CHAPTER 2

Pattern Formation

2.1 Introduction

The PDE models discussed in this chapter pertain to pattern formation, in this case, a pattern of cells that is defined by a chemoattractant and a stimulant, in analogous manner to the chemotaxis model of Chapter 1. We consider two 1D models based on two PDEs ([2], p 268, eqs. (5.20), (5.21)) and three PDEs ([2], p 264, eqs. (5.11), (5.12), (5.13)). The intent is to demonstrate

- The inclusion of nonlinear terms in the PDEs.
- Adding a PDE to a model that might occur, for example, during model development.
- The calculation and display of the numerical solution of the PDE model, including an examination of the individual terms in the PDEs to provide insight into the origin of the solution properties.

The starting point is the following coupled nonlinear PDE diffusion system ([2], p 264, eqs. (5.11)–(5.13))

$$\frac{\partial u_1}{\partial t} = D_1 \nabla^2 u_1 - \nabla \cdot \left[\frac{k_1 u_1}{(k_2 + u_2)^2} \nabla u_2 \right] + k_3 u_1 \left[\frac{k_4 u_3^2}{k_9 + u_3^2} - u_1 \right] \tag{2.1a}$$

$$\frac{\partial u_2}{\partial t} = D_2 \nabla^2 u_2 + k_5 u_3 \left[\frac{u_1^2}{k_6 + u_1^2} \right] - k_7 u_1 u_2 \tag{2.1b}$$

$$\frac{\partial u_3}{\partial t} = D_3\nabla^2 u_3 - k_8 u_1\left[\frac{u_3^2}{k_9 + u_3^2}\right] \tag{2.1c}$$

where

TABLE 2.1 Notation in eqs. (2.1).

Variable	Interpretation
u_1	density of cells
u_2	concentration of chemoattractant
u_3	concentration of stimulant
∇	div operating on a scalar
$\nabla\cdot$	div operating on a vector
$\nabla\cdot\nabla = \nabla^2$	Laplacian
t	time
D_1, D_2, D_3	diffusivities for u_1, u_2, u_3, respectively
$k_1, \ldots, k_9$	rate constants

where $(\cdot)$ denotes a vector dot product.

The ∇ operators in Cartesian coordinates (x, y, z) are given in Table 2.2.

TABLE 2.2 ∇ Operators in Cartesian coordinates.

∇ Operator	Cartesian coordinate representation
∇	$\mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}$
$\nabla\cdot$	$\left(\mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}\right)\cdot$
$\nabla\cdot\nabla = \nabla^2$	$\left(\mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}\right)\cdot\left(\mathbf{i}\frac{\partial}{\partial x} + \mathbf{j}\frac{\partial}{\partial y} + \mathbf{k}\frac{\partial}{\partial z}\right) = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$

$\mathbf{i}, \mathbf{j}, \mathbf{k}$ are the orthogonal Cartesian unit vectors with the properties, for example, $\mathbf{i}\cdot\mathbf{i} = 1$ and $\mathbf{i}\cdot\mathbf{j} = \mathbf{i}\cdot\mathbf{k} = 0$.

We note that eqs. (2.1) are just three diffusion equations (Fick's second law, $\partial u/\partial t = D\nabla^2 u$) augmented with various nonlinear terms. For example, in eq. (2.1a), $-\nabla \cdot \left[\frac{k_1 u_1}{(k_2+u_2)^2}\nabla u_2\right]$ is a rate of change of the cells (u_1) because of the combined effects of (1) the cells (u_1) from $k_1 u_1$ and (2) the attractant (u_2) from $\frac{1}{(k_2+u_2)^2}\nabla u_2$. The particular form of the nonlinear term, and the numerical values of the parameters, is a matter of experience and judgment employed during the model formulation and most likely will require some trial and error, particularly with regard to reconciliation with experimental data.

Similarly, the nonlinear term $+k_3 u_1\left[\frac{k_4 u_3^2}{k_9+u_3^2} - u_1\right]$ is a rate of change of the cells (u_1) from the combined effects of (1) the cells (u_1) from $+k_3 u_1$ and $-u_1$ (these two terms have opposite signs and therefore opposite effects with $k_3 > 0$) and (2) the stimulant (u_3) from $\frac{k_4 u_3^2}{k_9+u_3^2}$.

In summary, the effect and interaction of these nonlinear terms is complicated. This complexity can be elucidated by computing and examining the individual terms as illustrated in the subsequent programming (for the 3-PDE model).

2.2 Two PDE Model

To start with the 2-PDE model in 1D (x only so that $\nabla^2 = \partial^2/\partial x^2$), dimensionless $u_1(x,t)$ and $u_2(x,t)$ are given by variants of eqs. (2.1a) and (2.1b) ([2], p 268, eqs. (5.20) and (5.21))

$$\frac{\partial u_1}{\partial t} = D_1\frac{\partial^2 u_1}{\partial x^2} - \alpha\frac{\partial}{\partial x}\left[\frac{u_1}{(1+u_2)^2}\frac{\partial u_2}{\partial x}\right] \tag{2.2a}$$

$$\frac{\partial u_2}{\partial t} = \frac{\partial^2 u_2}{\partial x^2} + w\frac{u_1^2}{\mu + u_1^2} \tag{2.2b}$$

where D_1, α, w, and μ are dimensionless constants.

Eqs. (2.2) are first order in t and second order in x. Therefore, each of them requires one IC and two BCs ([1], p 239, eq. (6)).

$$u_1(x,t=0)=f_1(x); \quad u_2(x,t=0)=f_2(x) \tag{2.3a,b}$$

$$\frac{\partial u_1(x=0,t)}{\partial x}=\frac{\partial u_1(x=L,t)}{\partial x}=0 \tag{2.4a,b}$$

$$\frac{\partial u_2(x=0,t)}{\partial x}=\frac{\partial u_2(x=L,t)}{\partial x}=0 \tag{2.4c,d}$$

where L is the length of the experimental system. Eqs. (2.4) are zero-flux (no diffusion, homogeneous Neumann) BCs at the physical boundaries of the experimental system.

The nonlinear term in eq. (2.2a) can be expanded as

$$\frac{\partial}{\partial x}\left[\frac{u_1}{(1+u_2)^2}\frac{\partial u_2}{\partial x}\right]=\left[\frac{u_1}{(1+u_2)^2}\frac{\partial^2 u_2}{\partial x^2}\right]+\left[\frac{1}{(1+u_2)^2}\frac{\partial u_1}{\partial x}\frac{\partial u_2}{\partial x}\right] + \left[u_1\frac{-2}{(1+u_2)^3}\left(\frac{\partial u_2}{\partial x}\right)^2\right] \tag{2.5}$$

for use in the subsequent programming.

Eqs. (2.2) to (2.5) constitute the 2-PDE model. We now consider the numerical solution of these equations, starting with the ODE routine based on the MOL.

2.2.1 ODE Routine

The ODE routine with the programming of eqs. (2.2) and (2.5) is in Listing 2.1.

```
  p_form_1=function(t,u,parms){
#
# Function p_form_1 computes the t derivative vector
# of the u1,u2 vectors
#
# One vector to two vectors
  u1=rep(0,nx);u2=rep(0,nx);
  for(i in 1:nx){
    u1[i]=u[i];
    u2[i]=u[i+nx];
  }
#
# u1x, u2x
  u1x=dss004(xl,xu,nx,u1);
  u2x=dss004(xl,xu,nx,u2);
#
# Boundary conditions
  u1x[1]=0;u1x[nx]=0;
  u2x[1]=0;u2x[nx]=0;
  nl=2;nu=2;
#
# u1xx, u2xx
  u1xx=dss044(xl,xu,nx,u1,u1x,nl,nu);
  u2xx=dss044(xl,xu,nx,u2,u2x,nl,nu);
#
# RHS terms
  term1=rep(0,nx);term2=rep(0,nx);term3=rep(0,nx);
  for(i in 1:nx){
    den=1/(1+u2[i])^2;
    term1[i]=u1[i]*den*u2xx[i];
    term2[i]=den*u1x[i]*u2x[i];
    term3[i]=-2*u1[i]*den/(1+u2[i])*u2x[i]^2;
  }
#
# PDEs
  u1t=rep(0,nx);u2t=rep(0,nx);
  for(i in 1:nx){
    u1t[i]=d1*u1xx[i]-alpha*(term1[i]+term2[i]+term3[i]);
    u2t[i]=u2xx[i]+w1*u1[i]^2/(mu+u1[i]^2);
  }
#
# Two vectors to one vector
  ut=rep(0,2*nx);
  for(i in 1:nx){
    ut[i]   =u1t[i];
    ut[i+nx]=u2t[i];
  }
#
# Increment calls to p_form_1
  ncall <<- ncall+1;
#
# Return derivative vector
  return(list(c(ut)));
}
```

Listing 2.1 ODE routine `p_form_1`.

We can note the following details about p_form_1.

- The function is defined.

```
  p_form_1=function(t,u,parms){
#
# Function p_form_1 computes the t derivative vector
# of the u1,u2 vectors
```

  The input arguments are in conformity with the R ODE integrators, in this case lsodes called by the main program discussed subsequently. lsodes in turn calls p_form_1. u is a vector of 102 elements, that is, 51 points in x for each of eqs. (2.2) (102 ODEs in the MOL approximation of eqs. (2.2)). parms is unused.

- u is placed in two vectors, u1 and u2, to facilitate subsequent programming in terms of problem-oriented variables, that is, the dependent variables of eqs. (2.2). nx=51 is set in the main program.

```
#
# One vector to two vectors
  u1=rep(0,nx);u2=rep(0,nx);
  for(i in 1:nx){
    u1[i]=u[i];
    u2[i]=u[i+nx];
  }
```

- $\partial u_1/\partial x, \partial u_2/\partial x$ are computed by the library differentiator dss004. xl and xu are set in the main program.

```
#
# u1x, u2x
  u1x=dss004(xl,xu,nx,u1);
  u2x=dss004(xl,xu,nx,u2);
```

- BCs (2.4) are programmed.

```
#
# Boundary conditions
  u1x[1]=0;u1x[nx]=0;
  u2x[1]=0;u2x[nx]=0;
  nl=2;nu=2;
```

Since the BCs specify the partial derivatives at the boundaries, that is, $\partial u_1(x = 0, t)/\partial x = 0$, $\partial u_1(x = L, t)/\partial x = 0$, and similar conditions for u_2, Neumann BCs are designated with `nl` and `nu`.

- $\partial^2 u_1/\partial x^2$ and $\partial^2 u_2/\partial x^2$ are computed by the library differentiator `dss044`.

```
#
# u1xx, u2xx
  u1xx=dss044(xl,xu,nx,u1,u1x,nl,nu);
  u2xx=dss044(xl,xu,nx,u2,u2x,nl,nu);
```

Note that the derivatives at the boundaries, `u1x[1],u1x[nx]` and `u2x[1],u2x[nx]`, are inputs to `dss044`.

- The programming of the three RHS terms in eq. (2.5) is programmed with a `for` with index `i`.

```
#
# RHS terms
  term1=rep(0,nx);term2=rep(0,nx);term3=rep(0,nx);
  for(i in 1:nx){
    den=1/(1+u2[i])^2;
    term1[i]=u1[i]*den*u2xx[i];
    term2[i]=den*u1x[i]*u2x[i];
    term3[i]=-2*u1[i]*den/(1+u2[i])*u2x[i]^2;
  }
```

The correspondence between the terms in eq. (2.5) and their coding is

$$\frac{1}{(1+u_2)^2} \Rightarrow$$

`den=1/(1+u2[i])^2;`

$$\left[\frac{u_1}{(1+u_2)^2}\frac{\partial^2 u_2}{\partial x^2}\right] \Rightarrow$$

`term1[i]=u1[i]*den*u2xx[i];`

$$\left[\frac{1}{(1+u_2)^2}\frac{\partial u_1}{\partial x}\frac{\partial u_2}{\partial x}\right] \Rightarrow$$

```
term2[i]=den*u1x[i]*u2x[i];
```

$$\left[u_1\frac{-2}{(1+u_2)^3}\left(\frac{\partial u_2}{\partial x}\right)^2\right] \Rightarrow$$

```
term3[i]=-2*u1[i]*den/(1+u2[i])*u2x[i]^2;
```

- Eqs. (2.2) are programmed in a `for` with index `i`.

```
#
# PDEs
  u1t=rep(0,nx);u2t=rep(0,nx);
  for(i in 1:nx){
    u1t[i]=d1*u1xx[i]-alpha*(term1[i]+term2[i]
       +term3[i]);
    u2t[i]=u2xx[i]+w1*u1[i]^2/(mu+u1[i]^2);
  }
```

`d1,alpha,w1,mu` are defined numerically in the main program. The derivatives $\partial u_1/\partial t, \partial u_2/\partial t$ in `u1t,u2t` are the final result. They are passed to `lsodes` for the integration of the `102` ODEs.

