4

Two Nonlinear, Variable-Coefficient, Inhomogeneous Partial Differential Equations

This 2×2 partial differential equation (PDE) problem (two PDEs in two unknowns) introduces the following mathematical concepts and computational methods:

1. The two PDEs have a variety of mathematical properties; that is, they are
   - simultaneous,
   - nonlinear,
   - variable coefficient, and
   - inhomogeneous.

   These features are briefly explained in the discussion that follows. The subsequent programming illustrates how these features can be included in a computer code.
2. The two PDEs have an exact solution that can be used to assess the accuracy of the method of lines (MOL) numerical solution.
3. The effectiveness of higher-order finite differences (FDs) in improving the accuracy of numerical MOL solutions to PDEs.

The PDEs are [1]:

$$u_t = ((v-1)u_x)_x + (16xt - 2t - 16(v-1)(u-1))(u-1) + 10x\,e^{-4x} \tag{4.1a}$$

$$v_t = v_{xx} + u_x + 4u - 4 + x^2 - 2t - 10t\,e^{-4x} \tag{4.1b}$$

Here we have used the subscript notation of partial derivatives; for example,

$$u_t \leftrightarrow \frac{\partial u}{\partial t}, \quad ((v-1)u_x)_x \leftrightarrow \frac{\partial}{\partial x}\left[(v-1)\frac{\partial u}{\partial x}\right]$$

Also, we consider t to be an *initial-value variable* and x to be a *boundary-value variable* (the distinction is explained subsequently). Since each PDE is first order in t, each requires one initial condition (IC); these ICs are taken as

$$u(x, t=0) = 1, \qquad v(x, t=0) = 1 \tag{4.2a,b}$$

Equations (4.2a) and (4.2b) are termed *inhomogeneous* ICs since they are not zero on the RHS.

Equations (4.1a) and (4.1b) are second order in x and therefore each equation requires two boundary conditions (BCs); these BCs are taken as

$$u(x=0,t) = v(x=0,t) = 1 \tag{4.3a,b}$$

$$3u(x=1,t) + u_x(x=1,t) = 3; \qquad 5v_x(x=1,t) = e^4(u(x=1,t)-1) \tag{4.3c,d}$$

Note that Eqs. (4.3a) and (4.3b) are applied at $x = 0$, while Eqs. (4.3c) and (4.3d) are applied at $x = 1$, that is two different values of x which is why they are called boundary conditions; generally, BCs represent conditions at different boundaries of a physical system.

Equations (4.3a) and (4.3b) are termed *BCs of the first type* or *Dirichlet BCs* because they involve just the dependent variables, $u(x,t)$ and $v(x,t)$ at $x = 0$. Equations (4.3c) and (4.3d) are termed *BCs of the third type* or *Robin BCs* because they involve the dependent variables $u(x,t)$ and $v(x,t)$ and their first spatial derivatives $u_x(x,t)$ and $v_x(x,t)$ at $x = 1$. To complete the picture, BCs that involve only the derivatives $u_x(x,t)$ and $v_x(x,t)$ are termed *BCs of the second type* or *Neumann BCs*; examples would be the homogeneous Neumann BCs $u_x(x=1,t) = 0$ and $v_x(x = 0,t)$. All of the BCs as stated are *linear*; that is, the dependent variables and their derivatives are to the first power. BCs can also be nonlinear; for example, the third-type BCs $u_x(x=1,t) = h(u_a^4 - u(x=1,t)^4)$ and $v_x(x=1,t) = h(v_a^4 - v(x=1,t)^4)$ are nonlinear, where h and u_a and v_a are given positive constants.

To continue this discussion of terminology a bit further, we return to the PDEs, Eqs. (4.1a) and (4.1b). Some of the terminology is listed in item 1. Equations (4.1a) and (4.1b) are

- *Simultaneous* since the dependent variables $u(x,t)$ and $v(x,t)$ appear in both equations so that the PDEs must be solved together.
- *Nonlinear* since the dependent variables $u(x,t)$ and $v(x,t)$ and their derivatives are to other than the first power, for example, in Eq. (4.1a) $((v-1)u_x)_x$ and $(v-1)(u-1)$; note however that Eq. (4.1b) is linear.
- *Variable coefficient* since the dependent variables $u(x,t)$ and $v(x,t)$ are multiplied by functions of the *independent variables*, for example, in Eq. (4.1a) $(16xt - 2t)(u-1)$.
- *Inhomogeneous* since terms dependent on only the *independent variables* appear in the equations, for example, $10xe^{-4x}$ in both equations; these terms are also termed *nonhomogenenous*.
- *Mixed type*. To explain, since Eqs. (4.1a) and (4.1b) contain first-order derivatives in t and second-order derivatives in x, they can be termed *parabolic*. But Eqs. (4.1a) and (4.1b) also contain first-order derivatives in x so that they can be termed *first-order hyperbolic*. Thus, Eqs. (4.1a) and (4.1b) can be termed *hyperbolic–parabolic*, or in more physical terms, *convective–diffusive*.

