8

The Linear Wave Equation

This partial differential equation (PDE) application introduces the following mathematical concepts and computational methods:

1. A PDE with an exact solution that can be used to assess the accuracy of a numerical method of lines (MOL) solution.
2. Some of the basic properties of *traveling waves*, a fundamental concept in the theory of PDEs and an important idea in many areas of physics, engineering, and the biological sciences.
3. The *characteristics of a hyperbolic PDE*.
4. Illustrative programming of a traveling wave solution.
5. The effect of smoothness (continuity in derivatives) in determining the effectiveness of numerical methods for computing PDE solutions.
6. Computer analysis of a PDE over an essentially infinite spatial domain.
7. Programming of a *PDE* second order in *t*.

The *one-dimensional (1D), second-order, linear wave equation* is [1]

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

or in subscript notation,

$$u_{tt} = c^2 u_{xx} \tag{8.1}$$

where

u	dependent variable
x	boundary-value (spatial) independent variable
t	initial-value independent variable
c	velocity

c is the *velocity of wave propagation* with the representative units `m/s` (meters/second). Note how these units are consistent with those of the derivatives in Eq. (8.1). u is typically a displacement, for example, the height of a water wave or amplitude of an electromagnetic wave.

Equation (8.1) is second order in t and x. It therefore requires two *initial conditions* (ICs) and two *boundary conditions* (BCs). For the ICs, we use

$$u(x, t = 0) = f(x), \tag{8.2}$$

$$u_t(x, t = 0) = 0 \tag{8.3}$$

Thus, the PDE starts with an initial displacement $f(x)$ and zero initial velocity.

To complete the specification of the problem, we would naturally consider the two required BCs for Eq. (8.1). However, if the PDE is analyzed over an essentially infinite domain, $-\infty \le x \le \infty$, and if changes in the solution occur only over a finite interval in x, then *BCs at infinity have no effect*; in other words, we do not have to actually specify BCs (since they have no effect). This situation will be clarified through the PDE solution.

Consequently, Eqs. (8.1)–(8.3) constitute the complete PDE problem. The solution to Eqs. (8.1)–(8.3) was first reported by the French mathematician *Jean-le-Rond d'Alembert* (1717–1783) in 1747 in a treatise on *vibrating strings* [1, 2]. d'Alembert's remarkable solution that used a method specific to the wave equation (based on the chain rule for differentiation) is

$$u(x, t) = \frac{1}{2}\left[f(x - ct) + f(x + ct)\right] + \frac{1}{2c}\int_{x-ct}^{x+ct} g(\xi)d\xi \tag{8.4}$$

It can also be obtained by the *Fourier transform* method or by the *separation of variables* method, which are more general [3].

Equation (8.4) is actually for a somewhat more general case than Eqs. (8.1)–(8.3). In particular, IC (8.3) is generalized to $u_t(x, t = 0) = g(x)$. For most of the subsequent discussion, we consider the homogeneous IC $u_t(x, t = 0) = 0$; that is, $g(t) = 0$.

We now consider some Matlab routines for a numerical MOL solution of Eqs. (8.1)–(8.3). A main program, pde_1_main, is given in Listing 8.1.

```
%
% Clear previous files
  clear all
  clc
%
% Parameters shared with the ODE routine
  global  ncall ncase  ndss     c     x    xl    xu     n
  c=1.0;
%
% Select case
%
%   Case 1 - rectangular pulse
%
%   Case 2 - triangular pulse
%
%   Case 3 - sine pulse
%
%   Case 4 - Gaussian pulse
```

```
%
  for ncase=1:4
%
% Boundaries, number of grid points
  xl=0.0;
  xu=100.0;
  n=401;
%
% Initial condition
  t0=0.0;
  u0=inital_1(t0);
%
% Independent variable for ODE integration
  tf=30.0;
  tout=[t0:10.0:tf]';
  nout=4;
  ncall=0;
%
% ODE integration
  mf=3;
  reltol=1.0e-04; abstol=1.0e-04;
  options=odeset('RelTol',reltol,'AbsTol',abstol);
  if(mf==1) ndss=0; % explicit FDs
    [t,u]=ode45(@pde_1,tout,u0,options); end
  if(mf==2) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode45(@pde_2,tout,u0,options); end
  if(mf==3) ndss=44; % ndss = 42, 44, 46, 48 or 50 required
    [t,u]=ode45(@pde_3,tout,u0,options); end
%
% One vector to two vectors
  for it=1:nout
  for i=1:n
    u1(it,i)=u(it,i);
    u2(it,i)=u(it,i+n);
  end
  end
%
% Display selected output
  fprintf('\n  ncase = %2d,  c = %2d,  ndss = %2d\n\n', ...
          ncase,c,ndss);
  for it=1:nout
    fprintf('    t       x       u(i,j)  u_anal(i,j)
            err(i,j)\n');
    u_anal(it,:)=ua(t(it),x);
    for i=1:10:n
      err(it,i)=u1(it,i)-u_anal(it,i);
      fprintf('%6.2f%8.3f%15.6f%15.6f%15.6f\n',...
              t(it),x(i),u1(it,i),u_anal(it,i),err(it,i));
      end
      fprintf('\n');
    end
    fprintf('  ncall = %4d\n\n',ncall);
  %
  % Plot numerical and analytical solutions
    figure(ncase)
    plot(x,u1,'-',x,u_anal,'o')
    xlabel('x')
    ylabel('u(x,t)')
    title('Wave equation; t = 0, 10, 20, 30; o - numerical;
          solid - analytical')
    print -deps -r300 pde.eps; print -dps -r300 pde.ps;
    print -dpng -r300 pde.png
  %
  % Next case
    end
```

Listing 8.1. Main program pde_1_main

We can note the following points about this program:

1. After specifying some *global* variables, the program cycles through four cases for different ICs ($f(x)$ in Eq. (8.2)). These ICs are discussed subsequently. Note also that the velocity c is set to 1.

```
  %
  % Clear previous files
    clear all
    clc
  %
  % Parameters shared with the ODE routine
    global  ncall ncase  ndss     c     x    xl    xu     n
    c=1.0;
  %
  % Select case
  %
  %   Case 1 - rectangular pulse
  %
  %   Case 2 - triangular pulse
  %
  %   Case 3 - sine pulse
  %
  %   Case 4 - Gaussian pulse
  %
    for ncase=1:4
```

2. The spatial interval is defined as $0 \le x \le 100$ over 401 points; this is effectively an infinite interval ($-\infty \le x \le \infty$), as explained subsequently. The IC ($f(x)$ in Eq. (8.2)) is then specified through a call to `inital_1`. Finally, the t interval is defined as $0 \le t \le 30$ with solution outputs at $t = 0, 10, 20, 30$.

```
%
% Boundaries, number of grid points
  xl=0.0;
  xu=100.0;
  n=401;
%
% Initial condition
  t0=0.0;
  u0=inital_1(t0);
%
% Independent variable for ODE integration
  tf=30.0;
  tout=[t0:10.0:tf]';
  nout=4;
  ncall=0;
```

3. The 401 ordinary differential equations (ODEs) are integrated by a call to `ode45`; the three variations (`mf = 1,2,3`) correspond to differences in the programming of Eq. (8.1) in the ODE routines `pde_1, pde_2, pde_3`, as explained subsequently.

```
%
% ODE integration
  mf=3;
  reltol=1.0e-04; abstol=1.0e-04;
  options=odeset('RelTol',reltol,'AbsTol',abstol);
  if(mf==1) ndss=0; % explicit FDs
    [t,u]=ode45(@pde_1,tout,u0,options); end
  if(mf==2) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode45(@pde_2,tout,u0,options); end
  if(mf==3) ndss=44; % ndss = 42, 44, 46, 48 or 50 required
    [t,u]=ode45(@pde_3,tout,u0,options); end
%
% One vector to two vectors
  for it=1:nout
  for i=1:n
    u1(it,i)=u(it,i);
    u2(it,i)=u(it,i+n);
  end
  end
```

The solution vector in array u is then transferred to two arrays u1, u2 that are used because Eq. (8.1) is second order in *t*. This approach is explained when inital_1 is discussed.