- The derivatives `u1t,u2t` are placed in a single vector `ut` that is returned to `lsodes` for the integration of the `102` ODEs.

```
#
# Two vectors to one vector
  ut=rep(0,2*nx);
  for(i in 1:nx){
    ut[i]   =u1t[i];
    ut[i+nx]=u2t[i];
  }
```

- The number of calls to `p_form_1` is incremented, and the value is returned to the main program with `<<-`. The derivative vector is then returned from p_form_1 as a list (as required by the R ODE integrators including lsodes).

```
#
# Increment calls to p_form_1
  ncall <<- ncall+1;
#
# Return derivative vector
  return(list(c(ut)));
}
```

The final } concludes p_form_1.

In summary, the model consisting of eqs. (2.2), (2.4), and (2.5) is programmed in p_form_1. The only part of the model not included in this routine are the ICs for eqs. (2.2), that is, eqs. (2.3), to start the numerical solution. We now consider the effect of different ICs and some features of the numerical and graphical outputs produced by the main program.

2.2.2 Main Program

p_form_1 is called by the main program in Listing 2.2 as an input argument of lsodes.

```
#
# Access ODE integrator
  library("deSolve");
#
# Access functions for numerical  solutions
  setwd("c:/R/bme_pde/chap2/v_2pde");
  source("p_form_1.R");
  source("dss004.R");
  source("dss044.R");
#
# Level of output
#
#   ip = 1 - graphical (plotted) solutions
#             (u1(x,t), u2(x,t)) only
#
```

```
#   ip = 2 - numerical and graphical solutions
#
  ip=2;
#
# Initial condition (IC)
#
#   ncase = 1 - spatially uniform
#
#   ncase = 2 - Gaussian
#
#   ncase = 3 - step
#
  ncase=1;
#
# Grid (in x)
  nx=51;xl=0;xu=1;
  xg=seq(from=xl,to=xu,by=0.02);
#
# Parameters
  alpha=10;d1=0.5;w1=1;mu=0.1;
  cat(sprintf(
    "\n\n  alpha = %5.2f  d1 = %5.2f   w1 = %5.2f
       mu = %5.2f\n",alpha,d1,w1,mu));
#
# Independent variable for ODE integration
  nout=11;
  tout=seq(from=0,to=0.25,by=0.025);
#
# Initial condition
  u0=rep(0,2*nx);u10=rep(0,nx);u20=rep(0,nx);
  for(i in 1:nx){
#
#   Solution remains spatially uniform
    if(ncase==1){
    u10[i]=1;u20[i]=0;}
#
#   Initial Gaussian in u1
    if(ncase==2){
    u10[i]=exp(-5*(xg[i]-0.5)^2);u20[i]=0;}
#
#   Initial band in u1
```

```
    if(ncase==3){
    if(i<=11){
      u10[i]=1;u20[i]=0;
    }else{
      u10[i]=0;u20[i]=0;
    } }
    u0[i]   =u10[i];
    u0[i+nx]=u20[i];
  }
  t=0;
  ncall=0;
#
# ODE integration
  out=lsodes(y=u0,times=tout,func=p_form_1,parms=NULL)
  nrow(out)
  ncol(out)
#
# Arrays for plotting numerical solution
  u1_plot=matrix(0,nrow=nx,ncol=nout);
  u2_plot=matrix(0,nrow=nx,ncol=nout);
  for(it in 1:nout){
    for(ix in 1:nx){
       u1_plot[ix,it]=out[it,ix+1];
       u2_plot[ix,it]=out[it,ix+1+nx];
    }
  }
#
# Display numerical solution
  if(ip==2){
    for(it in 1:nout){
     cat(sprintf("\n   t    x   u1(x,t)  u2(x,t)\n"));
      for(ix in 1:nx){
        cat(sprintf("%7.3f%8.2f%12.5f%12.5f\n",
        tout[it],xg[ix],u1_plot[ix,it],u2_plot[ix,it]));
      }
    }
  }
#
# Calls to ODE routine
  cat(sprintf("\n\n ncall = %5d\n\n",ncall));
#
```

```
# Plot u1 numerical
  par(mfrow=c(1,1));
  matplot(x=xg,y=u1_plot,type="l",xlab="x",
          ylab="u1(x,t), t=0,0.025,...,0.25",xlim=c
             (xl,xu),lty=1,main="u1(x,t); t=0,0.025,...,
                0.25;",lwd=2);
#
# Plot u2 numerical
  par(mfrow=c(1,1));
  matplot(x=xg,y=u2_plot,type="l",xlab="x",
          ylab="u2(x,t), t=0,0.025,...,0.25",xlim=c
             (xl,xu),lty=1,main="u2(x,t); t=0,0.025,...,
                0.25;",lwd=2);
```

Listing 2.2 Main program for eqs. (2.2).

We can note the following details about this main program.

- The R ODE integrator library, `deSolve`, is accessed to provide `lsodes`. `p_form_1` of Listing 2.1 and the two spatial differentiation routines used in `p_form_1` are accessed by `setwd` (set working directory) and `source` statements.

  ```
  #
  # Access ODE integrator
    library("deSolve");
  #
  # Access functions for numerical solutions
    setwd("c:/R/bme_pde/chap2/v_2pde");
    source("p_form_1.R");
    source("dss004.R");
    source("dss044.R");
  ```

  Note the use of the forward slash `/` in the `setwd`.
- The level of output is specified with `ip`.

  ```
  #
  # Level of output
  #
  ```

```
#   ip = 1 - graphical (plotted) solutions
#            (u1(x,t), u2(x,t)) only
#
#   ip = 2 - numerical and graphical solutions
#
  ip=2;
```

- One of the three ICs is selected with `ncase`. The details of each IC will be clear from the subsequent programming.

```
#
# Initial condition (IC)
#
#   ncase = 1 - spatially uniform
#
#   ncase = 2 - Gaussian
#
#   ncase = 3 - step
#
  ncase=1;
```

- The grid in x is defined as $0 \leq x \leq 1$, with `51` points so that $x = 0, 0.02, \ldots, 1$.

```
#
# Grid (in x)
  nx=51;xl=0;xu=1;
  xg=seq(from=xl,to=xu,by=0.02);
```

- The parameters of eqs. (2.2) are assigned numerical values as suggested in [2], p269 and then displayed at the beginning of the solution.

```
#
# Parameters
  alpha=10;d1=0.5;w1=1;mu=0.1;
  cat(sprintf(
   "\n\n  alpha = %5.2f  d1 = %5.2f   w1 = %5.2f
      mu = %5.2f\n",alpha,d1,w1,mu));
```

- The interval in t is $0 \leq t \leq 0.25$ with `11` values, $t = 0, 0.025, \ldots, 0.25$, for the output.

```
#
# Independent variable for ODE integration
  nout=11;
  tout=seq(from=0,to=0.25,by=0.025);
```

- Three ICs are programmed.

  `ncase=1`, $u_1(x, t = 0) = 1, u_2(x, t = 0) = 0.$

  `ncase=2`, $u_1(x, t = 0) =$ gaussian function, $u_2(x, t = 0) = 0.$

  `ncase=3`, $u_1(x, t = 0) = 1, u_2(x, t = 0) = 0,\ 0 \leq x \leq 0.025,$

  $u_1(x, t = 0) = 0, u_2(x, t = 0) = 0,\ 0.025 < x \leq 0.25.$

```
#
# Initial condition
  u0=rep(0,2*nx);u10=rep(0,nx);u20=rep(0,nx);
  for(i in 1:nx){
#
#   Solution remains spatially uniform
    if(ncase==1){
    u10[i]=1;u20[i]=0;}
#
#   Initial Gaussian in u1
    if(ncase==2){
    u10[i]=exp(-5*(xg[i]-0.5)^2);u20[i]=0;}
#
#   Initial band in u1
    if(ncase==3){
    if(i<=11){
      u10[i]=1;u20[i]=0;
    }else{
      u10[i]=0;u20[i]=0;
    } }
    u0[i]   =u10[i];
    u0[i+nx]=u20[i];
  }
  t=0;
  ncall=0;
```

The ICs, u10,u20, are then placed in a single vector, u0, of length $2(51) = 102$. Finally, the independent variable t and the number of calls to p_form_1 are initialized.

- The ODEs are integrated by lsodes, which is informed of the number of ODEs (102) by the length of the IC vector u0.

```
#
# ODE integration
  out=lsodes(y=u0,times=tout,func=p_form_1,
     parms=NULL)
  nrow(out)
  ncol(out)
```

Note the use of the IC vector, u0, the vector of output points, tout, and the ODE routine p_form_1 of Listing 2.1. y,times,func,parms are reserved names. parms is unused. The dimensions of the solution array, out, are displayed for verification.

- The solution is put into two 2D arrays, u1_plot,u2_plot, for subsequent plotting.

```
#
# Arrays for plotting numerical solution
  u1_plot=matrix(0,nrow=nx,ncol=nout);
  u2_plot=matrix(0,nrow=nx,ncol=nout);
  for(it in 1:nout){
    for(ix in 1:nx){
       u1_plot[ix,it]=out[it,ix+1];
       u2_plot[ix,it]=out[it,ix+1+nx];
    }
  }
```

Note the offset of 1 in the second subscript of out, e.g., ix+1, since the values of t are included in out as out[it,1] (this is the usual operation of lsodes and the other R ODE integrators). In other words, out has the dimensions out[nout,2*nx+1]=out[6,2*51+1]=out[6,103].

- For ip=2, the numerical solution is displayed in tabular form.

```
#
# Display numerical solution
  if(ip==2){
    for(it in 1:nout){
      cat(sprintf("\n    t    x     u1(x,t)
         u2(x,t)\n"));
      for(ix in 1:nx){
        cat(sprintf("%7.3f%8.2f%12.5f%12.5f\n",
        tout[it],xg[ix],u1_plot[ix,it],u2_plot
           [ix,it]));
      }
    }
  }
```

- The number of calls to `p_form_1` is displayed at the end of the solution as a measure of the computational effort to compute the solution.

```
#
# Calls to ODE routine
  cat(sprintf("\n\n ncall = %5d\n\n",ncall));
```

- $u_1(x,t), u_2(x,t)$ are plotted as a function of x with t as a parameter.

```
#
# Plot u1 numerical
  par(mfrow=c(1,1));
  matplot(x=xg,y=u1_plot,type="l",xlab="x",
         ylab="u1(x,t), t=0,0.025,...,0.25",xlim=c
            (xl,xu),lty=1,main="u1(x,t); t=0,0.025,...,
               0.25;",lwd=2);
#
# Plot u2 numerical
  par(mfrow=c(1,1));
  matplot(x=xg,y=u2_plot,type="l",xlab="x",
         ylab="u2(x,t), t=0,0.025,...,0.25",xlim=c
            (xl,xu),lty=1,main="u2(x,t); t=0,0.025,...,
               0.25;",lwd=2);
```

`par(mfrow=c(1,1))` specifies a 1×1 matrix of plots, that is, a single plot. `matplot` produces the parametric plots by using `xg` as the abscissa (horizontal, `x` variable) and `u1_plot,u2_plot` as the ordinate (vertical, `y` variable), with the requirement that the number of rows of `x` and `y` must be the same, in this case `51`.

This concludes the programming of eqs. (2.2) to (2.5). The numerical and graphical outputs are reviewed in the following sections.

2.2.3 Numerical Solution

For `ncase=1`, abbreviated numerical output (from `ip=2` in Listing 2.2) is listed in Table 2.3.

We can note the following details about this output.

- The dimensions of `out` are `out[11,103]` as expected (and explained previously).
- The ICs, $u_1(x,t=0)=1, u_2(x,t=0)=0$ for `ncase=1`, are correct (always a good idea to check the ICs so the solution starts correctly).

TABLE 2.3 Abbreviated output from Listing 2.2 with `ip=2` for `ncase=1`.

```
alpha = 10.00  d1 =  0.50   w1 =  1.00  mu =  0.10

> nrow(out)
[1] 11
> ncol(out)
[1] 103

    t        x     u1(x,t)     u2(x,t)
0.000     0.00     1.00000     0.00000
0.000     0.02     1.00000     0.00000
0.000     0.04     1.00000     0.00000
0.000     0.06     1.00000     0.00000
0.000     0.08     1.00000     0.00000
0.000     0.10     1.00000     0.00000
             .        .
             .        .
             .        .
```

(*continued*)

TABLE 2.3 (*Continued*)

```
   Output for x = 0.12 to 0.88 removed
            .        .
            .        .
            .        .
  0.000    0.90     1.00000     0.00000
  0.000    0.92     1.00000     0.00000
  0.000    0.94     1.00000     0.00000
  0.000    0.96     1.00000     0.00000
  0.000    0.98     1.00000     0.00000
  0.000    1.00     1.00000     0.00000
            .        .
            .        .
            .        .
   Output for t = 0.025 to 0.225 removed
            .        .
            .        .
            .        .
      t       x     u1(x,t)     u2(x,t)
  0.250    0.00     1.00000     0.22727
  0.250    0.02     1.00000     0.22727
  0.250    0.04     1.00000     0.22727
  0.250    0.06     1.00000     0.22727
  0.250    0.08     1.00000     0.22727
  0.250    0.10     1.00000     0.22727
            .        .
            .        .
            .        .
   Output for x = 0.12 to 0.88 removed
            .        .
            .        .
            .        .
  0.250    0.90     1.00000     0.22727
  0.250    0.92     1.00000     0.22727
  0.250    0.94     1.00000     0.22727
  0.250    0.96     1.00000     0.22727
  0.250    0.98     1.00000     0.22727
  0.250    1.00     1.00000     0.22727
```

TABLE 2.3 (*Continued*)

```
ncall =   129

Warning message:
In plot.window(...) :
relative range of values =  27 * EPS, is small (axis 2)
```

- The solutions, $u_1(x,t), u_2(x,t)$, remain invariant with x to six figures. This result follows from eqs. (2.2). For the ICs that are invariant with x, $\partial u_1(x,t=0)/\partial x = \partial u_2(x,t=0)/\partial x = 0$. Thus, the zero derivatives in x in eq (2.2a) lead to $\partial u_1(x,t)/\partial t = 0$ for all x, that is, $u_1(x,t)$ is invariant in t so that $u_1(x,t=0) = u_1(x,t) = 1$ for all t as well as x as observed in Table 2.3.