The subsequent programming demonstrates that all of these variations in mathematical features can be accommodated by the MOL approach.

Equations (4.1)–(4.3) have an analytical solution

$$u(x,t) = 1 + 10xte^{-4x}, \qquad v(x,t) = 1 + x^2t \tag{4.4a,b}$$

This analytical solution is used to assess the accuracy of the MOL numerical solution in the subsequent programming. Also, Eqs. (4.4a) and (4.4b) suggest that this problem is artificial (fabricated or contrived) in the sense that it does not originate from a physical application. However, it requires coding of all of the mathematical features discussed previously and therefore we view it as a very useful and comprehensive test problem.

A program to implement Eqs. (4.1) and (4.4) follows. First the main program (pde_1_main) is given in Listing 4.1.

```
%
% Clear previous files
  clear all
  clc
%
% Parameters shared with other routines
  global ncall   ncase...
             n       x    ndss
%
% Two cases: ncase = 1, second order finite differences
%
%            ncase = 2, fourth order finite difference
%
  for ncase=1:2
%
%   Initial condition
    n=41;
    t0=0.0;
    y0=inital_1(t0);
%
%   Independent variable for ODE integration
    tf=1.0;
    nout=21;
    tout=linspace(t0,tf,nout);
    ncall=0;
%
%   ODE integration ;
    reltol=1.0e-04; abstol=1.0e-04;
    options=odeset('RelTol',reltol,'AbsTol',abstol);
    if(ncase==1) ndss=2; % ndss = 2, 4, 6, 8 or 10 required
      [t,y]=ode15s(@pde_1,tout,y0,options); end
    if(ncase==2) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
      [t,y]=ode15s(@pde_1,tout,y0,options); end
%
%   One vector to two vectors
    for it=1:nout
    for i=1:n
      u(it,i)=y(it,i);
      v(it,i)=y(it,i+n);
    end
    end
%
%   Display a heading for the numerical and analytical
%   solutions
    fprintf('\n\n  ncase = %2d\n',ncase);
    fprintf('\n  u =  numerical u(x,t)');
    fprintf('\n ua = analytical u(x,t)');
    fprintf('\n diff u = u - ua');
    fprintf('\n  v =  numerical v(x,t)');
    fprintf('\n va = analytical v(x,t)');
    fprintf('\n diff v = v - uv\n');
%
%   Analytical solutions, differences between these solutions
%   at selected x
    for it=1:nout
      fprintf('\n    t       x        u      ua     diff u');
      fprintf('\n                     v      va     diff v');
    for i=1:10:n
      ua(i)=1.0+10.0*x(i)*t(it)*exp(-4.0*x(i));
      du(i)=u(it,i)-ua(i);
      va(i)=1.0+(x(i)^2)*t(it);
      dv(i)=v(it,i)-va(i);
%
%   Display the numerical solutions, analytical solutions,
%   differences between the solutions
    fprintf('\n %4.2f%9.3f%9.4f%9.4f%12.2e\n
            %23.4f%9.4f%12.2e\n',t(it),x(i),u(it,i),ua(i), ...
            du(i),v(it,i),va(i),dv(i));
    end
    end
    fprintf('  ncall = %5d\n',ncall);
%
% Plot solutions
  if(ncase==1)
%
%   Parametric plots
    figure(1);
    subplot(1,2,1)
    plot(x,u,'-k'); axis tight
    title('u(x,t) vs x, t = 0, 0.05, 0.1, ..., 1.0');
          xlabel('x'); ylabel('u(x,t)')
    subplot(1,2,2)
    plot(x,v,'-k'); axis tight
    title('v(x,t) vs x, t = 0, 0.05, 0.1, ..., 1.0');
          xlabel('x'); ylabel('v(x,t)')
%
%   Surface plots
    figure(2);
    surf(u); axis tight
    xlabel('x grid number');
    ylabel('t grid number');
    zlabel('u(x,t)');
    title('Two nonlinear PDEs');
    view([-115 36]);
    colormap gray
    rotate3d on;
    figure(3);
    surf(v); axis tight
    xlabel('x grid number');
    ylabel('t grid number');
    zlabel('v(x,t)');
    title('Two nonlinear PDEs');
    view([170 24]);
    colormap gray
    rotate3d on;
%   print -r300 -deps pde.eps; print -r300 -dps pde.ps;
%   print -r300 -dpng pde.png
  end
%
% Next case
  end
```