4. Selected numerical output is then displayed followed by plotted output, and the program begins the next case (through the change in ncase).

```
%
% Display selected output
  fprintf('\n   ncase = %2d,   c = %2d,   ndss = %2d\n\n', ...
          ncase,c,ndss);
  for it=1:nout
    fprintf('    t       x         u(i,j)   u_anal(i,j)
            err(i,j)\n');
    u_anal(it,:)=ua(t(it),x);
    for
      err(it,i)=u1(it,i)-u_anal(it,i);
      fprintf('%6.2f%8.3f%15.6f%15.6f%15.6f\n',...
            t(it),x(i),u1(it,i),u_anal(it,i),err(it,i));
    end
    fprintf('\n');
  end
  fprintf('  ncall = %4d\n\n',ncall);
%
% Plot numerical and analytical solutions
  figure(ncase)
  plot(x,u1,'-',x,u_anal,'o')
  xlabel('x')
  ylabel('u(x,t)')
  title('Wave equation; t = 0, 10, 20, 30; o - numerical;
        solid - analytical')
  print -deps -r300 pde.eps; print -dps -r300 pde.ps;
  print -dpng -r300 pde.png
%
% Next case
  end
```

Before going on to the other routines called by main program pde_1_main, we will consider the output that helps to explain these routines. For ncase = 1 (the first pass of for ncase=1:4), a rectangular pulse is programmed in inital_1 as demonstrated in Table 8.1.

Table 8.1. Partial output from `pde_1_main` and `pde_1` (`ncase = 1`)

```
  ncase =   1,    c =   1

     t        x          u(i,j)    u_anal(i,j)       err(i,j)
  0.00    0.000        0.000000       0.000000       0.000000
  0.00    2.500        0.000000       0.000000       0.000000
  0.00    5.000        0.000000       0.000000       0.000000
  0.00    7.500        0.000000       0.000000       0.000000
  0.00   10.000        0.000000       0.000000       0.000000
             .                            .
             .                            .
             .                            .
  0.00   40.000        0.000000       0.000000       0.000000
  0.00   42.500        0.000000       0.000000       0.000000
  0.00   45.000        1.000000       1.000000       0.000000
  0.00   47.500        1.000000       1.000000       0.000000
  0.00   50.000        1.000000       1.000000       0.000000
  0.00   52.500        1.000000       1.000000       0.000000
  0.00   55.000        1.000000       1.000000       0.000000
  0.00   57.500        0.000000       0.000000       0.000000
  0.00   60.000        0.000000       0.000000       0.000000
             .                            .
             .                            .
             .                            .
  0.00   90.000        0.000000       0.000000       0.000000
  0.00   92.500        0.000000       0.000000       0.000000
  0.00   95.000        0.000000       0.000000       0.000000
  0.00   97.500        0.000000       0.000000       0.000000
  0.00  100.000        0.000000       0.000000       0.000000

             .                            .
             .                            .
             .                            .

        Output at t=10,20 removed from here.

             .                            .
             .                            .
             .                            .

     t        x          u(i,j)    u_anal(i,j)       err(i,j)
 30.00    0.000        0.000000       0.000000       0.000000
 30.00    2.500        0.000000       0.000000       0.000000
 30.00    5.000        0.000000       0.000000       0.000000
 30.00    7.500       -0.000000       0.000000      -0.000000
 30.00   10.000       -0.000000       0.000000      -0.000000
 30.00   12.500        0.000025       0.000000       0.000025
```

(*continued*)

Table 8.1 (*continued*)

```
30.00  15.000      0.278997     0.500000     -0.221003
30.00  17.500      0.441472     0.500000     -0.058528
30.00  20.000      0.541156     0.500000      0.041156
30.00  22.500      0.525924     0.500000      0.025924
30.00  25.000      0.396482     0.500000     -0.103518
30.00  27.500     -0.021875     0.000000     -0.021875
30.00  30.000     -0.001194     0.000000     -0.001194
30.00  32.500      0.025994     0.000000      0.025994
30.00  35.000     -0.025431     0.000000     -0.025431
30.00  37.500      0.003114     0.000000      0.003114
30.00  40.000     -0.034373     0.000000     -0.034373
             .                       .
             .                       .
             .                       .

30.00  60.000     -0.034373     0.000000     -0.034373
30.00  62.500      0.003114     0.000000      0.003114
30.00  65.000     -0.025431     0.000000     -0.025431
30.00  67.500      0.025994     0.000000      0.025994
30.00  70.000     -0.001194     0.000000     -0.001194
30.00  72.500     -0.021875     0.000000     -0.021875
30.00  75.000      0.396482     0.500000     -0.103518
30.00  77.500      0.525924     0.500000      0.025924
30.00  80.000      0.541156     0.500000      0.041156
30.00  82.500      0.441472     0.500000     -0.058528
30.00  85.000      0.278997     0.500000     -0.221003
30.00  87.500      0.000025     0.000000      0.000025
30.00  90.000     -0.000000     0.000000     -0.000000
30.00  92.500     -0.000000     0.000000     -0.000000
30.00  95.000      0.000000     0.000000      0.000000
30.00  97.500      0.000000     0.000000      0.000000
30.00 100.000      0.000000     0.000000      0.000000

 ncall = 2677
```

We can note the following points about this output:

1. Considering first the IC (at $t = 0$), a rectangular pulse is programmed with the properties given in Table 8.2. The pulse is defined numerically in `pde_1_main` over 401 points in x so that the grid spacing in x is $100/(401-1) = 0.25$ rather than the spacing of 2.5 in the preceding output (since only every 10th point is displayed in the output from `for i=1:10:n` in `pde_1_main`); still, the IC is only an approximation to a rectangular pulse since the discontinuous jump from 0 to 1 and back to 0 is approximated within an interval of 0.25.

Table 8.2. Properties of the rectangular pulse IC $f(x)$ of Eq. (8.2) (from Table 8.1) for t=0

$0 \le x \le 42.5$	$f(x) = 0$
$45 \le x \le 55$	$f(x) = 1$
$57.5 \le x \le 100$	$f(x) = 0$

2. The numerical MOL and analytical solutions agree at $t = 0$ (as they should); any subsequent differences between these two solutions (for $t > 0$) will be due to errors in the numerical solution.
3. These differences (errors) are evident in the output at $t = 30$. These errors are elucidated in the plotted output in Figure 8.1.
4. The analytical solution has the important properties at $t = 30$, as given in Table 8.3.

   Some additional elaboration is required to explain the significance of these properties:

   (a) The analytical solution is now in two parts: one part with $u(x, t) = 0.5$ is centered around $x = 20$ and a second part with $u(x, t) = 0.5$ is centered around $x = 80$; note, in particular, that the amplitude for both pulses is 0.5 (while the original pulse at $t = 0$ had an amplitude of 1). Outside these two pulses, the solution is zero ($u(x, t) = 0$). These features are a

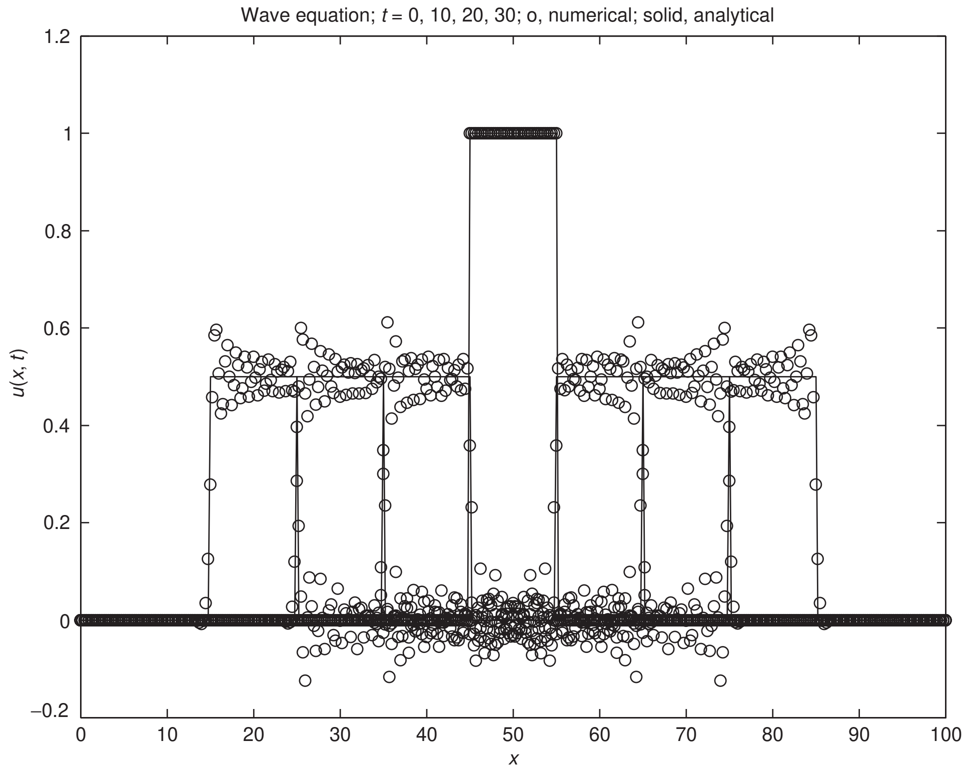

Figure 8.1. Output of main program `pde_1_main` for `ncase = 1`

Table 8.3. Properties of the solution of Eqs. (8.1)–(8.3) at $t = 30$ (from Eq. (8.4) and Table 8.1) for the IC $f(x)$ of Table 8.2

