delaunay triangulation whose triangles do not get too thin. A good general rule is to deploy node points in more or less squarelike configurations. The sizes of adjacent squares should not change too abruptly. Of course, when approaching the boundary, special care must be exercised. For boundary value problems, nodes need to be put on the boundary as well. It is also possible to increase the density of nodes in certain parts of the domain in regions where coefficients of the PDE are more active (oscillatory). The next example will create three different triangulations for the same domain, a disk.

[5] Actually, the `voronoi` command will show only the bounded Voronoi regions (i.e., those that have finite areas). There is an easy way to get MATLAB to show all of the regions; see Exercise for the Reader 13.2.

EXAMPLE 13.2:

Let Ω denote the unit disk $\{p=(x,y)\in\mathbb{R}^2 : \|p\|_2 \leqslant 1\}$. [6] Use MATLAB to create and plot three different triangulations of Ω each having between 1000 and 2000 nodes for each of the three requirements:
(a) The nodes are more or less uniformly distributed.
(b) The density of the nodes increases as $\|p\|_2$ increases, i.e., as we approach the boundary.
(c) The distribution of the nodes increases near the boundary point (1,0). NOTE: We left the exact number of nodes somewhat flexible since we want to stress node deployment schemes and do not wish to be distracted with trying to use a precise number of nodes.

SOLUTION: Part (a): We will give two different strategies for this part.

Method I: We use a squarelike configuration for the nodes. For the most part, this will be quite a simple scheme, but near the boundary circle $\|p\|_2 = 1$, things get a bit awkward. The square $S = \{p=(x,y)\in\mathbb{R}^2 : -1 \leqslant x,y \leqslant 1\}$includes the disk Ω as its inscribed circle. Since the ratio of the areas of Ω to S is $\pi\cdot 1^2/(2\cdot 2) = \pi/4 = 0.785\cdots$, it follows that if we uniformly distribute a large number of nodes in the interior of$S$, roughly 78.5% of them will be in the interior of Ω . Since it is a simple matter to uniformly distribute any (square) number of nodes in S, we will begin by uniformly distributing about 2000 nodes in S , and let δ denote the square side length that is used. Of these nodes we will keep all of them that lie inside of Ω but at a distance of at least $\delta/2$ from the boundary circle $\|p\|_2 = 1$. Then we add on a set of nodes on the boundary circle, which are uniformly spaced with gaps about equal to δ. The total amount of nodes thus constructed for Ω will be (well) over 1000 and certainly less than 2000.

To begin, since $\sqrt{2000} = 44.721...$, we will first construct a square grid of $N_0 = 45^2 = 2025$ interior nodes in S . The horizontal and vertical gaps should be $\delta = 2/(45+1)$. The following MATLAB commands will create these nodes, and store them in two vectors `xO, yO` .

```
>> delta =2/46; counter=l;
for i=l:45
for j=l:45
  xO(counter)=-l+i*delta; yO(counter)=-l+j*delta;
  counter=counter+l ;
end
end
```

Next, from these two vectors we extract those components corresponding to points which lie within the slightly smaller circle $\|p\|_2 = 1-\delta/2$ the newly created vectors will be labeled as x and y.

```
>> counter=l;
>> for i=l:2025
  if norm([xO(i) y0(i)],2) < l-delta/2
    x(counter)=x0(i); y(counter)=y0(i);
    counter=counter+l;
  end
end
```

Finally, since $2\pi/\delta = 144.5133...$, we tack onto the existing vectors x and y an additional 145 entries corresponding to 145 equally spaced points on the unit circle $\|p\|_2 = 1$. Figure 13.10(a) shows a plot of this node set.

```
>> for i=l:145
  x(i+1597)=cos(2*pi/145*i); y (i + 1597)=sin(2*pi/145*i) ;
end
>> plot(x,y,'bo'), axis('equal')
```

