5

Euler, Navier Stokes, and Burgers Equations

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

1. We start with a general PDE system in three dimensions (3D), with some simplifying assumptions, reduces to a 1D nonlinear PDE. Here is some terminology applied to this starting point and the final result:
   (a) We start with two classic (we might say without exaggeration "famous") PDE systems: the *Euler equations* and the *Navier Stokes equations* of fluid mechanics.
   (b) After reduction to one dimension, followed by some additional simplifications, we arrive at the *Burgers equation*, a PDE widely used as a test problem for numerical methods.
2. The analysis is in terms of coordinate-free PDEs that can then be specialized to a particular coordinate system; for the following analysis, this is Cartesian coordinates.
3. The 1D nonlinear PDE, Burgers' equation, has an analytical solution that can be used to test the numerical method of lines (MOL) algorithms developed and implemented in a series of computer routines.
4. The analytical solution is a traveling wave that has an increasingly steep moving front, usually termed *front sharpening*. The conditions under which front sharpening might occur are analyzed briefly.
5. Spatial convergence of the Burgers equation numerical solution is investigated by h- and p-refinement.
6. Stagewise differentiation is used to calculate higher-order spatial derivatives.
7. Dirichlet and Neumann boundary conditions are investigated as two cases of the numerical solution.

The starting point for the analysis is the *multidimensional, conservation equations written in conservative form*:

$$\mathbf{u}_t + \nabla \cdot [\mathbf{f}(\mathbf{u})] = \mathbf{0} \tag{5.1}$$

where **u** is the *vector of conserved quantities*, which typically includes mass, momentum, and/or energy. The terms in Eq. (5.1) represent

$\mathbf{u}_t$	time rate of accumulation (or depletion, depending on the sign of this term) of the conserved quantity
$\nabla \cdot [\mathbf{f}(\mathbf{u})]$	net flux of the conserved quantity into or out of a differential volume

Here we have used subscript notation for partial derivatives, so that, for example, $u_t = \partial u/\partial t$.

In the case of a simplified fluid mechanics model, **u** in Eq. (5.1) is usually mass and momentum, defined as

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho\mathbf{v} \end{bmatrix}, \quad \mathbf{f}(\mathbf{u}) = \begin{bmatrix} \rho\mathbf{v} \\ \rho\mathbf{v}\mathbf{v} + p \end{bmatrix}$$

Variables in boldface represent a vector or matrix; for example, **v** is a velocity vector (introductory discussion of vector/tensor notation is given in Appendix 1). Substitution of **u** and **f**(**u**) into Eq. (5.1) gives the Euler equations

$$\rho_t + \nabla \cdot (\rho\mathbf{v}) = 0 \qquad \text{continuity} \tag{5.2a}$$

$$(\rho\mathbf{v})_t + \nabla \cdot (\rho\mathbf{v}\mathbf{v}) + \nabla p = 0 \qquad \text{momentum} \tag{5.2b}$$

where ∇ is a differential operator applied to the mass vector $\rho\mathbf{v}$ in the continuity equation, and to a scalar (pressure, p) and a tensor ($\rho\mathbf{v}\mathbf{v}$) in the momentum equation. Considering ∇ a bit further, with the dot $\cdot$ it forms the usual dot product with a vector (and the result is a scalar), while without the dot, it operates on a scalar to give a vector, or on a tensor as explained subsequently.

Also, ∇ is a *coordinate-free operator*; that is, it can be expressed in any *3D orthogonal coordinate system*. For example, in *Cartesian coordinates*, ∇ is

$$\nabla = \mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}$$

Since the fluid density ρ is a scalar, and with $\mathbf{v} = \mathbf{i}v_x + \mathbf{j}v_y + \mathbf{k}v_z$,

$$\begin{aligned} \nabla \cdot (\rho\mathbf{v}) &= \left(\mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}\right) \cdot (\mathbf{i}\rho v_x + \mathbf{j}\rho v_y + \mathbf{k}\rho v_z) \\ &= \frac{\partial(\rho v_x)}{\partial x} + \frac{\partial(\rho v_y)}{\partial y} + \frac{\partial(\rho v_z)}{\partial z} \end{aligned}$$

where we have made use of the dot product between orthogonal unit vectors

$$\mathbf{i} \cdot \mathbf{j} = \begin{cases} 0, & \mathbf{i} \neq \mathbf{j} \\ 1, & \mathbf{i} = \mathbf{j} \end{cases}$$

Then, the (scalar) continuity equation is from Eq. (5.2a):

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho v_x)}{\partial x} + \frac{\partial (\rho v_y)}{\partial y} + \frac{\partial (\rho v_z)}{\partial z} = 0 \tag{5.3}$$

Note that for a constant density (incompressible) fluid,

$$\frac{\partial \rho}{\partial t} = 0$$

so that Eq. (5.2a) becomes

$$\frac{\partial (\rho v_x)}{\partial x} + \frac{\partial (\rho v_y)}{\partial y} + \frac{\partial (\rho v_z)}{\partial z} = \rho \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \right) + v_x \frac{\partial \rho}{\partial x} + v_y \frac{\partial \rho}{\partial x} + v_z \frac{\partial \rho}{\partial x} = 0$$

or (with the derivatives of ρ zero),

$$\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} = 0$$

$$\nabla \cdot \mathbf{v} = 0 \tag{5.4}$$

The property of Eq. (5.4) for an incompressible fluid is termed *divergence free*. It can be used to check a numerical solution to a PDE system for an incompressible fluid (a numerical solution should be divergence free).

We now consider the momentum balance of Eq. (5.2), starting with the term

$$\nabla \cdot (\rho \mathbf{v}\mathbf{v})$$

The product $\mathbf{vv}$ is actually a second-order (nine-component) tensor. Thus,

$$\rho \mathbf{v}\mathbf{v} = \begin{bmatrix} \rho \mathbf{i} v_x \mathbf{i} v_x & \rho \mathbf{i} v_x \mathbf{j} v_y & \rho \mathbf{i} v_x \mathbf{k} v_z \\ \rho \mathbf{j} v_y \mathbf{i} v_x & \rho \mathbf{j} v_y \mathbf{j} v_y & \rho \mathbf{j} v_y \mathbf{k} v_z \\ \rho \mathbf{k} v_z \mathbf{i} v_x & \rho \mathbf{k} v_z \mathbf{j} v_y & \rho \mathbf{k} v_z \mathbf{k} v_z \end{bmatrix}$$

and applying $\nabla\cdot$ to this tensor,

$$\nabla \cdot (\rho \mathbf{v}\mathbf{v}) = \begin{bmatrix} \mathbf{i} \left\{ \frac{\partial}{\partial x}(\rho v_x v_x) + \frac{\partial}{\partial y}(\rho v_x v_y) + \frac{\partial}{\partial z}(\rho v_x v_z) \right\} \\ \mathbf{j} \left\{ \frac{\partial}{\partial x}(\rho v_y v_x) + \frac{\partial}{\partial y}(\rho v_y v_y) + \frac{\partial}{\partial z}(\rho v_y v_z) \right\} \\ \mathbf{k} \left\{ \frac{\partial}{\partial x}(\rho v_z v_x) + \frac{\partial}{\partial y}(\rho v_z v_y) + \frac{\partial}{\partial z}(\rho v_z v_z) \right\} \end{bmatrix}$$