  For eq. (2.2b), the zero derivative in x gives the PDE $\dfrac{\partial u_2}{\partial t} = w\dfrac{u_1^2}{\mu + u_1^2}$. With $w = 1$, $\mu = 0.1$ (from Listing 2.2), $\dfrac{\partial u_2}{\partial t} = (1)1^2/(0.1 + 1^2) = 1/1.1 = 0.9090\ldots$. In other words, $u_2(x,t)$ changes by $0.9090\ldots$ for each unit change in t. For a change in t from 0 to 0.250 (in Table 2.3), the change in u_2 from 0 is $(0.9090\ldots)(0.25) = 0.227272\ldots$ as indicated in Table 2.3.
- The computational effort is modest with `ncall = 129`.
- Unexpectedly, the `matplot` utility applied to u_1 could not automatically scale the constant $u_1(x,t) = 1.00000$ solution vertically and therefore issued an error message.

  ```
  Warning message:
  In plot.window(...) :
  relative range of values =  27 * EPS, is small
     (axis 2)
  ```

  The resulting plot is in Fig. 2.1. Various attempts to rescale $u_1(x,t)$ so that `matplot` would produce a satisfactory plot (rather than Fig. 2.1), including the use of the argument `ylim` for `matplot` (forced rather than automatic scaling), were unsuccessful (perhaps the reader would like to retry some rescaling of $u_1(x,t)$). Interestingly, `matplot` had no difficulty in plotting $u_2(x,t)$ as indicated in Fig. 2.2.

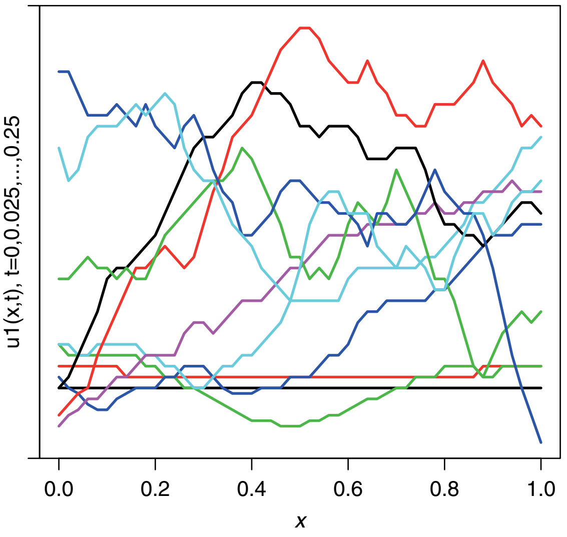

Figure 2.1 $u_1(x,t)$ versus x with t as a parameter, `ncase=1`.

Fig. 2.2 indicates the constant $u_2(x,t)$ in x of Table 2.3, with the variation of 0.9090 . . . for each unit in t as discussed previously.

For `ncase=2`, the abbreviated numerical output (from `ip=2` in Listing 2.2) is listed in Table 2.4.

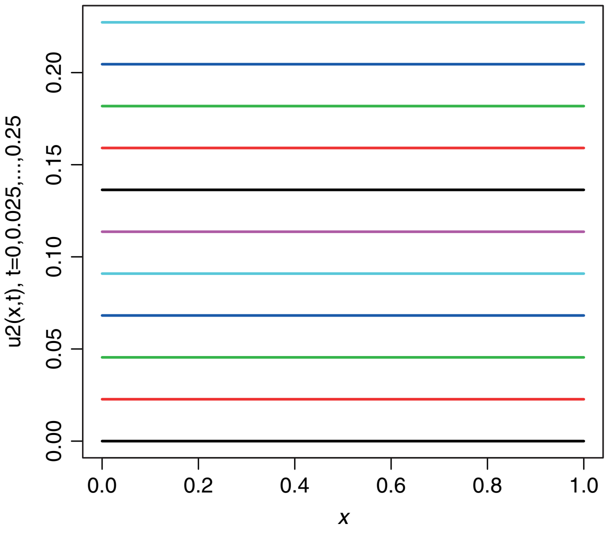

Figure 2.2 $u_2(x,t)$ versus x with t as a parameter, `ncase=1`.

TABLE 2.4 Abbreviated output from Listing 2.2 with `ip=2` for `ncase=2`.

```
alpha = 10.00  d1 =  0.50   w1 =  1.00  mu =  0.10

> nrow(out)
[1] 11
> ncol(out)
[1] 103

    t       x      u1(x,t)     u2(x,t)
0.000    0.00     0.28650     0.00000
0.000    0.02     0.31600     0.00000
0.000    0.04     0.34715     0.00000
0.000    0.06     0.37984     0.00000
0.000    0.08     0.41395     0.00000
0.000    0.10     0.44933     0.00000
           .         .
           .         .
           .         .
 Output for x = 0.12 to 0.88 removed
           .         .
           .         .
           .         .
0.000    0.90     0.44933     0.00000
0.000    0.92     0.41395     0.00000
0.000    0.94     0.37984     0.00000
0.000    0.96     0.34715     0.00000
0.000    0.98     0.31600     0.00000
0.000    1.00     0.28650     0.00000
           .         .
           .         .
           .         .
 Output for t = 0.025 to 0.225 removed
           .         .
           .         .
           .         .
    t       x      u1(x,t)     u2(x,t)
0.250    0.00     0.69716     0.20672
0.250    0.02     0.69720     0.20673
```

(*continued*)

TABLE 2.4 (*Continued*)

```
 0.250    0.04     0.69732     0.20673
 0.250    0.06     0.69753     0.20673
 0.250    0.08     0.69781     0.20674
 0.250    0.10     0.69816     0.20674
           .        .
           .        .
           .        .
  Output for x = 0.12 to 0.88 removed
           .        .
           .        .
           .        .
 0.250    0.90     0.69816     0.20674
 0.250    0.92     0.69781     0.20674
 0.250    0.94     0.69753     0.20673
 0.250    0.96     0.69732     0.20673
 0.250    0.98     0.69720     0.20673

 ncall =   258
```

We can note the following details about this output.

- The ICs, $u_1(x, t = 0) =$ gaussian function, $u_2(x, t = 0) = 0$ for `ncase=2`, appear to be correct, including symmetry around $x = 0.5$ for $u_1(x, t = 0)$ (v. the Gaussian IC function programmed in Listing 2.2 to explain this symmetry); the numbers for $u_1(x, t = 0)$ can be easily checked.
- $u_1(x, t)$ varies with x (starting at $t = 0$) so that the x derivatives in eq. (2.2a) also vary with x (and are therefore nonzero). Thus, $\partial u_1/\partial t$ from eq. (2.2a) now changes with x and t as reflected in Table 2.4. For eq. (2.2b), the coupling between eqs. (2.2a) and (2.2b), and the nonzero derivative in x, gives the PDE $\dfrac{\partial u_2}{\partial t}$ a variation in x and t.
- The computational effort is modest with `ncall = 258`.
- The plots of $u_1(x, t)$ and $u_2(x, t)$ in Figs. 2.3 and 2.4 give a more complete picture than the abbreviated output in Table 2.4. $u_1(x, t)$ has the pronounced Gaussian function at $t = 0$ (the

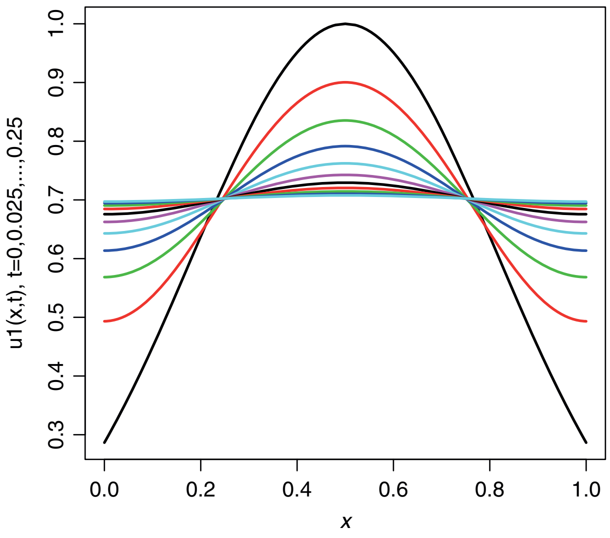

Figure 2.3 $u_1(x,t)$ versus x with t as a parameter, `ncase=2`.

IC for eq. (2.2a)). Through the effect of diffusion (expressed through the derivatives in x), $u_1(x,t)$ moves toward a uniform value with increasing t. Note the discontinuity between the IC (Gaussian function) at $x = 0, L$ and the BCs (eqs. (2.4)).

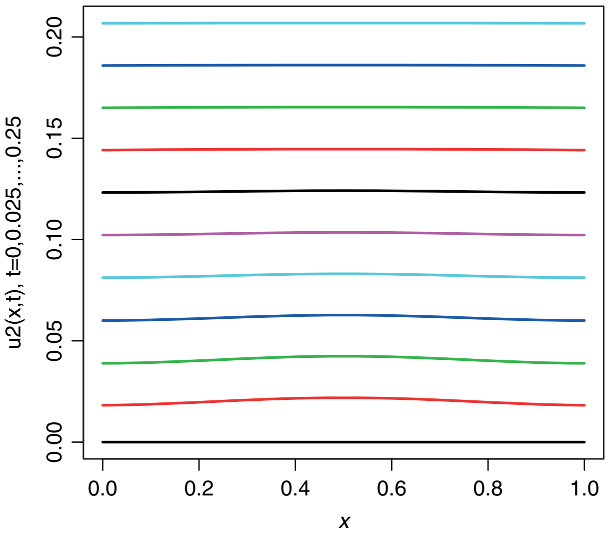

Figure 2.4 $u_2(x,t)$ versus x with t as a parameter, `ncase=2`.

At $t = 0$, the derivatives from the IC, $\partial u_1(x = 0, t = 0)/\partial x$ and $\partial u_1(x = L, t = 0)/\partial x$, are not zero (from the Gaussian function), whereas they are zero from BCs (eqs. 2.4). This discontinuity could be accommodated numerically because of the smoothing effect of the diffusion (expressed through the derivatives in x). This smoothing is a general property of parabolic PDEs.

The variation of $u_2(x, t)$ with x is small, and for increasing t, this variation essentially approaches zero as reflected in Table 2.4 (e.g., at $t = 0.250$).

For `ncase=3`, the abbreviated numerical output (from `ip=2` in Listing 2.2) is listed in Table 2.5.

We can note the following details about this output.

- The ICs, $u_1(x, t = 0) =$ unit step at $x = 0.025$, $u_2(x, t = 0) = 0$ for `ncase=3`, appear to be correct (the ICs at $t = 0$ are abbreviated to conserve space).
- $u_1(x, t)$ varies with x (starting as a unit step at $t = 0$) so that the x derivatives in eq. (2.2a) also vary with x (and are therefore nonzero). Thus, $\partial u_1/\partial t$ from eq. (2.2a) now changes with x and t as reflected in Table 2.5. For eq. (2.2b), the coupling between eqs. (2.2a) and (2.2b) and the nonzero derivative in x give the PDE $\dfrac{\partial u_2}{\partial t}$ a variation in x and t.
- The computational effort is modest with `ncall = 302`.
- The plots of $u_1(x, t)$ and $u_2(x, t)$ in Figs. 2.5 and 2.6 give a more complete picture than the abbreviated output in Table 2.5. $u_1(x, t)$ is the unit step at $t = 0$ (the IC for eq. (2.2a)). Through the effect of diffusion (expressed through the derivatives in x), $u_1(x, t)$ moves toward a uniform value with increasing t. The fact that this discontinuity could be accommodated numerically is due to the smoothing effect of the diffusion, a property of parabolic PDEs.

  Also, the IC (unit step) and the BCs are consistent at $x = 0, L$ (the derivatives in x from the IC and BCs (2.4) are zero, that is, $\partial u_1(x = 0, t = 0)/\partial x = \partial u_1(x = L, t = 0)/\partial x = 0$).