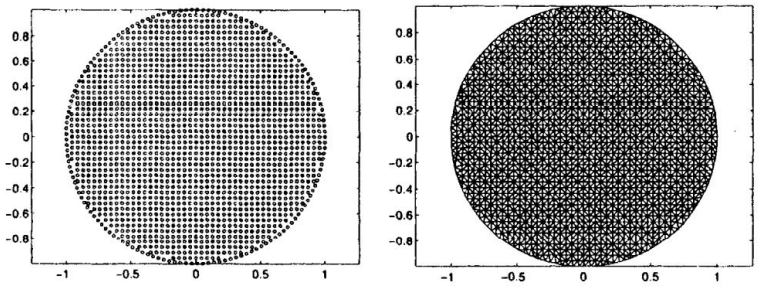

Figure 2.10: ((a) (left) Grid of nodes from Method 1 of Example 13.2(a), essentially a square pattern except near the boundary. There are 1742 nodes, (b) (right) A corresponding Delaunay triangulation that has 3337 triangles.

[6] We are using here the norm notation from Chapter 7: $\|p\| = \|(x,y)\|_2 \equiv \sqrt{x^2+y^2}$ is in 2-norm which is simply the (planar) Euclidean distance from $p=(x,y)$ to the origin (0,0) .

The corresponding Delaunay triangulation will result from the following MATLAB commands and is shown in Figure 13.10b.

```
>> tri=delaunay(x,y); trimesh(tri,x,y), axis('equal')
```

Method 2: Here we will deploy nodes on circles of increasing radii. The gaps between nodes on a given circle and the gaps between radii of adjacent circles of deployment should be all about equal (uniformity). The final circle will be the boundary of $\Omega : \|p\|_2 = 1$. The only mathematical preliminaries are to decide how many circles to deploy. Letting δ denote the common gap size, since the radii of the circles increase steadily from 0 to 1, the average radius will be about 1/2, which means the average circumference will be about $2\pi(1/2) = \pi$. Thus the average number of nodes on a circle will be π/δ. Likewise, the number of circles of deployment is about MS, so that the total number of nodes will be, roughly, $(\pi/\delta)\cdot(1/\delta) = \pi/\delta^2$. Setting this equal to 1800, say (we want it close to 2000, but to insure the actual number of nodes remains under 2000 we play it a bit safe), and solving for delta gives $\delta = \sqrt{\pi/1800} = 0.04177...$. We may now turn the node deployment over to MATLAB with this scheme:

```
>> delta=sqrt(pi/1800); x(1)=0; y(1)=0;
>> nodecount=1; ncirc=floor(1/delta); minrad=1/ncirc;
>> for i=1:ncirc
  rad=i*minrad;
  nnodes=floor(2*pi*rad/delta);
  anglegap=2*pi/nnodes;
  for k=1:nnodes
    x(nodecount+1)=rad*cos(k*anglegap);
    y(nodecount+1)=rad*sin(k*anglegap);
    nodecount = nodecount+1;
  end
end
```

The plotting of the nodes and then the Delaunay triangulation is done just as in Method 1 above; the results are shown in Figure 13.11.

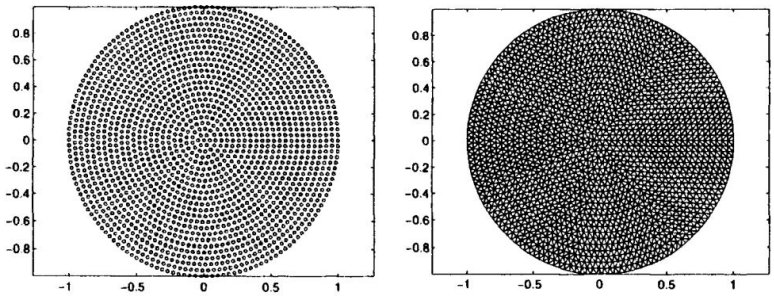

Figure 2.11: (a) (left) Grid of nodes from Method 2 of Example 13.2(a). There are 1887 nodes, (b) (right) A corresponding Delaunay triangulation that has 3438 triangles. Both the node distribution as well as the Delaunay triangulation take on an aesthetically more appealing pattern than those of Method 1, since this method better respected the symmetry of the domain.