Listing 4.1. Main program pde_1_main

We can note the following points about this program:

1. After placing some of the problem parameters and variables in global, two cases are programmed with a for loop for the calculation of the spatial derivatives by second- and fourth-piorder FDs.

```
%
% Clear previous files
  clear all
  clc
%
% Parameters shared with other routines
  global ncall   ncase...
             n       x    ndss
%
```

```
% Two cases: ncase = 1, second order finite differences
%
%            ncase = 2, fourth order finite difference
%
  for ncase=1:2
```

2. ICs (4.2a) and (4.2b) are defined on a 41-point spatial grid (through a call to `inital_1` discussed subsequently) and a 21-point time grid (for $0 \le t \le 1$).

```
%
%   Initial condition
    n=41;
    t0=0.0;
    y0=inital_1(t0);
%
%   Independent variable for ODE integration
    tf=1.0;
    nout=21;
    tout=linspace(t0,tf,nout);
    ncall=0;
```

3. The ODE integration within the MOL PDE solution is performed by `ode15s`; note that the IC vector `y0` is of length `2n` ($n = 41$), with the first `n` values assigned to `u` and the second `n` values assigned to `v`.

```
%
%   ODE integration ;
    reltol=1.0e-04; abstol=1.0e-04;
    options=odeset('RelTol',reltol,'AbsTol',abstol);
    if(ncase==1) ndss=2; % ndss = 2, 4, 6, 8 or 10 required
      [t,y]=ode15s(@pde_1,tout,y0,options); end
    if(ncase==2) ndss=4; % ndss = 2, 4, 6, 8 or 10 required
      [t,y]=ode15s(@pde_1,tout,y0,options); end
%
%   One vector to two vectors
    for it=1:nout
    for i=1:n
      u(it,i)=y(it,i);
      v(it,i)=y(it,i+n);
    end
    end
```

After the ODE numerical solution is returned in y, the solution of Eqs. (4.1a) and (4.1b) is put into arrays u and v.

4. The numerical solution is then displayed along with the analytical solution (computed according to Eq. (4.4)).

```
%
%   Display a heading for the numerical and
%   analytical solutions
    fprintf('\n\n  ncase = %2d\n',ncase);
    fprintf('\n  u =  numerical u(x,t)');
    fprintf('\n ua = analytical u(x,t)');
    fprintf('\n diff u = u - ua');
    fprintf('\n  v =  numerical v(x,t)');
    fprintf('\n va = analytical v(x,t)');
    fprintf('\n diff v = v - uv\n');
%
%   Analytical solutions, differences between these solutions
%   at selected x
    for it=1:nout
      fprintf('\n    t       x        u      ua     diff u');
      fprintf('\n                     v      va     diff v');
    for i=1:10:n
      ua(i)=1.0+10.0*x(i)*t(it)*exp(-4.0*x(i));
      du(i)=u(it,i)-ua(i);
      va(i)=1.0+(x(i)^2)*t(it);
      dv(i)=v(it,i)-va(i);
%
%   Display the numerical solutions, analytical solutions,
%   differences between the solutions
    fprintf('\n %4.2f%9.3f%9.4f%9.4f%12.2e\n
           %23.4f%9.4f%12.2e\n',t(it),x(i),u(it,i),ua(i), ...
           du(i),v(it,i),va(i),dv(i));
    end
    end
    fprintf('  ncall = %5d\n',ncall);
```

5. The numerical solutions are plotted as a function of x with t as a parameter.

```
%
% Plot solutions
  if(ncase==1)
```

```
%
%   Parametric plots
    figure(1);
    subplot(1,2,1)
    plot(x,u,'-k'); axis tight
    title('u(x,t) vs x, t = 0, 0.05, 0.1, ..., 1.0');
          xlabel('x'); ylabel('u(x,t)')
    subplot(1,2,2)
    plot(x,v,'-k'); axis tight
    title('v(x,t) vs x, t = 0, 0.05, 0.1, ..., 1.0');
          xlabel('x'); ylabel('v(x,t)')
```