TABLE 2.5 Abbreviated output from Listing 2.2 with `ip=2` for `ncase=3`.

```
alpha = 10.00  d1 =  0.50   w1 =  1.00  mu =  0.10

> nrow(out)
[1] 11
> ncol(out)
[1] 103

    t       x     u1(x,t)     u2(x,t)
0.000    0.00     1.00000     0.00000
0.000    0.02     1.00000     0.00000
0.000    0.04     1.00000     0.00000
0.000    0.06     1.00000     0.00000
0.000    0.08     1.00000     0.00000
0.000    0.10     1.00000     0.00000
          .        .
          .        .
          .        .
 Output for x = 0.12 to 0.88 removed
          .        .
          .        .
          .        .
0.000    0.90     0.00000     0.00000
0.000    0.92     0.00000     0.00000
0.000    0.94     0.00000     0.00000
0.000    0.96     0.00000     0.00000
0.000    0.98     0.00000     0.00000
0.000    1.00     0.00000     0.00000
          .        .
          .        .
          .        .
 Output for t = 0.025 to 0.225 removed
          .        .
          .        .
          .        .
    t       x     u1(x,t)     u2(x,t)
0.250    0.00     0.43586     0.10479
```

(*continued*)

TABLE 2.5 (***Continued***)

```
  0.250    0.02     0.43513     0.10471
  0.250    0.04     0.43295     0.10449
  0.250    0.06     0.42935     0.10412
  0.250    0.08     0.42436     0.10360
  0.250    0.10     0.41806     0.10294
            .        .
            .        .
            .        .
   Output for x = 0.12 to 0.88 removed
            .        .
            .        .
            .        .
  0.250    0.90     0.05976     0.03969
  0.250    0.92     0.05834     0.03921
  0.250    0.94     0.05724     0.03885
  0.250    0.96     0.05646     0.03858
  0.250    0.98     0.05599     0.03843
  0.250    1.00     0.05584     0.03837

  ncall =   302
```

The smoothing of the unit step at $x = 0.025$ is clear in Figs. 2.5 and 2.6.

In conclusion, the interaction between eqs. (2.2a) and (2.2b) is rather complicated but can be studied numerically and graphically through experimentation with the R routines in Listings 2.1 and 2.2. For example, the three ICs for `ncase=1,2,3` could be easily studied. The same is true for variations in the PDE parameters and even for the number and form (structure) of the PDEs. This is demonstrated next where the number of PDEs is increased from two to three.

Also, the question of the accuracy of the numerical solutions was not addressed. To address this, the number of grid points in x could be changed (from 51) and the effect on the solutions observed, an application of h refinement. Similarly, the order of the spatial differentiators (e.g., `dss004,dss044`) could be changed (e.g., by calling

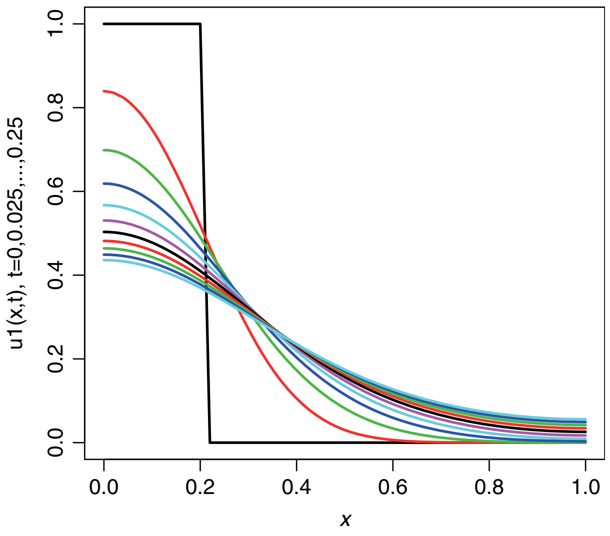

Figure 2.5 $u_1(x,t)$ versus x with t as a parameter, `ncase=3`.

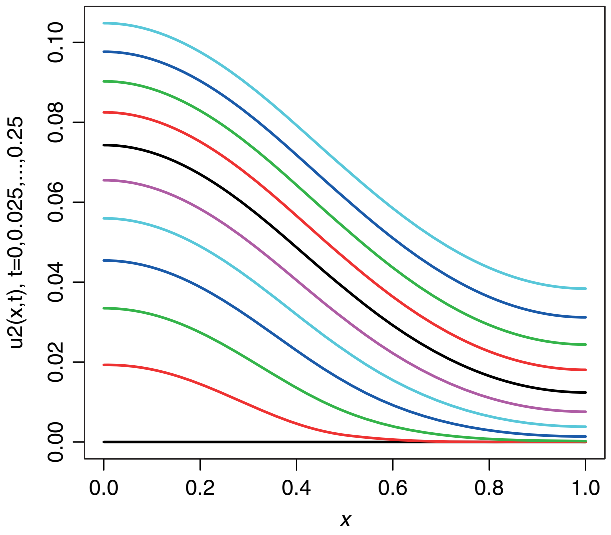

Figure 2.6 $u_2(x,t)$ versus x with t as a parameter, `ncase=3`.

routines with different orders in `p_form_1`), a form of p refinement. This form of error analysis should be a standard procedure for new PDE applications and is demonstrated in the analysis of the 3-PDE model that follows.

2.3 Three PDE Model

The model of eqs. (2.2) to (2.5) is now extended by adding a PDE for the stimulant Also, dimensional variables will be used rather than the dimensionless variables of eqs. (2.2) to (2.5). The resulting 3-PDE model follows ([2], p 264).

$$\frac{\partial u_1}{\partial t} = D_1 \nabla^2 u_1 - \nabla\left[\frac{k_1 u_1}{(k_2+u_2)^2}\nabla u_2\right] + k_3 u_1\left(\frac{k_4 u_3^2}{k_9+u_3^2} - u_1\right) \tag{2.6a}$$

$$\frac{\partial u_2}{\partial t} = D_2 \nabla^2 u_2 + k_5 u_3\left(\frac{u_1^2}{k_6+u_1^2} - k_7 u_1 u_2\right) \tag{2.6b}$$

$$\frac{\partial u_3}{\partial t} = D_3 \nabla^2 u_3 - k_8 u_1\left(\frac{u_3^2}{k_9+u_3^2}\right) \tag{2.6c}$$

where u_3 is the stimulant concentration.

The parameters of eqs. (2.6) with numerical values and units are listed in Table (2.6 [2], p 266, Table 5.1). Five parameters, u_{20}, k_5, k_6, k_7, and k_8, are not given by Murray and therefore are estimated as follows.

- u_2 is normalized by k_2 in eq. (5.19). Therefore, we take $u_{20} = k_2 = 5 \times 10^{-6}$ M.

TABLE 2.6 Parameters in eqs. (2.6).

Parameter	Value
k_1	3.9×10^{-9} M cm^2 sec^{-1}
k_2	5×10^{-6} M
k_3	1.62×10^{-9} hr ml^{-1} cell^{-1}
k_4	3.5×10^{8} cells ml^{-1}
k_9	4×10^{-6} M^2
D_1	$2 - 4 \times 10^{-6}$ cm^2 s^{-1}
D_2	8.9×10^{-6} cm^2 s^{-1}
D_3	$\approx 9 \times 10^{-6}$ cm^2 s^{-1}
u_{10}	10^9 cells ml^{-1}
u_{30}	$1 - 3 \times 10^{-3}$ M

- From the term $\dfrac{u_1^2}{k_6 + u_1^2}$ in eq. (2.6b), for $k_6 \approx u_1^2$, we take $k_6 = u_{10}^2 = 10^{18}$.
- From the term $\dfrac{u_3^2}{k_9 + u_3^2}$ in eq. (2.6c), for $k_9 \approx u_3^2$, we take $k_9 = u_{30}^2 = 10^{-6}$ (adjusted to $k_9 = 4 \times 10^{-6}$).
- From the term $k_5 u_3 \dfrac{u_1^2}{k_6 + u_1^2}$ in eq. (2.6b), if $\dfrac{\partial u_2}{\partial t} \approx 5 \times 10^{-9}$ M s^{-1}, we take $k_5(1 \times 10^{-3}) = 5 \times 10^{-9}$ or $k_5 = 5 \times 10^{-6}$ (adjusted to $k_5 = 5 \times 10^{-7}$).
- From the term $k_7 u_1 u_2$ in eq. (2.6b), we take $(k_7)(10^9)(5 \times 10^{-6}) = 5 \times 10^{-9}$ or $k_7 = 10^{-12}$ (adjusted to $k_7 = 10^{-13}$).
- From the term $k_8 u_1 \dfrac{u_3^2}{k_9 + u_3^2}$ in eq. (2.6c), if $\dfrac{\partial u_3}{\partial t} \approx 10^{-6}$ M s^{-1}, we take $k_8(10^9) = 10^{-6}$ or $k_8 = 10^{-15}$ (adjusted to $k_8 = 10^{-14}$).

In this way, u_{20}, k_5, k_6, k_7, and k_8 are estimated by an order-of-magnitude analysis and assumed values of the LHS derivatives of eqs. (2.6b) and (2.6c), followed possibly by some adjustment (and are not estimated from physical/chemical considerations or data). As a detail, the designation M in Table 2.6 is taken as molar, that is, $\mathrm{M} = \mathrm{gmol/cm^3} = \mathrm{gmol/ml}$ where g is gram.

Eqs. (2.6) each require an IC and two BCs. The IC for all three PDEs is taken as a Gaussian function.

$$u_1(x, t = 0) = u_{10} e^{-\lambda x^2};\; u_2(x, t = 0) = u_{20} e^{-\lambda x^2};$$
$$u_3(x, t = 0) = u_{30} e^{-\lambda x^2} \tag{2.7a,b,c}$$

where u_{10}, u_{20}, u_{30}, and λ are constants to be specified.

The zero-flux BCs (2.4) are again used for the three PDEs.

$$\frac{\partial u_1(x = 0, t)}{\partial x} = \frac{\partial u_1(x = L, t)}{\partial x} = 0 \tag{2.8a,b}$$
$$\frac{\partial u_2(x = 0, t)}{\partial x} = \frac{\partial u_2(x = L, t)}{\partial x} = 0 \tag{2.8c,d}$$
$$\frac{\partial u_3(x = 0, t)}{\partial x} = \frac{\partial u_3(x = L, t)}{\partial x} = 0 \tag{2.8e,f}$$

Eqs. (2.6) to (2.8) constitute the 3-PDE model. We now consider the numerical solution of these equations, starting with the ODE routine based on the method of lines (MOL).

2.3.1 ODE Routine

The following ODE routine for eqs. (2.6) and (2.8) parallels the 2-PDE ODE routine of Listing 2.1 (Listing 2.3 next).

```
  p_form_2=function(t,u,parms){
#
# Function p_form_2 computes the t derivative vector of
# the u1, u2, u3 vectors
#
# One vector to three vectors
  u1=rep(0,nx);u2=rep(0,nx);u3=rep(0,nx);
  for(i in 1:nx){
    u1[i]=u[i];
    u2[i]=u[i+nx];
    u3[i]=u[i+2*nx];
  }
#
# u1x, u2x, u3x
  u1x=dss004(xl,xu,nx,u1);
  u2x=dss004(xl,xu,nx,u2);
  u3x=dss004(xl,xu,nx,u3);
#
# Boundary conditions
  u1x[1]=0;u1x[nx]=0;
  u2x[1]=0;u2x[nx]=0;
  u3x[1]=0;u3x[nx]=0;
  nl=2;nu=2;
#
# u1xx, u2xx, u3xx
  u1xx=dss044(xl,xu,nx,u1,u1x,nl,nu);
  u2xx=dss044(xl,xu,nx,u2,u2x,nl,nu);
  u3xx=dss044(xl,xu,nx,u3,u3x,nl,nu);
#
# RHS terms
  term1=rep(0,nx);term2=rep(0,nx);term3=rep(0,nx);
  for(i in 1:nx){
    den=1/(k[2]+u2[i])^2;
    term1[i]=k[1]*u1[i]*den*u2xx[i];
    term2[i]=k[1]*den*u1x[i]*u2x[i];
    term3[i]=-2*k[1]*u1[i]*den/(k[2]+u2[i])*u2x[i]^2;
  {
#
# PDEs
  u1t=rep(0,nx);u2t=rep(0,nx);u3t=rep(0,nx);
  for(i in 1:nx){
    u1t[i]=D1*u1xx[i]-(term1[i]+term2[i]+term3[i])+
           k[3]*u1[i]*(k[4]*u3[i]^2/(k[9]+u3[i]^2)-u1[i]);
    u2t[i]=D2*u2xx[i]+k[5]*u3[i]*(u1[i]^2/(k[6]+u1[i]^2)-
           k[7]*u1[i]*u2[i]);
    u3t[i]=D3*u3xx[i]-k[8]*u1[i]*(u3[i]^2/(k[9]+u3[i]^2));
  }
#
# Three vectors to one vector
  ut=rep(0,3*nx);
  for(i in 1:nx){
    ut[i]     =u1t[i];
    ut[i+nx]  =u2t[i];
    ut[i+2*nx]=u3t[i];
  }
#
# Increment calls to p_form_2
  ncall <<- ncall+1;
#
# Return derivative vector
  return(list(c(ut)));
}
```

Listing 2.3 ODE routine p_form_2.