$0 \le x \le 12.5$	$u(x, t) = 0$
$15 \le x \le 25$	$u(x, t) = 0.5$
$27.5 \le x \le 72.5$	$u(x, t) = 0$
$75 \le x \le 85$	$u(x, t) = 0.5$
$87.5 \le x \le 100$	$u(x, t) = 0$

consequence of the analytical solution of Eq. (8.4) (with $g(x) = 0$ so the integral is zero).

(b) The pulse centered at $x = 20$ comes from the term $(1/2)f(x + ct)$ in Eq. (8.4) for which $45 \le x + ct \le 55$. Since $ct = (1)(30)$, we have that the position of the pulse is $45 - (1)(30) \le x \le 55 - (1)(30)$ or $15 \le x \le 25$, which is the interval in x we observe, for which the exact solution is 0.5; that is, the left pulse is traveling right to left at velocity c, and at $t = 30$, it is centered at $x = 20$. Thus, $f(x + ct)$ can be thought of as a *traveling wave* and PDE solutions with arguments such as $x - ct$ and $x + ct$ are generally termed *traveling wave solutions*.

(c) For the maximum value of t of $t = 30$, the pulse moving right to left does not reach the boundary at $x = 0$. Thus, this boundary has no effect on the solution and is effectively at $x = -\infty$; a BC is therefore not required at the left end of the interval $0 \le x \le 100$.

(d) The term $(1/2)f(x - ct)$ in Eq. (8.4) is zero for $15 \le x \le 25$ since it has the values $(1/2)f(15 - (1)(30))$ to $(1/2)f(25 - (1)(30))$ or $(1/2)f(-15)$ to $(1/2)f(-5)$. In other words, $(1/2)f(x - ct)$ has a negative argument over the interval $15 \le x \le 25$ that is outside the defined interval $0 \le x \le 100$, so we take $f(x - ct) = 0$ (as implied by the properties of Table 8.2). Thus, for the interval $15 \le x \le 25$, $f(x + ct) = 1$ and $f(x - ct) = 0$, which when substituted in Eq. (8.4) gives $u(x, t) = 0.5$ over $15 \le x \le 25$ (as observed in Table 8.1).

(e) The preceding discussion can be cast in a slightly different way. If we consider the relation $x - ct = c_1$, where c_1 is a prescribed constant, the property $f(x - ct) =$ constant follows. In other words, for x and t related by $x - ct = c_1$, the term $f(x - ct)$ in Eq. (8.4) is constant. Also, for $x + ct = c_2$, the property $f(x + ct) =$ constant follows. The conditions $x - ct = c_1$ and $x + ct = c_2$ are termed the *characteristics* of Eq. (8.1), and if we follow the evolution of the solution to Eq. (8.1), $u(x, t)$, along its characteristics, this approach to producing the solution is called the *method of characteristics*; the important feature of this approach to a PDE solution is that along the characteristics, the *solution is usually simplified*, for example, often a constant. Typically, in an analysis based on the method of characteristics, the evolution of the solution is depicted in an x–t plane with t as the ordinate (vertical coordinate) and x as the abscissa (horizontal coordinate); the numerical value of the solution appears in parametric form along with the characteristics in the x–t plane.

(f) We can view this approach to a PDE solution as *moving through the x–t plane along the characteristics* that is generally termed a *Lagrangian coordinate system*; if we consider the solution at fixed (stationary) values of x, this is generally termed an *Eulerian coordinate system*. The MOL solution of Eq. (8.1) discussed here is Eulerian since it is implemented on a fixed grid in x; that is, $0 \leq x \leq 100$. A method of characteristics version of MOL has also been reported, sometimes termed the *pseudo-characteristic method of lines*. We will not consider further the method of characteristics, but it is discussed extensively in references pertaining to the solution of hyperbolic PDEs (such as Eq. (8.1)).

(g) The preceding reasoning applies in the same way to the solution over $75 \leq x \leq 85$ for the pulse traveling left to right with amplitude 0.5 (as demonstrated by the properties of Table 8.3 and the numerical solution at $t = 30$ in Table 8.1). $f(x - ct) = 1$ is now the term in Eq. (8.4) that defines the pulse moving left to right and $f(x + ct) = 0$.

5. Finally, we can note that the characteristics $x - ct = c_1$ and $x + ct = c_2$ are *straight lines* in the x–t plane; this is a property *resulting from the linearity of the PDE*, for example, Eq. (8.1). If the PDE is nonlinear, the characteristics in the x–t plane generally *will not be linear*, but will be a function of the solution. Thus, as the solution evolves in x and t, it is necessary to compute the characteristics in using the method of characteristics.

The graphical output from `pde_1_main` in Listing 8.1 is shown in Figure 8.1. Figure 8.1 is somewhat difficult to interpret because the MOL solution is not very accurate. In fact, because the *rectangular pulse is discontinuous*, the MOL solution oscillates markedly at the points of discontinuity (which can be considered a form of the *Gibbs phenomenon*); this oscillation is also evident in the numerical output of Table 8.1. About the best we do for the interpretation of Figure 8.1 is to conclude that the *MOL does not perform satisfactorily in the case of solutions with discontinuities if finite differences (FDs) are used to approximate the spatial derivatives.* In other words, for solutions with discontinuities, the PDE spatial derivatives require special treatment, such as approximations based on *flux limiters* or *weighted essentially nonoscillatory methods*, which will not be considered in the present discussion. Numerical methods for the resolution of discontinuities and moving fronts are discussed extensively in the literature, for example, references [4–7], and are implemented in available computer codes.

The coding that produced the rectangular pulse with the properties of Table 8.2 is in the IC function `inital_1` (as might be expected since it is $f(x)$ in IC (8.2)) (see Listing 8.2).

```
  function u0=inital_1(t0)
%
% Function inital_1 sets the initial conditions for the wave
% equation
%
  global  ncall ncase  ndss     c     x    xl    xu     n
%
```

```
% Spatial domain and initial condition
  dx=(xu-xl)/(n-1);
  for i=1:n
    x(i)=(i-1)*dx;
  end
  u1=ua(0.0,x);
  for i=1:n
    u2(i)=0.0;
    u0(i)  =u1(i);
    u0(i+n)=u2(i);
  end
```

Listing 8.2. Routine `inital_1` for ICs (8.2) and (8.3)

We can note the following details about `inital_1`:

1. After definition of the function and the specification of some global variables, the spatial grid in x is defined over $n = 401$ points (with `xl = 0, xu = 100` set in the main program of Listing 8.1).

```
  function u0=inital_1(t0)
%
% Function inital_1 sets the initial conditions for the wave
% equation
%
  global  ncall ncase  ndss     c     x    xl    xu     n
%
% Spatial domain and initial condition
  dx=(xu-xl)/(n-1);
  for i=1:n
    x(i)=(i-1)*dx;
  end
```

2. The analytical solution at $t = 0$ is defined by a call to function `ua` (discussed subsequently).

```
  u1=ua(0.0,x);
  for i=1:n
    u2(i)=0.0;
    u0(i)  =u1(i);
    u0(i+n)=u2(i);
  end
```

The numerical solution consists of two parts: $u(x, t = 0)$ stored as `u1`, and $u_t(x, t = 0)$ stored as `u2`. Note that the homogeneous IC of Eq. (8.3) is used.