Part (b): The requirement is rather vague. We will use a deployment scheme similar to that of Method 2 in part (a). The new difficulty is that there will need to

be more circles of nodes of larger radii so it will take more work to estimate the total number of nodes. Such an estimate will depend first on how we plan to distribute the radii for the circles of nodes. Here is (but) one scheme. We start off with a single node at the origin (0,0). Then we move to the circle $\|p\|_2 = 1/2 = rad(1) = 1 - 1/2$ and deploy 8 (equally spaced) nodes on this circle. Our next circle is $\|p\|_2 = 3/4 = rad(2) = 1 - 1/4$ on which we deploy 2-8 = 16 nodes. After this we deploy $2\cdot = 32$ nodes on the circle. We continue this pattern, so that at the wth circle of deployment will be $\|p\|_2 = 1/2 = radn1) = 1 - 1/2^n$ on which we will deploy 2^{n+2} nodes. This will continue until the number of remaining nodes is still greater than the number of most recently installed nodes (on the last circle of deployment). The final step will be to put all of the remaining nodes on the unit circle $\|p\|_2 = 2$. This plan will create exactly 2000 nodes. Here now is the MATLAB code needed to create such a set of nodes.

```
>> xb(1)=0; yb(1)=0; rnodes=1999; %remaining nodes
>> newnodes=8; %nodes to be added on next circle
>> radcount=1; Icounter for circles
>> oldnodes=1; %number of nodes already deployed
>> while newnodes < rnodes/2
  rad - 1 - 2 A(-radcount);
  for i=1:newnodes
    xb(oldnodes+i)=rad*cos(2*pi*i/newnodes);
    yb(oldnodes+i)=rad*sin(2*pi*i/newnodes);
  end
  oldnodes=oldnodes + newnodes; %update oldnodes
  modes = modes - newnodes; %update modes
```

```
  radcount=radcount+1; %update radcount
  newnodes = 2*newnodes; %update newnodes
  end
% now deploy remaining nodes on boundary
>> for i = 1: modes
    xb(oldnodes*i)=cos(2*pi*i/rnodes);
    yb(oldnodes+i)=sin(2*pi*i/rnodes);
end
```

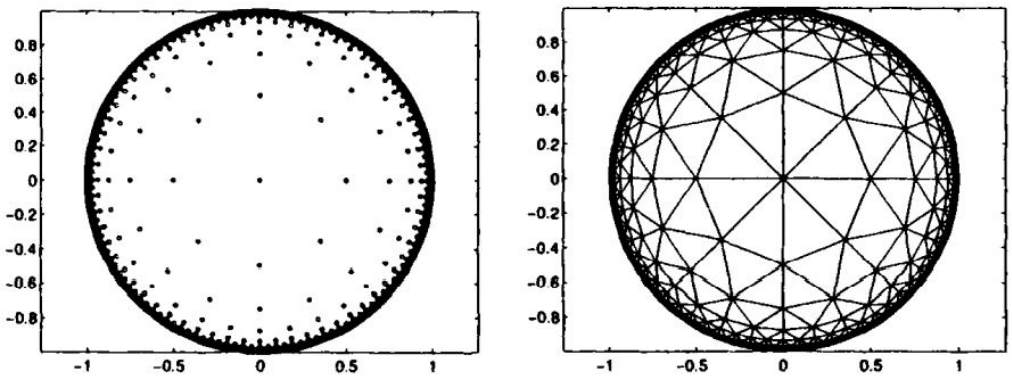

Figure 2.12: (a) (left) Grid of nodes for Example 13.2(b). There are 2000 nodes, (b) (right) A corresponding Delaunay triangulation that has 3015 triangles. Such a triangulation is useful for BVPs which are particularly sensitive to boundary data.