Then, using this result (a vector) in the momentum equation,

$$(\rho \mathbf{v})_t + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) + \nabla p = 0$$

and equating corresponding components,

$$\frac{\partial}{\partial t}(\rho v_x) + \frac{\partial}{\partial x}(\rho v_x v_x) + \frac{\partial}{\partial y}(\rho v_x v_y) + \frac{\partial}{\partial z}(\rho v_x v_z) + \frac{\partial p}{\partial x} = 0 \ (\mathbf{i} \text{ component}) \quad (5.5a)$$

$$\frac{\partial}{\partial t}(\rho v_y) + \frac{\partial}{\partial x}(\rho v_y v_x) + \frac{\partial}{\partial y}(\rho v_y v_y) + \frac{\partial}{\partial z}(\rho v_y v_z) + \frac{\partial p}{\partial y} = 0 \ (\mathbf{j} \text{ component}) \quad (5.5b)$$

$$\frac{\partial}{\partial t}(\rho v_z) + \frac{\partial}{\partial x}(\rho v_z v_x) + \frac{\partial}{\partial y}(\rho v_z v_y) + \frac{\partial}{\partial z}(\rho v_z v_z) + \frac{\partial p}{\partial z} = 0 \ (\mathbf{k} \text{ component}) \quad (5.5c)$$

Equations (5.5a)–(5.5c) have several noteworthy properties. They are

1. Hyperbolic (first order in x, y, z, and t)
2. A source of sharp moving fronts and discontinuities (which are characteristic of hyperbolic PDEs)
3. Highly nonlinear (consider the products of the velocity components, e.g., $v_x v_x$, in Eqs. (5.5a)–(5.5c)
4. Difficult to solve numerically (and generally impossible analytically)
5. Limited in the sense that they apply only to isothermal, nonreacting systems

Equations (5.5a)–(5.5c) can be simplified through continuity equation (5.3); for example Eq. (5.5a) can be written as

$$\begin{aligned} &v_x \frac{\partial}{\partial t}(\rho) + v_x \frac{\partial}{\partial x}(\rho v_x) + v_x \frac{\partial}{\partial y}(\rho v_y) + v_x \frac{\partial}{\partial z}(\rho v_z) \\ &\quad + \rho \frac{\partial}{\partial t}(v_x) + \rho v_x \frac{\partial}{\partial x}(v_x) + \rho v_y \frac{\partial}{\partial y}(v_x) + \rho v_z \frac{\partial}{\partial z}(v_x) + \frac{\partial p}{\partial x} = 0 \end{aligned}$$

or

$$\begin{aligned} &v_x \left\{ \frac{\partial \rho}{\partial t} + \frac{\partial(\rho v_x)}{\partial x} + \frac{\partial(\rho v_y)}{\partial y} + \frac{\partial(\rho v_z)}{\partial z} \right\} \\ &\quad + \rho \frac{\partial v_x}{\partial t} + \rho v_x \frac{\partial v_x}{\partial x} + \rho v_y \frac{\partial v_x}{\partial y} + \rho v_z \frac{\partial v_x}{\partial z} + \frac{\partial p}{\partial x} = 0 \end{aligned}$$

and the first row is zero through the continuity equation (5.3).

This result can be generalized to

$$\rho \mathbf{v}_t + \rho \mathbf{v} \cdot \nabla \mathbf{v} + \nabla p = 0$$

which is an alternate form of the Euler equations.

Finally, if a Newtonian viscosity term is added to the momentum equation

$$(\rho \mathbf{v})_t + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) + \nabla p - \mu \nabla^2 \mathbf{v} = 0 \tag{5.6}$$

which along with Eq. (5.3) are the *Navier Stokes equations* for incompressible flow where gravitational effects are negligible. ∇^2 is the *Laplacian operator* that in Cartesian coordinates is

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{5.7}$$

Note that ∇^2 is a scalar, and it can be applied to a vector, as in Eq. (5.6), or it can be applied to a scalar, as in the parabolic heat equation $\partial u/\partial t = \nabla^2 u$.

We can use Eq. (5.6) as the starting point for a PDE analysis. For example, simplification of the x component of Eq. (5.6) (combined with Eq. (5.7)) gives

$$\rho \frac{\partial v_x}{\partial t} + \rho v_x \frac{\partial v_x}{\partial x} + \rho v_y \frac{\partial v_x}{\partial y} + \rho v_z \frac{\partial v_x}{\partial z} + \frac{\partial p}{\partial x} - \mu \left(\frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_x}{\partial y^2} + \frac{\partial^2 v_x}{\partial z^2} \right) = 0 \tag{5.8}$$

If we consider a 1D problem with no pressure gradient, Eq. (5.8) reduces to

$$\rho \frac{\partial v_x}{\partial t} + \rho v_x \frac{\partial v_x}{\partial x} - \mu \frac{\partial^2 v_x}{\partial x^2} = 0 \tag{5.9}$$

which is *Burgers' equation*. This is an unusual (and widely studied) PDE because

1. It is nonlinear, yet has a known, exact (analytical) solution.
2. The solution exhibits moving fronts that can be made arbitrarily sharp by decreasing the *kinematic viscosity* $\nu = \mu/\rho$ (Eq. (5.9) can be divided by ρ to give the kinematic viscosity as a coefficient of the second-derivative term).

Burgers' equation for $\nu = \mu/\rho = 0$ reduces to the *inviscid Burgers equation* or *nonlinear advection equation*:

$$\frac{\partial v_x}{\partial t} + v_x \frac{\partial v_x}{\partial x} = 0 \tag{5.10}$$

We now consider the numerical solution of Burgers' equation (5.9) through a series of MOL routines. To evaluate the numerical solution, we will use an analytical solution to Eq. (5.9), v_{xa} [1].

$$v_{xa}(x,t) = \frac{0.1\, e^{-A} + 0.5\, e^{-B} + e^{-C}}{e^{-A} + e^{-B} + e^{-C}} \tag{5.11}$$

where

$$A = \frac{0.05}{\nu}(x - 0.5 + 4.95t) \tag{5.12a}$$

$$B = \frac{0.25}{\nu}(x - 0.5 + 0.75t) \tag{5.12b}$$

$$C = \frac{0.5}{\nu}(x - 0.375) \tag{5.12c}$$

For the initial condition (IC) and boundary conditions (BCs) for Eq. (5.9), we use Eqs. (5.11) and (5.12).

$$v_x(t = 0, x) = v_{xa}(t = 0, x) \tag{5.13a}$$

$$v_x(t, x = 0) = v_{xa}(t, x = 0) \tag{5.13b}$$

$$v_x(t, x = 1) = v_{xa}(t, x = 1) \tag{5.13c}$$

Note from Eqs. (5.13b) and (5.13c) $0 \leq x \leq 1$. For the numerical solution, we take $\nu = 0.003$, which, as we shall observe, produces a solution with a *steep moving front that sharpens with increasing t*.

