1

AN INTRODUCTION TO MOL ANALYSIS OF PDEs: WAVE FRONT RESOLUTION IN CHROMATOGRAPHY

This first chapter introduces a partial differential equation (PDE) model for chromatography which is a basic analytical method in biomedical science and engineering (BSME). For example, chromatography can be used to analyze a stream of various proteins through selective adsorption. Thus, the model can also be applied to adsorption as a basic procedure for separating biochemical species such as proteins. The computer implementation (programming, coding) of the model is in R[1].

The intent of this chapter is to

- Derive a basic chromatography PDE model, including the required initial conditions (ICs) and boundary conditions (BCs).
- Illustrate the coding of the model within the method of lines (MOL) through a series of R routines, including the use of library routines for integration of the PDE derivatives in time and space.
- Present the computed model solution in numerical and graphical (plotted) format.
- Discuss the features of the numerical solution and the performance of the algorithms used to compute the solution.
- Consider extensions of the model and the numerical algorithms.

[1]R is an open source scientific programming system that can be downloaded (at no cost) from the Internet. The R routines discussed in this book are available as a download from the author and the publisher.

Method of Lines PDE Analysis in Biomedical Science and Engineering, First Edition. William E. Schiesser.
© 2016 John Wiley & Sons, Inc. Published 2016 by John Wiley & Sons, Inc.
Companion website: www.wiley.com/go/Schiesser/PDE_Analysis

(1.1) 1D 2-PDE MODEL

The configuration of a chromatography column is illustrated in Fig. 1.1.

We can note the following details about the column represented in Fig. 1.1:

- The column is one dimensional (1D) with distance along the column, z, as the spatial (boundary value) independent variable. Time t is an initial value independent variable. A solid adsorbent is represented as spherical particles that fill the column. A fluid stream flows through the column in the interstices (voids) between the adsorbent particles. The flowing stream enters the base of the column at $z = 0$, and exits the top at $z = z_L$.
- The two PDE dependent variables are:
  - $u_1(z,t)$: concentration of the adsorbate (the chemical component to be processed) in the fluid stream.
  - $u_2(z,t)$: adsorbate concentration on the adsorbent.

  $u_1(z,t)$ and $u_2(z,t)$ are the PDE dependent variables. The PDEs that define these dependent variables are derived subsequently[2].
- The adsorbate enters the column at $z = 0$ with a prescribed (entering) concentration $u_{1e}(t)$ that serves as a boundary condition (BC) for the u_1 PDE[3]. Note that the boundary value can be a function of t.

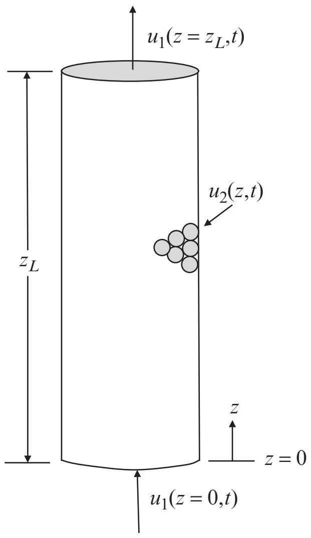

Figure 1.1 Diagram of a chromatographic column

[2]In accordance with the usual convention in PDE modeling, the dependent variables are designated with the letter u and a numerical subscript for each variable, e.g., $u_1(z,t)$, $u_2(z,t)$. The solution to the models PDEs will be the dependent variables in numerical form as a function of the independent variables, e.g., z, t.

[3]The term *boundary condition* follows from the use of a mathematical condition specified at the physical boundary of the system, in this case the adsorbate at $z = 0$, $u_1(z = 0, t) = u_{1e}(t)$.

- The exiting stream at $z = z_L$ has the concentration $u_1(z = z_L, t)$ which is a function of t. The t variation of this exiting stream is of primary interest when using the model. A plot of $u_1(z = z_L, t)$ against t is termed a *breakthrough curve*.
- An overall objective in formulating the model and computing numerical solutions is to determine $u_1(z = z_L, t)$, and in particular, how effective the chromatographic column is in alterting the entering stream with concentration $u_1(z = 0, t) = u_{1e}(t)$.

In summary, the numerical solution of the PDE model will give the dependent variables $u_1(z, t)$ and $u_2(z, t)$ as a function of z, t. $u_1(z = z_L, t)$ is a primary output from the model, that is, the outflow adsorbate concentration as a function of time.

A mass balance on the adsorbate stream[4] gives

$$\epsilon A \Delta z \frac{\partial u_1}{\partial t} = \epsilon A v u_1|_z - \epsilon A v u_1|_{z+\Delta z} - (1-\epsilon) A \Delta z (k_f u_1 (u_2^e - u_2) - k_r u_2) \quad (1.1a)$$

where

u_1	concentration of adsorbate in the flowing stream
u_2	concentration of adsorbate on the adsorbent
u_2^e	equilibrium (saturation) concentration of adsorbate on the adsorbent
t	time
z	axial position along the column
A	cross sectional area of column (transverse to z)
v	superficial velocity of flow
ϵ	void fraction of the adsorbent interstices
k_f	forward rate constant for the adsorbate transfer from the fluid to the adsorbent
k_r	reverse rate constant for the adsorbate transfer from the adsorbent to the flowing stream

Table 1.1: Variables and parameters of eq. (1.1a)

Eq. (1.1a) is a mass conservation balance for the flowing adsorbate with the terms explained further in the following comments.

LHS-1: $\epsilon A \Delta z \frac{\partial u_1}{\partial t}$ - accumulation of adsorbate in the incremental volume $\epsilon A \Delta z$. The CGS units of this term are (cm^2)(cm)(gmol/cm^3)(1/s) =gmol/s, that is, the accumulation of adsorbate per second within the incremental volume $\epsilon A \Delta z$. If the derivative

[4] A 3D PDE is derived in Appendix A that can be reduced to eq. (1.1a) as a special case.

$\frac{\partial u_1}{\partial t}$ is negative, the adsorbate is depleted (reduced). Also, some elaboration of the units of length is possible.

- ϵ: cm$^3_{fluid}$/cm$^3_{column}$ (so that the void fraction is not dimensionless)
- A: cm$^2_{column}$
- Δz: cm$_{column}$
- u_1: g-mol/cm$^3_{fluid}$

Thus, more detailed units of the LHS t derivative of eq. (1.1a) are:

(cm$^3_{fluid}$/cm$^3_{column}$)(cm$^2_{column}$)(cm$_{column}$)(g-mol/cm$^3_{fluid}$)(1/s)=gmol/s

The distinction between cm$_{fluid}$ and cm$_{column}$ (and later, cm$_{adsorbent}$) will not generally be retained in the subsequent discussion (only cm will be used), but this distinction should be kept in mind when analyzing units in the model.

RHS-1: $\epsilon A v u_1|_z$ - flow (by convection) of absorbate into the incremental volume at z. The units of this term are (cm^2)(cm/s)(gmol/cm^3) =gmol/s, that is, the flow of adsorbate per second into the incremental volume. Note that v has the units cm$_{column}$/s This is generally termed a *superficial* or *linear* velocity and is assumed constant across the chromatographic column (any wall effects are neglected).

RHS-2: $-\epsilon A v u_1|_{z+\Delta z}$ - flow (by convection) of absorbate out of the incremental volume at $z+\Delta z$. Again, the units of this term are (cm^2)(cm/s)(gmol/cm^3)=gmol/s, that is, the flow of adsorbate per second out of the incremental volume.

RHS-3: $-(1-\epsilon)A\Delta z(k_f u_1(u_2^e - u_2) - k_r u_2)$ - volumetric rate of adsorption (when this term is negative, adsorbate moves from the fluid to the adsorbent) or desorption (when this term is positive). The units of this term are (cm^2)(cm)(1/s)(gmol/cm^3)=gmol/s, that is, the transfer of adsorbate per second within the incremental volume.

Three additional points about this term can be observed.

- u_2^e and u_2 are volumetric (not surface) adsorbent concentrations with the units gmol/ cm$^3_{absorbent}$. k_f has the units cm$^3_{fluid}$/gmol-s and k_r has the units 1/s (explained next).

  By definition,

  $$\text{cm}^3_{fluid}\text{+cm}^3_{absorbent}\text{=cm}^3_{column}$$

  and

  $$\begin{aligned}(1\text{-}\epsilon)\rightarrow (1\text{-cm}^3_{fluid}/\text{cm}^3_{column}) &= (\text{cm}^3_{column}\text{-cm}^3_{fluid})/\text{cm}^3_{column}\\ &= \text{cm}^3_{adsorbent}/\text{cm}^3_{column}\end{aligned}$$

Then the units of the term $-(1-\epsilon)A\Delta z(k_f u_1(u_2^e - u_2) - k_r u_2)$ are:

$$(\text{cm}^3_{absorbent}/\text{cm}^3_{column})\ (\text{cm}^2_{column})(\text{cm}_{column})((\text{cm}^3_{fluid}/\text{gmol-s})$$
$$(\text{gmol/cm}^3_{fluid})(\text{gmol/cm}^3_{adsorbent})\text{-}(1/\text{s})(\text{gmol/cm}^3_{adsorbent}))\text{=gmol/s}$$

- The forward rate of adsorption, $k_f u_1(u_2^e - u_2)$, is usually termed a *logistic* rate. Note that it is nonlinear from the product of the two dependent variables, $u_1 u_2$, which means that an analytical solution to the PDE model is probably precluded, but a numerical solution can be easily programmed and calculated. Also, for $(u_2^e - u_2) > 0$ this forward rate is positive giving adsorption from this term in eq. (1.1a), and for $(u_2^e - u_2) < 0$ this term reflects desorption (when the adsorbate concentration u_2 exceeds the equilibrium adsorbent concentration, u_2^e).
- When $(k_f u_1(u_2^e - u_2) - k_r u_2) > 0$, adsorption takes place (with a reduction in $\dfrac{\partial u_1}{\partial t}$ from eq. (1.1a) since this term is multiplied by a minus). Conversely, when $(k_f u_1(u_2^e - u_2) - k_r u_2) < 0$, this term reflects desorption (and an increase in $\dfrac{\partial u_1}{\partial t}$ from eq. (1.1a)).

If eq. (1.1a) is divided by $\epsilon A\Delta z$,

$$\frac{\partial u_1}{\partial t} = -\frac{vu_1|_{z+\Delta z} - vu_1|_z}{\Delta z} - \frac{(1-\epsilon)}{\epsilon}(k_f u_1(u_2^e - u_2) - k_r u_2)$$

or for $\Delta z \to 0$,

$$\frac{\partial u_1}{\partial t} = -\frac{\partial (vu_1)}{\partial z} - \frac{(1-\epsilon)}{\epsilon}(k_f u_1(u_2^e - u_2) - k_r u_2) \qquad (1.1b)$$

Eq. (1.1b) is the PDE for the calculation of $u_1(z,t)$. For the subsequent analysis and programming, we will take v as independent of z so it can be taken outside the derivative in z (even though the transfer of adsorbate could affect v, but this will not be considered). v as a function of t is an interesting case that could be investigated through the use of eq. (1.1b). Note also that the column cross sectional area, A, canceled in going from eq. (1.1a) to eq. (1.1b), that is, we come to the somewhat unexpected conclusion that A does not appear in eq. (1.1b).

Also, in eq. (1.1b),

$$\frac{(1-\epsilon)}{\epsilon} \to \frac{cm^3_{adsorbent}/cm^3_{column}}{cm^3_{fluid}/cm^3_{column}} = \frac{cm^3_{adsorbent}}{cm^3_{fluid}}$$

as expected for consistent units in eq. (1.1b), that is, the units in the various terms in eq. (1.1b) are gmol/cm$^3_{fluid}$-s since eq. (1.1b) is a mass balance on the fluid.

A PDE for u_2 follows from an analogous mass balance for the adsorbent. The starting point is

$$(1-\epsilon)A\Delta z\frac{\partial u_2}{\partial t} = (1-\epsilon)A\Delta z(k_f u_1(u_2^e - u_2) - k_r u_2) \qquad (1.2a)$$

Division by $(1-\epsilon)A\Delta z$ gives

$$\frac{\partial u_2}{\partial t} = k_f u_1(u_2^e - u_2) - k_r u_2 \tag{1.2b}$$

Note that the adsorption terms in eqs. (1.1b) and (1.2b) are opposite in sign which indicates that the rate absorbate leaves (or enters) the fluid stream equals the rate adsorbate is transferred to (or leaves) the adsorbent. Also, the LHS and RHS terms in eq. (1.2b) have the units gmol/cm$^3_{adsorbent}$-s, since eq. (1.2b) is a mass balance on the adsorbent in an incremental volume $(1-\epsilon)A\Delta z$. Again, A cancels in going from eq. (1.2a) to eq. (1.2b).