The plotting of the nodes and creation of the Delaunay triangulation is obtained in the same fashion as in part (a). The results are shown in Figure 13.12.
Part (c): The way we will deploy nodes is to first partition Ω into subsets determined by the regions between pairs of circles centered at (1,0). For each positive integer n, we define the following subset $\Omega_n \subseteq \Omega$:

$$\Omega_n = \left\{(x,y) \in \Omega : 1/2^2 < dist((x,y),(1,0)) < 2 \cdot (1/2^n)\right\}$$

In words, Ω_n is just the portion of points inside Ω that lie in between the two concentric circles with center (1,0), and radii $1/2^n$ and $2 \cdot (1/2^n)$. A typical such region is illustrated in Figure 13.13. The idea will be to deploy roughly a fixed number of nodes (we will use about 100) in each of these regions, up to a certain value of the index. We will also need to put some nodes in the remaining part of Ω (which actually can be written as Ω_0); here we again put roughly the same number, about 100 nodes. To decide how to put the nodes on the boundary circle of Ω as well as in the interior of the domains Ω_n, we use the following estimates. From Figure 13.13, it is clear that Ω_n, is always contained in the half annulus (enclosed between the two dotted circles in the figure), and consequently,

$$Area(\Omega) \leqslant \frac{1}{2}[\pi \cdot (2 \cdot 2^{-n})^2 - \pi \cdot (2^{-n})^2] = \frac{3\pi}{2} 2^{-2n}.$$

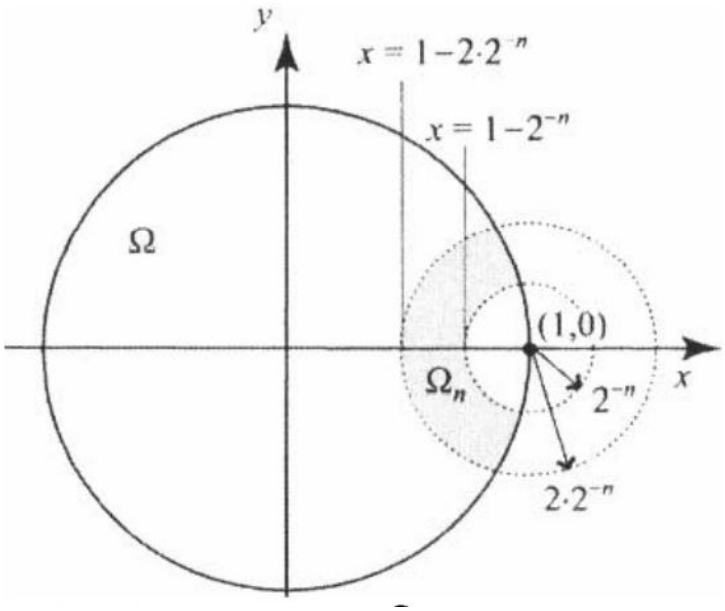

Figure 2.13: Illustration of a typical region Ω_n (shaded) for the triangulation scheme of part (c) of Example 13.2. Such regions are useful for general triangulation schemes when it is desired to have large finer meshes near a special point of the domain.

Also, the estimate becomes increasingly accurate as n gets larger. Now, if we deploy a square grid of nodes with (horizontal = vertical) grid spacing = s in the interior Ω_n, each node would give rise to a square of area s^2 inside of Ω_n (to be specific, let's say that the node gets associated with the square of side length s having the node as its lower left vertex). Thus, if we were to put a grid of 100 such nodes in the interior of Ω_n, the area bound above would yield the following estimate for s:

$$100\cdots^2 \leqslant Area(\Omega_n) \leqslant \frac{3\pi}{2}2^{-2n} \Rightarrow s \leqslant \sqrt{3\pi}2^{-n}/(10\sqrt{2})$$

We use this as a scheme for the horizontal/vertical grid gap to put between nodes that lie inside each Ω_n. The actual number of nodes on each deployment will be less than 100 because the above estimates are inequalities. Since we will essentially be placing 100 nodes at each iteration on Qw(and the corresponding portion of the boundary circle adjacent to Ω_n) starting with $n = 0$, it follows that we should let n run up to about 15 in this scheme. For deploying nodes on the boundary circle that lie adjacent to Ω_n, we also use s as the gap (this time the circular arclength gap) between boundary nodes. Since the boundary circle has radius one, angles are equal to the corresponding boundary arclengths. We will create the nodes using nested loops. On each iteration for n (master loop), the loops will first create and store the corresponding value of s, determined by using an equality in the above inequality for s.