A main program for the solution of Eq. (5.9) is given in Listing 5.1 (in the subsequent programming, we use the traditional variable u as the PDE dependent variable rather than v_x of Eqs. (5.10) and (5.13), and $\phi(x, t)$ (= `phi`) is the analytical solution of Eq. (5.11)).

```
%
% Clear previous files
  clear all
  clc
%
% Parameters shared with the ODE routine
  global n x xl xu ncall ncase ndss vis
%
% Select case
%
% ncase = 1 - analytical solution used in BCs
%
% ncase = 2 - homogeneous Neumann BCs
  ncase=1;
%
% Parameters
  n=201;
  xl=0.0;
  xu=1.0;
  vis=0.003;
%
% Initial condition
  t0=0.0;
  u0=inital_1(t0);
%
% Independent variable for ODE integration
```

```
  tf=1.0;
  tout=[t0:0.1:tf]';
  nout=11;
  ncall=0;
%
% ODE integration
  reltol=1.0e-04; abstol=1.0e-04;
  options=odeset('RelTol',reltol,'AbsTol',abstol);
  if(ncase==1)
    ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode15s(@pde_1,tout,u0,options); end
  if(ncase==2)
    ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode15s(@pde_2,tout,u0,options); end
%
% Store analytical solution, errors
  for it=1:nout
    for i=1:n
      u_anal(it,i)=phi(x(i),t(it));
      err(it,i)=u(it,i)-u_anal(it,i);
    end
  end
%
% Display selected output
  fprintf('\n vis = %8.4f   ndss = %2d\n\n',vis,ndss);
  for it=1:2:nout
    fprintf('   it   i   t(it)   x(i)       u(it,i)
            u_anal(it,i)    err(it,i)\n');
    for i=1:5:n
      fprintf('%5d%5d%8.2f%8.3f%15.6f%15.6f%15.6f\n', ...
              it,i,t(it),x(i),u(it,i),u_anal(it,i),err(it,i));
    end
    fprintf('\n');
  end
  fprintf('  ncall = %4d\n\n',ncall);
%
% Plot numerical and analytical solutions
  plot(x,u,'o',x,u_anal,'-')
  xlabel('x')
  ylabel('u(x,t)')
  title('Burgers equation; t = 0, 0.1, ..., 1;
        o - numerical; solid - analytical')
% print -deps -r300 pde.eps; print -dps -r300 pde.ps;
% print -dpng -r300 pde.png
```

Listing 5.1. Main program `pde_1_main.m` for the solution of Eqs. (5.10)–(5.13)

We can note the following points about this main program:

1. After defining a *global* area by which variables and parameters can be shared with other routines, a case for the solution is selected; for `ncase=1`, analytical solution (5.11) is used to define the BCs in the ordinary differential equation (ODE) routine `pde_1` (discussed subsequently).

```
%
% Clear previous files
  clear all
  clc
%
% Parameters shared with the ODE routine
  global n x xl xu ncall ncase ndss vis
%
% Select case
%
% ncase = 1 - analytical solution used in BCs
%
% ncase = 2 - homogeneous Neumann BCs
  ncase=1;
```

2. A grid in x of 201 points, over the interval $0 \leq x \leq 1$, is defined and the kinematic viscosity in Eq. (5.9) set as $\nu = \mu/\rho = 0.003$ specifies a solution that has a steep front, as demonstrated subsequently.

```
%
% Parameters
  n=201;
  xl=0.0;
  xu=1.0;
  vis=0.003;
```

3. The IC for Eq. (5.9) is taken as the analytical solution, Eq. (5.11), with $t = 0$. This IC is in function `inital_1`, discussed subsequently.

```
%
% Initial condition
  t0=0.0;
  u0=inital_1(t0);
%
% Independent variable for ODE integration
```

```
tf=1.0;
tout=[t0:0.1:tf]';
nout=11;
ncall=0;
```

The time variation is then defined over the interval $0 \leq t \leq 1$ with an output interval of 0.1 so that there are a total of 11 outputs (counting the IC at $t = 0$).

4. The 201 ODEs are integrated by the stiff integrator `ode15s` for `ncase=1` or 2. These cases are explained when the corresponding ODE routines `pde_1` and `pde_2` are discussed subsequently.

```
%
% ODE integration
  reltol=1.0e-04; abstol=1.0e-04;
  options=odeset('RelTol',reltol,'AbsTol',abstol);
  if(ncase==1)
    ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode15s(@pde_1,tout,u0,options); end
  if(ncase==2)
    ndss=4; % ndss = 2, 4, 6, 8 or 10 required
    [t,u]=ode15s(@pde_2,tout,u0,options); end
```

The spatial derivatives in Eq. (5.9) are computed by routine `dss004` as specified by `ndss=4` (passed as a global variable to the ODE routines `pde_1`, `pde_2`).

5. The analytical solution, Eq. (5.11), is then evaluated via function `phi` (discussed subsequently), and the difference between the numerical and analytical solutions is computed for subsequent output.

```
%
% Store analytical solution, errors
  for it=1:nout
    for i=1:n
      u_anal(it,i)=phi(x(i),t(it));
      err(it,i)=u(it,i)-u_anal(it,i);
    end
  end
```

6. Selected portions of the numerical and analytical solutions are displayed in tabular form, and these two solutions are plotted so that they can be compared graphically.

```
%
% Display selected output
  fprintf('\n vis = %8.4f   ndss = %2d\n\n',vis,ndss);
  for it=1:2:nout
    fprintf('  it   i   t(it)   x(i)      u(it,i)
           u_anal(it,i)    err(it,i)\n');
    for i=1:5:n
      fprintf('%5d%5d%8.2f%8.3f%15.6f%15.6f%15.6f\n',...
            it,i,t(it),x(i),u(it,i),u_anal(it,i),err(it,i));
    end
    fprintf('\n');
  end
  fprintf('  ncall = %4d\n\n',ncall);
%
% Plot numerical and analytical solutions
  plot(x,u,'o',x,u_anal,'-')
  xlabel('x')
  ylabel('u(x,t)')
  title('Burgers equation; t = 0, 0.1, ..., 1;
       o - numerical; solid - analytical')
% print -deps -r300 pde.eps; print -dps -r300 pde.ps;
% print -dpng -r300 pde.png
```

We briefly review the output before going on to discussions of the routines `inital_1, pde_1, pde_2, phi`. A portion of the output (for `ncase=1`) is given in Table 5.1.