Eqs. (1.1b) and (1.2b) are a 2×2 (two equations in two unknowns) for the concentrations u_1, u_2. One other variable, u_2^e, appears in the adsorption rate in these two PDEs. This adsorbent equilibrium concentration might be assumed to be a constant, for example, corresponding to a monolayer of the adsorbate on the adsorbent. Or u_2^e can be considered a variable from an equilibrium relation such as, for example, a *Langmuir isotherm* of the form

$$u_2^e = \frac{c_1 u_1}{1 + c_2 u_1} \tag{1.3}$$

where c_1, c_2 are constants typically measured experimentally.

Eq. (1.1b) is first order in t and z (and is termed a *first-order, hyperbolic PDE*). Therefore, it requires one *initial condition* (IC)[5] and one *boundary condition* (BC).[6,7]

The IC is taken as

$$u_1(z, t=0) = f_1(z) \tag{1.4a}$$

The BC is taken as

$$u_1(z=0, t) = g_1(t) \tag{1.4b}$$

where $f_1(z)$ and $g_1(t)$ are prescibed functions of z and t, respectively.

Eq. (1.2b) is first order in t so it requires one IC

$$u_2(z, t=0) = f_2(z) \tag{1.4c}$$

[5]An initial condition defines the value of the dependent variable, $u_1(z,t)$, for a particular value of the initial value independent variable, t, typically time in a physical application. t is defined over an open interval $t_0 \le t \le \infty$. The initial or beginning value, t_0, in the present case will be taken as $t_0 = 0$. Note that t can continue without limit.

[6]A boundary condition defines the value of the dependent variable, $u_1(z,t)$, for a particular value of the boundary value independent variable, z, typically at a physical boundary in an application. z can be defined over a finite interval, $z_0 \le z \le z_L$, a semi infinite interval, $z_0 \le z \le \infty$, or a fully infinite interval, $-\infty \le z \le \infty$. In the present case, we will use a finite interval corresponding to the length of the chromatographic column, $0 \le z \le z_L$ where z_L is the specified length of the column.

[7]In general, the number of required ICs equals the order of the highest order derivative in the initial value variable, one IC in the case of eq. (1.1b) for the first order derivative $\frac{\partial u_1}{\partial t}$. The number of required BCs equals the order of the highest order derivative in the boundary value variable, one BC in the case of eq. (1.1b) for the first order derivative $\frac{\partial u_1}{\partial z}$.

Eq. (1.4b) is a *Dirichlet* BC since the dependent variable u_1 is specified at the boundary $z = 0$. Other types of BCs are discussed in subsequent chapters.

Eqs. (1.1) to (1.4) constitute the PDE model for the chromatographic column. We next consider the programming of these equations within the MOL framework.

(1.2) MOL routines

The discussion of the routines for eqs. (1.1) to (1.4) starts with the main program.

(1.2.1) Main program

The listing of the main program follows.

```
#
# Delete previous workspaces
  rm(list=ls(all=TRUE))
#
#   1D, one component, chromatography model
#
#   The ODE/PDE system is
#
#   u1_t = -v*u1_z - (1 - eps)/eps*rate                      (1.1b)
#
#   u2_t = rate                                              (1.2b)
#
#   rate = kf*u1*(u2eq - u2) - kr*u2
#
#   u2eq = c1*u1/(1 + c2*u1)                                 (1.3)
#
#   Boundary condition
#
#     u1(z=0,t) = step(t)                                    (1.4b)
#
#   Initial conditions
#
#     u1(z,t=0) = 0                                          (1.4a)
#
#     u2(z,t=0) = 0                                          (1.4c)
#
#   The method of lines (MOL) solution for eqs. (1.1) to
#   (1.4) is coded below.  Specifically, the spatial
#   derivative in the fluid balance, u1_z in eq. (1.1b),
#   is replaced by one of four approximations as selected
#   by the variable ifd.
#
# Access ODE integrator
```

```
  library("deSolve");
#
# Access files
  setwd("g:/chap1");
  source("pde_1.R") ;source("step.R")  ;
  source("dss004.R");source("dss012.R");
  source("dss020.R");source("vanl.R")  ;
  source("max3.R")  ;
#
# Step through cases
  for(ncase in 1:2){
#
#   Model parameters
     v=1; eps=0.4; u10=0; u20=0;
    c1=1;    c2=1; zL=50;  n=41;
    if(ncase==1){ kf=0; kr=0; }
    if(ncase==2){ kf=1; kr=1; }
#
# Select an approximation for the convective derivative u1z
#
#   ifd = 1: Two point upwind approximation
#
#   ifd = 2: Centered approximation
#
#   ifd = 3: Five point, biased upwind approximation
#
#   ifd = 4: van Leer flux limiter
#
  ifd=1;
#
# Level of output
#
#   Detailed output   - ip = 1
#
#   Brief (IC) output - ip = 2
#
  ip=1;
#
# Initial condition
  u0=rep(0,2*n);
  for(i in 1:n){
      u0[i]=u10;
    u0[i+n]=u20;
  }
  t0=0;tf=150;nout=51;
  tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
  ncall=0;
#
# ODE integration
  out=ode(func=pde_1,times=tout,y=u0);
#
# Store solution
  u1=matrix(0,nrow=nout,ncol=n);
  u2=matrix(0,nrow=nout,ncol=n);
  t=rep(0,nout);
  for(it in 1:nout){
  for(iz in 1:n){
    u1[it,iz]=out[it,iz+1];
    u2[it,iz]=out[it,iz+1+n];
  }
    t[it]=out[it,1];
  }
#
# Display ifd, ncase
  cat(sprintf("\n ifd = %2d   ncase = %2d",ifd,ncase));
#
# Display numerical solution
  if(ip==1){
  cat(sprintf(
    "\n\n     t     u1(z=zL,t)  rate(z=zL,t)\n"));
  u2eq=rep(0,nout);rate=rep(0,nout);
  for(it in 1:nout){
    u2eq[it]=c1*u1[it,n]/(1+c2*u1[it,n]);
    rate[it]=kf*u1[it,n]*(u2eq[it]-u2[it,n])-kr*u2[it,n];
    cat(sprintf(
      "%7.2f%12.4f%12.4f\n",t[it],u1[it,n],rate[it]));
  }
  }
#
# Store solution for plotting
  u1plot=rep(0,nout);tplot=rep(0,nout);
  for(it in 1:nout){
    u1plot[it]=u1[it,n];
     tplot[it]=t[it];
  }
#
# Calls to ODE routine
  cat(sprintf("\n ncall = %4d\n",ncall));
#
# Plot for u1(z=zL,t)
# ncase = 1
  if(ncase==1){
```

```
  par(mfrow=c(1,1))
  plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t)",
    lwd=2,main="u1(z=zL,t) vs t, ncase=1\n
    line - anal, o - num",type="l",
    xlim=c(0,tplot[nout]));#,ylim=c(0,1));
  points(tplot,u1plot, pch="o",lwd=2);
  }
#
# Analytical solution, ncase=1
  if(ncase==1){
  u1expl=rep(0,nout);
  for(it in 1:nout){
    u1expl[it]=step(tplot[it],zL,v);
  }
  lines(tplot,u1expl,lwd=2,type="l");
  }
#
# ncase = 2
  if(ncase==2){
  par(mfrow=c(1,1))
  plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t)",
    lwd=2,main="u1(z=zL,t) vs t, ncase=2",
    type="l",xlim=c(0,tplot[nout]));#,ylim=c(0,1));
  points(tplot,u1plot, pch="o",lwd=2);
  }
#
# Next case
  }
```

Listing 1.1: Main program pde_1_main for eqs. (1.1) to (1.4)

We can note the following details about pde_1_main.

- Previous files are cleared and a series of documentation comments for the ODE/PDE system is included that restates eqs. (1.1) to (1.4) in the text.

```
#
# Delete previous workspaces
  rm(list=ls(all=TRUE))
#
#   1D, one component, chromatography model
#
#   The ODE/PDE system is
#
#   u1_t = -v*u1_z - (1 - eps)/eps*rate                   (1.1b)
#
```

```
#   u2_t = rate                                          (1.2b)
#
#   rate = kf*u1*(u2eq - u2) - kr*u2
#
#   u2eq = c1*u1/(1 + c2*u1)                             (1.3)
#
#   Boundary condition
#
#     u1(z=0,t) = step(t)                                (1.4b)
#
#   Initial conditions
#
#     u1(z,t=0) = 0                                      (1.4a)
#
#     u2(z,t=0) = 0                                      (1.4c)
#
#   The method of lines (MOL) solution for eqs. (1.1) to
#   (1.4) is coded below.  Specifically, the spatial
#   derivative in the fluid balance, u1_z in eq. (1.1b),
#   is replaced by one of four approximations as selected
#   by the variable ifd.
```

The IC and BC functions of eqs. (1.4), $f_1(z) = 0, g_1(t) = h(t), f_2(z) = 0$, are explained subsequently ($h(t)$ is the *unit step function* or *Heaviside function*).

- The R ODE integrator library `deSolve` and a series of routine discussed subsequently are accessed.

```
#
# Access ODE integrator
  library("deSolve");
#
# Access files
  setwd("g:/chap1");
  source("pde_1.R") ;source("step.R")   ;
  source("dss004.R");source("dss012.R");
  source("dss020.R");source("vanl.R")   ;
  source("max3.R")  ;
```

The set working directory, `setwd`, will have to be edited for the local computer. Note the forward slash, /, rather than the usual backslash, \. The `source` utility is used to select individual files that make up the complete code for the model of eqs. (1.1) to (1.4). These files are explained subsequently.

- A `for` is used to step through a series of (two) cases, `ncase=1,2`.

```
#
# Step through cases
```

```
  for(ncase in 1:2){
#
#   Model parameters
     v=1; eps=0.4; u10=0; u20=0;
    c1=1;    c2=1; zL=50;  n=41;
    if(ncase==1){ kf=0; kr=0; }
    if(ncase==2){ kf=1; kr=1; }
```

The parameters in eqs. (1.1) to (1.4) for each case are defined numerically. In particular, for ncase=1, no adsorption takes place so that the fluid with adsorbate concentration $u_1(z,t)$ merely flows through the column and there is no up take of adsorbate with concentration $u_2(z,t)$ onto the adsorbent. This special condition is used to check the coding of the model as discussed subsequently. For ncase=2, the effect of adsorbate transfer to the adsorbent can be observed in the fluid outlet with concentration $u_1(z=z_L,t)$.

- An approximation for the spatial derivative $\frac{\partial u_1}{\partial z}$ in eq. (1.1b) (v constant and therefore outside of the derivative) is selected with index ifd. The performance of the four approximations is discussed subsequently.

```
#
# Select an approximation for the convective derivative u1z
#
#   ifd = 1: Two point upwind approximation
#
#   ifd = 2: Centered approximation
#
#   ifd = 3: Five point, biased upwind approximation
#
#   ifd = 4: van Leer flux limiter
#
  ifd=1;
```

- A level of numerical output is selected with ip. Initially, ip=1 is used to give detailed numerical output along with graphical (plotted) output. ip=2 can be used to give only graphical output (when experimenting with the model).

```
#
# Level of output
#
#   Detailed output   - ip = 1
#
#   Brief (IC) output - ip = 2
#
  ip=1;
```

- ICs (1.4a) and (1.4c) are programmed as zero (*homogeneous*) ICs (since `u10 = u20 = 0`). Note that these ICs are placed in a single vector or 1D array `u0` as required by the ODE integrator `ode` discussed next. This vector is first declared (allocated, sized) with the utility `rep` (`2*n = 2*41 = 82` zero elements).

```
#
# Initial condition
  u0=rep(0,2*n);
  for(i in 1:n){
      u0[i]=u10;
    u0[i+n]=u20;
  }
  t0=0;tf=150;nout=51;
  tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
  ncall=0;
```

  The time scale is defined as $0 \leq t \leq 150$ with `nout=51` points in t for the numerical solution. The utility `seq` is used to define the 51 values $t = 0, 3, 6, ..., 150$. Finally, the counter for the calls to the ODE routine `pde_1` is initialized. The use of this counter is discussed later.

- The `2*n = 82` ODEs are integrated numerically with the library integrator `ode` (part of the `deSolve` library specified previously).

```
#
# ODE integration
  out=ode(func=pde_1,times=tout,y=u0);
```

  The input arguments for `ode` require some explanation.

  – The routine for the MOL/ODEs that approximate PDEs (1.1b), (1.2b), `pde_1`, is declared for the parameter `func` (which is a reserved argument name). `func` does not have to be the first input argument, but by convention, it usually is when calling one of the R integrators (`ode` in this case).

  – The vector of output values of t, `tout`, (defined previously) is assigned to the input argument `times`. Again, `times` is a reserved name and can be placed anywhere in the input argument list.

  – The IC vector, `u0`, is assigned to the parameter `y`. The length of this IC vector tells `ode` how many ODEs are to be integrated, in this case `2*n = 82`. Note that the number of ODEs is not specified explicitly in the input argument list.