We can note the following details about p_form_2.

- The function is defined.

```
    p_form_2=function(t,u,parms){
  #
  # Function p_form_2 computes the t derivative vector
  # of the u1, u2, u3 vectors
```

The input arguments are in conformity with the R ODE integrators, in this case `lsodes` called by the main program discussed subsequently. `lsodes` in turn calls `p_form_2`. For the number of points in x, `nx=51`, `u` is a vector of `(3)(nx)=(3)(51)=153` elements, that is, `51` points in x for each of eqs. (2.6) (`153` ODEs in the MOL approximation of eqs. (2.6)). `nx` is defined numerically in the main program discussed subsequently. `parms` is unused.

- `u` is placed in three vectors, `u1,u2,u3`, to facilitate subsequent programming in terms of problem oriented variables, that is, the dependent variables of eqs. (2.6).

```
#
# One vector to three vectors
  u1=rep(0,nx);u2=rep(0,nx);u3=rep(0,nx);
  for(i in 1:nx){
    u1[i]=u[i];
    u2[i]=u[i+nx];
    u3[i]=u[i+2*nx];
  }
```

Note the ease with which a PDE can be added to the 2-PDE model of eqs. (2.2).

- $\partial u_1/\partial x, \partial u_2/\partial x, \partial u_3/\partial x$ are computed by the library differentiator `dss004`. `xl,xu` are set in the main program.

```
#
# u1x, u2x, u3x
  u1x=dss004(xl,xu,nx,u1);
  u2x=dss004(xl,xu,nx,u2);
  u3x=dss004(xl,xu,nx,u3);
```

- BCs (2.8) are programmed.

```
#
# Boundary conditions
  u1x[1]=0;u1x[nx]=0;
  u2x[1]=0;u2x[nx]=0;
  u3x[1]=0;u3x[nx]=0;
  nl=2;nu=2;
```

Again, since the BCs specify the partial derivatives at the boundaries, that is, $\partial u_1(x=0,t)/\partial x = 0$, $\partial u_1(x=L,t)/\partial x = 0$, and similar conditions for u_2, u_3, Neumann BCs are designated with `nl,nu`.

- $\partial^2 u_1/\partial x^2, \partial^2 u_2/\partial x^2, \partial^2 u_3/\partial x^2$ are computed by the library differentiator `dss044`.

```
#
# u1xx, u2xx, u3xx
  u1xx=dss044(xl,xu,nx,u1,u1x,nl,nu);
  u2xx=dss044(xl,xu,nx,u2,u2x,nl,nu);
  u3xx=dss044(xl,xu,nx,u3,u3x,nl,nu);
```

Note that the derivatives at the boundaries, `u1x[1],u1x[nx]`, `u2x[1],u2x[nx]`, and `u3x[1],u3x[nx]`, are inputs to `dss044`.

- The programming of three RHS terms of eq. (2.6a) is similar to the previous programming of eq. (2.5). A `for` with index `i` is used for the interval $0 \le x \le L$.

```
#
# RHS terms
  term1=rep(0,nx);term2=rep(0,nx);term3=rep(0,nx);
  for(i in 1:nx){
    den=1/(k[2]+u2[i])^2;
    term1[i]=k[1]*u1[i]*den*u2xx[i];
    term2[i]=k[1]*den*u1x[i]*u2x[i];
    term3[i]=-2*k[1]*u1[i]*den/(k[2]+u2[i])*u2x[i]^2;
  }
```

- Eqs (2.6) are programmed in a `for` with index `i`.

```
#
# PDEs
  u1t=rep(0,nx);u2t=rep(0,nx);u3t=rep(0,nx);
  for(i in 1:nx){
    u1t[i]=D1*u1xx[i]-(term1[i]+term2[i]+term3[i])+
           k[3]*u1[i]*(k[4]*u3[i]^2/(k[9]+u3[i]^2)
              -u1[i]);
    u2t[i]=D2*u2xx[i]+k[5]*u3[i]*(u1[i]^2/(k[6]+
       u1[i]^2)-k[7]*u1[i]*u2[i]);
    u3t[i]=D3*u3xx[i]-k[8]*u1[i]*(u3[i]^2/(k[9]+
       u3[i]^2));
  }
```

$D_1, D_2, D_3, k_3, k_4, k_5, k_6, k_7, k_8, k_9$ are defined numerically in the main program. The derivatives $\partial u_1/\partial t, \partial u_2/\partial t, \partial u_3/\partial t$ in `u1t,u2t,u3t` are the final result. They are passed to `lsodes` for the integration of the `153` ODEs.

- The derivatives `u1t,u2t,u3t` are placed in a single vector `ut` that is returned to `lsodes` for the integration of the `153` ODEs.

```
#
# Three vectors to one vector
  ut=rep(0,3*nx);
  for(i in 1:nx){
    ut[i]     =u1t[i];
    ut[i+nx]  =u2t[i];
    ut[i+2*nx]=u3t[i];
  }
```

An important check is that the number of dependent variable (e.g., in `u`) equals the number of derivatives (e.g., in `ut`) and that each dependent variable and its associated derivative are in the same positions in the dependent variable and derivative vectors. While this positioning is rather obvious in the present case, generally for models with relatively large numbers of ODEs, the correct positioning may not be so obvious and, of course, even one dependent variable misplaced with respect to its derivative will produce an incorrect solution.

- The number of calls to `p_form_2` is incremented and the value returned to the main program with `<<-`. The derivative vector is then returned from `p_form_2` as a list (as required by the R ODE integrators including `lsodes`).

```
#
# Increment calls to p_form_2
  ncall <<- ncall+1;
```

```
#
# Return derivative vector
  return(list(c(ut)));
}
```

The final } concludes p_form_2.

In summary, the model consisting of eqs. (2.6) and (2.8) is programmed in p_form_2. The only part of the model not included in this routine are the ICs for eqs. (2.6), that is, eqs. (2.7), to start the numerical solution. We now consider the effect of different parameter sets and some features of the numerical and graphical outputs produced by the main program. In addition, the RHS terms of eqs. (2.6) are computed and displayed to give further insight into the origin of various features of the PDE solutions.

2.3.2 Main Program

The main program that calls p_form_2 of Listing 2.3 through lsodes is similar to the main program of Listing 2.2.

```
#
# Access ODE integrator
  library("deSolve");
#
# Access functions for analytical solutions
  setwd("c:/R/bme_pde/chap2/v_3pde");
  source("p_form_2.R");
  source("pde_terms.R");
  source("dss004.R");
  source("dss044.R");
#
# Level of output
#
#   ip = 1 - graphical (plotted) solutions
#            (u1(x,t), u2(x,t), u3(x,t)) only
#
#   ip = 2 - numerical and graphical solutions
#
  ip=2;
```

```
#
# Parameters
#
# ncase = 1: Fickian diffusion
#
# ncase = 2: Fickian diffusion
#
#            Chemotaxis diffusion
#
# ncase = 3: Fickian diffusion
#
#            Chemotaxis diffusion
#
#            Source term in u1 PDE
#
# ncase = 4: Fickian diffusion
#
#            Chemotaxis diffusion
#
#            Source term in u1 PDE
#
#            Source term in u2 PDE
#
# ncase = 5: Fickian diffusion
#
#            Chemotaxis diffusion
#
#            Source term in u1 PDE
#
#            Source term in u2 PDE
#
#            Source term in u3 PDE
#
# ncase = 6: ncase = 5 plus the LHS and RHS terms of the
#            three pdes
#
#            Seven PDE RHS terms in chemo1 to chemo7
#
#            Three PDE LHS (derivatives in t) in
#            chemo8 to chemo10
#
```

```
  ncase=6;
#
# ncase = 1
  if(ncase==1){
    D1=2.0e-06;D2=8.9e-06;D3=9.0e-06;
    k=rep(0,9);
  }
#
# ncase = 2
  if(ncase==2){
    D1=2.0e-06;D2=8.9e-06;D3=9.0e-06;
    k=rep(0,9);
    k[1]=3.9e-09;k[2]=5.0e-06;
  }
#
# ncase = 3
  if(ncase==3){
    D1=2.0e-06;D2=8.9e-06;D3=9.0e-06;
    k=rep(0,9);
    k[1]=3.9e-09;k[2]=5.0e-06;k[3]=1.62e-09;
    k[4]=3.5e+08;k[9]=4.0e-06;
  }
#
# ncase = 4
  if(ncase==4){
    D1=2.0e-06;D2=8.9e-06;D3=9.0e-06;
    k=rep(0,9);
    k[1]=3.9e-09;k[2]=5.0e-06;k[3]=1.62e-09;
    k[4]=3.5e+08;k[9]=4.0e-06;
    k[5]=5.0e-07;k[6]=1.0e+18;k[7]=1.0e-13;
  }
#
# ncase = 5, 6
  if(ncase==5 || ncase==6){
    D1=2.0e-06;D2=8.9e-06;D3=9.0e-06;
    k=rep(0,9);
    k[1]=3.9e-09;k[2]=5.0e-06;k[3]=1.62e-09;
    k[4]=3.5e+08;k[9]= 4.0e-06;
    k[5]=5.0e-07;k[6]=1.0e+18;k[7]=1.0e-13;
    k[8]=1.0e-14;
  }
```

```
#
# Write parameters
  cat(sprintf("\n\n D1 = %8.3e  D2 = %8.3e  D3 = %8.3e \n",
          D1,D2,D3));
#
# Write heading
  if(ip==1){
    cat(sprintf("\n Graphical output only\n"));
  }
#
# Grid (in x)
  nx=51;xl=0;xu=1;
  xg=seq(from=xl,to=xu,by=0.02);
#
# Independent variable for ODE integration
  nout=11;
  tout=seq(from=0,to=5*3600,by=0.5*3600);
#
# Initial condition
  u0=rep(0,3*nx);
  u10=1.0e+08;u20=5.0e-06;u30=1.0e-03;
  for(i in 1:nx){
    u0[i]     =u10*exp(-5*xg[i]^2);
    u0[i+nx]  =u20*exp(-5*xg[i]^2);
    u0[i+2*nx]=u30*exp(-5*xg[i]^2);
  }
  t=0;
  ncall=0;
#
# ODE integration
  out=lsodes(y=u0,times=tout,func=p_form_2,parms=NULL)
  nrow(out)
  ncol(out)
#
# Arrays for plotting numerical solution
  u1_plot=matrix(0,nrow=nx,ncol=nout);
  u2_plot=matrix(0,nrow=nx,ncol=nout);
  u3_plot=matrix(0,nrow=nx,ncol=nout);
  for(it in 1:nout){
    for(ix in 1:nx){
       u1_plot[ix,it]=out[it,ix+1];
```

```
        u2_plot[ix,it]=out[it,ix+1+nx];
        u3_plot[ix,it]=out[it,ix+1+2*nx];
      }
    }
#
# Display numerical solution
  if(ip==2){
    for(it in 1:nout){
      cat(sprintf(
      "\n      t       x     u1(x,t)     u2(x,t)
         u3(x,t)\n"));
      for(ix in 1:nx){
        cat(sprintf("%7.2f%8.3f%12.3e%12.3e%12.3e\n",
        tout[it]/3600,xg[ix],u1_plot[ix,it],u2_plot[ix,it],
                             u3_plot[ix,it]));
      }
    }
  }
#
# Calls to ODE routine
  cat(sprintf("\n\n ncall = %5d\n\n",ncall));
#
# Plot u1
  par(mfrow=c(1,1));
  matplot(x=xg,y=u1_plot,type="l",xlab="x",
          ylab="u1(x,t), t=0,0.5,...,5",xlim=c(xl,xu),
             lty=1,main="u1(x,t); t=0,0.5,...,5;",lwd=2);
#
# Plot u2
  par(mfrow=c(1,1));
  matplot(x=xg,y=u2_plot,type="l",xlab="x",
          ylab="u2(x,t), t=0,0.5,...,5",xlim=c(xl,xu),
             lty=1,main="u2(x,t); t=0,0.5,...,5;",lwd=2);
#
# Plot u3
  par(mfrow=c(1,1));
  matplot(x=xg,y=u3_plot,type="l",xlab="x",
          ylab="u3(x,t), t=0,0.5,...,5",xlim=c(xl,xu),
             lty=1,main="u3(x,t); t=0,0.5,...,5;",lwd=2);
#
# Supplemental calculations
```

```
  if(ncase==6){
    chemo1_2d=matrix(0,nrow=nx,ncol=nout);
    chemo2_2d=matrix(0,nrow=nx,ncol=nout);
    chemo3_2d=matrix(0,nrow=nx,ncol=nout);
    chemo4_2d=matrix(0,nrow=nx,ncol=nout);
    chemo5_2d=matrix(0,nrow=nx,ncol=nout);
    chemo6_2d=matrix(0,nrow=nx,ncol=nout);
    chemo7_2d=matrix(0,nrow=nx,ncol=nout);
    chemo8_2d=matrix(0,nrow=nx,ncol=nout);
    chemo9_2d=matrix(0,nrow=nx,ncol=nout);
   chemo10_2d=matrix(0,nrow=nx,ncol=nout);
#
#   Step through t
    for(it in 1:nout){
      pde_terms(tout[it],c(u1_plot[,it],u2_plot[,it],
                           u3_plot[,it]));
      chemo1_2d[,it]=chemo1; chemo2_2d[,it]=chemo2;
      chemo3_2d[,it]=chemo3; chemo4_2d[,it]=chemo4;
      chemo5_2d[,it]=chemo5; chemo6_2d[,it]=chemo6;
      chemo7_2d[,it]=chemo7; chemo8_2d[,it]=chemo8;
      chemo9_2d[,it]=chemo9;chemo10_2d[,it]=chemo10;
    }
#
# Plot chemo1 (D1*u1_xx)
  par(mfrow=c(1,1));
  matplot(x=xg,y=chemo1_2d[,-1],type="l",xlab="x",
          ylab="D1*u1_{xx}, t=0.5,...,5",xlim=c(xl,xu),
             lty=1,main="D1*u1_{xx}; t=0.5,...,5;",lwd=2);
#
# Plot chemo8 (u1_t)
  par(mfrow=c(1,1));
  matplot(x=xg,y=chemo8_2d[,-1],type="l",xlab="x",
          ylab="u1_t, t=0.5,...,5",xlim=c(xl,xu),lty=1,
          main="u1_t; t=0.5,...,5;",lwd=2);
#
# Plot chemo9 (u2_t)
  par(mfrow=c(1,1));
  matplot(x=xg,y=chemo9_2d[,-1],type="l",xlab="x",
          ylab="u2_t, t=0.5,...,5",xlim=c(xl,xu),lty=1,
        main="u2_t; t=0.5,...,5;",lwd=2);
#
# Plot chemo10 (u3_t)
  par(mfrow=c(1,1));
  matplot(x=xg,y=chemo10_2d[,-1],type="l",xlab="x",
        ylab="u3_t, t=0.5,...,5",xlim=c(xl,xu),lty=1,
        main="u3_t; t=0.5,...,5;",lwd=2);
```