We can note the following details about the output is given in Table 5.1:

1. The IC (at $t = 0$) is the same for the numerical and analytical solutions as expected (since the analytical solution, Eq. (5.11), is used to generate the numerical solution IC).
2. The IC has an important feature. In moving left to right (increasing x), the IC decreases (from 1 to 0.1). This decreasing value of $u(x, t = 0)$ with increasing x is the *condition that leads to front sharpening* for subsequent t. This can be noted in the output for $t = 1$ that has a considerably sharper front than at $t = 0$. In fact, if t were extended to higher values (than $t = 1$), a *shock or discontinuity would eventually develop*. This is one of the principal reasons why Burgers' equation (5.9) is a stringent test problem; that is, the solution steepens and therefore becomes increasingly difficult to resolve spatially (the accurate resolution of the solution $u(x, t)$ as a function of x as t increases becomes progressively more difficult).
3. For the spatial grid of 201 points, the agreement between the numerical and analytical solutions is quite satisfactory, and remains that way throughout the solution, for example, from $t = 0.2$ to $t = 1$. This is an important point since it indicates that *numerical errors did not grow or accumulate with increasing t*.

Table 5.1. Selected output from `pde_1_main` for `ncase=1`

```
vis =   0.0030   ndss =  4

 it     i   t(it)     x(i)       u(it,i)    u_anal(it,i)     err(it,i)
  1     1    0.00    0.000      1.000000       1.000000      0.000000
  1     6    0.00    0.025      1.000000       1.000000      0.000000
  1    11    0.00    0.050      1.000000       1.000000      0.000000
  1    16    0.00    0.075      1.000000       1.000000      0.000000
  1    21    0.00    0.100      0.999998       0.999998      0.000000
  1    26    0.00    0.125      0.999985       0.999985      0.000000
  1    31    0.00    0.150      0.999880       0.999880      0.000000
  1    36    0.00    0.175      0.999037       0.999037      0.000000
  1    41    0.00    0.200      0.992366       0.992366      0.000000
  1    46    0.00    0.225      0.944636       0.944636      0.000000
  1    51    0.00    0.250      0.750000       0.750000      0.000000
  1    56    0.00    0.275      0.555364       0.555364      0.000000
  1    61    0.00    0.300      0.507633       0.507633      0.000000
  1    66    0.00    0.325      0.500960       0.500960      0.000000
  1    71    0.00    0.350      0.500102       0.500102      0.000000
  1    76    0.00    0.375      0.499919       0.499919      0.000000
  1    81    0.00    0.400      0.499493       0.499493      0.000000
  1    86    0.00    0.425      0.497323       0.497323      0.000000
  1    91    0.00    0.450      0.486222       0.486222      0.000000
  1    96    0.00    0.475      0.436452       0.436452      0.000000
  1   101    0.00    0.500      0.300000       0.300000      0.000000
  1   106    0.00    0.525      0.163548       0.163548      0.000000
  1   111    0.00    0.550      0.113778       0.113778      0.000000
  1   116    0.00    0.575      0.102677       0.102677      0.000000
  1   121    0.00    0.600      0.100508       0.100508      0.000000
  1   126    0.00    0.625      0.100096       0.100096      0.000000
  1   131    0.00    0.650      0.100018       0.100018      0.000000
  1   136    0.00    0.675      0.100003       0.100003      0.000000
  1   141    0.00    0.700      0.100001       0.100001      0.000000
  1   146    0.00    0.725      0.100000       0.100000      0.000000
  1   151    0.00    0.750      0.100000       0.100000      0.000000
  1   156    0.00    0.775      0.100000       0.100000      0.000000
  1   161    0.00    0.800      0.100000       0.100000      0.000000
  1   166    0.00    0.825      0.100000       0.100000      0.000000
  1   171    0.00    0.850      0.100000       0.100000      0.000000
  1   176    0.00    0.875      0.100000       0.100000      0.000000
  1   181    0.00    0.900      0.100000       0.100000      0.000000
  1   186    0.00    0.925      0.100000       0.100000      0.000000
  1   191    0.00    0.950      0.100000       0.100000      0.000000
  1   196    0.00    0.975      0.100000       0.100000      0.000000
  1   201    0.00    1.000      0.100000       0.100000      0.000000
```

it	i	t(it)	x(i)	u(it,i)	u_anal(it,i)	err(it,i)
3	1	0.20	0.000	1.000000	1.000000	-0.000000
3	6	0.20	0.025	1.000000	1.000000	0.000000
3	11	0.20	0.050	1.000000	1.000000	-0.000000
3	16	0.20	0.075	1.000000	1.000000	-0.000000
3	21	0.20	0.100	1.000000	1.000000	-0.000000
3	26	0.20	0.125	1.000000	1.000000	-0.000000
3	31	0.20	0.150	1.000000	1.000000	-0.000000
3	36	0.20	0.175	1.000000	1.000000	-0.000000
3	41	0.20	0.200	1.000000	1.000000	-0.000000
3	46	0.20	0.225	1.000000	1.000000	-0.000000
3	51	0.20	0.250	0.999998	0.999998	-0.000000
3	56	0.20	0.275	0.999985	0.999985	-0.000000
3	61	0.20	0.300	0.999877	0.999880	-0.000003
3	66	0.20	0.325	0.999015	0.999037	-0.000022
3	71	0.20	0.350	0.992269	0.992366	-0.000097
3	76	0.20	0.375	0.944801	0.944636	0.000165
3	81	0.20	0.400	0.749763	0.749992	-0.000229
3	86	0.20	0.425	0.555319	0.555314	0.000005
3	91	0.20	0.450	0.507447	0.507371	0.000075
3	96	0.20	0.475	0.499597	0.499584	0.000013
3	101	0.20	0.500	0.492921	0.492925	-0.000004
3	106	0.20	0.525	0.464658	0.464655	0.000003
3	111	0.20	0.550	0.364335	0.364304	0.000031
3	116	0.20	0.575	0.207536	0.207577	-0.000041
3	121	0.20	0.600	0.126005	0.125988	0.000017
3	126	0.20	0.625	0.105184	0.105181	0.000003
3	131	0.20	0.650	0.100989	0.100989	-0.000000
3	136	0.20	0.675	0.100187	0.100187	-0.000000
3	141	0.20	0.700	0.100035	0.100035	-0.000000
3	146	0.20	0.725	0.100007	0.100007	-0.000000
3	151	0.20	0.750	0.100001	0.100001	-0.000000
3	156	0.20	0.775	0.100000	0.100000	-0.000000
3	161	0.20	0.800	0.100000	0.100000	-0.000000
3	166	0.20	0.825	0.100000	0.100000	-0.000000
3	171	0.20	0.850	0.100000	0.100000	-0.000000
3	176	0.20	0.875	0.100000	0.100000	-0.000000
3	181	0.20	0.900	0.100000	0.100000	-0.000000
3	186	0.20	0.925	0.100000	0.100000	-0.000000
3	191	0.20	0.950	0.100000	0.100000	-0.000000
3	196	0.20	0.975	0.100000	0.100000	-0.000000
3	201	0.20	1.000	0.100000	0.100000	-0.000000
		.				.
		.				.
		.				.