  Numerical solutions to eqs. (1.1) to (1.4) are returned by `ode` in the 2D array `out`. The content of this solution array is explained next. The various ODE integrators in `deSolve` generally follow this format.

- The numerical solution is placed in matrices and a vector. These arrays are first declared with the utilities `matrix` (for u_1, u_2 of eqs. (1.1b), (1.2b)) and `rep` (for t of eqs. (1.1b) and (1.2b)).

```
#
# Store solution
  u1=matrix(0,nrow=nout,ncol=n);
  u2=matrix(0,nrow=nout,ncol=n);
  t=rep(0,nout);
  for(it in 1:nout){
  for(iz in 1:n){
    u1[it,iz]=out[it,iz+1];
    u2[it,iz]=out[it,iz+1+n];
  }
    t[it]=out[it,1];
  }
```

A pair of nested `for`s is used to place the numerical solutions in `u1,u2`. The outer `for` with index `it` steps through t for $0 \le t \le 150$. The inner `for` with index `iz` steps through z for $0 \le z \le z_L$ with $z_L = 50$ (defined previously) and a spatial increment $(50 - 0)/(41 - 1) = 1.25$, that is, $z = 0, 1.25, 2.50, ..., 50$ (based on `n=41` points in z).

The solution array `out` has the dimensions `out(nout,2*n+1) = out(51,82+1)`, that is, 82 ODEs at `nout=51` points in t (including $t = 0$). The offset of 1 in `iz+1,iz+1+n,2*n+1,82+1` reflects the additional space for t, so that `out[it,1]` contains the 51 values of t. This ordering of the output array `out` is a unique feature of the ODE integrators in `deSolve`, including `ode`.

- The index for the spatial differentiator, `ifd`, and the value of `ncase` are displayed.

```
#
# Display ifd, ncase
  cat(sprintf("\n ifd = %2d   ncase = %2d",ifd,ncase));
```

- For `ip=1`, the numerical solution is displayed as (1) t, (2) $u_1(z = z_L, t)$, and (3) $rate(t) = k_f u_1(z = z_L, t)(u_2^e - u_2(z = z_L, t)) - k_r u_2(z = z_L, t)$ (note the use of `n` for $z = z_L$). Vectors are first defined with the utility `rep`.

```
#
# Display numerical solution
  if(ip==1){
  cat(sprintf(
    "\n\n     t     u1(z=zL,t)  rate(z=zL,t)\n"));
  u2eq=rep(0,nout);rate=rep(0,nout);
  for(it in 1:nout){
    u2eq[it]=c1*u1[it,n]/(1+c2*u1[it,n]);
    rate[it]=kf*u1[it,n]*(u2eq[it]-u2[it,n])-kr*u2[it,n];
    cat(sprintf(
      "%7.2f%12.4f%12.4f\n",t[it],u1[it,n],rate[it]));
  }
  }
```

u_2^e = `u2eq` is computed from the isotherm of eq. (1.3).

- $u_1(z = z_L, t)$, t are stored for subsequent plotting.

```
#
# Store solution for plotting
  u1plot=rep(0,nout);tplot=rep(0,nout);
  for(it in 1:nout){
    u1plot[it]=u1[it,n];
     tplot[it]=t[it];
  }
```

- At the end of the solution (after the call to `ode`), the number of calls to the MOL/ODE routine `pde_1` is displayed (this routine is discussed next).

```
#
# Calls to ODE routine
  cat(sprintf("\n ncall = %4d\n",ncall));
```

- $u_1(z = z_L, t)$ is plotted against t for `ncase=1` (no adsorption). For this case, an analytical solution is available that is plotted as a solid line while the numerical solution is plotted as points on a solid line (this is clear in Fig. 1.2).

```
#
# Plot for u1(z=zL,t)
# ncase = 1
  if(ncase==1){
  par(mfrow=c(1,1))
  plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t)",
    lwd=2,main="u1(z=zL,t) vs t, ncase=1\n
    line - anal, o - num",type="l",
    xlim=c(0,tplot[nout]));#,ylim=c(0,1));
  points(tplot,u1plot, pch="o",lwd=2);
  }
```

  The scaling of the y axis is deactivated as a comment, `#,ylim=c(0,1));` so that oscillations in the solution outside $0 \leq u_1(z = z_L, t) \leq 1$ can be accommodated with the default scaling for the y axis (the oscillations are a numerical artifact that is an incorrect part of the numerical solution as discussed subsequently).

- For `ncase=1` (no adsorption), the analytical solution to eq. (1.1b) is computed by a call to `step` (as explained subsequently). The resulting plot of the analytical solution is superimposed on the preceding plot of $u_1(z = z_L, t)$ (see Fig. 1.2).

```
#
# Analytical solution, ncase=1
  if(ncase==1){
  u1expl=rep(0,nout);
  for(it in 1:nout){
    u1expl[it]=step(tplot[it],zL,v);
  }
  lines(tplot,u1expl,lwd=2,type="l");
  }
```

- For ncase=2 (with adsorption), the numerical solution $u_1(z = z_L, t)$ is plotted against t as points on a solid line.

```
  #
  # ncase = 2
    if(ncase==2){
    par(mfrow=c(1,1))
    plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t)",
      lwd=2,main="u1(z=zL,t) vs t, ncase=2",
      type="l",xlim=c(0,tplot[nout]));#,ylim=c(0,1));
    points(tplot,u1plot, pch="o",lwd=2);
    }
  #
  # Next case
    }
```

The final } concludes the for in ncase.
The ODE routine pde_1 called by ode (Listing 1.1) is considered next.

(1.2.2) MOL/ODE routine

The ODE routine pde_1 called by ode (Listing 1.1) is in Listing 1.2.

```
  pde_1=function(t,u,parms){
#
# Function pde_1 computes the t derivative vector of the u vector
#
# One vector to two PDEs
  u1=rep(0,n);u2=rep(0,n);
  for (i in 1:n){
    u1[i]=u[i];
    u2[i]=u[i+n];
  }
#
# Boundary condition
  u1[1]=step(t,0,v);
#
# First order spatial derivative
#
#   ifd = 1: Two point upwind finite difference (2pu)
    if(ifd==1){ u1z=dss012(0,zL,n,u1,v); }
#
#   ifd = 2: Three point center finite difference (3pc)
    if(ifd==2){ u1z=dss004(0,zL,n,u1); }
#
#   ifd = 3: Five point biased upwind approximation (5pbu)
    if(ifd==3){ u1z=dss020(0,zL,n,u1,v); }
#
#   ifd = 4: van Leer flux limiter
    if(ifd==4){ u1z=vanl(0,zL,n,u1,v); }
#
# Temporal derivatives, mass transfer rate
    u1t=rep(0,n); u2t=rep(0,n);
   u2eq=rep(0,n);rate=rep(0,n);
#
#    u1t, u2t
     for(i in 1:n){
       u2eq[i]=c1*u1[i]/(1+c2*u1[i]);
       rate[i]=kf*u1[i]*(u2eq[i]-u2[i])-kr*u2[i];
       if(i==1){
         u1t[i]=0;
       }else{
         u1t[i]=-v*u1z[i]-(1-eps)/eps*rate[i];
       }
         u2t[i]=rate[i];
     }
#
# Two PDEs to one vector
  ut=rep(0,2*n);
  for(i in 1:n){
      ut[i]=u1t[i];
    ut[i+n]=u2t[i];
  }
#
# Increment calls to pde_1
  ncall<<-ncall+1;
#
# Return derivative vector
  return(list(c(ut)));
}
```

Listing 1.2: ODE routine pde_1 for eqs. (1.1) to (1.4)

We can note the following points about pde_1.

- The function is defined.

```
  pde_1=function(t,u,parms){
#
# Function pde_1 computes the t derivative vector of the u
     vector
```

The input argument `t` is the current value of t along the numerical solution. `u` is the current vector of (82) ODE dependent variables. `parm` is a set of input parameters for the ODEs; in this case it is unused (but is required in the input arguments). Note that the input arguments are not assigned to reserved names (as in the call to `ode` in Listing 1.1), so the order that they are specified must be maintained, e.g., `t` first followed by `u`.

- The single vector `u` that is the second input argument to `pde_1` is placed in two vectors, `u1` for eq. (1.1b) and `u2` for eq. (1.2b). This is not a required step, but rather, is used so that the subsequent programming can be in terms of variables closely resembling u_1, u_2 in eqs. (1.1b), (1.2b).

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

The two vectors `u1,u2` are first declared with the `rep` utility.

- The BC for eq. (1.1b), eq. (1.4b) with $u_1(z = 0, t) = h(t)$ = `u1(1)`, is specified as a unit step in function `step` (discussed subsequently).

```
#
# Boundary condition
  u1[1]=step(t,0,v);
```

The arguments of `step` are for the current value of t, the value $z = 0$, and the velocity v in eq. (1.1b) (and numerically defined previously).

- The first derivatives in z in eq. (1.1b), $\frac{\partial u_1}{\partial z}$, is computed by one of four spatial differentiators as selected by `ifd` (set previously). Some of the details of these differentiators and their performance as evaluated by comparison of the numerical solution with the analytical solution (for `ncase=1`) are considered subsequently.

```
#
# First order spatial derivative
#
#   ifd = 1: Two point upwind finite difference (2pu)
    if(ifd==1){ u1z=dss012(0,zL,n,u1,v); }
#
#   ifd = 2: Three point center finite difference (3pc)
    if(ifd==2){ u1z=dss004(0,zL,n,u1); }
#
#   ifd = 3: Five point biased upwind approximation (5pbu)
    if(ifd==3){ u1z=dss020(0,zL,n,u1,v); }
```

```
#
#   ifd = 4: van Leer flux limiter
    if(ifd==4){ u1z=vanl(0,zL,n,u1,v); }
```

- Vectors for the LHS derivatives in t in eqs. (1.1b), (1.2b), the equilibrium concentration u_1^e, and the rate of adsorption in eqs. (1.1b), (1.2b), are declared with the `rep` utility over the `n=41` points in z.

```
#
# Temporal derivatives, mass transfer rate
    u1t=rep(0,n); u2t=rep(0,n);
   u2eq=rep(0,n);rate=rep(0,n);
```

- The LHS derivatives in t in eqs. (1.1b), (1.2b), $\frac{\partial u_1}{\partial t}$, $\frac{\partial u_2}{\partial t}$, are programmed in a `for` over the `n=41` points in z.

```
#
#   u1t, u2t
    for(i in 1:n){
       u2eq[i]=c1*u1[i]/(1+c2*u1[i]);
       rate[i]=kf*u1[i]*(u2eq[i]-u2[i])-kr*u2[i];
       if(i==1){
         u1t[i]=0;
       }else{
         u1t[i]=-v*u1z[i]-(1-eps)/eps*rate[i];
       }
         u2t[i]=rate[i];
     }
```

We can note the following details in this programming.

– The equilibrium concentration u_2^e in eq. (1.3) is programmed first.

```
u2eq[i]=c1*u1[i]/(1+c2*u1[i]);
```

– The adsorption rate in eqs. (1.1b) and (1.2b) is then programmed.

```
rate[i]=kf*u1[i]*(u2eq[i]-u2[i])-kr*u2[i];
```

– The 41 MOL/ODEs are programmed.

```
 if(i==1){
   u1t[i]=0;
 }else{
   u1t[i]=-v*u1z[i]-(1-eps)/eps*rate[i];
 }
   u2t[i]=rate[i];
}
```

Since $u_1(z = 0, t)$ is specified through BC (1.4b), its derivative in t is set to zero so that the ODE integrator, `ode`, will not move it away from its prescribed BC value, that is, `u1t[i]=0;`. Otherwise, eq. (1.1b) is programmed as

```
u1t[i]=-v*u1z[i]-(1-eps)/eps*rate[i];
```

Eq. (1.2b) is programmed as

```
u2t[i]=rate[i];
```

The close resemblance of this programming to the PDEs, eqs. (1.1b), (1.2b), is one of the principal advantages of the MOL.

- The two derivatives vectors, `u1t,u2t`, are placed in a single vector `ut` (of length `2*n = 2*41 = 82`) to be returned to the ODE integrator `ode` called in the main program of Listing 1.1.

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

The derivative vector `ut` is first declared with a `rep`.

- The counter for the calls to `pde_1` is incremented and its value is returned to the main program of Listing 1.1 with `<<-`.

```
#
# Increment calls to pde_1
  ncall<<-ncall+1;
#
# Return derivative vector
  return(list(c(ut)));
}
```

The derivative vector `ut` is returned to `ode` as a list which is a requirement of `ode` (and generally, the ODE integrators in `deSolve`). `c()` is the vector operator in R.

The final } concludes function `pde_1`.

Additional subordinate routines called in the preceding program are now considered.