Thus, since Eq. (8.1) is second order in t, we need two ICs (Eqs. (8.2) and (8.3)), and we will actually be integrating $2n = 802$ ODEs, with their ICs stored in u0.

The subordinate routine ua for the analytical solution of Eq. (8.4) (with $g(x) = 0$) is given in Listing 8.3.

```
  function uanal=ua(t,x)
%
% Function ua computes the analytical solution to the wave
% equation
%
  global  ncall ncase  ndss     c    xl    xu     n
%
% The following coding is specific to the interval
% 0 le x le 100
%
% Rectangular pulse
  if(ncase==1)
    for i=1:n
      if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>=(45.0-c*t) & x(i)<=(50.0-c*t)
             pulse1(i)=0.5;
      elseif x(i)>=(50.0-c*t) & x(i)<=(55.0-c*t)
             pulse1(i)=0.5;
      end
      if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>=(45.0+c*t) & x(i)<=(50.0+c*t)
             pulse2(i)=0.5;
      elseif x(i)>=(50.0+c*t) & x(i)<=(55.0+c*t)
             pulse2(i)=0.5;
      end
      uanal(i)=pulse1(i)+pulse2(i);
    end
  end
%
% Triangular pulse
  if(ncase==2)
    for i=1:n
      if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>=(45.0-c*t) & x(i)<=(50.0-c*t)
             pulse1(i)=(x(i)-(45.0-c*t))/10.0;
      elseif x(i)>=(50.0-c*t) & x(i)<=(55.0-c*t)
             pulse1(i)=((55.0-c*t)-x(i))/10.0;
```

```
        end
        if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
        elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
        elseif x(i)>=(45.0+c*t) & x(i)<=(50.0+c*t)
               pulse2(i)=(x(i)-(45.0+c*t))/10.0;
        elseif x(i)>=(50.0+c*t) & x(i)<=(55.0+c*t)
               pulse2(i)=((55.0+c*t)-x(i))/10.0;
        end
        uanal(i)=pulse1(i)+pulse2(i);
      end
    end
  %
  % Sine pulse
    if(ncase==3)
      for i=1:n
        if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
        elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
        elseif x(i)>=(45.0-c*t) & x(i)<=(50.0-c*t)
               pulse1(i)=(1.0/2.0)*sin(pi*(x(i)-(45.0-c*t))/10.0);
        elseif x(i)>=(50.0-c*t) & x(i)<=(55.0-c*t)
               pulse1(i)=(1.0/2.0)*sin(pi*((55.0-c*t)-x(i))/10.0);
        end
        if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
        elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
        elseif x(i)>=(45.0+c*t) & x(i)<=(50.0+c*t)
               pulse2(i)=(1.0/2.0)*sin(pi*(x(i)-(45.0+c*t))/10.0);
        elseif x(i)>=(50.0+c*t) & x(i)<=(55.0+c*t)
               pulse2(i)=(1.0/2.0)*sin(pi*((55.0+c*t)-x(i))/10.0);
        end
        uanal(i)=pulse1(i)+pulse2(i);
      end
    end
  %
  % Gaussian pulse
    if(ncase==4)
      a=2.0;
      for i=1:n
        if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
        elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
        elseif x(i)>=(45.0-c*t) & x(i)<=(55.0-c*t)
               pulse1(i)=(1.0/2.0)* ...
                          exp(-a*(x(i)-(50.0-c*t))^2/10.0);
        end
        if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
        elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
        elseif x(i)>=(45.0+c*t) & x(i)<=(55.0+c*t)
               pulse2(i)=(1.0/2.0)* ...
```

```
                    exp(-a*(x(i)-(50.0+c*t))^2/10.0);
    end
    uanal(i)=pulse1(i)+pulse2(i);
  end
end
```

Listing 8.3. Function ua for the analytical solution, Eq. (8.4), (with $g(x) = 0$)

We can note the following points about ua:

1. After the function and some global variables are defined, the coding for the rectangular pulse is executed if `ncase = 1` (from `for ncase=1:4` in the main program of Listing 8.1).

```
  function uanal=ua(t,x)
%
% Function ua computes the analytical solution to the wave
% equation
%
  global  ncall ncase  ndss     c    xl    xu     n
%
% The following coding is specific to the interval
% 0 le x le 100
%
% Rectangular pulse
  if(ncase==1)
    for i=1:n
      if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>=(45.0-c*t) & x(i)<=(50.0-c*t)...
             pulse1(i)=0.5;
      elseif x(i)>=(50.0-c*t) & x(i)<=(55.0-c*t)...
             pulse1(i)=0.5;
      end
      if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>=(45.0+c*t) & x(i)<=(50.0+c*t)...
             pulse2(i)=0.5;
      elseif x(i)>=(50.0+c*t) & x(i)<=(55.0+c*t)...
             pulse2(i)=0.5;
      end
      uanal(i)=pulse1(i)+pulse2(i);
    end
  end
```

Table 8.4. Properties of the triangular pulse IC $f(x)$ of Eq. (8.2) for $t = 0$

$0 \le x \le 42.5$	$f(x) = 0$
$45 \le x \le 50$	$f(x) = (x - 45)/(50 - 45)$
$50 \le x \le 55$	$f(x) = 1 - (x - 50)/(55 - 50)$
$57.5 \le x \le 100$	$f(x) = 0$

The `for` loop computes $f(x)$ of Eq. (8.2) over the 401-point grid in `x` (the second argument of `ua`) when `t=0` is the input value (through the first argument of `ua`).

2. The calculation of $f(x)$ is in two parts: `pulse1` is $f(x - ct)$ in Eq. (8.4), while `pulse2` is $f(x + ct)$ (with `t = 0`). Note that these two variables have the nonzero value 0.5 for x corresponding to the properties of Table 8.2 and elsewhere they are zero, thereby defining the rectangular pulse.
3. In other words, the statement `uanal(i)=pulse1(i)+pulse2(i);` is the coding of Eq. (8.4), and for `t = 0`, it is `uanal(i)= 0.5 + 0.5 = 1` for `i` corresponding to $45 \le x \le 55$ as shown in Table 8.1; elsewhere, `uanal(i) = 0`.
4. For $t > 0$, the coding of `pulse1` and `pulse2` corresponds to the properties of $f(x - ct)$ and $f(x + ct)$ of Table 8.3.

We can consider other functions $f(x)$ in Eq. (8.4) (than the rectangular pulse); for example, we can consider the piecewise linear function given in Table 8.4. The coding for this triangular pulse in function `ua` (Listing 8.3, `ncase = 2`) is as follows:

```
%
% Triangular pulse
  if(ncase==2)
    for i=1:n
      if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>=(45.0-c*t) & x(i)<=(50.0-c*t)
             pulse1(i)=(x(i)-(45.0-c*t))/10.0;
      elseif x(i)>=(50.0-c*t) & x(i)<=(55.0-c*t)
             pulse1(i)=((55.0-c*t)-x(i))/10.0;
      end
      if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>=(45.0+c*t) & x(i)<=(50.0+c*t)
             pulse2(i)=(x(i)-(45.0+c*t))/10.0;
      elseif x(i)>=(50.0+c*t) & x(i)<=(55.0+c*t)
             pulse2(i)=((55.0+c*t)-x(i))/10.0;
      end
```

```
        uanal(i)=pulse1(i)+pulse2(i);
      end
    end
```

When the main program of Listing 8.1 is executed in the second pass through `for ncase=1:4`, the numerical and graphical output given in Table 8.5 results.

We can note the following points about the output given in Table 8.5:

1. For `t = 0`, the triangular pulse is centered at `x = 50` (as was the rectangular pulse) and the amplitude is unity.
2. For `t = 30`, the analytical solution is again in two parts as expected from Eq. (8.4), with the triangular pulses centered at `x = 20` and `x = 80` and each with a maximum value of 0.5.
3. For `t = 30`, the numerical solution agrees much better with the analytical solution than it did for the rectangular pulse. However, there is some appreciable difference between the numerical and analytical solutions; for example, the difference is -0.015836 at $x = 20, 80$; as might be expected, this difference is a maximum at the peaks of the two pulses at $x = 20, 80$ because the *first derivative has a discontinuity*, and switches from $+1$ to -1 (this is clear in the plotted output shown in Figure 8.2).