(*continued*)

Table 5.1 (*continued*)

```
                  Output at t = 0.4, 0.6, 0.8 removed
             .                                          .
             .                                          .
             .                                          .

 it     i   t(it)     x(i)      u(it,i)   u_anal(it,i)    err(it,i)
 11     1    1.00    0.000     1.000049       1.000000     0.000049
 11     6    1.00    0.025     0.999999       1.000000    -0.000001
 11    11    1.00    0.050     1.000000       1.000000    -0.000000
 11    16    1.00    0.075     1.000000       1.000000    -0.000000
 11    21    1.00    0.100     1.000000       1.000000     0.000000
 11    26    1.00    0.125     1.000000       1.000000    -0.000000
 11    31    1.00    0.150     1.000000       1.000000     0.000000
 11    36    1.00    0.175     1.000000       1.000000    -0.000000
 11    41    1.00    0.200     1.000000       1.000000    -0.000000
 11    46    1.00    0.225     1.000000       1.000000     0.000000
 11    51    1.00    0.250     1.000000       1.000000    -0.000000
 11    56    1.00    0.275     1.000000       1.000000     0.000000
 11    61    1.00    0.300     1.000000       1.000000    -0.000000
 11    66    1.00    0.325     1.000000       1.000000     0.000000
 11    71    1.00    0.350     1.000000       1.000000    -0.000000
 11    76    1.00    0.375     1.000000       1.000000     0.000000
 11    81    1.00    0.400     1.000000       1.000000    -0.000000
 11    86    1.00    0.425     1.000000       1.000000     0.000000
 11    91    1.00    0.450     1.000000       1.000000    -0.000000
 11    96    1.00    0.475     1.000000       1.000000     0.000000
 11   101    1.00    0.500     1.000000       1.000000    -0.000000
 11   106    1.00    0.525     1.000000       1.000000     0.000000
 11   111    1.00    0.550     1.000000       1.000000    -0.000000
 11   116    1.00    0.575     1.000000       1.000000     0.000000
 11   121    1.00    0.600     1.000000       1.000000    -0.000000
 11   126    1.00    0.625     1.000000       1.000000     0.000000
 11   131    1.00    0.650     1.000000       1.000000    -0.000000
 11   136    1.00    0.675     1.000000       1.000000     0.000000
 11   141    1.00    0.700     1.000000       1.000000     0.000000
 11   146    1.00    0.725     1.000000       1.000000     0.000000
 11   151    1.00    0.750     1.000000       1.000000     0.000000
 11   156    1.00    0.775     1.000000       1.000000     0.000000
 11   161    1.00    0.800     1.000001       1.000000     0.000001
 11   166    1.00    0.825     1.000002       0.999998     0.000005
 11   171    1.00    0.850     0.999920       0.999904     0.000015
 11   176    1.00    0.875     0.995958       0.996005    -0.000048
 11   181    1.00    0.900     0.858691       0.856946     0.001746
 11   186    1.00    0.925     0.200038       0.199719     0.000319
 11   191    1.00    0.950     0.102603       0.102646    -0.000043
 11   196    1.00    0.975     0.100059       0.100065    -0.000006
 11   201    1.00    1.000     0.100002       0.100002    -0.000000

 ncall =  728
```

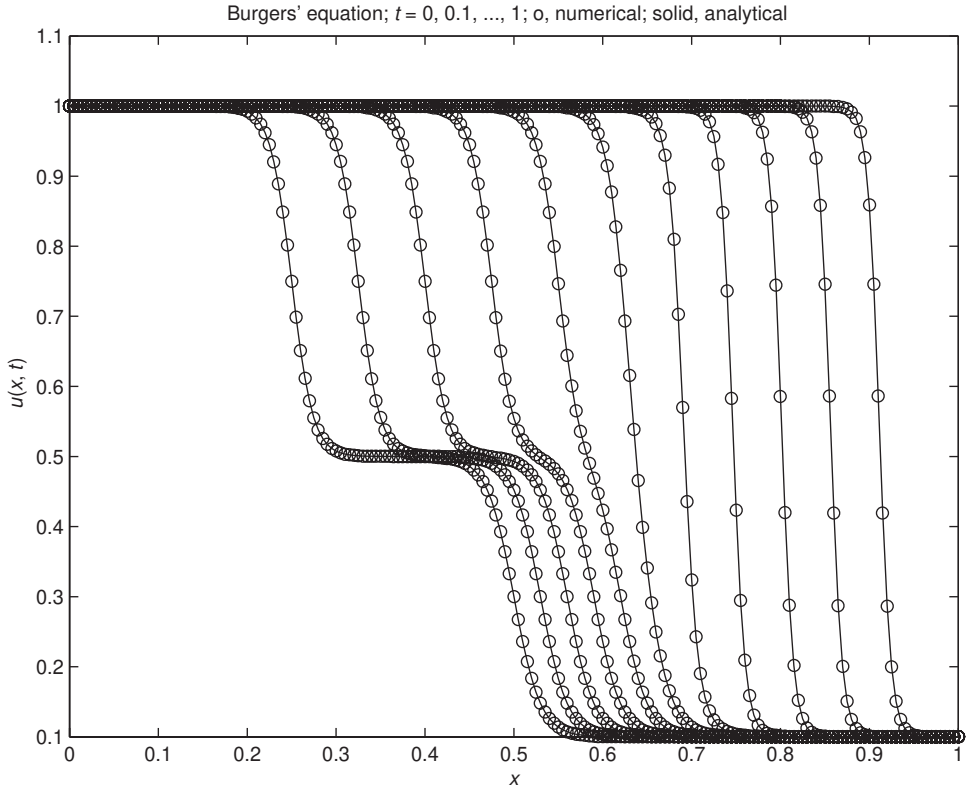

Figure 5.1. Plot of the numerical and analytical solutions of Burgers' equation (5.9) from `pde_1_main` and `pde_1` for `ncase=1`

The agreement between the numerical and analytical solutions is illustrated by the plotted output indicated in Figure 5.1 (for `ncase=1`).

The front sharpening of the solution is evident in Figure 5.1 (the IC at $t = 0$ is the leftmost curve and the rightmost curve is the solution at $t = 1$).

We now discuss the remaining routines to produce the solution of Table 5.1 and Figure 5.1. The IC routine, `inital_1`, is given in Listing 5.2.

```
  function u0=inital_1(t0)
%
% Function inital_1 sets the initial condition for...
% Burgers' equation
%
  global n x xl xu ncall ncase ndss vis
%
% Analytical solution initially
  dx=(xu-xl)/(n-1);
  for i=1:n
```

```
  x(i)=xl+(i-1)*dx;
  u0(i)=phi(x(i),0.0);
end
```

Listing 5.2. `inital_1` for the IC of Eq. (5.9) based on the analytical solution of Eq. (5.11) with $t = 0$

This routine is straightforward. Note that it calls `phi` for analytical solution (5.11) with $t = 0$ over the 201-point grid in x.