6. Finally, the numerical solutions are plotted in three-dimensional (3D) perspective.

```
%
%   Surface plots
    figure(2);
    surf(u); axis tight
    xlabel('x grid number');
    ylabel('t grid number');
    zlabel('u(x,t)');
    title('Two nonlinear PDEs');
    view([-115 36]);
    colormap gray
    rotate3d on;
    figure(3);
    surf(v); axis tight
    xlabel('x grid number');
    ylabel('t grid number');
    zlabel('v(x,t)');
    title('Two nonlinear PDEs');
    view([170 24]);
    colormap gray
    rotate3d on;
%   print -r300 -deps pde.eps; print -r300 -dps pde.ps;
%   print -r300 -dpng pde.png
  end
%
% Next case
  end
```

The initialization routine `inital_1` is given in Listing 4.2.

```
  function y=inital_1(t0)
%
% Function inital_1 is called by the main program to define
% the initial conditions in the MOL solution of two nonlinear
% PDEs
%
  global ncall   ncase...
             u       v...
             n       x      e4    ndss
%
% Spatial increment
  dx=1.0/(n-1);
%
% Values of x along the spatial grid
  x=[0.0:dx:1.0];
%
% Initial conditions
  for i=1:n
    u(i)=1.0;
    v(i)=1.0;
    y(i)  =u(i);
    y(i+n)=v(i);
  end
%
% Constant e^4 used in boundary condition
  e4=exp(1.0)^4;
%
% Initialize calls to pde_1
  ncall=0;
```

Listing 4.2. Initialization routine `inital_1`

The following points can be noted:

1. After defining the function and the global parameters and variables, the spatial grid in x is defined for $0 \leq x \leq 1$.

```
  function y=inital_1(t0)
%
% Function inital_1 is called by the main program to define
% the initial conditions in the MOL solution of two
% nonlinear PDEs
%
```

```
  global ncall   ncase...
             u       v...
             n       x      e4    ndss
%
% Spatial increment
  dx=1.0/(n-1);
%
% Values of x along the spatial grid
  x=[0.0:dx:1.0];
```

2. ICs (4.2a) and (4.2b) are then set for $u(x, t)$ and $v(x, t)$ and subsequently put into one dependent-variable array y.

```
%
% Initial conditions
  for i=1:n
    u(i)=1.0;
    v(i)=1.0;
    y(i)  =u(i);
    y(i+n)=v(i);
  end
%
% Constant e^4 used in boundary condition
  e4=exp(1.0)^4;
%
% Initialize calls to pde_1
  ncall=0;
```

Finally, e^4 is computed here to avoid repeated calculation in the ODE routine pde_1 (listed next) and the counter for the calls to pde_1 is initialized.

The ODE routine pde_1 called by ode15s in pde_1_main is given in Listing 4.3.

```
  function yt=pde_1(t,y)
%
% Function pde_1 defines the ODEs in the MOL solution of two
% nonlinear PDEs
%
  global ncall   ncase...
             u       v...
            ut      vt...
            ux     uxx      vx     vxx...
            n       x      e4    ndss
%
% One vector to two vectors
  for i=1:n
    u(i)=y(i);
    v(i)=y(i+n);
  end
%
% Boundary conditions at x = 0
  u(1)=1.0;
  ut(1)=0.0;
  v(1)=1.0;
  vt(1)=0.0;
%
% First order spatial derivatives
  xl=x(1);
  xu=x(n);
%
% Three point centered differences
  if(ncase==1)ux=dss002(xl,xu,n,u); end
  if(ncase==1)vx=dss002(xl,xu,n,v); end
%
% Five point centered differences
  if(ncase==2)ux=dss004(xl,xu,n,u); end
  if(ncase==2)vx=dss004(xl,xu,n,v); end
%
% Boundary conditions at x = 1
  ux(n)=3.0-3.0*u(n);
  vx(n)=e4*(u(n)-1.0)/5.0;
%
% Second order spatial derivatives
%
% Three point centered  differences
  if(ncase==1)vxx=dss002(xl,xu,n,vx); end
%
% Five point centered differences
  if(ncase==2)vxx=dss004(xl,xu,n,vx); end
%
% Array vx is used as temporary storage in the calculation
% of the term ((v - 1)*u )  which is finally stored in
% array uxx             x x
  for i=1:n
    vx(i)=(v(i)-1.0)*ux(i);
  end
  if(ncase==1)uxx=dss002(xl,xu,n,vx); end
  if(ncase==2)uxx=dss004(xl,xu,n,vx); end
%
```

```
% Two PDEs
  for i=2:n
    ex=exp(-4.0*x(i));
      ut(i)=uxx(i)+(16.0*x(i)*t-2.0*t-16.0*(v(i)-1.0))* ...
            (u(i)-1.0)+10.*x(i)*ex;vt(i)=vxx(i)+ux(i) ...
            +4.0*u(i)-4.0+x(i)^2-2.0*t-10.0*t*ex;
  end
%
% Two vectors to one vector
  for i=1:n
    yt(i)   =ut(i);
    yt(i+n) =vt(i);
  end
  yt=yt';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Listing 4.3. ODE routine pde_1

We can note the following points:

1. After the routine and the global parameters and variables are defined, the dependent-variable vector y is divided into the two problem-oriented arrays u and v (to facilitate programming in terms of these problem-oriented variables).