However, these differences are relatively small, and do not show up substantially in the plotted output of Figure 8.2. In other words, the plotted output shows relatively close agreement between the numerical and analytical solutions.

Note again that the solution starts as a single triangular pulse at $t = 0$ and centered at $x = 50$ in accordance with IC (8.2). This pulse then separates into two pulses that travel left and right with velocity `c=1` according to Eq. (8.4).

Figure 8.2 suggests that the smoothness of the IC function $f(x)$ of Eq. (8.2) plays an important role in the agreement between the numerical and analytical solutions. Before we go on to the case of a still smoother function to test this idea, we can note one other detail about the solutions for the rectangular and triangular pulses.

If the ICs for Eq. (8.1) are taken as

$$u(x, t = 0) = 0, \tag{8.5}$$

$$u_t(x, t = 0) = g(x) \tag{8.6}$$

then the analytical solution from Eq. (8.4) is

$$u(x, t) = \frac{1}{2c}\int_{x-ct}^{x+ct} g(\xi)d\xi \tag{8.7}$$

If $g(x)$ is the rectangular pulse considered previously, Eq. (8.7) indicates that it will be integrated to produce the analytical solution. Integration of the rectangular pulse

Table 8.5. Partial output from `pde_1_main` and `pde_1` (`ncase = 2`)

```
  ncase =  2,   c =  1

    t        x          u(i,j)    u_anal(i,j)       err(i,j)
 0.00    0.000        0.000000       0.000000       0.000000
 0.00    2.500        0.000000       0.000000       0.000000
 0.00    5.000        0.000000       0.000000       0.000000
 0.00    7.500        0.000000       0.000000       0.000000
 0.00   10.000        0.000000       0.000000       0.000000
          .                              .
          .                              .
          .                              .
 0.00   40.000        0.000000       0.000000       0.000000
 0.00   42.500        0.000000       0.000000       0.000000
 0.00   45.000        0.000000       0.000000       0.000000
 0.00   47.500        0.500000       0.500000       0.000000
 0.00   50.000        1.000000       1.000000       0.000000
 0.00   52.500        0.500000       0.500000       0.000000
 0.00   55.000        0.000000       0.000000       0.000000
 0.00   57.500        0.000000       0.000000       0.000000
 0.00   60.000        0.000000       0.000000       0.000000
          .                              .
          .                              .
          .                              .
 0.00   90.000        0.000000       0.000000       0.000000
 0.00   92.500        0.000000       0.000000       0.000000
 0.00   95.000        0.000000       0.000000       0.000000
 0.00   97.500        0.000000       0.000000       0.000000
 0.00  100.000        0.000000       0.000000       0.000000

          .                              .
          .                              .
          .                              .

        Output at t=10,20 removed from here.

          .                              .
          .                              .
          .                              .

    t        x          u(i,j)    u_anal(i,j)       err(i,j)
30.00    0.000       -0.000000       0.000000      -0.000000
30.00    2.500       -0.000000       0.000000      -0.000000
30.00    5.000       -0.000000       0.000000      -0.000000
30.00    7.500       -0.000000       0.000000      -0.000000
30.00   10.000        0.000000       0.000000       0.000000
30.00   12.500       -0.000000       0.000000      -0.000000
30.00   15.000        0.007346       0.000000       0.007346
30.00   17.500        0.251863       0.250000       0.001863
```

```
30.00  20.000       0.484164       0.500000      -0.015836
30.00  22.500       0.245241       0.250000      -0.004759
30.00  25.000       0.009375       0.000000       0.009375
30.00  27.500       0.004243       0.000000       0.004243
30.00  30.000      -0.000070       0.000000      -0.000070
            .                            .
            .                            .
            .                            .
30.00  70.000      -0.000070       0.000000      -0.000070
30.00  72.500       0.004243       0.000000       0.004243
30.00  75.000       0.009375       0.000000       0.009375
30.00  77.500       0.245241       0.250000      -0.004759
30.00  80.000       0.484164       0.500000      -0.015836
30.00  82.500       0.251863       0.250000       0.001863
30.00  85.000       0.007346       0.000000       0.007346
30.00  87.500      -0.000000       0.000000      -0.000000
30.00  90.000       0.000000       0.000000       0.000000
30.00  92.500      -0.000000       0.000000      -0.000000
30.00  95.000      -0.000000       0.000000      -0.000000
30.00  97.500      -0.000000       0.000000      -0.000000
30.00 100.000      -0.000000       0.000000      -0.000000

 ncall = 1489
```

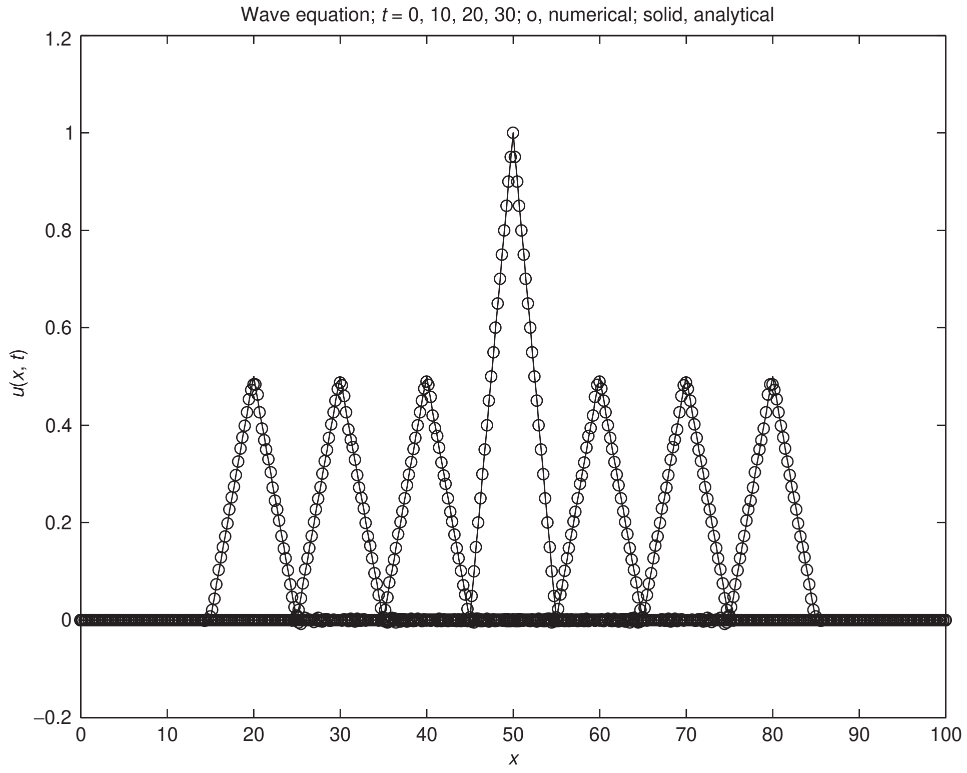

Figure 8.2. Output of main program `pde_1_main` for `ncase = 2`

Table 8.6. Properties of the sine pulse IC $f(x)$ of Eq. (8.2) for $t = 0$

$0 \le x \le 42.5$	$f(x) = 0$
$45 \le x \le 50$	$f(x) = \sin(\pi(x - 45)/(55 - 45)$
$57.5 \le x \le 100$	$f(x) = 0$

produces a linear function that is smoother than the rectangular pulse so that we would expect the analytical and numerical solutions to better agree for ICs (8.5) and (8.6) than for (8.2) and (8.3).

An IC function $f(x)$ that is smoother than either the rectangular or triangular pulses can be constructed based on the use of a sine function. Refer to Table 8.6 for an example. Now the derivatives of $f(x)$ of all orders will be smooth (a property of the sine function) except at $x = 45, 55$, where the first derivative is discontinuous due to the switch from the sine function to the constant function $f(x) = 0$.