Refer to Listing 5.3 for ODE routine `pde_1`.

```
  function ut=pde_1(t,u)
%
% Function pde_1 computes the t derivative vector for
% Burgers' equation
%
  global n x xl xu ncall ncase ndss vis
%
% BC at x = 0
  u(1)=phi(xl,t);
%
% BC at x = 1
  u(n)=phi(xu,t);
%
% ux
  if      (ndss== 2) ux=dss002(xl,xu,n,u); % second order
    elseif(ndss== 4) ux=dss004(xl,xu,n,u); % fourth order
    elseif(ndss== 6) ux=dss006(xl,xu,n,u); % sixth order
    elseif(ndss== 8) ux=dss008(xl,xu,n,u); % eighth order
    elseif(ndss==10) ux=dss010(xl,xu,n,u); % tenth order
  end
%
% uxx
  if      (ndss== 2) uxx=dss002(xl,xu,n,ux); % second order
    elseif(ndss== 4) uxx=dss004(xl,xu,n,ux); % fourth order
    elseif(ndss== 6) uxx=dss006(xl,xu,n,ux); % sixth order

    elseif(ndss== 8) uxx=dss008(xl,xu,n,ux); % eighth order
    elseif(ndss==10) uxx=dss010(xl,xu,n,ux); % tenth order
  end
%
% PDE
  for i=1:n
    ut(i)=vis*uxx(i)-u(i)*ux(i);
  end
```

```
  ut=ut';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Listing 5.3. pde_1 for the MOL solution of Eq. (5.9) based on BCs from analytical solution (5.11)

We can note the following points about pde_1:

1. After the call definition of the function and a global area, two Dirichlet BCs for Eq. (5.9) at $x = xl = 0$ and $x = xu = 1$ are defined by using the analytical solution of Eq. (5.11) at these values of x (through calls to phi for Eq. (5.11)).

```
  function ut=pde_1(t,u)
%
% Function pde_1 computes the t derivative vector for
% Burgers' equation
%
  global n x xl xu ncall ncase ndss vis
%
% BC at x = 0
  u(1)=phi(xl,t);
%
% BC at x = 1
  u(n)=phi(xu,t);
```

2. The first derivative ux in Eq. (5.9) is computed by a differentiation routine, in this case dss004 since ndss=4 from pde_1_main is passed as a global variable to pde_1.

```
%
% ux
  if      (ndss== 2) ux=dss002(xl,xu,n,u); % second order
    elseif(ndss== 4) ux=dss004(xl,xu,n,u); % fourth order
    elseif(ndss== 6) ux=dss006(xl,xu,n,u); % sixth order
    elseif(ndss== 8) ux=dss008(xl,xu,n,u); % eighth order
    elseif(ndss==10) ux=dss010(xl,xu,n,u); % tenth order
  end
```

3. The second derivative in Eq. (5.9), uxx, is then computed by differentiating ux; that is, *stagewise differentiation* is used.

```
%
% uxx
  if      (ndss== 2) uxx=dss002(xl,xu,n,ux); % second order
    elseif(ndss== 4) uxx=dss004(xl,xu,n,ux); % fourth order
    elseif(ndss== 6) uxx=dss006(xl,xu,n,ux); % sixth order
    elseif(ndss== 8) uxx=dss008(xl,xu,n,ux); % eighth order
    elseif(ndss==10) uxx=dss010(xl,xu,n,ux); % tenth order
  end
```

4. Equation (5.9) is then programmed, and the usual transpose is included at the end (as required by `ode15s`). The counter `ncall` has a modest final value of 728 from Table 5.1.

```
%
% PDE
  for i=1:n
    ut(i)=vis*uxx(i)-u(i)*ux(i);
  end
  ut=ut';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

The close correspondence of the coding of Eq. (5.9) and its mathematical form indicates one of the salient features of the MOL. Note also how easily the nonlinear term `u(i)*ux(i)` can be included, which is also a salient feature of the numerical approach.

Routine `phi` for analytical solution (5.11) is given in Listing 5.4.

```
  function ua=phi(x,t)
%
% Function phi computes the exact solution of Burgers'
% equation for comparison with the numerical solution. It is
% also used to define the initial and boundary conditions for
% the numerical solution.
%
  global vis
%
% Analytical solution
  a=(0.05/vis)*(x-0.5+4.95*t);
  b=(0.25/vis)*(x-0.5+0.75*t);
  c=( 0.5/vis)*(x-0.375);
  ea=exp(-a);
```

```
  eb=exp(-b);
  ec=exp(-c);
  ua=(0.1*ea+0.5*eb+ec)/(ea+eb+ec);
```

Listing 5.4. Function `phi` for the analytical solution of Eqs. (5.11)–(5.13) (with $v_{xa}(x,t)$ = `phi`)

`phi` is a straightforward implementation of Eqs. (5.11)–(5.13).

An alternative approach to the solution of Eq. (5.9) is to note that the solution in Figure 5.1 has a zero slope at $x = 0$ and $x = 1$ (because the moving front does not reach either of these boundaries for $0 \leq t \leq 1$). Thus, we can make use of this property of the solution to program the BCs as *homogeneous Neumann BCs*. This is illustrated in `pde_2` called by `pde_1_main` for `ncase=2` (see Listing 5.5).

```
  function ut=pde_2(t,u)
%
% Function pde_2 computes the t derivative vector for Burgers'
% equation
%
  global n x xl xu ncall ncase ndss vis
%
% Calculate ux
  if      (ndss== 2) ux=dss002(xl,xu,n,u); % second order
    elseif(ndss== 4) ux=dss004(xl,xu,n,u); % fourth order
    elseif(ndss== 6) ux=dss006(xl,xu,n,u); % sixth order
    elseif(ndss== 8) ux=dss008(xl,xu,n,u); % eighth order
    elseif(ndss==10) ux=dss010(xl,xu,n,u); % tenth order
  end
%
% BC at x = 0
  ux(1)=0.0;
%
% BC at x = 1
  ux(n)=0.0;
%
% Calculate uxx
  if      (ndss== 2) uxx=dss002(xl,xu,n,ux); % second order
    elseif(ndss== 4) uxx=dss004(xl,xu,n,ux); % fourth order
    elseif(ndss== 6) uxx=dss006(xl,xu,n,ux); % sixth order
    elseif(ndss== 8) uxx=dss008(xl,xu,n,ux); % eighth order
    elseif(ndss==10) uxx=dss010(xl,xu,n,ux); % tenth order
  end
%
% PDE
  for i=1:n
```