(1.2.3) Subordinate routines

The unit step (Heaviside function) $h(t)$ in BC (1.4b) with $g_1(t) = h(t)$ is programmed in `step`.

```
  step=function(t,z,v) {
#
# Function step approximates a unit step function
#
  tzv=t-z/v;
  if(tzv <0){u1s=0;  }
  if(tzv >0){u1s=1;  }
  if(tzv==0){u1s=0.5;}
#
# Return step
  return(c(u1s));
}
```

Listing 1.3: Function `step` for a unit step

We can note the following details about Listing 1.3.

- The unit step is a traveling wave with argument `tzv=t-z/v`[8]. The step is a finite discontinuity that occurs at $t - z/v = 0$. This discontinuity, which occurs at $z = 0$ in BC (1.4b), is approximated by three `ifs`. For $tzv < 0$, the function is 0, for $tzv > 0$, the function is 1, and for $tzv = 0$, the function is 0.5. This approximation is required since the unit step at $t - z/v = 0$ is undefined (it is not single valued). Analysis of the unit step and the associated solution of eqs. (1.1) to (1.4) is given subsequently.
- The value of the function, `u1s`, is returned as a 1-vector (through the operator `c()`).

The form of the unit step from function `step` will be clear from the graphical output of the numerical solutions of eqs. (1.1) to (1.4), e.g., in Fig. 1.2. In summary, `step` gives an approximation to a unit step as output for a given point along the chromatographic column z as a function of time t. This traveling unit step will be clear from the subsequent solution of eqs. (1.1) to (1.4).

The other routines called in the preceding programming, that is, `dss004` to `max3` as accessed by a `source` in the main program of Listing 1.1, are library routines and are briefly discussed later when the numerical solutions to eqs. (1.1) to (1.4) are considered. The source code for these routines is available from a download site (see the publisher's Web site for this book).

We next consider the output from the R routines in Listings 1.1, 1.2 and 1.3.

(1.3) Model output, single component chromatography

We first consider FDs as implemented with `ifd = 1,2,3`

(1.3.1) FDs, step BC

Abbreviated numerical output from the execution of the routines in Listings 1.1, 1.2 and 1.3 is given in Table 1.2.

[8]A detailed discussion of traveling wave solutions to PDEs is given in [1].

```
ifd =  1   ncase =  1

    t     u1(z=zL,t)  rate(z=zL,t)
  0.00      0.0000      0.0000
  3.00      0.0000      0.0000
              .            .
              .            .
              .            .
  (output for t = 6 to 21 deleted)
              .            .
              .            .
              .            .
 24.00      0.0000      0.0000
 27.00      0.0003      0.0000
 30.00      0.0017      0.0000
 33.00      0.0081      0.0000
 36.00      0.0277      0.0000
 39.00      0.0728      0.0000
 42.00      0.1543      0.0000
 45.00      0.2737      0.0000
 48.00      0.4194      0.0000
 51.00      0.5708      0.0000
 54.00      0.7073      0.0000
 57.00      0.8158      0.0000
 60.00      0.8927      0.0000
 63.00      0.9420      0.0000
 66.00      0.9708      0.0000
 69.00      0.9862      0.0000
 72.00      0.9939      0.0000
 75.00      0.9975      0.0000
 78.00      0.9990      0.0000
 81.00      0.9996      0.0000
 84.00      0.9999      0.0000
 87.00      1.0000      0.0000
 90.00      1.0000      0.0000
              .            .
              .            .
              .            .
       (output for t = 93 to
           144 deleted)
              .            .
              .            .
              .            .
147.00      1.0000      0.0000
150.00      1.0000      0.0000
```

```
ncall =  609

ifd =  1   ncase =  2

    t     u1(z=zL,t)  rate(z=zL,t)
  0.00      0.0000      0.0000
  3.00      0.0000      0.0000
              .           .
              .           .
              .           .
  (output for t = 6 to 24 deleted)
              .           .
              .           .
              .           .
 27.00      0.0001      0.0000
 30.00      0.0008      0.0000
 33.00      0.0031      0.0000
 36.00      0.0089      0.0000
 39.00      0.0196      0.0001
 42.00      0.0359      0.0003
 45.00      0.0577      0.0007
 48.00      0.0852      0.0012
 51.00      0.1185      0.0018
 54.00      0.1584      0.0027
 57.00      0.2057      0.0038
 60.00      0.2617      0.0050
 63.00      0.3269      0.0063
 66.00      0.4014      0.0075
 69.00      0.4839      0.0084
 72.00      0.5710      0.0087
 75.00      0.6580      0.0084
 78.00      0.7393      0.0075
 81.00      0.8101      0.0062
 84.00      0.8676      0.0048
 87.00      0.9114      0.0035
 90.00      0.9430      0.0024
 93.00      0.9646      0.0016
 96.00      0.9787      0.0010
 99.00      0.9875      0.0006
102.00      0.9929      0.0004
105.00      0.9961      0.0002
108.00      0.9979      0.0001
111.00      0.9989      0.0001
114.00      0.9994      0.0000
117.00      0.9997      0.0000
120.00      0.9999      0.0000
```

```
123.00      0.9999      0.0000
126.00      1.0000      0.0000
129.00      1.0000      0.0000
132.00      1.0000      0.0000
135.00      1.0000      0.0000
138.00      1.0000      0.0000
141.00      1.0000      0.0000
144.00      1.0000      0.0000
147.00      1.0000      0.0000
150.00      1.0000      0.0000

ncall = 1143
```

Table 1.2: Selected numerical output for eqs. (1.1) to (1.4) from `pde_1_main` and `pde_1` for `ncase=1,2`

We can note the following details about Table 1.2.

- The output is for $0 \leq t \leq 150$ as programmed in Listing 1.1.
- The ICs of eqs. (1.4a) and (1.4c) are confirmed (checking the ICs of a numerical solution is always a good idea since if the ICs are incorrect, the solution will start incorrectly and will therefore most likely be entirely incorrect).
- The rate of adsorption for `ncase=1` is zero as expected since no adsorption takes place ($rate = k_f = k_r = 0$ in eqs. (1.1b), (1.2b)).
- The rate of adsorption for `ncase=2` goes through a maximum of `0.0087` at $t = 72$. This maximum is expected as the adsorption on the adsorbate increases initially, then reaches a point where desorption begins to reduce the rate. Eventually, the rate returns to zero. Also, the maximum adsorption rate occurs at approximately the half way point in the transient, $u_1(z = z_L, t = 72) = 0.5710$, where the rate of change of u_1 is greatest (see Fig. 1.3).
- The solution is smooth (e.g., no oscillations) and approaches the expected final value, $u_1(z = z_L, t) = 1$. In other words, the two-point upwind finite difference (FD) approximation in `dss012` corresponding to `ifd=1` appears to function quite well. However, there is a significant error as reflected in Fig. 1.2 where the exact and analytical solutions are compared. This error is discussed in more detail later.
- The accuracy of a numerical PDE solution generally cannot be determined directly or explicitly (since this implies an analytical solution is available for computing the exact error). For `ncase=1`, an analytical solution is available, but this is unusual. In fact, numerical solutions are generally computed because analytical solutions are not available.

  However, each new numerical solution should be evaluated indirectly. Here are three procedures for an approximate error analysis that do not require an analytical solution for the full PDE problem.

  – The accuracy of the integration in z could be inferred by: (1) observing the effect of increasing the number of grid points, e.g., `n=41` to `n=81` (termed *h*

refinement since the grid spacing, which is typically given the symbol h, is varied when the number of grid points is changed), and (2) by changing the spatial derivative approximation, e.g., another value of `ifd` (termed *p refinement* since the order of the FD approximation[9], which is typically given the symbol p, is varied as the FD approximation is changed). The procedure then in (1) and (2) is to change h and p and observe the effect on the numerical solution. For example, if the solution does not change in the third figure, three-figure accuracy can be inferred. However, this is not a proof of numerical accuracy. Rather, it is just an estimate of accuracy.
  - The accuracy of the integration in t could be inferred by changing the error tolerances specified for the ODE integrator. In the case of the R integrator `ode` (called in Listing 1.1), default error tolerances of 1×10^{-6} (absolute and relative) are used in `ode` unless the defaults are reset to other values before `ode` is called. `ode` is a sophisticated ODE integrator that changes the integration interval (h refinement) and the algorithm order (p refinement) in an attempt to meet the default or user-specified error tolerances. If `ode` is unable to meet the error tolerances, it issues a warning message to this effect[10].
  - Special case analytical solutions can be used to test the numerical solutions. This is particularly useful in locating coding (programming) errors. For example, for `ncase=1`, the available analytical solution can be compared with the numerical solution as in Fig. 1.2.

- The total calls to `pde_1` is `ncall = 609` for `ncase=1` and `ncall = 1143` for `ncase=2` indicating that `ode` computed numerical solutions with modest computational effort. The solutions presumably have the accuracy indicated through h and p refinement (as discussed subsequently).

Additional features of the numerical solution are evident in Figs. 1.2 and 1.3. We can note the following details in Fig. 1.2:

- The analytical solution, plotted as a solid line, is an approximation to a unit step at $t = 50$. This solution is derived in the following way.

[9]The order of a FD approximation refers to the power p in a formula of the form

$$error = c_1 \Delta z^p$$

where *error* is the *truncation error*, c_1 is a constant, Δz is the grid spacing in z and p is the order of the FD approximation. This formula for the error suggests that the error decreases with decreasing Δz (as the number of grid points in z is increased). For the two-point upwind FD approximations in `dss012`, $p = 1$ so the truncation error varies linearly with Δz and they are termed *first-order correct*. The term truncation error refers to the error resulting from truncating the Taylor series from which the FD is derived. For the five-point FD approximations in `dss004` (with `idf=2`), $p = 4$ and for the five-point FD approximations in `dss020` (with `ifd=3`), $p = 4$. These FD approximations of various orders are discussed subsequently.

[10]The adjustment of the FD integration intervals in `ode` is also termed *r refinement* where the r designates automatic (algorithmic, adaptive) refinement of the grid. This term is generally applied to the refinement of spatial grids. However, the spatial grids in `dss004, dss012, dss020` are not refined automatically, i.e., they are fixed or constant grids that can be h-refined by changing the number of spatial grid points.

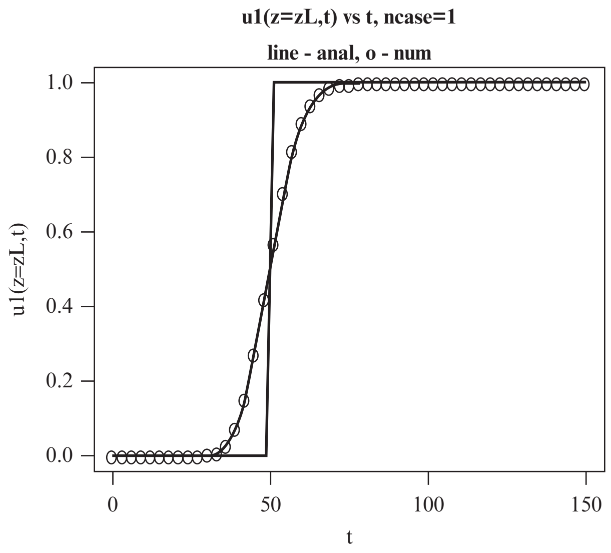

Figure 1.2 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b)

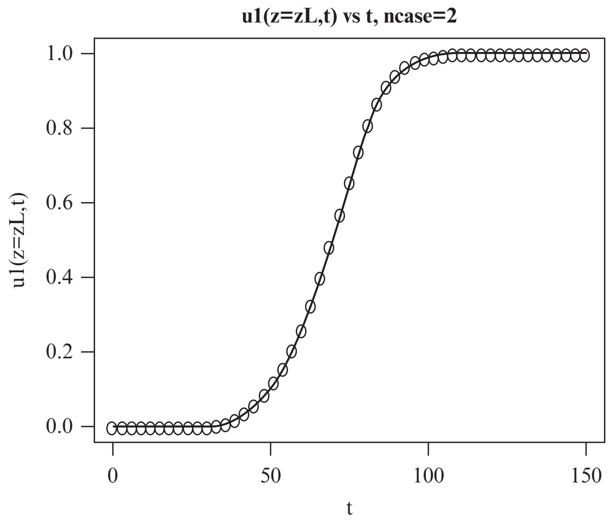

Figure 1.3 Numerical solution of eqs. (1.1b), (1.2b), `ncase=2`, `ifd=1`

Eq. (1.1b) for $k_f = k_r = 0$, v constant (`ncase=1` in Listing 1.1) is the *linear advection equation*,

$$\frac{\partial u_1}{\partial t} = -v\frac{\partial u_1}{\partial z} \tag{1.5a}$$

with IC (1.4a) and BC (1.4b). For the present analysis, we take (in eq. (1.4a))

$$f_1(z) = 0 \tag{1.5b}$$

and (in eq. (1.4b))

$$g_1(t) = h(t) \tag{1.5c}$$

where $h(t)$ is the unit step (Heaviside) function

$$h(t) = \begin{cases} 0, & t < 0 \\ 1, & t > 0 \end{cases} \tag{1.5d}$$

Note that $h(t)$ is not defined at $t = 0$.