The programming of the sine pulse in *ua* of Listing 8.3 is the traveling wave solution of Eq. (8.1) for the IC function of Table 8.6, produced by the main program of Listing 8.1 with `ncase = 3`.

```
%
% Sine pulse
  if(ncase==3)
    for i=1:n
      if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>=(45.0-c*t) & x(i)<=(50.0-c*t)
             pulse1(i)=(1.0/2.0)*sin(pi*(x(i)-(45.0-c*t))/10.0);
      elseif x(i)>=(50.0-c*t) & x(i)<=(55.0-c*t)
             pulse1(i)=(1.0/2.0)*sin(pi*((55.0-c*t)-x(i))/10.0);
      end
      if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>=(45.0+c*t) & x(i)<=(50.0+c*t)
             pulse2(i)=(1.0/2.0)*sin(pi*(x(i)-(45.0+c*t))/10.0);
      elseif x(i)>=(50.0+c*t) & x(i)<=(55.0+c*t)
             pulse2(i)=(1.0/2.0)*sin(pi*((55.0+c*t)-x(i))/10.0);
      end
      uanal(i)=pulse1(i)+pulse2(i);
    end
  end
```

Again, the traveling wave characteristic of this solution is evident from the use of the arguments $x - ct$ and $x + ct$.

Abbreviated numerical output from the main program of Listing 8.1 is given in Table 8.7.

Table 8.7. Partial output from `pde_1_main` and `pde_1` (`ncase = 3`)

```
  ncase =  3,   c =  1

     t       x          u(i,j)    u_anal(i,j)       err(i,j)
  0.00   0.000        0.000000       0.000000       0.000000
  0.00   2.500        0.000000       0.000000       0.000000
  0.00   5.000        0.000000       0.000000       0.000000
  0.00   7.500        0.000000       0.000000       0.000000
  0.00  10.000        0.000000       0.000000       0.000000
           .                            .
           .                            .
           .                            .
  0.00  40.000        0.000000       0.000000       0.000000
  0.00  42.500        0.000000       0.000000       0.000000
  0.00  45.000        0.000000       0.000000       0.000000
  0.00  47.500        0.707107       0.707107       0.000000
  0.00  50.000        1.000000       1.000000       0.000000
  0.00  52.500        0.707107       0.707107       0.000000
  0.00  55.000        0.000000       0.000000       0.000000
  0.00  57.500        0.000000       0.000000       0.000000
  0.00  60.000        0.000000       0.000000       0.000000
           .                            .
           .                            .
           .                            .
  0.00  90.000        0.000000       0.000000       0.000000
  0.00  92.500        0.000000       0.000000       0.000000
  0.00  95.000        0.000000       0.000000       0.000000
  0.00  97.500        0.000000       0.000000       0.000000
  0.00 100.000        0.000000       0.000000       0.000000


           .                            .
           .                            .
           .                            .

        Output at t=10,20 removed from here.

           .                            .
           .                            .
           .                            .

     t       x          u(i,j)    u_anal(i,j)       err(i,j)
 30.00   0.000       -0.000000       0.000000      -0.000000
 30.00   2.500        0.000000       0.000000       0.000000
 30.00   5.000       -0.000000       0.000000      -0.000000
 30.00   7.500       -0.000000       0.000000      -0.000000
```

(*continued*)

Table 8.7 (*continued*)

```
30.00  10.000      0.000000     0.000000     0.000000
30.00  12.500     -0.000001     0.000000    -0.000001
30.00  15.000      0.011563     0.000000     0.011563
30.00  17.500      0.356483     0.353553     0.002929
30.00  20.000      0.498205     0.500000    -0.001795
30.00  22.500      0.351926     0.353553    -0.001628
30.00  25.000      0.011172     0.000000     0.011172
30.00  27.500      0.003404     0.000000     0.003404
30.00  30.000     -0.000901     0.000000    -0.000901
          .                         .
          .                         .
          .                         .
30.00  70.000     -0.000901     0.000000    -0.000901
30.00  72.500      0.003404     0.000000     0.003404
30.00  75.000      0.011172     0.000000     0.011172
30.00  77.500      0.351926     0.353553    -0.001628
30.00  80.000      0.498205     0.500000    -0.001795
30.00  82.500      0.356483     0.353553     0.002929
30.00  85.000      0.011563     0.000000     0.011563
30.00  87.500     -0.000001     0.000000    -0.000001
30.00  90.000      0.000000     0.000000     0.000000
30.00  92.500     -0.000000     0.000000    -0.000000
30.00  95.000     -0.000000     0.000000    -0.000000
30.00  97.500      0.000000     0.000000     0.000000
30.00 100.000     -0.000000     0.000000    -0.000000

 ncall = 1423
```

We again observe that the sine pulse centered at $x = 50$ at $t = 0$ divides into two pulses centered at $x = 20, 80$. Also, the agreement between the numerical and analytical solutions is better than that for the triangular pulse; note, for example, that the numerical peak height is `0.498205`, while the analytical value is `0.500000`. These properties are evident in Figure 8.3 produced by the main program in Listing 8.1.

Finally, we can consider a still smoother solution for a Gaussian pulse that has derivatives of all orders throughout the spatial domain $0 \le x \le 100$.

$$0 \le x \le 100 \quad f(x) = \exp(-a(x-50)^2) \tag{8.8}$$

where a is a prescribed constant.

The programming of the Gaussian pulse in *ua* of Listing 8.3 is the traveling wave solution of Eq. (8.1) for the IC function of Eq. (8.8), produced by the main program of Listing 8.1 with `ncase = 4`.

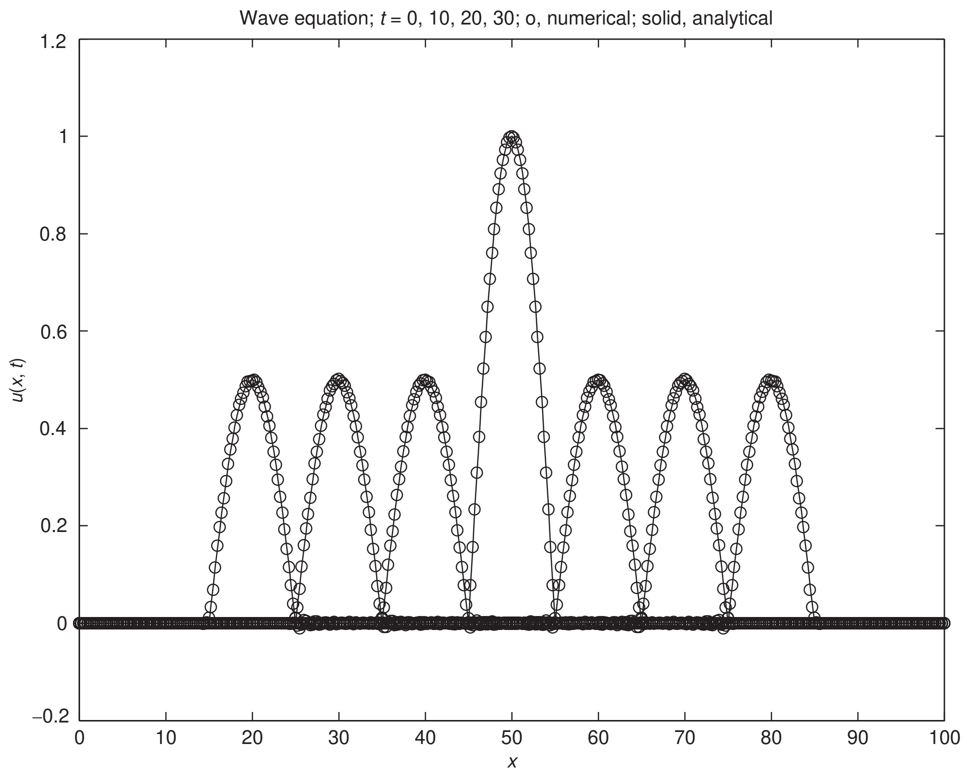

Figure 8.3. Output of main program `pde_1_main` for `ncase = 3`

```
%
% Gaussian pulse
  if(ncase==4)
    a=2.0;
    for i=1:n
      if     x(i)<(45.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>(55.0-c*t)  pulse1(i)=0.0;
      elseif x(i)>=(45.0-c*t) & x(i)<=(55.0-c*t)
             pulse1(i)=(1.0/2.0)*exp(-a*(x(i)-(50.0-c*t))^2/10.0);
      end
      if     x(i)<(45.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>(55.0+c*t)  pulse2(i)=0.0;
      elseif x(i)>=(45.0+c*t) & x(i)<=(55.0+c*t)
             pulse2(i)=(1.0/2.0)*exp(-a*(x(i)-(50.0+c*t))^2/10.0);
      end
      uanal(i)=pulse1(i)+pulse2(i);
    end
  end
```