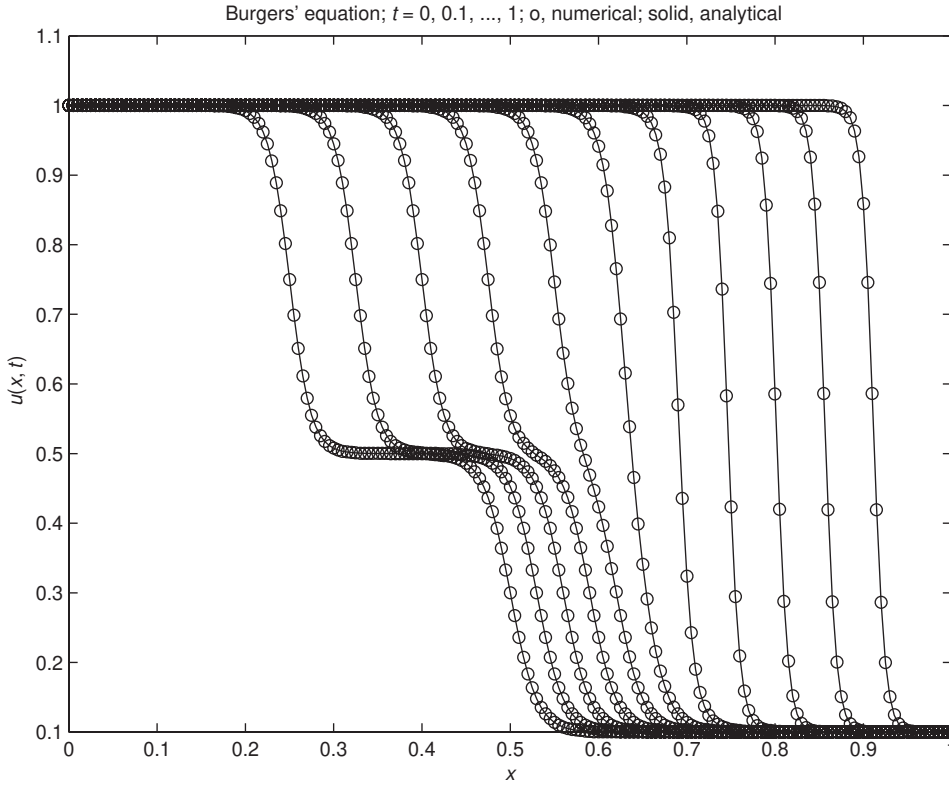

Figure 5.2. Plot of the numerical and analytical solutions of Burgers' equation (5.9) from `pde_1_main` and `pde_2` for `ncase=2`

```
    ut(i)=vis*uxx(i)-u(i)*ux(i);
  end
  ut=ut';
%
% Increment calls to pde_2
  ncall=ncall+1;
```

Listing 5.5. `pde_2` for the MOL solution of Eq. (5.9) based on homogeneous Neumann BCs

The only essential difference between `pde_1` (of Listing 5.3) and `pde_2` (of Listing 5.5) is the use of the Neumann BCs between the first and second calls to `dss004` to set `ux` at the boundaries rather than `u` (from the analytical solution, Eq. (5.11)). Thus, in this second approach, the analytical solution is used only for the IC, Eq. (5.11), with $t = 0$.

The numerical solution from `pde_2` is quite similar to the solution from `pde_1` in Table 5.1. Therefore, we present only the plotted output in Figure 5.2. The close agreement between the numerical and analytical solutions is again evident.

Figures 5.1 and 5.2 indicate that the solution front sharpens (steepens) with increasing t. In other words, at $t = 0$, the IC consists of two connected fronts, and as t increases, these two fronts merge into a single front that then steepens with

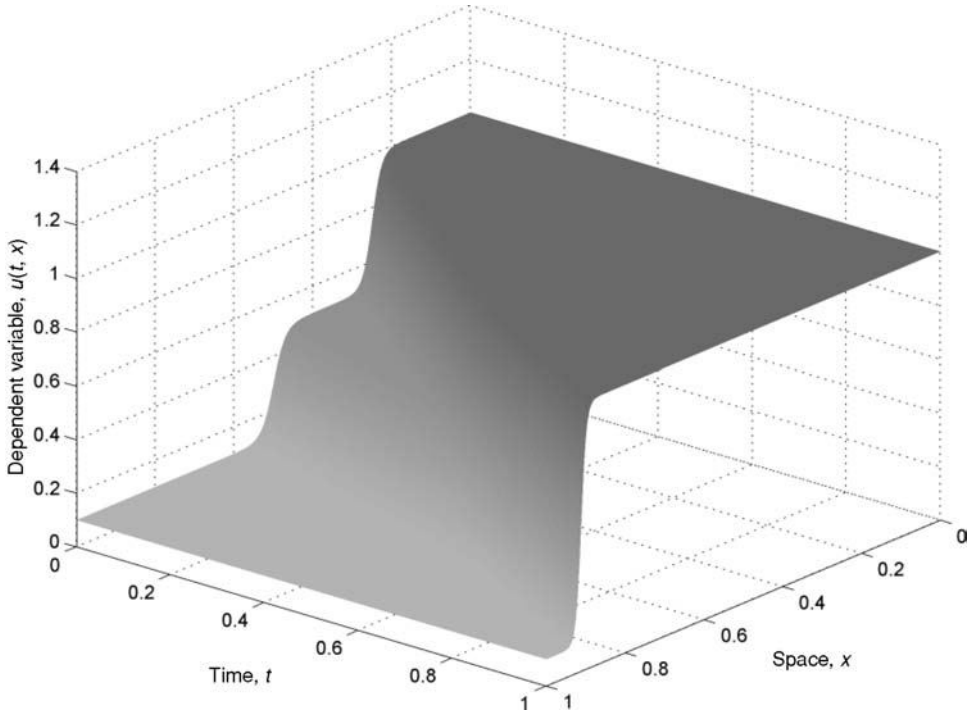

Figure 5.3. Three-dimensional plot of the numerical solution of Burgers' equation (5.9) from `pde_1_main` and `pde_1` for `ncase=1`

increasing t (note the relatively steep single front at $t = 1$). An explanation for this front sharpening can be inferred from the nonlinear first-derivative term of Eq. (5.9):

$$v_x \frac{\partial v_x}{\partial x}$$

This *convective term* has the important property that the velocity is v_x, the dependent variable of Eq. (5.9). If v_x decreases with increasing x as in the case of the solution of Figures 5.1 and 5.2, then the velocity is greater for small x. In other words, the solution "flows" more rapidly for small x, and this higher velocity leads to front sharpening. These properties of the solution of Eq. (5.9) are elucidated by the 3D plot in Figure 5.3.

Figure 5.3 was produced by using 101 points in t to produce better resolution for the 3D plotting (in `pde_1_main`). `t`

```
%
% Independent variable for ODE integration
  tf=1.0;
  tout=[t0:0.01:tf]';
  nout=101;
  ncall=0;
```

The 3D plot of Figure 5.3 was then produced by the following code (added after the plotting for Figure 5.1 in pde_1_main).

```
%
  figure(2);
  surf(x,t,u,'EdgeColor','none');
  set(get(gca,'XLabel'),'String','space, x')
  set(get(gca,'YLabel'),'String','time, t')
  set(get(gca,'ZLabel'),'String','dependent variable, u(t,x)')
  view(15,15);
  colormap('Cool')
```

Additional enhanced plotting of the Eq. (5.9) solution can be accomplished by animation (a movie). The details for producing an animation are given in Appendix 6.

Finally, an interesting study can be carried out by varying the kinematic viscosity $\nu = \mu/\rho$ in Eq. (5.9). As this parameter is reduced, the front sharpening of the solution increases (because $\nu = \mu/\rho$ multiplies the diffusive term $\partial^2 v_x/\partial x^2$ in Eq. (5.9)), and eventually the spatial grid will not be fine enough to resolve the solution. In this case, an alternative solution method may be required such as a special approximation of the derivatives in x in Eq. (5.9). The resolution of steep fronts is beyond the scope of the present discussion; it is the subject of an extensive published literature and available codes.