If we assume a solution to eq. (1.5a) of the form

$$u_1(z,t) = g_1(t - z/v) = g_1(\lambda);\ \lambda = t - z/v \tag{1.6a}$$

with $g_1(\lambda < 0) = 0$ (recall g_1 is the BC function of eq. (1.4b)), substitution of eq. (1.6a) in eq. (1.5a) gives

$$\begin{aligned}\frac{\partial u_1}{\partial t} &= \frac{dg_1}{d\lambda}\frac{\partial \lambda}{\partial t} \\ &= -v\frac{\partial u_1}{\partial z} = -v\frac{dg_1}{d\lambda}\frac{\partial \lambda}{\partial z}\end{aligned}$$

Since

$$\frac{\partial \lambda}{\partial t} = 1; \quad \frac{\partial \lambda}{\partial z} = -1/v$$

substitution in the preceding equation gives

$$\frac{\partial u_1}{\partial t} = \frac{dg_1}{d\lambda}(1)$$

$$= -v\frac{\partial u_1}{\partial z} = -v\frac{dg_1}{d\lambda}(-1/v)\ (QED^{11})$$

so that eq. (1.6a) is a solution to eq. (1.5a).

Also, eq. (1.6a) satisfies IC (1.4a) (with $f_1(z) = 0$) and BC (1.4b) (with $z = 0$). Therefore, eq. (1.6a) is the analytical solution to eqs. (1.5a) to (1.5c). It is termed a *traveling wave solution* [1] since it depends only on the *Lagrangian* variable $\lambda = t - z/v$.

For the special case of $g_1(t) = h(t)$ of eq. (1.5d), the analytical solution is

$$u_1(z,t) = h(t - z/v) \tag{1.6b}$$

[11] QED ="quod erat demonstrandu" (Latin) or "that which was to be demonstrated".

Eq. (1.6b) defines a unit step[12], starting at $z = 0$ and traveling in the direction of increasing z (up the chromatographic column of Fig. 1.1) with velocity v. The unit step occurs at $\lambda = t - z/v = 0$. Thus, at the exit of the column, $\lambda = t - z_L/v = 0$ or at $t = z_L/v = 50/1 = 50$). This unit step is evident in Fig. 1.2, except that the step is actually undefined (not single valued) and is approximated over three grid points as programmed in `step` of Listing 1.3.

The fact that $u_1(z = z_L, t)$ is not a discontinuity at $t = 50$ is clear in Fig. 1.2 (the slope is finite). However, the approximation is required since a function that is not single valued (and with an infinite slope) cannot be programmed (it can be plotted using two points, $u_1(\lambda = 0) = 0$ and $u_1(\lambda = 0) = 1$ (both values used at $t = 50$), but this is an artifice just to give the step the required appearance).

As another perspective, eqs. (1.5) constitute an impossible problem numerically since the derivative (slope) $\dfrac{\partial u_1}{\partial z}$ at $t - z/v = 0$ is infinite. For the purpose of computing a numerical solution, this discontinuity is approximated by a function with a finite slope as in function `step` of Listing 1.3. In the subsequent discussion, we will consider how closely the methods for calculating $\dfrac{\partial u_1}{\partial z}$ produce the solution of eq. (1.6b), that is, for `ifd = 1,2,3,4` in Listing 1.1.

In summary, eq. (1.5a) is an elementary hyperbolic PDE for which an analytical solution is easily derived, yet it is one of the most difficult PDEs to integrate numerically (since it propagates steep fronts and discontinuities, a general feature of hyperbolic PDEs).

- The numerical solution of eqs. (1.1) to (1.4) (in Fig. 1.2) plotted with o is a smoothed (rounded) approximation of the unit step solution of eq. (1.6b). This smoothing is generally termed *numerical diffusion* and is one of two distortions (numerical artifacts) of solutions with steep moving fronts or discontinuities. The other distortion is *numerical oscillation* that is described next. Numerical diffusion may preclude the accurate calculation of moving front solutions in applications for which this is unacceptable, e.g., chromatography, as discussed subsequently.
- The two-point upwind FD approximations used in the numerical solution of eqs. (1.1) to (1.4) are illustrated by the following code taken from `dss012 (ifd=1)` listed in Appendix B.

```
for(i in 2:n){
    ux[i]=(u[i]-u[i-1])/dx;
}
```

with `n=41` in `pde_1` of Listing 1.2 where `dss012` is called. The derivative, `ux[i]`, is approximated at grid point `i` by a FD based on a weighted sum of `u[i]` and `u[i-1]` with a spacing `dx` (for `v > 0`). Point `i-1` is upwind (upstream) of point `i`. This upwinding is essential in the case of steep moving fronts and discontinuities in the

[12]A PDE with a solution that depends only on an IC is generally termed a *Cauchy problem*, that is, an initial value problem. If the IC is discontinuous, the PDE is termed a *Riemann problem*. Eqs. (1.5) are an example of a Riemann problem since they define a PDE problem with a solution that has a discontinuity for $t \geq 0$, that is, a discontinuity at $t - z/v = 0$ as stated in eq. (1.6b).

solution. If `u[i+1]` is used in place of `u[i-1]`, the numerical solution will become unstable. Physically, this makes sense since what happens at `i` is determined by what is happening upstream at `i-1` and not downstream at `i+1`.

- This use of upwinding requires a priori knowledge of the direction of flow, e.g., bottom to top in the chromatographic column of Fig. 1.1. An incorrect direction used in the FD approximation leads to unstable solutions. For example, if `v < 0`, the preceding FD approximation will produce an unstable numerical solution. Rather, `u[i]` and `u[i+1]` are used in the FD since `i+1` is now in the upstream direction (see the listing of `dss012` in Appendix B). Approximations that are *centered* (rather than upwinded), and therefore do not require knowledge of the direction of flow, are discussed subsequently (for `ifd=2`).
- For `i=2` (one point inside the left boundary in z), `u[1]` is required (in the preceding code). `u[1]` is set as a BC, as illustrated in `pde_1` of Listing 1.2.

In summary, two-point upwinding generally produces a stable solution with no numerical oscillation, but with numerical diffusion that may be excessive, depending on the application, such as for hyperbolic (convective) PDEs that propagate steep fronts and discontinuities.

For `ncase=2` in Listing 1.1, Fig. 1.3 results (the numerical output is in Table 1.2). We can note the following details in Fig. 1.3.

- The step input, $u_1(z=0,t)=h(t)$, is smoothed beyond that from the two-point upwind approximation of Fig. 1.2 by the adsorption onto the adsorbate. Whether this is an accurate solution is difficult to assess without some form of *error analysis*, for example, h and p refinement, which are considered next. As is generally the case, an analytical solution is not readily available that can be used to calculate an exact error as in Fig. 1.2. A principal reason for not having an analytical solution is the nonlinearity of the rate, that is, the term $u_1(u_1^e-u_2)$ with the u_1u_2 product, and the isotherm of eq. (1.3) which is nonlinear in u_2 for $c_2 \neq 0$. In other words, an analytical solution is precluded, and we accept a numerical solution that we expect will be of reasonable accuracy, e.g., 3-4 significant figures. However, this expectation must be justified (as it should for all PDE numerical solutions).
- An important consideration in evaluating the numerical solution of Table 1.2 and Fig. 1.3 is whether the numerical diffusion from the two-point upwind approximation is significant relative to the physical smoothing from the adsorption. This point requires further investigation through h and p refinement which is facilitated by having the choice of four approximations selected with `ifd`. To start, we consider the numerical solution of eqs. (1.1) to (1.4) with `ifd=2` in the main program of Listing (1.1) (everything else remains the same).

`idf = 2` gives a call to `dss004` in `pde_1`. `dss004` has five-point centered (fourth-order correct) FD approximations (except near the boundaries $(z=0,z_L)$ where noncentered FDs are used) so we would expect improved accuracy of the solution relative to `ifd=1` (five points rather than two). However, we will observe this is incorrect.

The numerical output for `ifd=2` is given in Table 1.3

```
ifd =  2   ncase =  1

    t      u1(z=zL,t)  rate(z=zL,t)
  0.00       0.0000      0.0000
  3.00       0.0000      0.0000
               .            .
               .            .
               .            .
  (output for t = 6 to 36 deleted)
               .            .
               .            .
               .            .
 39.00       0.0003      0.0000
 42.00      -0.0019      0.0000
 45.00      -0.0149      0.0000
 48.00       0.0850      0.0000
 51.00       0.6330      0.0000
 54.00       1.1941      0.0000
 57.00       0.9481      0.0000
 60.00       0.9382      0.0000
 63.00       1.1158      0.0000
 66.00       0.8752      0.0000
 69.00       1.1072      0.0000
 72.00       0.9229      0.0000
 75.00       1.0437      0.0000
 78.00       0.9880      0.0000
 81.00       0.9843      0.0000
 84.00       1.0384      0.0000
 87.00       0.9439      0.0000
 90.00       1.0693      0.0000
 93.00       0.9215      0.0000
 96.00       1.0844      0.0000
 99.00       0.9125      0.0000
102.00       1.0885      0.0000
105.00       0.9122      0.0000
108.00       1.0858      0.0000
111.00       0.9171      0.0000
114.00       1.0794      0.0000
117.00       0.9248      0.0000
120.00       1.0710      0.0000
123.00       0.9321      0.0000
126.00       1.0620      0.0000
129.00       0.9456      0.0000
132.00       1.0548      0.0000
135.00       0.9422      0.0000
```

```
138.00      1.0490      0.0000
141.00      0.9667      0.0000
144.00      1.0267      0.0000
147.00      0.9660      0.0000
150.00      1.0459      0.0000

ncall = 1311


ifd =  2   ncase =  2

    t     u1(z=zL,t)  rate(z=zL,t)
  0.00      0.0000      0.0000
  3.00      0.0000      0.0000
               .           .
               .           .
               .           .
  (output for t = 6 to 36 deleted)
               .           .
               .           .
               .           .
 39.00      0.0001      0.0000
 42.00     -0.0005      0.0000
 45.00     -0.0049      0.0000
 48.00      0.0036      0.0000
 51.00      0.0337      0.0005
 54.00      0.0654      0.0010
 57.00      0.1012      0.0017
 60.00      0.1460      0.0029
 63.00      0.2051      0.0048
 66.00      0.2886      0.0081
 69.00      0.4133      0.0136
 72.00      0.5950      0.0200
 75.00      0.8073      0.0195
 78.00      0.9486      0.0089
 81.00      0.9866      0.0015
 84.00      0.9945      0.0007
 87.00      1.0004      0.0003
 90.00      0.9996     -0.0001
 93.00      0.9996      0.0001
 96.00      1.0004      0.0000
 99.00      0.9998     -0.0000
102.00      0.9999      0.0000
105.00      1.0001     -0.0000
108.00      0.9999     -0.0000
```

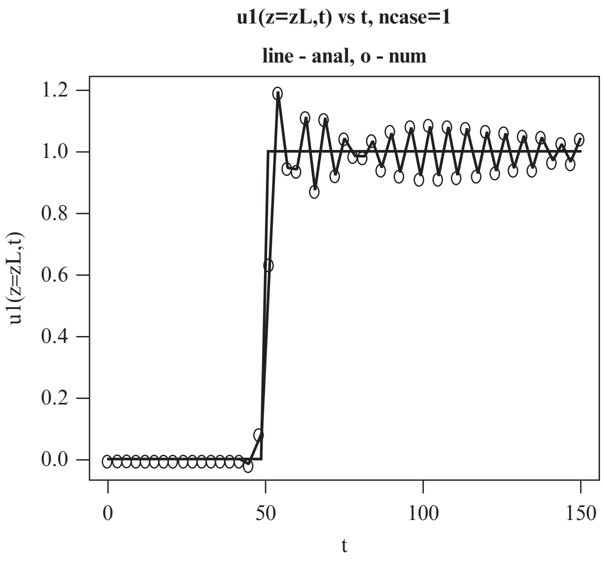

Figure 1.4 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b)

```
111.00      1.0000      0.0000
              .           .
              .           .
              .           .
(output for t = 114 to 144 deleted)
              .           .
              .           .
              .           .

147.00      1.0000     -0.0000
150.00      1.0000     -0.0000

ncall = 1035
```

Table 1.3: Numerical output for eqs. (1.1) to (1.4) for `ncase=1,2`, `ifd=2`

Figs. 1.4, 1.5 follow.
We can note the following points from Table 1.3 and Figs. 1.4, 1.5.

- In Table 1.3, `ncase=1`, and in Fig. 1.4, the numerical solution is highly oscillatory (termed *numerical oscillation*). This is the second form of numerical distortion (in addition to numerical diffusion illustrated in Fig. 1.2). Clearly, by comparison with the analytical solution, the numerical solution is unacceptable (and it is also unrealistic physically since we would not expect the output from the adsorption column of Fig. 1.1 to oscillate).