Listing 2.4 Main program for eqs. (2.6).

We can note the following points about this main program (with emphasis on the details pertaining to eqs. (2.6)).

- `p_form_2` of Listing 2.3 is accessed.

  ```
  setwd("c:/R/bme_pde/chap2/v_3pde");
  source("p_form_2.R");
  ```

- The level of output is specified with `ip` as in Listing 2.2. Then, six cases pertaining to parameter changes are programmed. For the subsequent discussion, `ncase=6` is used.

  ```
  #
  # ncase = 5, 6
    if(ncase==5 || ncase==6){
      D1=2.0e-06;D2=8.9e-06;D3=9.0e-06;
      k=rep(0,9);
      k[1]=3.9e-09;k[2]=5.0e-06;k[3]=1.62e-09;
      k[4]=3.5e+08;k[9]= 4.0e-06;
      k[5]=5.0e-07;k[6]=1.0e+18;k[7]=1.0e-13;
      k[8]=1.0e-14;
    }
  ```

  The `or` operator in R is `||`. The nine elements of the rate constant vector `k` are given specific values (in contrast with `ncase=1,2,3,4` for which some of the zero default values are used). Also, for `ncase=6`, the RHS terms of eqs. (2.6) are computed and displayed (discussed subsequently).

- The grid in x is defined as $0 \le x \le 1$, with `51` points so that $x = 0, 0.02, \ldots, 1$.

```
#
# Grid (in x)
  nx=51;xl=0;xu=1;
  xg=seq(from=xl,to=xu,by=0.02);
```

- The interval in t is $0 \le t \le (5)(3600)$ `s` (seconds), corresponding to a total interval of `5` h, with `11` values, $t = 0, (0.5)(3600), \ldots, (5)(3600)$, for the output.

```
#
# Independent variable for ODE integration
  nout=11;
  tout=seq(from=0,to=5*3600,by=0.5*3600);
```

The basic unit of time t for the calculations is `s` (seconds) because the model parameters are in `s` as listed in Table 2.6.

- Three ICs for eqs. (2.6) are Gaussian functions in accordance with eqs. (2.7).

```
#
# Initial condition
  u0=rep(0,3*nx);
  u10=1.0e+08;u20=5.0e-06;u30=1.0e-03;
  for(i in 1:nx){
    u0[i]     =u10*exp(-5*xg[i]^2);
    u0[i+nx]  =u20*exp(-5*xg[i]^2);
    u0[i+2*nx]=u30*exp(-5*xg[i]^2);
  }
  t=0;
  ncall=0;
```

Note that `u10=1.0e+08` and `u20=5.0e-06` so that the dependent variables range over approximately 14 orders of magnitude. But even with this wide variation in magnitude, `lsodes` is able to compute a numerical solution to eqs. (2.6). Also, this large range illustrates a common difference between dimensional variables such as u_1, u_2, u_3 of eqs. (2.6) and dimensionless variables such as u_1, u_2 of eqs. (2.2); the latter are often normalized to have values close to 1.

- The ODEs are integrated by `lsodes`, which is informed of the number of ODEs `((3)(51)=153)` by the length of the IC vector `u0`. Also, the output vector `tout` has length `11` (programmed previously).

```
#
# ODE integration
  out=lsodes(y=u0,times=tout,func=p_form_2,parms=NULL)
  nrow(out)
  ncol(out
```

  Note the use of the ODE routine `p_form_2` of Listing 2.3.

- The solution is put into three 2D arrays, `u1_plot,u2_plot`, `u3_plot`, for subsequent plotting.

```
#
# Arrays for plotting numerical solution
  u1_plot=matrix(0,nrow=nx,ncol=nout);
  u2_plot=matrix(0,nrow=nx,ncol=nout);
  u3_plot=matrix(0,nrow=nx,ncol=nout);
  for(it in 1:nout){
    for(ix in 1:nx){
       u1_plot[ix,it]=out[it,ix+1];
       u2_plot[ix,it]=out[it,ix+1+nx];
       u3_plot[ix,it]=out[it,ix+1+2*nx];
    }
  }
```

  Note again the offset of `1` in the second subscript of `out`, for example, `ix+1`, because the values of t are included in `out` as `out[it,1]`. Thus, `out` has the dimensions `out[nout,3*nx+1]= out[11,3*51+1]=out[11,154]`.

- For `ip=2`, the numerical solution is displayed in tabular form.

```
#
# Display numerical solution
  if(ip==2){
    for(it in 1:nout){
      cat(sprintf(
      "\n   t     x   u1(x,t)   u2(x,t)   u3(x,t)\n"));
```

```
        for(ix in 1:nx){
          cat(sprintf("%7.2f%8.3f%12.3e%12.3e%12.3e\n",
          tout[it]/3600,xg[ix],u1_plot[ix,it],u2_plot
             [ix,it],u3_plot[ix,it]));
        }
      }
    }
```

t in `tout` is converted to hours before it is displayed. Also, the `%12.3e` format is used because of the wide variation in the magnitude of the three dependent variables u_1, u_2, u_3 as discussed previously.

- The number of calls to `p_form_2` is displayed at the end of the solution as a measure of the computational effort to compute the solution. Then, u_1, u_2, u_3 are plotted separately by using `par(mfrow=c(1,1))`.

```
  #
  # Plot u1
    par(mfrow=c(1,1));
    matplot(x=xg,y=u1_plot,type="l",xlab="x",
            ylab="u1(x,t), t=0,0.5,...,5",xlim=c(xl,xu),
               lty=1,main="u1(x,t); t=0,0.5,...,5;",
                  lwd=2);

      etc. for u2, u3
```

- For `ncase=6`, `10` arrays are declared (preallocated) for various terms in eqs. (2.6). Two-dimensional arrays are used to facilitate subsequent plotting.

```
  #
  # Supplemental calculations
    if(ncase==6){
      chemo1_2d=matrix(0,nrow=nx,ncol=nout);
      chemo2_2d=matrix(0,nrow=nx,ncol=nout);
      chemo3_2d=matrix(0,nrow=nx,ncol=nout);
      chemo4_2d=matrix(0,nrow=nx,ncol=nout);
      chemo5_2d=matrix(0,nrow=nx,ncol=nout);
      chemo6_2d=matrix(0,nrow=nx,ncol=nout);
```

```
     chemo7_2d=matrix(0,nrow=nx,ncol=nout);
     chemo8_2d=matrix(0,nrow=nx,ncol=nout);
     chemo9_2d=matrix(0,nrow=nx,ncol=nout);
    chemo10_2d=matrix(0,nrow=nx,ncol=nout);
```

As an incidental programming note, two or more of these matrix statements could be placed in individual lines, but then the lines would be too long for listing and printing.

- A call to routine pde_terms (discussed subsequently) computes the values of the 10 terms chemo1 to chemo10 (returned from pde_terms). The for with index it is used for the nout=11 values of t in the interval $0 \leq t \leq (5)(3600)$.

```
#
#   Step through t
    for(it in 1:nout){
      pde_terms(tout[it],c(u1_plot[,it],u2_plot[,it],
                           u3_plot[,it]));
      chemo1_2d[,it]=chemo1; chemo2_2d[,it]=chemo2;
      chemo3_2d[,it]=chemo3; chemo4_2d[,it]=chemo4;
      chemo5_2d[,it]=chemo5; chemo6_2d[,it]=chemo6;
      chemo7_2d[,it]=chemo7; chemo8_2d[,it]=chemo8;
      chemo9_2d[,it]=chemo9;chemo10_2d[,it]=chemo10;
    }
```

The 10 terms are then placed in the 2D arrays. pde_terms performs the derivative calculations of p_form_2, but it also computes the 10 terms so that it is provided as a separate routine (i.e., a call to p_form_1 alone would not be sufficient). Also, note the use of the [,it] subscripts for the 51 values of x at each value of t.

- The 10 terms could all be plotted, but here only four are selected, chemo1, chem08, chemo9, and chemo10.

```
#
# Plot chemo1 (D1*u1_xx)
  par(mfrow=c(1,1));
  matplot(x=xg,y=chemo1_2d[,-1],type="l",xlab="x",
          ylab="D1*u1_{xx}, t=0.5,...,5",xlim=c(xl,xu),
             lty=1,main="D1*u1_{xx}; t=0.5,...,5;",
                lwd=2);

     etc. for chemo8, chemo9, chemo10
```

This level of output illustrates how PDEs can be studied in detail (rather than just considering the dependent variables, such as $u_1(x,t), u_2(x,t), u_3(x,t)$, as a function of the independent variables, such as x,t).

Note also that the first subscript in t is not used in the plotting of `chem01,chemo8,chemo9,chemo10`, that is, t starts at $t = 0.5$ rather than $t = 0$. For example, `chemo1_2d[,-1]` excludes `[,1]`. The reason for this form of plotting to exclude $t = 0$ is to avoid the discontinuity between the ICs of eqs. (2.7) and the BCs of eqs. (2.8) (at $x = 0, x = L = 1$). For $t > 0$, this discontinuity is smoothed (by diffusion) and therefore does not produce a disruption in the plotting (the reader could try `chemo1_2d` to observe the output at $t = 0$).

The routine `pde_terms` for the calculation of the `10` terms `chemo1` to `chemo10` follows next.