```
  function yt=pde_1(t,y)
%
% Function pde_1 defines the ODEs in the MOL solution of two
% nonlinear PDEs
%
  global ncall   ncase...
             u       v...
            ut      vt...
            ux     uxx      vx     vxx...
             n       x      e4    ndss
%
% One vector to two vectors
  for i=1:n
    u(i)=y(i);
    v(i)=y(i+n);
  end
```

2. BCs (4.3a) and (4.3b) at $x = 0$ are then set.

```
%
% Boundary conditions at x = 0
  u(1)=1.0;
  ut(1)=0.0;
  v(1)=1.0;
  vt(1)=0.0;
```

Note that the derivatives in t, `ut(1)` and `vt(1)`, are set to zero to maintain the constant values of `u(1)` and `v(1)` at the boundary $x = 0$.

3. The first derivatives in x are computed by calls to `dss002` for second-order FDs or `dss004` for fourth-order FDs.

```
%
% First order spatial derivatives
  xl=x(1);
  xu=x(n);
%
% Three point centered differences
  if(ncase==1)ux=dss002(xl,xu,n,u); end
  if(ncase==1)vx=dss002(xl,xu,n,v); end
%
% Five point centered differences
  if(ncase==2)ux=dss004(xl,xu,n,u); end
  if(ncase==2)vx=dss004(xl,xu,n,v); end
```

4. BCs (4.3c) and (4.3d) are applied to give the first-order derivatives at $x = 1$.

```
%
% Boundary conditions at x = 1
  ux(n)=3.0-3.0*u(n);
  vx(n)=e4*(u(n)-1.0)/5.0;
```

5. The second-order derivatives in x are then computed by differentiating the first-order derivatives, that is, using *stagewise differentiation*.

```
%
% Second order spatial derivatives
%
% Three point centered  differences
```

```
  if(ncase==1)vxx=dss002(xl,xu,n,vx); end
%
% Five point centered differences
  if(ncase==2)vxx=dss004(xl,xu,n,vx); end
%
% Array vx is used as temporary storage in the calculation
% of the term ((v - 1)*u )  which is finally stored in
% array uxx             x x
  for i=1:n
    vx(i)=(v(i)-1.0)*ux(i);
  end
  if(ncase==1)uxx=dss002(xl,xu,n,vx); end
  if(ncase==2)uxx=dss004(xl,xu,n,vx); end
```

Note in particular how the nonlinear term $((v-1)u_x)_x$ in Eq. (4.1a) is computed.

6. Finally, the two PDEs, Eqs. (4.1a) and (4.1b), are programmed to give the derivative vectors `ut` and `vt`, which are in turn stored in the single-derivative array `yt`.

```
%
% Two PDEs
  for i=2:n
    ex=exp(-4.0*x(i));
      ut(i)=uxx(i)+(16.0*x(i)*t-2.0*t-16.0*(v(i)-1.0))* ...
            (u(i)-1.0)+10.*x(i)*ex;vt(i)=vxx(i)+ux(i)+4.0* ...
            u(i)-4.0+x(i)^2-2.0*t-10.0*t*ex;
  end
%
% Two vectors to one vector
  for i=1:n
    yt(i)   =ut(i);
    yt(i+n) =vt(i);
  end
  yt=yt';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Note in particular that `pde_1` has the input vector `y` and returns the derivative vector `yt` in accordance with the requirements of the ODE integrator `ode15s`.