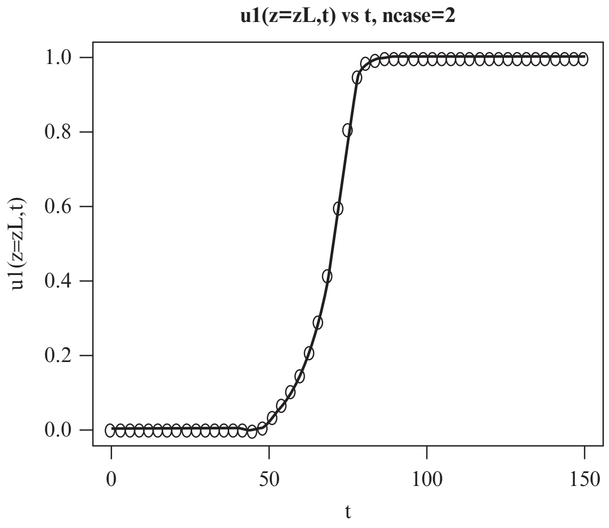

Figure 1.5 Numerical solution of eqs. (1.1b), (1.2b) for `ncase=2`, `ifd=2`

- In Fig. 1.5, the numerical oscillation is essentially eliminated by the adsorption (`ncase=2`), which demonstrates that the performance of a spatial differentiator may be strongly dependent on the characteristics (features) of the PDE problem. For example, with no adsorption (`ncase=1`) (so that the solution is the propagation of a unit step or discontinuity), the two-point upwind (2pu) approximation gives excessive numerical diffusion (Fig. 1.2) and the five-point centered approximation (5pc) gives numerical oscillation (Fig. 1.4). However, for the 2pu with adsorption (`ncase=2`), the diffusion still appears to be unacceptable (Fig. 1.3) while for 5pc, the physical smoothing was sufficient to give what appears to be an accurate solution (Fig. 1.5). But this use of p refinement (2pu to 5pc) is inconclusive and further analysis is required.

The general conclusion we reach is that centered FD approximations should not be used for strongly hyperbolic (convective) PDEs with a solution that includes a steep moving front or discontinuity. This conclusion remains valid if the number of grid points is increased (e.g., `n=41` to `81`). In fact, the oscillations usually become even more pronounced[13]. This conclusion also remains valid if the order of the FD approximation is increased, e.g., seven-point centered approximations oscillate as much as the five-point FD approximations.

To complete this discussion of five-point centered approximations, a section of code from `dss004` (listed in Appendix B) is given below.

[13]We should not conclude that five-point centered FDs are always unsatisfactory. In fact, they generally work very well for parabolic (diffusive) PDEs such as the *heat conduction equation* (*Fourier's second law*) and the *diffusion equation* (*Fick's second law*). This point will be illustrated in later applications.

```
#
# Interior points (x=xl+2*dx,...,x=xu-2*dx)
  for(i in 3:(n-2))ux[i]=r12dx*(-u[i+2]+8*u[i+1]-8*u[i-1]+u[i-2]);
```

Note that the derivative `ux[i]` is computed as a weighted sum of five values of `u` with weighting coefficients `-1 8 0 -8 1` that are skew symmetric with respect to the center point `i` where the coefficient is `0` (and is therefore not programmed). At the boundary points `i=1,2,n-1,n`, noncentered approximations are used. Details are given in `dss004` listed in Appendix B.

The question then naturally arises if there is a FD approximation with acceptable levels of numerical diffusion and oscillation. If the two-point upwind approximations (in `dss012`) do not oscillate and the five-point approximations (in `dss004`) have a relatively low level of numerical diffusion (see Fig. 1.4 along the near vertical analytical solution) perhaps somehow combining the approximations would be worth trying. To this end, we consider the five-point biased upwind (5pbu) approximations in `dss020` listed in Appendix B. A section of the coding from `dss020` for the derivative at `i` is listed below.

```
for(i in 4:(n-1)){
 ux[ i]=r12dx*(  -u[i-3] +6*u[i-2]-18*u[i-1]+10*u[i ]+3*u[i+1]); }
```

The derivative `ux[i]` is the weighted sum of five values of `u` with grid indices `i-3,i-2,i-1,i,i+1` (for `v > 0`). That is, three points upstream of `i` and one point downstream are used. This biasing in the upstream direction, designated as 5pbu, is intended to maintain the effect of the flow. The effectiveness of this approach is reflected in the numerical output in Table 1.4 and Figs. 1.6 and 1.7 (produced with `ifd`=3 in main program of Listing 1.1).

```
 ifd =  3   ncase =  1

     t     u1(z=zL,t)  rate(z=zL,t)
   0.00      0.0000      0.0000
   3.00      0.0000      0.0000
               .           .
               .           .
               .           .
   (output for t = 6 to 24 deleted)
               .           .
               .           .
               .           .
  27.00      0.0001      0.0000
  30.00     -0.0005      0.0000
  33.00      0.0015      0.0000
  36.00     -0.0019      0.0000
  39.00     -0.0074      0.0000
  42.00      0.0522      0.0000
  45.00     -0.1178      0.0000
  48.00      0.1066      0.0000
  51.00      0.7831      0.0000
  54.00      1.0120      0.0000
  57.00      1.0039      0.0000
  60.00      1.0026      0.0000
  63.00      0.9980      0.0000
  66.00      1.0012      0.0000
  69.00      0.9994      0.0000
  72.00      1.0003      0.0000
  75.00      0.9999      0.0000
  78.00      1.0000      0.0000
               .           .
               .           .
               .           .
 (output for t = 81 to 144 deleted)
               .           .
               .           .
               .           .
 147.00      1.0000      0.0000
 150.00      1.0000      0.0000

 ncall =  914


 ifd =  3   ncase =  2

    t     u1(z=zL,t)  rate(z=zL,t)
   0.00      0.0000      0.0000
   3.00      0.0000      0.0000
               .           .
               .           .
               .           .
   (output for t = 6 to 24 deleted)
               .           .
               .           .
               .           .
  27.00      0.0001      0.0000
  30.00     -0.0002      0.0000
  33.00      0.0005      0.0000
  36.00     -0.0005     -0.0000
  39.00     -0.0020     -0.0000
  42.00      0.0081      0.0000
  45.00     -0.0086      0.0000
  48.00      0.0012     -0.0000
  51.00      0.0304      0.0004
  54.00      0.0606      0.0009
  57.00      0.0958      0.0016
  60.00      0.1400      0.0028
  63.00      0.1998      0.0048
  66.00      0.2879      0.0086
  69.00      0.4238      0.0149
  72.00      0.6143      0.0203
  75.00      0.8101      0.0171
  78.00      0.9366      0.0086
  81.00      0.9865      0.0027
  84.00      0.9987      0.0005
  87.00      1.0002      0.0000
  90.00      1.0001     -0.0000
  93.00      1.0000     -0.0000
               .           .
               .           .
               .           .
 (output for t = 96 to 144 deleted)
               .           .
               .           .
               .           .
 147.00      1.0000     -0.0000
 150.00      1.0000     -0.0000

 ncall = 1312
```

Table 1.4: Numerical output for eqs. (1.1) to (1.4) for ncase=1,2, ifd=3

Figs. 1.6, 1.7 follow.
We can note the following points from Table 1.4 and Figs. 1.6, 1.7.

- In Fig. 1.6, the 5pbu approximation has substantially reduced the numerical diffusion and oscillation of Figs. 1.2 and 1.4. However, there is an oscillation at the leading edge of the numerical solution that could still render the numerical solution unacceptable. But we should keep in mind that the numerical solution approximates a unit step, which is essentially an impossible requirement (as discussed previously), so that the numerical solution is a substantial improvement over the previous 2pu (ifd=1) and 5pc (ifd=2) solutions.
- In Fig. 1.7, the oscillation of Fig. 1.6 has been essentially eliminated by the adsorption (for ncase=2) as occurred in Fig. 1.5. Also, Figs. 1.5 and 1.7 are quite similar which suggests that they reflect an accurate solution (although this is certainly not a proof of accuracy and some additional cases with a number of grid points other than n=41 should be considered).

  To demonstrate this point of the similarity of the solutions for ifd=2,3, from Tables 1.3 and 1.4, we have at the portions of the solutions changing most rapidly (see Figs. 1.5, 1.7):

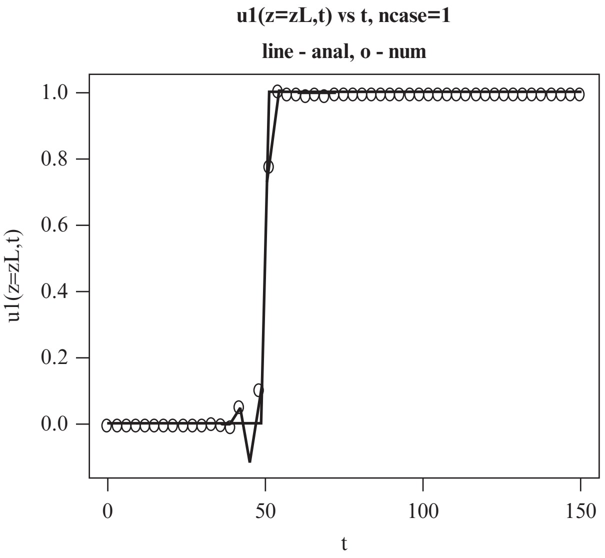

Figure 1.6 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b)

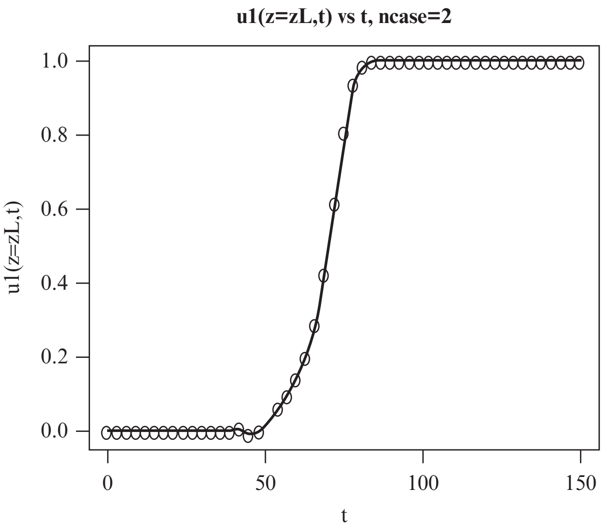

Figure 1.7 Numerical solution of eqs. (1.1b), (1.2b), `ncase=2`, `ifd=3`

```
t = 63
  5pc  (ifd=2)
  63.00      0.2051      0.0048
  5pbu (ifd=3)
  63.00      0.1998      0.0048 (n=41)
  63.00      0.2026      0.0048 (n=81)
```

```
t = 69
  5pc  (ifd=2)
  69.00      0.4133      0.0136
  5pbu (ifd=3)
  69.00      0.4238      0.0149 (n=41)
  69.00      0.4167      0.0142 (n=81)

t = 75
  5pc  (ifd=2)
  75.00      0.8073      0.0195
  5pbu (ifd=3)
  75.00      0.8101      0.0171 (n=41)
  75.00      0.8157      0.0182 (n=81)
```

For 5pbu, solutions are summarized with `n=41,81` to demonstrate the level of convergence from h refinement.

Experience has indicated that the 5pbu frequently works as required to produce an accurate numerical solution if the moving front of the solution is not very steep, which is the case in many physical applications. But the preceding results (for `ifd=1,2,3`) suggest that some experimentation and careful evaluation of the numerical solution is usually required, including the use of h refinement, that is, changing the number of grid points and observing the effect on the numerical solution. We have not done that here because of space limitations, but rather used only `n=41` (with results for `n=81` indicated in the preceding table). Changes in the number of grid points requires only changing `n` in the main program of Listing 1.1.

We conclude this discussion of FD approximation of hyperbolic PDEs that propagate steep fronts and discontinuities with Godunov's barrier theorem that pertains to numerical diffusion and oscillation. This theorem states ([1], p25): There is no linear approximation to the Riemann problem, higher than first order, that is nonoscillatory.

To explain the wording:

- `ncase=1` in the previous examples corresponds to the Riemann problem (the unit step or discontinuity BC of eqs. (1.4b) and (1.5b)).
- The FD approximations 2pu, 5pc and 5pbu are linear in the sense that `u[i]` in the RHS weighted sums for the calculation of `ux[i]` is to the first power (see the preceding portions of code).
- 2pu is first order and does not oscillate (Fig. 1.2).
- 5pc and 5pbu are fourth order (i.e., higher than first order) and oscillate (Figs. 1.4, 1.6).

Thus, if we are to use a higher order method to achieve better accuracy (e.g., less diffusion than 2pu), we will have to use a nonlinear approximation or algorithm to avoid oscillation. This is the approach considered next based on *flux limiters*.