Table 8.8. Partial Output from `pde_1_main` and `pde_1` (`ncase = 4`)

```
 ncase =  4,   c =  1

     t        x          u(i,j)    u_anal(i,j)       err(i,j)
  0.00    0.000        0.000000       0.000000       0.000000
  0.00    2.500        0.000000       0.000000       0.000000
  0.00    5.000        0.000000       0.000000       0.000000
  0.00    7.500        0.000000       0.000000       0.000000
  0.00   10.000        0.000000       0.000000       0.000000
             .                            .
             .                            .
             .                            .
  0.00   40.000        0.000000       0.000000       0.000000
  0.00   42.500        0.000000       0.000000       0.000000
  0.00   45.000        0.006738       0.006738       0.000000
  0.00   47.500        0.286505       0.286505       0.000000
  0.00   50.000        1.000000       1.000000       0.000000
  0.00   52.500        0.286505       0.286505       0.000000
  0.00   55.000        0.006738       0.006738       0.000000
  0.00   57.500        0.000000       0.000000       0.000000
  0.00   60.000        0.000000       0.000000       0.000000
             .                            .
             .                            .
             .                            .
  0.00   90.000        0.000000       0.000000       0.000000
  0.00   92.500        0.000000       0.000000       0.000000
  0.00   95.000        0.000000       0.000000       0.000000
  0.00   97.500        0.000000       0.000000       0.000000
  0.00  100.000        0.000000       0.000000       0.000000


             .                            .
             .                            .
             .                            .

        Output at t=10,20 removed from here.

             .                            .
             .                            .
             .                            .

     t        x          u(i,j)    u_anal(i,j)       err(i,j)
 30.00    0.000        0.000000       0.000000       0.000000
 30.00    2.500        0.000000       0.000000       0.000000
 30.00    5.000        0.000000       0.000000       0.000000
 30.00    7.500       -0.000000       0.000000      -0.000000
```

```
30.00  10.000     -0.000000     0.000000     -0.000000
30.00  12.500      0.000000     0.000000      0.000000
30.00  15.000      0.002408     0.003369     -0.000961
30.00  17.500      0.143043     0.143252     -0.000209
30.00  20.000      0.500208     0.500000      0.000208
30.00  22.500      0.143357     0.143252      0.000104
30.00  25.000      0.003029     0.003369     -0.000340
30.00  27.500      0.000008     0.000000      0.000008
30.00  30.000     -0.000209     0.000000     -0.000209
          .                         .
          .                         .
          .                         .
30.00  70.000     -0.000209     0.000000     -0.000209
30.00  72.500      0.000008     0.000000      0.000008
30.00  75.000      0.003029     0.003369     -0.000340
30.00  77.500      0.143357     0.143252      0.000104
30.00  80.000      0.500208     0.500000      0.000208
30.00  82.500      0.143043     0.143252     -0.000209
30.00  85.000      0.002408     0.003369     -0.000961
30.00  87.500      0.000000     0.000000      0.000000
30.00  90.000     -0.000000     0.000000     -0.000000
30.00  92.500     -0.000000     0.000000     -0.000000
30.00  95.000      0.000000     0.000000      0.000000
30.00  97.500      0.000000     0.000000      0.000000
30.00 100.000      0.000000     0.000000      0.000000

 ncall = 1039
```

The constant a is 2, which determines the rate at which the Gaussian pulse Eq. (8.8) decays and for this value, ensures that it is effectively zero near the boundaries $x = 0, 100$ so that for Eq. (8.1) BCs are not required; also, to maintain continuity with the programming of the rectangular, triangular, and sine pulses in Listing 8.3, three sections for `pulse1(i), pulse2(i)` are programmed, but a single section over all `i` could have been used just as well. Again, the traveling wave characteristic of this solution is evident from the use of the arguments $x - ct$ and $x + ct$. Abbreviated numerical output from the main program of Listing 8.1 is given in Table 8.8.

We again observe that the Gaussian pulse centered at $x = 50$ at $t = 0$ divides into two pulses centered at $x = 20, 80$. Also, the agreement between the numerical and analytical solutions is better than that for the triangular and sine pulses; note, for example, that the numerical peak height is `0.500208`, while the analytical value is `0.500000`. These properties are evident in Figure 8.4, produced by the main program in Listing 8.1.

Finally, the ODE routines for the MOL approximation of Eqs. (8.1)–(8.3) remain for some discussion. These routines, `pde_1, pde_2, pde_3`, called by `ode15s`

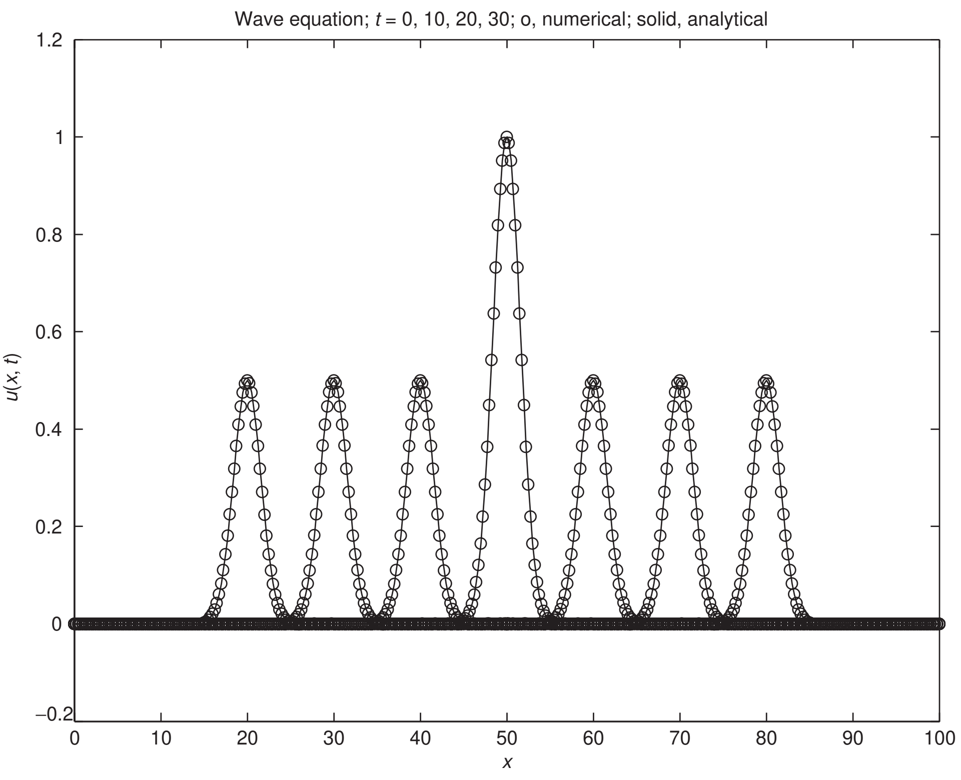

Figure 8.4. Output of main program `pde_1_main` for `ncase = 4`

from the main program of Listing 8.1 (for `mf=1,2,3`, respectively), are, of course, a central part of the MOL solutions discussed previously. However, these routines have been discussed in some detail in earlier PDE chapters, for example, the chapter on the diffusion equation, and they are therefore listed in an appendix at the end of this chapter, with a few explanatory comments.

In summary, we have observed the following:

1. Traveling wave solutions of the linear wave equation (8.1), as illustrated by the d'Alembert solution of Eq. (8.4).
2. The numerical MOL solutions with accuracy that is directly tied to the smoothness of the IC function, $f(x)$, of Eq. (8.2).
3. In particular, for an IC function that is discontinuous, for example, the rectangular pulse, the MOL solution based on FDs has severe numerical distortions (oscillations). This type of problem (with a discontinuous IC) is generally termed a *Riemann problem*, which, as the preceding discussion illustrates, can present significant numerical difficulties that require special numerical techninqes (in the form of a *Riemann solver*). In recent years major advances have been made in the development of so-called *high-resolution methods* that are able to resolve discontinuous solutions, often involving shock waves, to an increasingly high degree of accuracy and without oscillations [4, 5, 7]. A very

interesting and readable review of the state of the art in *computational fluid dynamics* is given in reference [8].