In summary, our intention in presenting this chapter is to indicate how:

1. A particular PDE, in this case Burgers' equation, can be obtained as a special case of a more general PDE system, in this case the Navier Stokes equations.
2. A numerical solution can be computed using either Dirichlet BCs (in this case set by the use of an analytical solution) or Neumann BCs. But the use of Neumann BCs required some knowledge of the solution (which might be through physical reasoning), for example, the solution does not change with x at the boundaries.
3. The use of the differentiation routines, for example, dss004 in pde_1 and pde_2, facilitates changing the order of the FD approximations; all that is required is to change the number of the routine since the arguments remain the same. Thus, the fourth-order FDs of dss004 can be changed to sixth-order FDs by changing dss004 to dss006 in Listings 5.3 and 5.5. Since the order of numerical approximations is often given the symbol p, this procedure of changing the order of the FDs is termed *p-refinement*.
4. Additionally, the number of grid points can be changed (in pde_1_main of Listing 5.1). Since the grid spacing in numerical approximations is often given the symbol h, changing the number of grid points is termed *h-refinement*.
5. The effect of h- and p-refinement on the numerical solution can be observed through repeated solutions. If the solution remains unchanged, for example, in the third figure, then three-figure accuracy can be inferred. Note that

this procedure does not require knowledge of an analytical solution (but, of course, it is not a mathematical proof of convergence).

6. To demonstrate an application of h- and p-refinement, we can consider a summary of some numerical output from `pde_1_main` and `pde_1` as given in Table 5.2.

   We can note the following points about this output

   (a) The kinematic viscosity, $\nu = \mu/\rho$ (= `nu`), is varied through the values 1.0, 0.1, 0.01, 0.003. As expected, this decrease in ν sharpens the front of the solution of Eq. (5.9) and therefore makes the calculation of the numerical solution more difficult, which is reflected in an increasing numerical error. For example, at `n = 201` grid points and a fourth-order FD approximation for the spatial derivatives in x, `ndss = 4`, the maximum errors in the solution are

nu	maxerror	ncall
1	$6.381e-006$	218
0.1	$1.703e-004$	228
0.01	$5.006e-004$	372
0.003	$3.135e-003$	728

   Although the maximum error increases by approximately 10^3 as ν is decreased from 1 to 0.003, this error is still quite acceptable for a solution that has a maximum value of the order of 1. Also, the computational effort increases with decreasing ν, but it is modest, even for $\nu = 0.003$ (`ncall = 728`).

   To complete the explanation of the output in Table 5.2, `xmax` is the value of x at which the maximum error occurs and `tmax` is the value of t at which this maximum error occurs.

   (b) As the number of grid points is increased, the maximum error decreases as expected. Thus, for $\nu = 0.003$ and `ndss = 4`,

n	maxerror	ncall
51	$2.356e-001$	451
101	$4.414e-002$	658
201	$3.135e-003$	728

   This reduction in error with increasing number of grid points is an example of h-refinement, which for the problem of Eq. (5.9) is quite effective (a reduction in the maximum error of approximately 10^{-2} with an increase in the number of grid points from `n = 51` to `n = 201`). In fact, this

Table 5.2. Maximum solution errors as a function of the kinematic viscosity, ν (= nu), number of grid points, n, and order of the FD approximation, ndss[1]

nu	n	ndss	maxerror	xmax	tmax	ncall
1.000	51	2	1.847e-006	0.0000	1.0	71
		4	6.379e-006	0.5400	0.3	68
		6	6.379e-006	0.5400	0.3	68
	101	2	7.825e-006	0.5800	0.3	122
		4	6.381e-006	0.5500	0.3	118
		6	6.381e-006	0.5500	0.3	118
	201	2	8.276e-006	0.5800	0.3	222
		4	6.381e-006	0.5500	0.3	218
		6	6.483e-006	0.5700	0.2	219
0.100	51	2	2.325e-004	0.0000	1.0	76
		4	1.702e-004	0.7200	0.2	78
		6	1.702e-004	0.7200	0.2	78
	101	2	9.413e-005	0.7700	0.7	130
		4	1.703e-004	0.7300	0.2	128
		6	1.703e-004	0.7300	0.2	128
	201	2	8.286e-005	0.7650	0.7	229
		4	1.703e-004	0.7300	0.2	228
		6	1.703e-004	0.7300	0.2	228
0.010	51	2	4.117e-002	0.9200	1.0	235
		4	5.725e-003	0.8800	1.0	218
		6	1.129e-002	1.0000	1.0	217
	101	2	9.256e-003	0.9300	1.0	273
		4	6.264e-004	0.8300	0.9	275
		6	4.967e-004	0.5700	0.5	272
	201	2	2.392e-003	0.9300	1.0	371
		4	5.006e-004	0.5750	0.5	372
		6	4.982e-004	0.5750	0.5	370
0.003	51	2	5.126e-001	0.7600	0.8	441
		4	2.356e-001	0.8800	1.0	451
		6	6.492e-001	0.0000	0.2	454
	101	2	1.738e-001	0.8900	1.0	669
		4	4.414e-002	0.8400	0.9	658
		6	2.409e-002	0.9000	1.0	635
	201	2	2.997e-002	0.9150	1.0	797
		4	3.135e-003	0.8500	0.9	728
		6	1.111e-003	0.8500	0.9	723

[1] Contributed by John Carroll, School of Mathematical Sciences, Dublin City University, Ireland.

reduction can be estimated approximately by considering the fourth-order FD approximation: $[(201-1)/(51-1)]^4 = 64$, while $2.356e-001/3.135e-003 = 75.2 \approx 64$.

Better agreement between the ratio of errors estimated from the FD order condition (`ndss = 4`) and the actual ratio of errors is not necessarily expected. Keep in mind that the FD order condition is for the derivatives in x in eq. (5.9) and the accuracy of these numerical derivatives is not necessarily the same as the accuracy of the numerical PDE solution. However, we would expect improvement in the numerical PDE solution with improved accuracy in the numerical derivatives in x, as observed in this case.

As a related point, the total computational effort does not increase substantially (`ncall = 451` to `ncall = 728`), which is due principally to the use of a stiff integrator (`ode15s`), since the stiffness of the ODEs most likely increased substantially with their number (`51` to `201`), but the stiff integrator handled the stiffness quite well (in general the ODE stiffness increases with decreasing grid spacing or increasing numbers of ODEs).

(c) As the order of the FD approximation is increased, the maximum error decreases as expected. Thus, for $\nu = 0.003$ and `n = 201`,

ndss	maxerror	ncall
2	$2.997e-002$	797
4	$3.135e-003$	728
6	$1.111e-003$	723

This reduction in error with increasing order of the FD approximation is an example of p-refinement, which for the problem of Eq. (5.9) is quite effective (a reduction in the maximum error of approximately 10^{-1} with an increase in the order of the FD approximation from `ndss = 2` to `ndss = 6`). Equally important, this improvement was achieved without an increase in `ncall` (but the formulas in `dss006` require more calculations than those in `dss002`). The extension of this improvement could be expected by using the eighth- and tenth-order FD approximations of routines `dss008` and `dss010`.

REFERENCE

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