(1.3.2) Flux limiters, step BC

Flux limiters provide a nonlinear approximation to the first-order spatial derivatives in convective (hyperbolic) PDEs, e.g., $\frac{\partial u_1}{\partial z}$ in eq. (1.1b). The nonlinear algorithm can be used to eliminate numerical oscillations as explained by Godunov's theorem cited previously.

For example, we can code the van Leer flux limiter ([1], pp 37-43) in the format of the 5pc and 5pbu FD approximations. This has been done in function `vanl` listed in Appendix B. We can then call `vanl` by using `ifd=4` in the main program of Listing 1.1. Abbreviated numerical output is listed in Table 1.5 below, and the graphical output is in Figs. 1.8 and 1.9.

```
ifd =  4   ncase =  1
    t     u1(z=zL,t)  rate(z=zL,t)
  0.00      0.0000      0.0000
  3.00     -0.0000      0.0000
               .           .
               .           .
               .           .
  (output for t = 6 to 42 deleted)
               .           .
               .           .
               .           .

 45.00      0.0022      0.0000
 48.00      0.2420      0.0000
 51.00      0.6735      0.0000
 54.00      0.9253      0.0000
 57.00      0.9899      0.0000
 60.00      0.9989      0.0000
 63.00      0.9999      0.0000
 66.00      1.0000      0.0000
               .           .
               .           .
               .           .
(output for t = 69 to 144 deleted)
               .           .
               .           .
               .           .
147.00      1.0000      0.0000
150.00      1.0000      0.0000

ncall = 2191


ifd =  4   ncase =  2
```

```
     t      u1(z=zL,t)  rate(z=zL,t)
   0.00      0.0000      0.0000
   3.00     -0.0000      0.0000
               .           .
               .           .
               .           .
   (output for t = 6 to 45 deleted)
               .           .
               .           .
               .           .
  48.00      0.0063      0.0000
  51.00      0.0334      0.0004
  54.00      0.0666      0.0011
  57.00      0.1055      0.0020
  60.00      0.1549      0.0033
  63.00      0.2223      0.0057
  66.00      0.3180      0.0096
  69.00      0.4511      0.0143
  72.00      0.6151      0.0169
  75.00      0.7770      0.0146
  78.00      0.8969      0.0091
  81.00      0.9621      0.0042
  84.00      0.9883      0.0015
  87.00      0.9968      0.0005
  90.00      0.9992      0.0001
  93.00      0.9998      0.0000
  96.00      0.9999      0.0000
  99.00      1.0000      0.0000
               .           .
               .           .
               .           .
 (output for t = 102 to 144 deleted)
               .           .
               .           .
               .           .
 147.00      1.0000      0.0000
 150.00      1.0000      0.0000

 ncall = 5620
```

Table 1.5: Numerical output for eqs. (1.1) to (1.4) for `ncase=1,2`, `ifd=4`

Fig. 1.8 indicates that the unit step of eq. (1.5d) is closely approximated, with little numerical diffusion and no oscillation. The latter is termed *essentially non-oscillatory* or ENO.

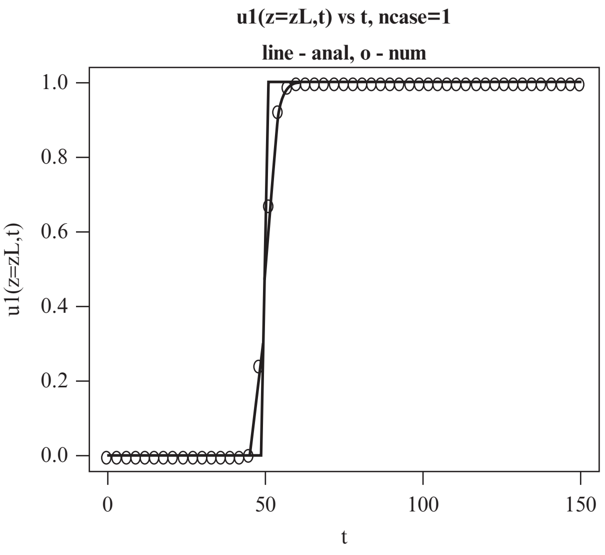

Figure 1.8 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b)

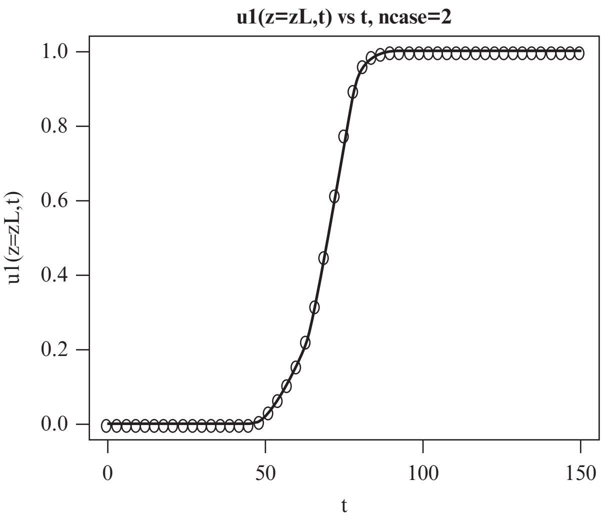

Figure 1.9 Numerical solution of eqs. (1.1b), (1.2b), `ncase=2`, `ifd=4`

Fig. 1.9 also indicates a smooth solution (no oscillation), but this was also achieved with 5pc and 5pbu (Figs. 1.5, 1.7) because of smoothing of the transfer to the adsorbent for `ncase=2`.

The three approaches, 2pc, 5pbu and van Leer, are briefly compared in Table 1.6

```
t = 63
  5pc  (ifd=2)
  63.00      0.2051      0.0048
  5pbu (ifd=3)
  63.00      0.1998      0.0048 (n=41)
  63.00      0.2026      0.0048 (n=81)
  van Leer (ifd=4)
  63.00      0.2223      0.0057

t = 69
  5pc  (ifd=2)
  69.00      0.4133      0.0136
  5pbu (ifd=3)
  69.00      0.4238      0.0149 (n=41)
  69.00      0.4167      0.0142 (n=81)
  van Leer (ifd=4)
  69.00      0.4511      0.0143

t = 75
  5pc  (ifd=2)
  75.00      0.8073      0.0195
  5pbu (ifd=3)
  75.00      0.8101      0.0171 (n=41)
  75.00      0.8157      0.0182 (n=81)
  van Leer (ifd=4)
  75.00      0.7770      0.0146

 Calls to pde_1, ncase=2
  5pc       ncall = 1035
  5pbu      ncall = 1312
  van Leer  ncall = 5620
```

Table 1.6: Abbreviated comparison of output for eqs. (1.1) to (1.4) for `ncase=2`

The differences in the numerical solutions in Table 1.6 (for `ifd=2,3,4`) suggest that these differences are substantial. However, this is not necessarily the case as indicated by the graphical output produced by the following variant of the main program of Listing 1.1.

```
#
# Delete previous workspaces
  rm(list=ls(all=TRUE))
#
#   1D, one component, chromatography model
#
#   The ODE/PDE system is
#
```

```
#   u1_t = -v*u1_z - (1 - eps)/eps*rate                          (1.1b)
#
#   u2_t = rate                                                  (1.2b)
#
#   rate = kf*u1*(u2eq - u2) - kr*u2
#
#   u2eq = c1*u1/(1 + c2*u1)                                     (1.3)
#
#   Boundary condition
#
#     u1(z=0,t) = step(t)                                        (1.4b)
#
#   Initial conditions
#
#     u1(z,t=0) = 0                                              (1.4a)
#
#     u2(z,t=0) = 0                                              (1.4c)
#
#   The method of lines (MOL) solution for eqs. (1.1) to
#   (1.4) is coded below.  Specifically, the spatial
#   derivative in the fluid balance, u1_z in eq. (1.1b),
#   is replaced by one of four approximations as selected
#   by the variable ifd.
#
# Access ODE integrator
  library("deSolve");
#
# Access files
  setwd("G:/comp3/chromatography/R/ex1");
  source("pde_1.R") ;source("step.R")  ;
  source("dss004.R");source("dss012.R");
  source("dss020.R");source("vanl.R")  ;
#
# Declare (preallocate) array for plotted solutions
  nout=51;
  plot_2=matrix(0,nrow=nout,ncol=2);
#
# Step through cases
  for(ncase in 1:2){
#
#   Model parameters
      v=1; eps=0.4; u10=0; u20=0;
     c1=1;    c2=1;  kf=1;  kr=1;
    zL=50;    n=41;
    if(ncase==1){ ifd=3; }
    if(ncase==2){ ifd=4; }
```

```
#
# Level of output
#
#   Detailed output   - ip = 1
#
#   Brief (IC) output - ip = 2
#
  ip=2;
#
# Initial condition
  u0=rep(0,2*n);
  for(i in 1:n){
      u0[i]=u10;
    u0[i+n]=u20;
  }
  t0=0;tf=150;
  tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
  ncall=0;
#
# ODE integration
  out=ode(func=pde_1,times=tout,y=u0);
#
# Store solution
  u1=matrix(0,nrow=nout,ncol=n);
  u2=matrix(0,nrow=nout,ncol=n);
  t=rep(0,nout);
  for(it in 1:nout){
  for(iz in 1:n){
    u1[it,iz]=out[it,iz+1];
    u2[it,iz]=out[it,iz+1+n];
  }
    t[it]=out[it,1];
  }
#
# Display ifd, ncase
  cat(sprintf("\n ifd = %2d   ncase = %2d",ifd,ncase));
#
# Display numerical solution
  if(ip==1){
  cat(sprintf(
    "\n\n     t     u1(z=zL,t)  rate(z=zL,t)\n"));
  u2eq=rep(0,nout);rate=rep(0,nout);
  for(it in 1:nout){
    u2eq[it]=c1*u1[it,n]/(1+c2*u1[it,n]);
    rate[it]=kf*u1[it,n]*(u2eq[it]-u2[it,n])-kr*u2[it,n];
    cat(sprintf(
      "%7.2f%12.4f%12.4f\n",t[it],u1[it,n],rate[it]));
  }
  }
#
# Store solution for plotting
  tplot=rep(0,nout);
  for(it in 1:nout){
    plot_2[it,ncase]=u1[it,n];
    tplot[it]=t[it];
  }
#
# Calls to ODE routine
  cat(sprintf("\n ncall = %4d\n",ncall));
#
# Next case
  }
#
# Plot for u1(z=zL,t)
  par(mfrow=c(1,1))
  plot(tplot,plot_2[,1],
      xlab="t",ylab="u1(z=zL,t)",
      xlim=c(0,tplot[nout]),ylim=c(0,1),
      main="1 - 5pbu, 2 - van Leer",
      type="l",lwd=2);
   points(tplot,plot_2[,1], pch="1",lwd=2);
    lines(tplot,plot_2[,2],type="l",lwd=2);
   points(tplot,plot_2[,2], pch="2",lwd=2);
```

Listing 1.4: Main program `pde_1_main` for comparison of the 5pbu and van Leer solutions for `ncase=2`

Listing 1.4 is similar to Listing 1.1, so we note only the differences here.

- The routine `vanl` for the van Leer flux limiter is included.

```
  source("dss020.R");source("vanl.R");
```

- An array (matrix) is defined with the `matrix` utility for the two solutions `ifd=3,4`, `ncase=2`.

```
#
# Declare (preallocate) array for plotted solutions
  nout=51;
  plot_2=matrix(0,nrow=nout,ncol=2);
```

In this way, the two solutions can be superimposed on the same plot (Figs. 1.10, 1.11).

- Two cases are programmed corresponding to `ifd=3,4` (5pbu, van Leer).

```
#
# Step through cases
  for(ncase in 1:2){
#
#   Model parameters
      v=1; eps=0.4; u10=0; u20=0;
     c1=1;    c2=1;  kf=1;  kr=1;
    zL=50;    n=41;
    if(ncase==1){ ifd=3; }
    if(ncase==2){ ifd=4; }
```

- Brief numerical output is selected.

```
  ip=2;
```

- The numerical solution for `ncase=1,2` (`ifd=3,4`) is placed in array `plot_2`.

```
#
# Store solution for plotting
  tplot=rep(0,nout);
  for(it in 1:nout){
    plot_2[it,ncase]=u1[it,n];
    tplot[it]=t[it];
  }
```

Note that this is at `i=n` corresponding to $z = z_L$.

- At the end of the second solution, both solutions (for 5pbu and van Leer) are plotted as a composite plot identified with `1` and `2`.

```
#
# Plot for u1(z=zL,t)
  par(mfrow=c(1,1))
  plot(tplot,plot_2[,1],
      xlab="t",ylab="u1(z=zL,t)",
      xlim=c(0,tplot[nout]),ylim=c(0,1),
      main="1 - 5pbu, 2 - van Leer",
      type="l",lwd=2);
   points(tplot,plot_2[,1], pch="1",lwd=2);
    lines(tplot,plot_2[,2],type="l",lwd=2);
   points(tplot,plot_2[,2], pch="2",lwd=2);
```

The composite plot is produced with a combination of three utilities, `plot`, `lines`, `points`. The result is in Fig. 1.10, and when the two `points` are not included (by making those statements comments), Fig. 1.11 results.