Next, a double loop will be set up that will run through a horizontal and vertical grid that will cover the domain Ω_n and have (horizontal = vertical) grid gap = s. For this part, note (again from Figure 13.13) that the domain Ω_n is always contained within the rectangle: $R_n = \{(x,y) : 1-2\cdot 2^{-n} \leqslant x \leqslant 1, -2\cdot 2^{-n} \leqslant y \leqslant 2\cdot 2^{-n}\}$ Grid points lying in the interior of Ω_n are added as nodes. Once this double nested loop has been executed and interior nodes have been added, the same master loop will then move on to install nodes on the two portions of the boundary of Ω_n that lie on the unit circle (= boundary of the main domain Ω). We will need to compute the angles (made from (0,0) to the positive jc-axis) of the two endpoints of the top boundary arc of Ω_n on the unit circle. (The node deployment on the bottom symmetric boundary arc can be gotten by simply negating the y-coordinates of the nodes in the upper arc.) It is easily shown using the law of cosines that these two angles θ_1 and θ_2 (which technically should be denoted by $\theta_{1,n}$ and $\theta_{2,n}$ to indicate their dependence on Ω_n satisfy: $cos(\theta_n) = 1-2/2^{2n}$ and $cos(\theta_2) = l-2^{-2n}/2$ The following MATLAB code is an implementation of the scheme described above.

```
>> n=0; nodecount-1;
>> while n<16
  s=sqrt(3*pi/2)/10/2An; hgrid=2/s/2An; vgrid=4/s/2An;
%these will be sufficient horizontal and vertical grid counts to
%create a rectangular grid (with gap size =s) that will cover the
%domain Omega_n
  for i=l:hgrid
  for j=l:vgrid
   xnew=1-2/2 A
   n+ i *s; ynew*-2/2 An+j *s;
  pij * [xnew ynew]; p=[l 0];
 if norm(pij,2)<l-s/2 & norm(pij-p,2)<2/2An & norm(pij-p,2)>l/2An+s/2
```

```
%The three conditions here check to see if the node should be added.
%The first says that the node should be in the unit circle (with a
%safe distance to the boundary to prevent interior nodes from getting
%too close to boundary nodes which will be added later) . The second
%and third state that the distance from the node to the special
%boundary point (1,0) should be between the two required radii. The
%last condition has a safety term added to the lower bound to prevent
%nodes from successive iterations from getting too close.
  x(nodecount)=xnew; y(nodecount)=ynew; nodecount=nodecount+l;
  end
 end
end
%The next part of the loop puts nodes on the boundary.
  thetal=acos(l-2/2A
  (2*n)); theta2=acos(1-2A
  (-2*n)/2);
  if n==0, thetal*thetal-s; end
  for theta = thetal:-s:(theta2+s/2)
      x(nodecount)=cos(theta); y(nodecount)=sin(theta);
    x(nodecount+l)=cos(theta); y(nodecount+1)=-sin(theta);
    nodecount=nodecount+2;
  end
  n=n+l;
end
%We need to put a node at the special unsymmetric point (-1,0).
x(nodecount)=-1; y(nodecount)=0; nodecount=nodecount+l;
%Finally we put nodes in the portion of the domain between (1,0)
%and the last Omega_n, and then on the boundary.
%We need first to bump n back down one unit.
n=n-l;
 for i=l:hgrid
 for j=l:vgrid
  xnew=l-2/2A n+i*s; ynew=-2/2An+j*s;
  pij = [xnew ynew]; p=[l 0] ;
  if norm(pij,2)<l-s/2 & norm(pij-p,2)< l/2A^n
   x(nodecount)=xnew; y(nodecount)=ynew; nodecount=nodecount+l;
  end
end
end
for theta = -theta2:s:theta2
  x(nodecount)=cos(theta); y(nodecount)=sin(theta);
  nodecount=nodecount+l;
end
```

Plotting of the nodes as well as the corresponding triangulation is accomplished exactly as it was done in the above two parts. The results are shown in Figures 13.14 and 13.15.