2.3.3 Supplemental Routine

`pde_terms` in Listing 2.5 calculates the `10` RHS terms `chemo1` to `chem10` for the plotting in the main program of Listing 2.4 discussed previously.

```
  pde_terms=function(t,u){
#
# Function pde_terms provides supplemental calculations
#    for the three-pde
# model based on the u1, u2, u3 vectors
#
# One vector to three vectors
  u1=rep(0,nx);u2=rep(0,nx);u3=rep(0,nx);
  for(i in 1:nx){
    u1[i]=u[i];
    u2[i]=u[i+nx];
    u3[i]=u[i+2*nx];
  }
#
# u1x, u2x
  u1x=dss004(xl,xu,nx,u1);
  u2x=dss004(xl,xu,nx,u2);
  u3x=dss004(xl,xu,nx,u3);
#
# Boundary conditions
  u1x[1]=0;u1x[nx]=0;
  u2x[1]=0;u2x[nx]=0;
  u3x[1]=0;u3x[nx]=0;
  nl=2;nu=2;
#
# u1xx, u2xx
  u1xx=dss044(xl,xu,nx,u1,u1x,nl,nu);
  u2xx=dss044(xl,xu,nx,u2,u2x,nl,nu);
  u3xx=dss044(xl,xu,nx,u3,u3x,nl,nu);
#
# RHS terms
  term1=rep(0,nx);term2=rep(0,nx);term3=rep(0,nx);
  for(i in 1:nx){
    den=1/(k[2]+u2[i])^2;
    term1[i]=k[1]*u1[i]*den*u2xx[i];
    term2[i]=k[1]*den*u1x[i]*u2x[i];
    term3[i]=-2*k[1]*u1[i]*den/(k[2]+u2[i])*u2x[i]^2;
  }
#
# PDEs
  u1t=rep(0,nx);u2t=rep(0,nx);u3t=rep(0,nx);
  for(i in 1:nx){
    u1t[i]=D1*u1xx[i]-(term1[i]+term2[i]+term3[i])+
           k[3]*u1[i]*(k[4]*u3[i]^2/(k[9]+u3[i]^2)-u1[i]);
    u2t[i]=D2*u2xx[i]+k[5]*u3[i]*(u1[i]^2/(k[6]+u1[i]^2)-
           k[7]*u1[i]*u2[i]);
    u3t[i]=D3*u3xx[i]-k[8]*u1[i]*(u3[i]^2/(k[9]+u3[i]^2));
  }
#
# PDE terms
  chemo1=rep(0,nx);chemo2=rep(0,nx);chemo3=rep(0,nx);
  chemo4=rep(0,nx);chemo5=rep(0,nx);chemo6=rep(0,nx);
  chemo7=rep(0,nx);chemo8=rep(0,nx);chemo9=rep(0,nx);
 chemo10=rep(0,nx);
  for(i in 1:nx){
    chemo1[i]=D1*u1xx[i];
    chemo2[i]=-(term1[i]+term2[i]+term3[i]);
    chemo3[i]=k[3]*u1[i]*(k[4]*u3[i]^2/(k[9]+u3[i]^2)-
       u1[i]);
    chemo4[i]=D2*u2xx[i];
    chemo5[i]=k[5]*u3[i]*(u1[i]^2/(k[6]+u1[i]^2)-k[7]
       *u1[i]*u2[i]);
    chemo6[i]=D3*u3xx[i];
    chemo7[i]=-k[8]*u1[i]*(u3[i]^2/(k[9]+u3[i]^2));
    chemo8[i]=u1t[i];
    chemo9[i]=u2t[i];
   chemo10[i]=u3t[i];
  }
#
# Return arrays of PDE terms
  chemo1 <<- chemo1;chemo2 <<- chemo2;chemo3 <<- chemo3;
  chemo4 <<- chemo4;chemo5 <<- chemo5;chemo6 <<- chemo6;
  chemo7 <<- chemo7;chemo8 <<- chemo8;chemo9 <<- chemo9;
  chemo10<<-chemo10;
}
```

Listing 2.5 `pde_terms` for the calculation of the terms in eqs. (2.6).

We can note the following details about `pde_terms`.

- `pde_terms` is essentially the same as `p_form_2` of Listing 2.3 up to and including the calculation of the derivative vectors `ut1,u2t,u3t`. Then, the `10` RHS terms of eqs. (2.6) are calculated as a function of x using a `for` with index `i`.

```
#
# PDE terms
  chemo1=rep(0,nx);chemo2=rep(0,nx);chemo3=rep(0,nx);
  chemo4=rep(0,nx);chemo5=rep(0,nx);chemo6=rep(0,nx);
  chemo7=rep(0,nx);chemo8=rep(0,nx);chemo9=rep(0,nx);
 chemo10=rep(0,nx);
  for(i in 1:nx){
    chemo1[i]=D1*u1xx[i];
    chemo2[i]=-(term1[i]+term2[i]+term3[i]);
    chemo3[i]=k[3]*u1[i]*(k[4]*u3[i]^2/(k[9]+u3[i]^2)
       -u1[i]);
    chemo4[i]=D2*u2xx[i];
    chemo5[i]=k[5]*u3[i]*(u1[i]^2/(k[6]+u1[i]^2)-k[7]
       *u1[i]*u2[i]);
    chemo6[i]=D3*u3xx[i];
    chemo7[i]=-k[8]*u1[i]*(u3[i]^2/(k[9]+u3[i]^2));
    chemo8[i]=u1t[i];
    chemo9[i]=u2t[i];
   chemo10[i]=u3t[i];
  }
```

For example, the diffusion term in eq. (2.6a), $D_1\partial^2 u_1/\partial x^2$, is calculated as `chemo1[i]=D1*u1xx[i]`. The derivatives in t, $\partial u_1/\partial t$, $\partial u_2/\partial t$, and $\partial u_3/\partial t$, are placed in `chemo8,chemo9,chemo10`, respectively. In this way, the dependent variables u_1, u_2, u_3 and their derivatives in t can be examined graphically.

- The `10` arrays are returned to the main program of Listing 2.4 by the operator `<<-`.

```
#
# Return arrays of PDE terms
  chemo1 <<- chemo1;chemo2 <<- chemo2;chemo3 <<-
     chemo3;
  chemo4 <<- chemo4;chemo5 <<- chemo5;chemo6 <<-
     chemo6;
  chemo7 <<- chemo7;chemo8 <<- chemo8;chemo9 <<-
     chemo9;
  chemo10<<-chemo10;
}
```

The final `}` concludes `pde_terms`. Note that the derivative vector of `p_form_2` in Listing 2.3, `ut`, is not formed and returned as a list (`pde_terms` is not used for the numerical integration of the `153` ODEs).

This concludes the programming of eqs. (2.6) to (2.8). The numerical and graphical outputs are reviewed next.

TABLE 2.7 **Abbreviated output from Listing 2.4 with `ncase=6`.**

```
D1 = 2.000e-06  D2 = 8.900e-06  D3 = 9.000e-06

> nrow(out)
[1] 11
> ncol(out)
[1] 154

     t        x     u1(x,t)     u2(x,t)     u3(x,t)
   0.00   0.000   1.000e+08   5.000e-06   1.000e-03
   0.00   0.020   9.980e+07   4.990e-06   9.980e-04
   0.00   0.040   9.920e+07   4.960e-06   9.920e-04
   0.00   0.060   9.822e+07   4.911e-06   9.822e-04
   0.00   0.080   9.685e+07   4.843e-06   9.685e-04
   0.00   0.100   9.512e+07   4.756e-06   9.512e-04
             .                     .
             .                     .
             .                     .
      Output from x = 0.120 to 0.880 removed
             .                     .
             .                     .
             .                     .
   0.00   0.900   1.742e+06   8.711e-08   1.742e-05
   0.00   0.920   1.452e+06   7.262e-08   1.452e-05
   0.00   0.940   1.206e+06   6.029e-08   1.206e-05
   0.00   0.960   9.972e+05   4.986e-08   9.972e-06
   0.00   0.980   8.213e+05   4.107e-08   8.213e-06
   0.00   1.000   6.738e+05   3.369e-08   6.738e-06
              .                     .
              .                     .
              .                     .
      Output from t = 0.5 to 4.5 removed
              .                     .
              .                     .
              .                     .
     t        x     u1(x,t)     u2(x,t)     u3(x,t)
   5.00   0.000   1.034e+07   2.480e-06   3.449e-04
   5.00   0.020   1.034e+07   2.479e-06   3.448e-04
   5.00   0.040   1.032e+07   2.476e-06   3.445e-04
   5.00   0.060   1.029e+07   2.471e-06   3.440e-04
```

TABLE 2.7 (*Continued*)

```
  5.00   0.080   1.025e+07   2.464e-06   3.434e-04
  5.00   0.100   1.020e+07   2.455e-06   3.426e-04
           .                     .
           .                     .
           .                     .
     Output from x = 0.120 to 0.880 removed
           .                     .
           .                     .
           .                     .
  5.00   0.900   4.944e+06   1.505e-06   2.465e-04
  5.00   0.920   4.901e+06   1.497e-06   2.455e-04
  5.00   0.940   4.868e+06   1.490e-06   2.448e-04
  5.00   0.960   4.844e+06   1.485e-06   2.442e-04
  5.00   0.980   4.830e+06   1.482e-06   2.439e-04
  5.00   1.000   4.825e+06   1.481e-06   2.438e-04

  ncall =   831
```

2.3.4 Numerical Solution

The numerical solutions from Listings 2.3, 2.4, and 2.5 are in Table 2.7 and Figs. 2.7 to 2.13.

Table 2.7 indicates the expected variation in t, $0 \le t \le 5$ (in hr) and x, $0 \le x \le 1$ (in cm). Also, the large range in the dependent variables, approximately 14 orders of magnitude, is clear. The computational effort even with this large range is quite modest, `ncall = 831`.

Generally, Figs. 2.7 to 2.9 indicate that $u_1(x,t), u_2(x,t), u_3(x,t)$ have the largest variation in x at $t = 0$ and then move toward uniform values in x with increasing t. The diffusion term in eq. (2.6a), $D_1 \partial^2 u_1(x,t)/\partial x^2$, in Fig. 2.10 (`chemo01`), has a complex form but approaches a uniform zero value with increasing t. Note that $t = 0.5, \ldots, 5$ as explained previously ($t = 0$ is not included). Generally, the derivatives in t (Figs. 2.11–2.13) have the largest nonzero values at $t = 0.5$ (from the Gaussian ICs of eqs. (2.7)) and approach a uniform zero value for $u_1(x, t \to \infty), u_2(x, t \to \infty)$, $u_3(x, t \to \infty)$.

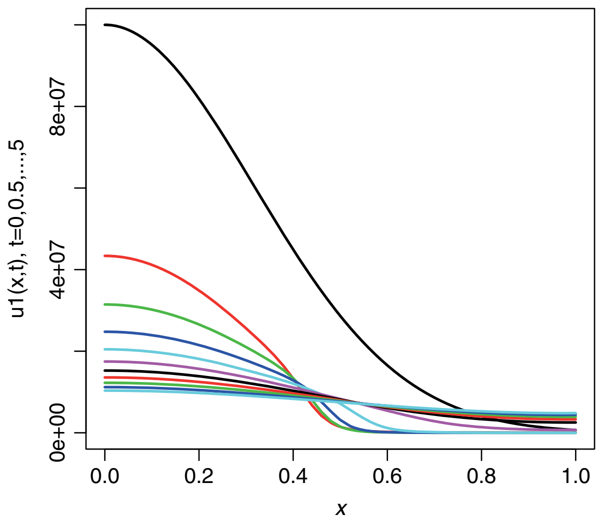

Figure 2.7 $u_1(x,t)$ versus x with t as a parameter, Gaussian IC.

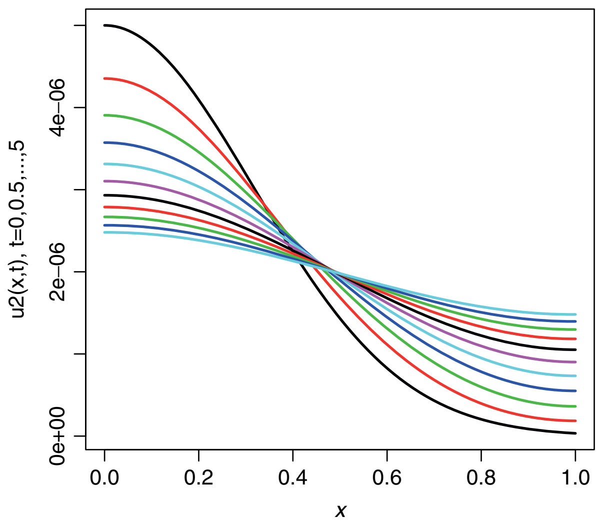

Figure 2.8 $u_2(x,t)$ versus x with t as a parameter, Gaussian IC.

In conclusion, this analysis demonstrates how PDEs can be studied in detail to understand the origin and properties of the solutions. We considered only the diffusion term in the eq. (2.6a), $D_1\partial^2 u_1(x,t)/\partial x^2$ in Fig. 2.10, but all of the other terms in the PDEs could have

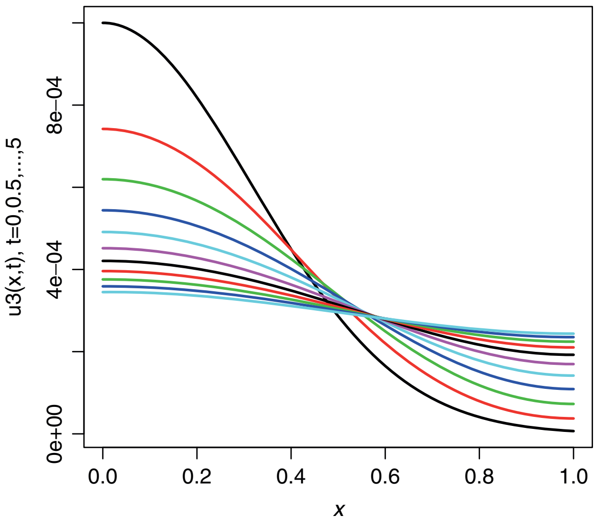

Figure 2.9 $u_3(x,t)$ versus x with t as a parameter, Gaussian IC.