This completes the MOL programming of Eqs. (4.1)–(4.3). Part of the numerical output from `pde_1_main` is given in Table 4.1.

Table 4.1. Abbreviated numerical output from `pde_1_main` and `pde_1`

```
 ncase =  1

 u =  numerical u(x,t)
ua = analytical u(x,t)
diff u = u - ua
 v =  numerical v(x,t)
va = analytical v(x,t)
diff v = v - uv

   t        x         u        ua       diff u
                      v        va       diff v
0.00     0.000    1.0000   1.0000    0.00e+000
                  1.0000   1.0000    0.00e+000

0.00     0.250    1.0000   1.0000    0.00e+000
                  1.0000   1.0000    0.00e+000

0.00     0.500    1.0000   1.0000    0.00e+000
                  1.0000   1.0000    0.00e+000

0.00     0.750    1.0000   1.0000    0.00e+000
                  1.0000   1.0000    0.00e+000

0.00     1.000    1.0000   1.0000    0.00e+000
                  1.0000   1.0000    0.00e+000

   t        x         u        ua       diff u
                      v        va       diff v
0.05     0.000    1.0000   1.0000    0.00e+000
                  1.0000   1.0000    0.00e+000

0.05     0.250    1.0460   1.0460   -7.26e-007
                  1.0031   1.0031    1.46e-005

0.05     0.500    1.0338   1.0338   -2.42e-007
                  1.0125   1.0125    6.58e-006

0.05     0.750    1.0187   1.0187   -3.14e-007
                  1.0281   1.0281    2.07e-006

0.05     1.000    1.0092   1.0092    2.81e-007
                  1.0500   1.0500    1.34e-006
  .                                     .
  .                                     .
  .                                     .
```

```
   t        x          u        ua      diff u
                       v        va      diff v
0.95    0.000     1.0000    1.0000    0.00e+000
                  1.0000    1.0000    0.00e+000

0.95    0.250     1.8714    1.8737   -2.34e-003
                  1.0598    1.0594    3.96e-004

0.95    0.500     1.6418    1.6428   -1.06e-003
                  1.2377    1.2375    1.93e-004

0.95    0.750     1.3543    1.3547   -4.71e-004
                  1.5342    1.5344   -1.51e-004

0.95    1.000     1.1738    1.1740   -1.68e-004
                  1.9494    1.9500   -5.64e-004

   t        x          u        ua      diff u
                       v        va      diff v
1.00    0.000     1.0000    1.0000    0.00e+000
                  1.0000    1.0000    0.00e+000

1.00    0.250     1.9171    1.9197   -2.58e-003
                  1.0629    1.0625    4.05e-004

1.00    0.500     1.6755    1.6767   -1.15e-003
                  1.2502    1.2500    1.69e-004

1.00    0.750     1.3729    1.3734   -5.02e-004
                  1.5623    1.5625   -2.12e-004

1.00    1.000     1.1830    1.1832   -1.79e-004
                  1.9993    2.0000   -6.57e-004
 ncall =   453


 ncase =  2

 u =  numerical u(x,t)
ua = analytical u(x,t)
diff u = u - ua
 v =  numerical v(x,t)
va = analytical v(x,t)
diff v = v - uv
```

(continued)

Table 4.1 (*continued*)

```
   t        x         u         ua      diff u
                      v         va      diff v
0.00    0.000    1.0000    1.0000   0.00e+000
                 1.0000    1.0000   0.00e+000

0.00    0.250    1.0000    1.0000   0.00e+000
                 1.0000    1.0000   0.00e+000

0.00    0.500    1.0000    1.0000   0.00e+000
                 1.0000    1.0000   0.00e+000

0.00    0.750    1.0000    1.0000   0.00e+000
                 1.0000    1.0000   0.00e+000

0.00    1.000    1.0000    1.0000   0.00e+000
                 1.0000    1.0000   0.00e+000

   t        x         u         ua      diff u
                      v         va      diff v
0.05    0.000    1.0000    1.0000   0.00e+000
                 1.0000    1.0000   0.00e+000

0.05    0.250    1.0460    1.0460   1.45e-008
                 1.0031    1.0031  -5.22e-008

0.05    0.500    1.0338    1.0338   1.38e-009
                 1.0125    1.0125  -2.82e-008

0.05    0.750    1.0187    1.0187   4.29e-010
                 1.0281    1.0281  -7.98e-009

0.05    1.000    1.0092    1.0092   3.89e-009
                 1.0500    1.0500   2.87e-009
  .                                     .
  .                                     .
  .                                     .

0.95    0.250    1.8737    1.8737   2.76e-005
                 1.0594    1.0594   1.84e-006

0.95    0.500    1.6429    1.6428   1.39e-005
                 1.2375    1.2375   1.24e-006

0.95    0.750    1.3547    1.3547   9.96e-006
                 1.5344    1.5344   1.25e-007
```

```
0.95    1.000   1.1740   1.1740  -1.03e-006
                1.9500   1.9500   2.35e-006

   t        x        u       ua      diff u
                     v       va      diff v
1.00    0.000   1.0000   1.0000   0.00e+000
                1.0000   1.0000   0.00e+000

1.00    0.250   1.9197   1.9197   2.97e-005
                1.0625   1.0625   2.30e-006

1.00    0.500   1.6767   1.6767   1.51e-005
                1.2500   1.2500   1.98e-006

1.00    0.750   1.3734   1.3734   1.13e-005
                1.5625   1.5625   9.03e-007

1.00    1.000   1.1832   1.1832  -1.30e-006
                2.0000   2.0000   3.04e-006
 ncall =   363
```