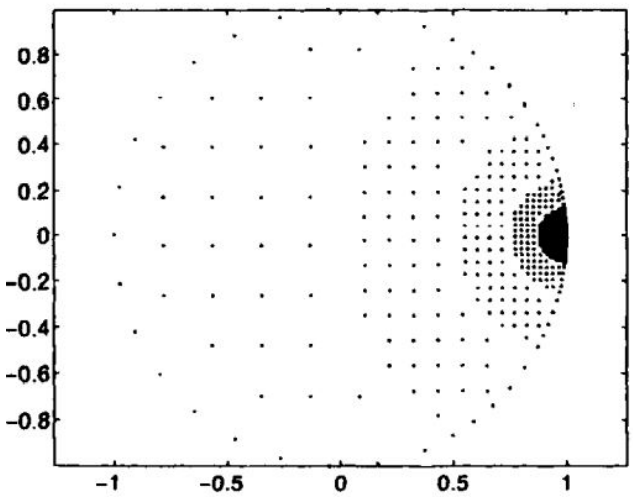

Figure 2.14: Node distribution from the solution of part (c) of Example 13.2. The 1457 nodes are constructed in clusters with each cluster getting its grid gap size cut in half as we move in towards the special boundary point (1,0). The exercises will examine some related schemes for this domain where there is a smoother transition in gap sizes of nodes as we progress toward (0,1).

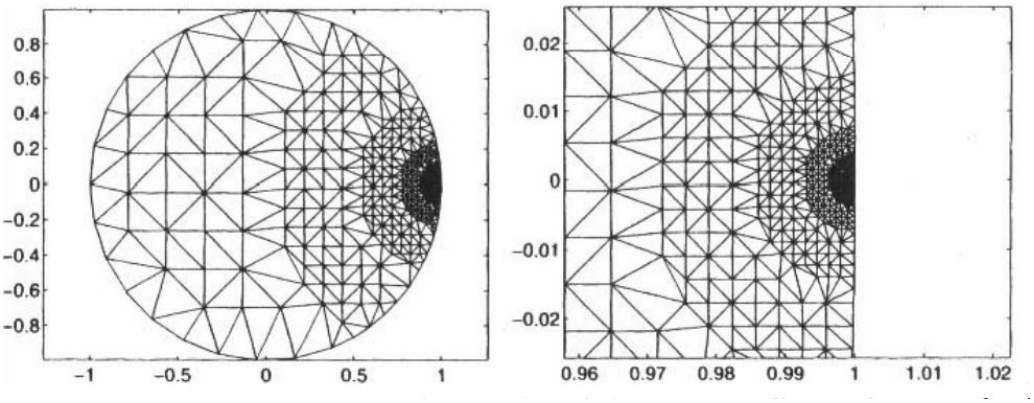

Figure 2.15: (a) (left) The Delaunay triangulation corresponding to the network of nodes of Figure 13.14 that has 2733 triangles, (b) (right) A $20\times$ magnification of the triangulation of (a) near the point of focus (1,0).

On the node sets that were constructed in the last example, the Delaunay triangulation worked quite nicely because the domain Ω was convex. In general, the Delaunay triangulation of a set of nodes will triangulate the convex hull of this set. Delaunay triangulation can still be used to triangulate a nonconvex domain Ω. This is usually done either by breaking up the domain into convex pieces, triangulating each piece, and merging these triangulations or simply by triangulating the convex hull and deleting triangles that are not part of the domain.7 With either strategy, some sort of (global) reindexing will be necessary when constructing the final triangulation. Our next example will illustrate the latter strategy and the exercise for the reader which follows will require also the former strategy.