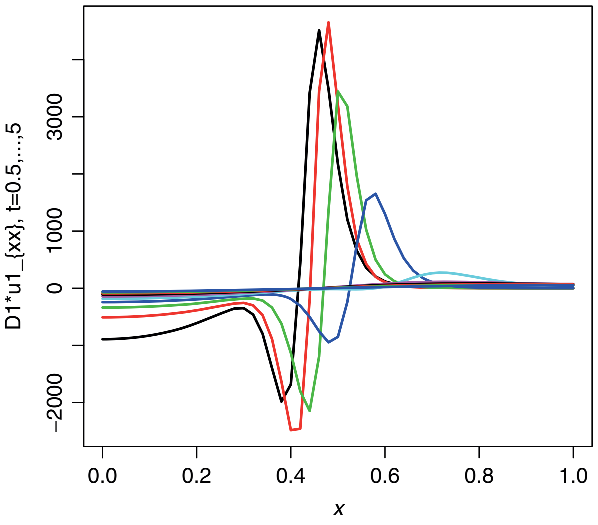

Figure 2.10 $D_1\partial^2 u_1(x,t)/\partial x^2$ versus x with $t = 0.5, \ldots, 5$ as a parameter.

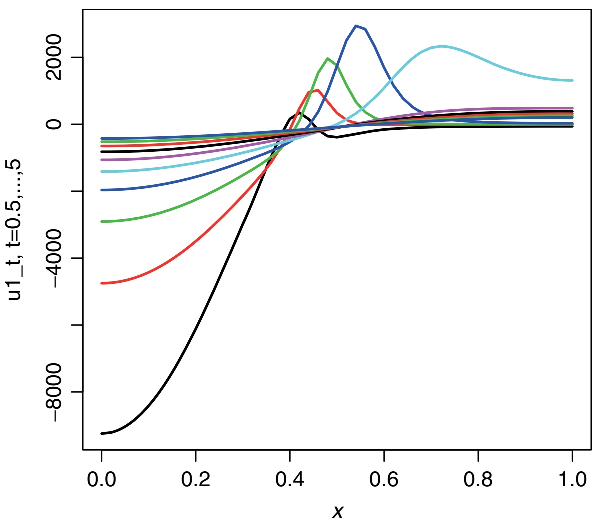

Figure 2.11 $\partial u_1(x,t)/\partial t$ versus x with $t = 0.5, \ldots, 5$ as a parameter.

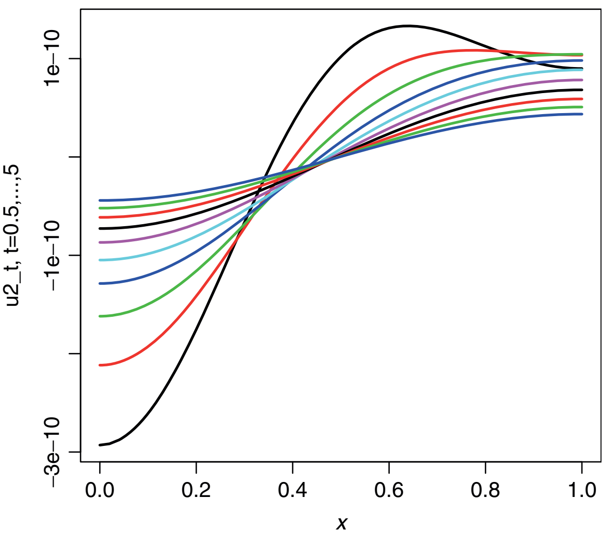

Figure 2.12 $\partial u_2(x,t)/\partial t$ versus x with $t = 0.5, \ldots, 5$ as a parameter.

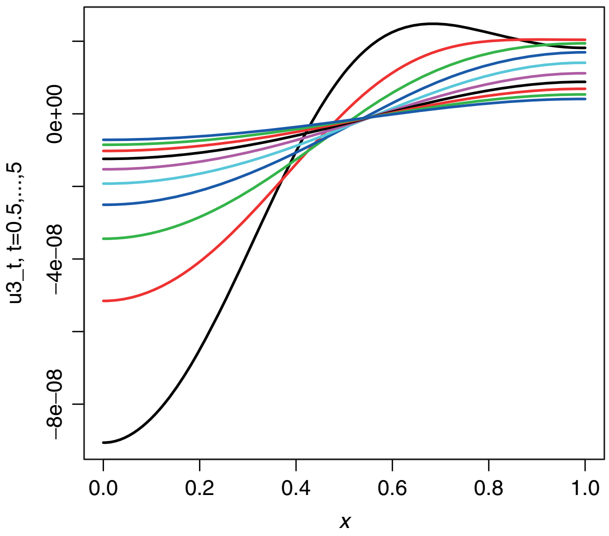

Figure 2.13 $\partial u_3(x,t)/\partial t$ versus x with $t = 0.5, \ldots, 5$ as a parameter.

been studied as well, that is, in `chemo2` to `chemo10`. Having detailed insight into the solutions generally provides guidance for the model development, revision, and interpretation, particularly for the reconciliation of the solutions with experimental data.

2.3.4.1 h Refinement The question of the accuracy of the preceding numerical solutions should also be addressed (to provide some assurance that the solutions have reasonable accuracy). To this end, we consider two procedures that can be applied without knowledge of an analytical solution (which is usually the situation for PDE models that are complicated so that analytical solutions are precluded).

First, we consider the effect of varying the number of points in x in the MOL approximation of the PDEs. This is easily achieved by changing the value of `nx` in Listing 2.4, which is accomplished, for example, with the following code.

```
#
# Grid (in x)
  nx=51;xl=0;xu=1;
  xg=seq(from=xl,to=xu,by=0.020);
```

Everything else in Listing 2.4 remains the same. We will not reconsider the previous numerical and graphical outputs for `nx=51` because the solution changes very little. This is clear from the following abbreviated numerical output for the two cases, `nx=41,51` (in Table 2.8).

In Table 2.8 only the solution for $x = 0, 0.1, \ldots, 1$ at $t = 5$ has been retained. The two solutions are essentially identical, suggesting `nx=51` was large enough to give good accuracy. Of course, the results of Table 2.8 do not prove the level of accuracy but only imply that it is acceptable.

Also, the computational effort is reduced slightly, from `ncall = 831 (nx = 51)` to `ncall = 754 (nx = 41)`. However, the reduced computation is actually more than indicated by the values of `ncall` because within the ODE routine `p_form_2` of Listing 2.3 the number of ODEs programmed in the calculations was also reduced. This suggests that some experimentation with the parameters of a numerical algorithm, for example, `nx`, can be worthwhile in reducing the computational requirements while maintaining acceptable accuracy.

The preceding analysis is usually termed *h refinement* because in the numerical analysis literature, the discrete step is often denoted with h.

2.3.4.2 p Refinement

As a second check on the accuracy, we can consider changing the order of the FD approximations of the derivatives in x in `p_form_2`. This is easily accomplished. For example, the fourth-order FD approximations in `dss004,dss044` can be replaced with sixth-order FD approximations by using `dss006,dss046`. This is demonstrated with the following changes.

```
#
# u1x, u2x, u3x
  u1x=dss004(xl,xu,nx,u1);
  u2x=dss004(xl,xu,nx,u2);
  u3x=dss004(xl,xu,nx,u3);
              .
              .
              .
#
```

TABLE 2.8 Abbreviated output from Listing 2.4 with `nx=41,51`.

```
nx=41

   t        x      u1(x,t)     u2(x,t)     u3(x,t)
5.00    0.000   1.034e+07   2.480e-06   3.449e-04
5.00    0.100   1.020e+07   2.455e-06   3.426e-04
5.00    0.200   9.782e+06   2.384e-06   3.358e-04
5.00    0.300   9.137e+06   2.273e-06   3.252e-04
5.00    0.400   8.333e+06   2.133e-06   3.116e-04
5.00    0.500   7.456e+06   1.979e-06   2.961e-04
5.00    0.600   6.603e+06   1.824e-06   2.803e-04
5.00    0.700   5.860e+06   1.686e-06   2.658e-04
5.00    0.800   5.294e+06   1.576e-06   2.541e-04
5.00    0.900   4.943e+06   1.505e-06   2.465e-04
5.00    1.000   4.825e+06   1.481e-06   2.438e-04

ncall =   754

nx=51

   t        x      u1(x,t)     u2(x,t)     u3(x,t)
5.00    0.000   1.034e+07   2.480e-06   3.449e-04
5.00    0.100   1.020e+07   2.455e-06   3.426e-04
5.00    0.200   9.783e+06   2.384e-06   3.358e-04
5.00    0.300   9.137e+06   2.273e-06   3.252e-04
5.00    0.400   8.333e+06   2.133e-06   3.116e-04
5.00    0.500   7.456e+06   1.979e-06   2.961e-04
5.00    0.600   6.603e+06   1.824e-06   2.803e-04
5.00    0.700   5.860e+06   1.686e-06   2.658e-04
5.00    0.800   5.294e+06   1.576e-06   2.541e-04
5.00    0.900   4.944e+06   1.505e-06   2.465e-04
5.00    1.000   4.825e+06   1.481e-06   2.438e-04

ncall =   831
```

```
# u1xx, u2xx, u3xx
  u1xx=dss044(xl,xu,nx,u1,u1x,nl,nu);
  u2xx=dss044(xl,xu,nx,u2,u2x,nl,nu);
  u3xx=dss044(xl,xu,nx,u3,u3x,nl,nu);
```

changed to

```
#
# u1x, u2x, u3x
  u1x=dss006(xl,xu,nx,u1);
  u2x=dss006(xl,xu,nx,u2);
  u3x=dss006(xl,xu,nx,u3);
              .
              .
              .
#
# u1xx, u2xx, u3xx
  u1xx=dss046(xl,xu,nx,u1,u1x,nl,nu);
  u2xx=dss046(xl,xu,nx,u2,u2x,nl,nu);
  u3xx=dss046(xl,xu,nx,u3,u3x,nl,nu);
```

Also, two `source` statements in Listing 2.4 have to be changed to access `dss006,dss046`.

Abbreviated output reflecting these changes is given as follows.

```
 dss004, dss044

     t        x      u1(x,t)     u2(x,t)     u3(x,t)
  5.00    0.000    1.034e+07   2.480e-06   3.449e-04
  5.00    0.100    1.020e+07   2.455e-06   3.426e-04
  5.00    0.200    9.783e+06   2.384e-06   3.358e-04
  5.00    0.300    9.137e+06   2.273e-06   3.252e-04
  5.00    0.400    8.333e+06   2.133e-06   3.116e-04
  5.00    0.500    7.456e+06   1.979e-06   2.961e-04
  5.00    0.600    6.603e+06   1.824e-06   2.803e-04
  5.00    0.700    5.860e+06   1.686e-06   2.658e-04
  5.00    0.800    5.294e+06   1.576e-06   2.541e-04
  5.00    0.900    4.944e+06   1.505e-06   2.465e-04
  5.00    1.000    4.825e+06   1.481e-06   2.438e-04

 ncall =   831
```

```
dss006, dss046

    t        x       u1(x,t)      u2(x,t)      u3(x,t)
 5.00    0.000    1.034e+07    2.480e-06    3.449e-04
 5.00    0.100    1.020e+07    2.455e-06    3.426e-04
 5.00    0.200    9.783e+06    2.384e-06    3.358e-04
 5.00    0.300    9.137e+06    2.273e-06    3.252e-04
 5.00    0.400    8.333e+06    2.133e-06    3.116e-04
 5.00    0.500    7.456e+06    1.979e-06    2.961e-04
 5.00    0.600    6.603e+06    1.824e-06    2.803e-04
 5.00    0.700    5.860e+06    1.686e-06    2.658e-04
 5.00    0.800    5.294e+06    1.576e-06    2.541e-04
 5.00    0.900    4.944e+06    1.505e-06    2.465e-04
 5.00    1.000    4.825e+06    1.481e-06    2.438e-04

ncall =   831
```

Briefly, there is no perceptible change in the solution suggesting that the fourth-order approximations of `dss004,dss044` provided acceptable solution accuracy. This procedure of changing the order of the MOL approximations is termed *p refinement* because in the numerical analysis literature, the approximation order is often designated with p. For `dss004,dss044`, $p = 4$, whereas for `dss006,dss046`, $p = 6$.

2.4 Conclusions

In conclusion, the apparent accuracy of the numerical solution is due in part to a relatively smooth problem, because eqs. (2.6) have diffusion terms that tend to smooth any rapid changes in x and the Gaussian IC of eqs. (2.7) is smooth. For more stringent problems, this level of accuracy might not be as easily achieved. In other words, each new problem should be investigated numerically, with careful documentation of the apparent accuracy.

This concludes the discussion of the pattern formation models of eqs. (2.2) to (2.8). The 2-PDE and 3-PDE models would be difficult to investigate analytically, but computation of the numerical solutions was straightforward.

References

[1] Keller, E.F., and L.A. Segel (1971), Traveling bands of chemotactic bacteria: a theoretical analysis, *J. Theor. Biol.*, **30**, 235–248.

[2] Murray, J.D. (2003), *Mathematical Biology, II: Spatial Models and Biomedical Applications*, 3rd edition, Springer-Verlag, Berlin, Heidelberg.