You may then question why a numerical MOL solution is considered for a Riemann problem even if it performs so poorly. The answer is that frequently all we have is a numerical solution since analytical solutions are generally unavailable for realistic physical problems. For example, the Euler equations of fluid mechanics (a system of hyperbolic PDEs) are often solved with a discontinuous IC, or under conditions for which a discontinuity develops (a *shock*). The only tractable approach to a solution to the Euler equations is usually numerical (because of their complexity, especially their nonlinearity). Thus, the development of numerical methods for this important class of (Riemann) problems is a very active area of research.

APPENDIX A

A.1. ODE Routines pde_1, pde_2, pde_3

```
  function ut=pde_1(t,u)
%
% Function pde_1 computes the t derivative vector for the wave
% equation
%
  global  ncall     c    xl    xu     n
%
% One vector to two vectors
  for i=1:n
    u1(i)=u(i);
    u2(i)=u(i+n);
  end
%
% Calculate u1xx with ux(1) = ux(n) = 0 BCs
  dxs=((xu-xl)/(n-1))^2;
  for i=1:n
     if(i==1)     u1xx(i)=2.0*(u1(i+1)-u1(i))/dxs;
     elseif(i==n) u1xx(i)=2.0*(u1(i-1)-u1(i))/dxs;
     else         u1xx(i)=(u1(i+1)-2.0*u1(i)+u1(i-1))/dxs;
     end
  end
%
% PDE
  cs=c^2;
  for i=1:n
    u1t(i)=u2(i);
    u2t(i)=cs*u1xx(i);
  end
```

```
%
% Two vectors to one vector
  for i=1:n
    ut(i)  =u1t(i);
    ut(i+n)=u2t(i);
  end
  ut=ut';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Listing 8.4. Function pde_1 for the MOL Solution of Eq. (8.1) based on explicit FD approximations of u_{xx}

```
  function ut=pde_2(t,u)
%
% Function pde_2 computes the t derivative vector for the wave
% equation
%
  global  ncall  ndss     c    xl    xu     n
%
% One vector to two vectors
  for i=1:n
    u1(i)=u(i);
    u2(i)=u(i+n);
  end
%
% Calculate u1x
  if    (ndss== 2) u1x=dss002(xl,xu,n,u1); % second order
  elseif(ndss== 4) u1x=dss004(xl,xu,n,u1); % fourth order
  elseif(ndss== 6) u1x=dss006(xl,xu,n,u1); % sixth order
  elseif(ndss== 8) u1x=dss008(xl,xu,n,u1); % eighth order
  elseif(ndss==10) u1x=dss010(xl,xu,n,u1); % tenth order
  end
%
% BC at x = 0
  u1x(1)=0.0;
%
% BC at x = 1
  u1x(n)=0.0;
%
% Calculate u1xx
  if    (ndss== 2) u1xx=dss002(xl,xu,n,u1x); % second order
  elseif(ndss== 4) u1xx=dss004(xl,xu,n,u1x); % fourth order
  elseif(ndss== 6) u1xx=dss006(xl,xu,n,u1x); % sixth order
  elseif(ndss== 8) u1xx=dss008(xl,xu,n,u1x); % eighth order
  elseif(ndss==10) u1xx=dss010(xl,xu,n,u1x); % tenth order
  end
%
% PDE
  cs=c^2;
  for i=1:n
    u1t(i)=u2(i);
    u2t(i)=cs*u1xx(i);
  end
%
% Two vectors to one vector
  for i=1:n
    ut(i)  =u1t(i);
    ut(i+n)=u2t(i);
  end
  ut=ut';
%
% Increment calls to pde_2
  ncall=ncall+1;
```

Listing 8.5. Function pde_2 for the MOL solution of Eq. (8.1) based on FD approximations of u_{xx} in dss002 to dss010

```
  function ut=pde_3(t,u)
%
% Function pde_3 computes the t derivative vector for the wave
% equation
%
  global  ncall  ndss     c    xl    xu     n
%
% One vector to two vectors
  for i=1:n
    u1(i)=u(i);
    u2(i)=u(i+n);
  end
%
% Calculate u1xx with ux(1) = ux(n) = 0 as BCs
  u1x=zeros(n,1);
  u1x(1)=0.0;
  u1x(n)=0.0;
  nl=2;
  nu=2;
  if    (ndss==42) u1xx=dss042(xl,xu,n,u1,u1x,nl,nu);
  % second order
  elseif(ndss==44) u1xx=dss044(xl,xu,n,u1,u1x,nl,nu);
  % fourth order
  elseif(ndss==46) u1xx=dss046(xl,xu,n,u1,u1x,nl,nu);
  % sixth order
  elseif(ndss==48) u1xx=dss048(xl,xu,n,u1,u1x,nl,nu);
  % eighth order
  elseif(ndss==50) u1xx=dss050(xl,xu,n,u1,u1x,nl,nu);
  % tenth order
  end
%
% PDE
  cs=c^2;
  for i=1:n
    u1t(i)=u2(i);
    u2t(i)=cs*u1xx(i);
  end
%
% Two vectors to one vector
  for i=1:n
    ut(i)  =u1t(i);
    ut(i+n)=u2t(i);
  end
  ut=ut';
%
% Increment calls to pde_3
  ncall=ncall+1;
```

Listing 8.6. Function pde_3 for the MOL solution of Eq. (8.1) based on FD approximations of u_{xx} in dss042 to dss050

In each of these three routines, there are three sections of code that pertain specifically to Eq. (8.1).

1. The transfer of the incoming dependent-variable vector u into two arrays u1 and u2 (again, because Eq. (8.1) is second order in t).

```
%
% One vector to two vectors
  for i=1:n
    u1(i)=u(i);
    u2(i)=u(i+n);
  end
```

2. The coding of Eq. (8.1) (recall $u \leftrightarrow u1,\; u_t \leftrightarrow u2$).

```
%
% PDE
  cs=c^2;
  for i=1:n
    u1t(i)=u2(i);
    u2t(i)=cs*u1xx(i);
  end
```

3. The transfer of the two derivative arrays `u1t` and `u2t` to the dependent-variable derivative array `ut`.

```
%
% Two vectors to one vector
  for i=1:n
    ut(i)  =u1t(i);
    ut(i+n)=u2t(i);
  end
  ut=ut';
```

In each of the three routines, the basic requirement of computing the ODE derivative vector `ut` as an output from the input dependent variable `u` as an input is accomplished. You may wish to review the calculation of the derivative u_{xx} as explained in earlier listings of these routines with the same names. All of the preceding numerical solutions were computed with `pde_3` (for `mf=3`). Very similar solutions would have resulted using `pde_1` or `pde_2` (since $n = 401$ was determined to be an adequate number of grid points in x to achieve acceptable spatial convergence and `ode15s` provided acceptable accuracy in the t integration using the tolerances specified as input arguments).

REFERENCES

[1] Farlow, S. J. (1993), More on the D'Alembert Equation *Partial Differential Equations for Scientists and Engineers*, Dover Publications, New York, Chapter 18, pp. 137–145

[2] Cajori, F. (1961), *A History of Mathematics*, MacMillan, New York

[3] Kreyszig, E. (1993), *Advanced Engineering Mathematics – Seventh Edition*, Wiley, New York

[4] Leveque, R. J. (2002), *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, Cambridge, UK

[5] Shu, C.-W. (1998), Essentially Non-Oscillatory and Weighted Essential Non-oscillatory Schemes for Hyperbolic Conservation Laws, In: B. Cockburn, C. Johnson, C.-W. Shu, and E. Tadmor (Eds.), *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, Lecture Notes in Mathematics, vol. 1697, Springer, New York, pp. 325–432

[6] Strauss, W. A. (1992), *Partial Differential Equations: An Introduction*, Wiley, New York
[7] Wesseling, P. (2001), *Principles of Computational Fluid Dynamics*, Springer, New York
[8] Koren, B. (2006), Computational Fluid Dynamics: Science and Tool, In: *Centrum Wiskunde & Informatica (CWI), Modeling, Analysis and Simulation [MAS]*, Report E 0602, ISSN: 1386-3703, January 2006; available online at `ftp://ftp.cwi.nl/pub/CWIreports/MAS/MAS-E0602.pdf`