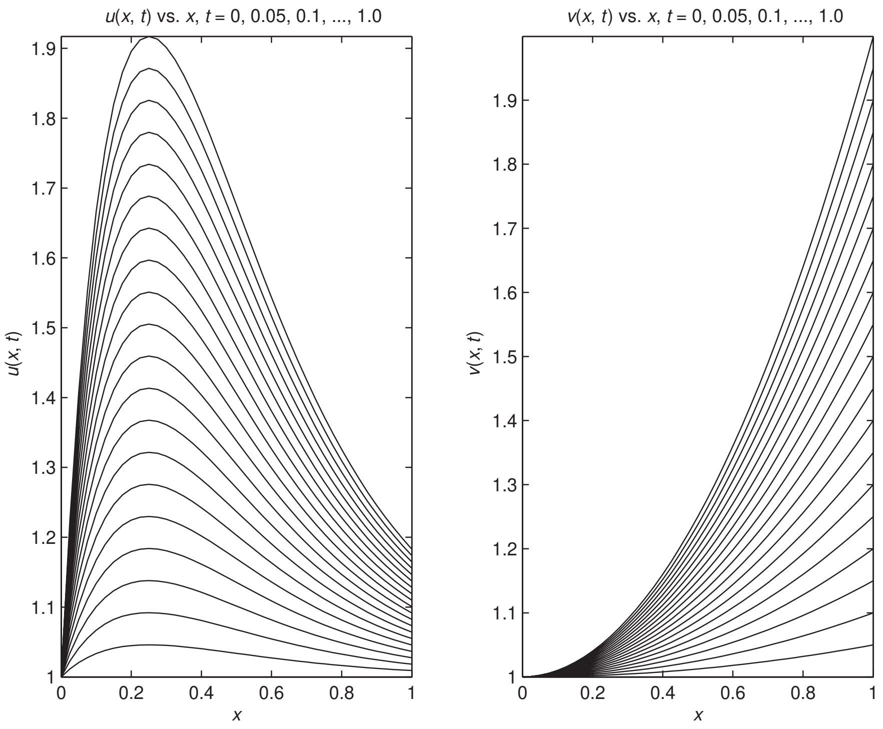

Figure 4.1. $u(x, t)$ and $v(x, t)$ profiles in x with t as a parameter

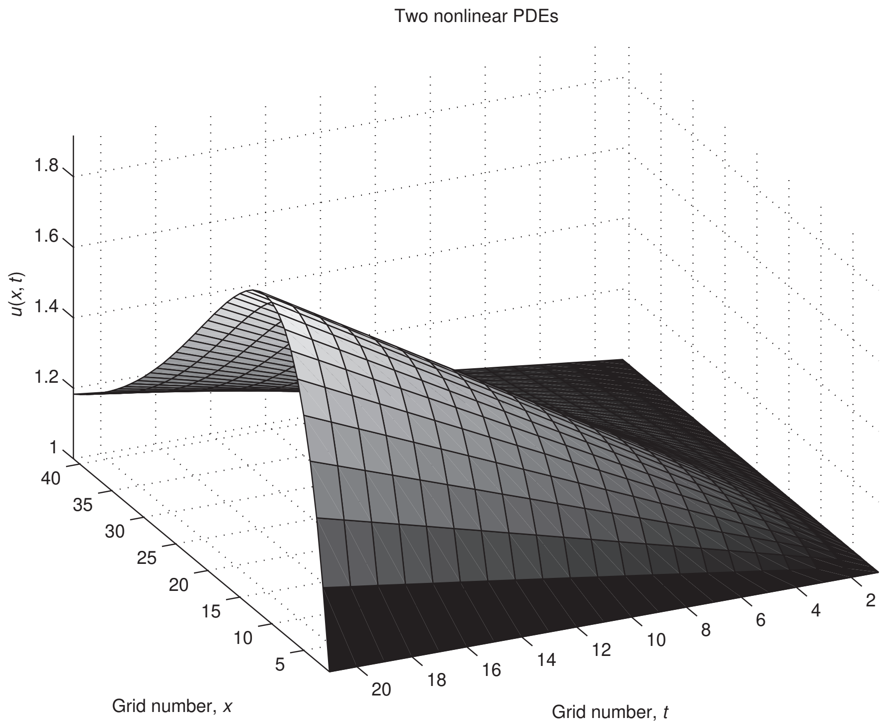

Figure 4.2. $u(x, t)$ in 3D perspective

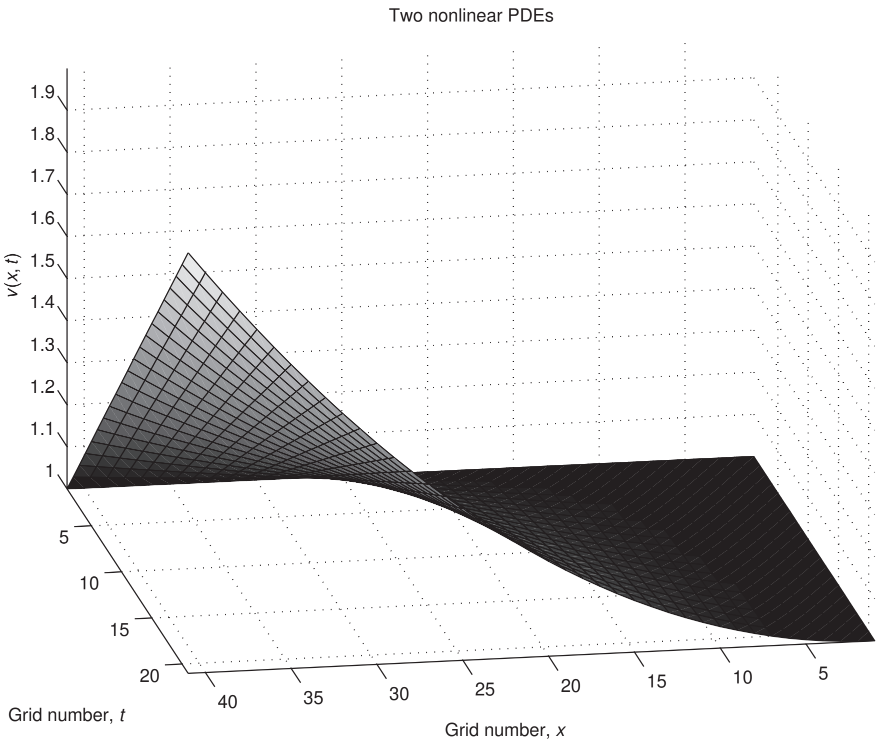

Figure 4.3. $v(x, t)$ in 3D perspective

We can note the following about the output provided in Table 4.1:

1. The agreement between the numerical and analytical solutions is better for the fourth-order FD approximations (`ncase = 2`).
2. The number of calls to `pde_1` is quite modest, and in fact, the fourth-order FDs required fewer (`ncall = 363`) than the second-order FDs (`ncall = 453`) so that for this problem at least, a more accurate solution was achieved with less computational effort.

The 2D plotted output from `pde_1_main` is shown in Figure 4.1. The solutions in Figure 4.1 are further elucidated by the 3D plots in Figures 4.2 and 4.3 (from `pde_1_main`).

In summary, this application based on Eqs. (4.1)–(4.3) demonstrates the applicability of the MOL to a system of PDEs with a variety of mathematical properties that are plainly programmed in `pde_1`.

REFERENCE

[1] Madsen, N. K. and R. F. Sincovec (1976), Software for Partial Differential Equations, In: L. Lapidus and W. E. Schiesser (Eds.), *Numerical Methods for Differential Systems*, Academic Press, New York, pp. 229–242