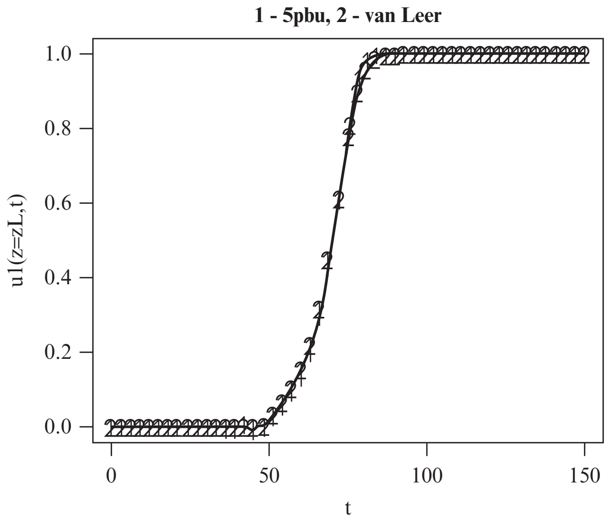

Figure 1.10 Comparison of 5pbu (`ifd=3`) and van Leer (`ifd=4`)

The numerical output is

```
ifd =  3   ncase =  1
ncall = 1312

ifd =  4   ncase =  2
ncall = 5620
```

so that the van Leer limiter (from `idf=4`) requires substantially more computation than 5pbu (from `ifd=3`).

Fig 1.10 indicates that the two solutions agree closely. This is further confirmed in Fig. 1.11 in which the numbered points have been surppressed.

The fact that the two solutions agree closely does not prove that they are accurate and correct. However, this agreement resulting from two different algorithms, 5pbu and van Leer, suggests that the two solutions are accurate. We can view this approach of comparing solutions from two different algorithms as a generalized form of p refinement in which not only is the order of the approximation changed (p usually denotes the order), but the algorithm itself is changed.

To study this approach, we could consider other flux limiters. A set of limiters is provided in [1], pp 40-43, and these can easily be used in place of the van Leer limiter in `vanl`.

This completes the discussion of the model of eqs. (1.1) to (1.4). In particular, the unit step of eq. (1.5d) provides a stringent test of the numerical algorithms (within the MOL format). We now go through a similar analysis using a less stringent BC function, a pulse in place of the unit step.

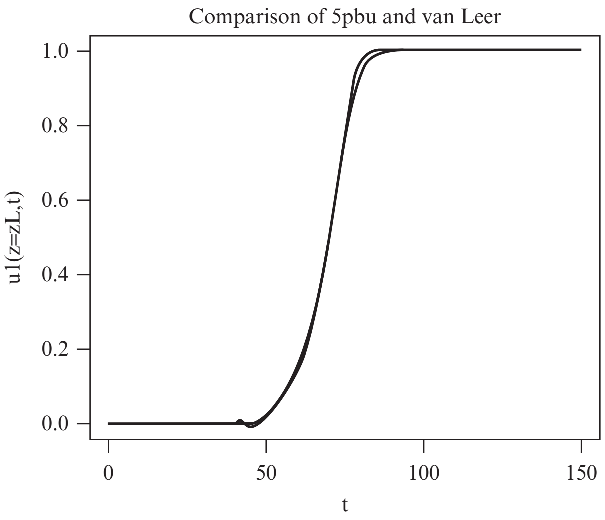

Figure 1.11 Comparison of 5pbu (`ifd=3`) and van Leer (`ifd=4`)

(1.3.3) FDs, pulse BC

$g_1(t)$ in BC (1.4b) is a cosine pulse, defined as a function of the Lagrangian variable $(t - z/v)$.

$$\text{pulse(t)} = \begin{cases} 0, & (t - z/v) < 0 \\ 1 - \cos(\omega(t - z/v), & 0 \leq (t - z/v) < \pi/2 \\ 1 + \cos(\omega(t - z/v), & \pi/2 \leq (t - z/v) \leq \pi \\ 0, & (t - z/v) > \pi \end{cases} \tag{1.7a}$$

pulse(t) is a smooth (continuous) function of t in contrast with the step of eq. (1.5d). Therefore, we would expect that calculating solutions to eqs. (1.1) to (1.4) would be easier than for the step function. However, it is included in this analysis since for the multi component case considered subsequently, we can observe the separation of the component pulses as would occur in a chromatographic column.

The pulse of eq. (1.7a) is programmed in function `pulse`.

```
  pulse=function(t,z,v) {
#
# Function pulse computes a pulse function
#
  w=0.05;tzv=t-z/v;wtzv=w*tzv;
  if((wtzv)< 0 ){u1p=0;}
  if((wtzv>=0   )&(wtzv< pi/2)) {u1p=1-cos(wtzv);}
  if((wtzv>=pi/2)&(wtzv<=pi  )) {u1p=1+cos(wtzv);}
  if((wtzv)>pi) {u1p=0;}
#
# Return pulse
  return(c(u1p));
}
```

Listing 1.5: Function pulse from eq. (1.7a)

The programming in Listing 1.5 follows directly from eq. (1.7a) and therefore does not require elaboration. Note that $\omega = 0.05$ which was selected to give a pulse with a suitable spread for the graphical output (plots of the following figures) for $0 \leq t \leq 150$.

pulse of Listing 1.5 is used in Listings 1.1 to 1.3 by merely replacing the use of step with pulse. So discussion of the programming details is not required. The exact solution for ncase=1 is, from eq. (1.6a)

$$u_1(z,t) = \text{pulse}(t - z/v) \tag{1.7b}$$

As before, the analytical solution of eq. (1.7b) can be used to give the exact error in the numerical solution for ncase=1 for the various spatial differentiation routines (ifd=1,2,3,4). We would expect that the agreement between the numerical and analytical solutions would be better than for the step since the pulse of eq. (1.7a) is smoother than the step of eq. (1.5d).

The numerical and graphical output from Listings 1.1 to 1.3 follows.

In Fig. 1.12 (ncase=1), the numerical diffusion for 2pu (ifd=1) is substantial, particularly at the peak which is reduced from 1 to approximately 0.7. The defined vertical scaling was used in producing this plot (ylim=c(0,1)) since the numerical solution is

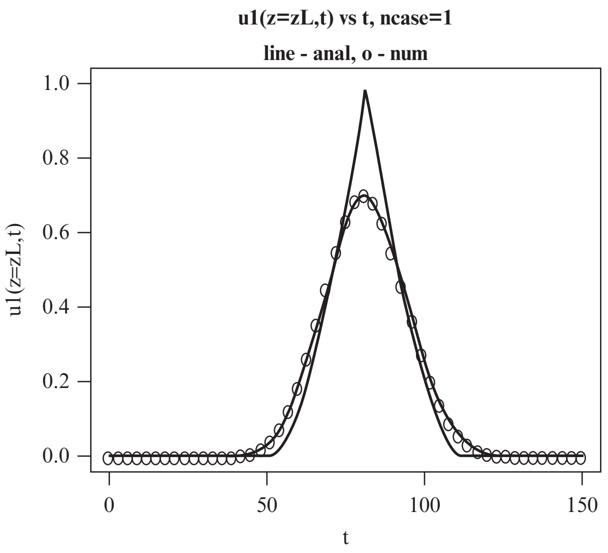

Figure 1.12 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b) ncase=1, ifd=1, pulse BC

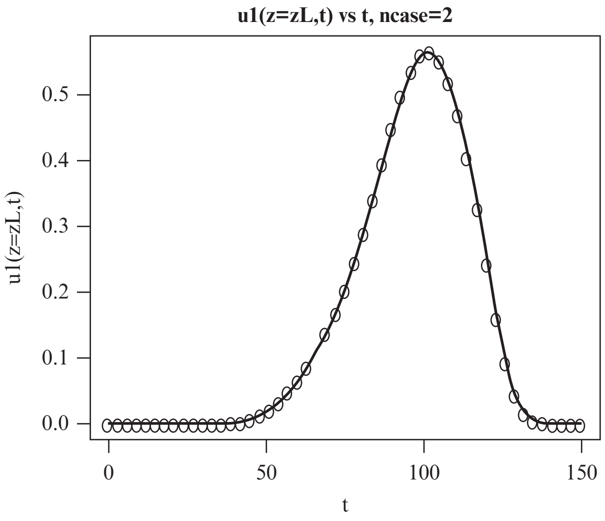

Figure 1.13 Numerical solution of eqs. (1.1b), (1.2b), `ncase=2`, `ifd=1`, pulse BC

plotted first. Automatic scaling gives a vertical axis of `0` to `0.8`, and then the peak of the exact solution plotted next is truncated (the plot cannot reach `1`).

In Fig. 1.13 (`ncase=2`), the numerical solution appears smooth, but the accuracy of the solution cannot be ascertained. Later we compare the `ncase=2` solutions for different spatial differentiators.

Execution of the routines for `ifd=2` gives the graphical output in Fig. 1.14 (the numerical output and the plotted solution for `ncase=2` are not given here).

In Fig. 1.14, the numerical and analytical solutions agree closely, except for some numerical oscillation in the downstream portion of the solution, which is not unexpected since we found previously that the 5pc approximations (`ifd=2`) oscillate (for steep moving fronts such as the unit step).

Execution of the routines for `ifd=3` gives the graphical output in Fig. 1.15 (the numerical output and the plotted solution for `ncase=2` are not given here).

In Fig. 1.15, the numerical and analytical solutions agree closely, with no apparent numerical diffusion or oscillation.

(1.3.4) Flux limiters, pulse BC

We next consider the solutions to eqs. (1.1b) and (1.2b) with the derivative $\frac{\partial u_1}{\partial z}$ in eq. (1.1b) approximated with a flux limiter. Execution of the routines for `ifd=4` gives the graphical output in Fig. 1.16 (the numerical output and the plotted solution for `ncase=2` are not given here).

In Fig. 1.16, the peak of the numerical solution is not resolved as closely as we might expect. Also, the van Leer limiter (`ifd=4`) required substantially more calculations (higher value of `ncall`) than the 5pbu FD (`ifd=3`).

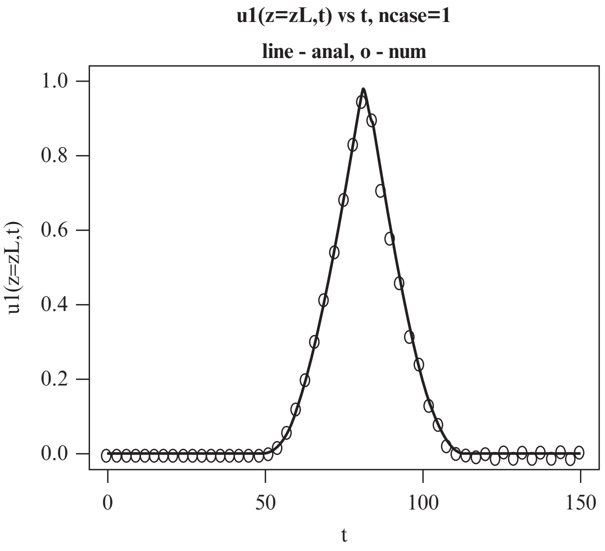

Figure 1.14 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b) `ncase=1`, `ifd=2`, pulse BC

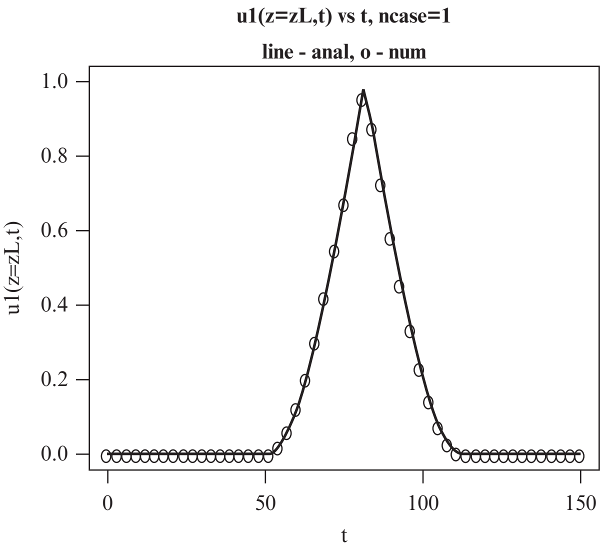

Figure 1.15 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b) `ncase=1`, `ifd=3`, pulse BC

The error in the numerical solution at the peak is not unexpected when we consider how rapidly the solution changes at the peak. In fact, the first derivative of the solution is discontinuous at the peak. To show this, the derivative from the segment $u_1 = 1 - \cos(\omega(t - z/v))$ (from eq. (1.7a)) is $\omega \sin(\omega(t - z/v))$ and at the peak, $\omega(t - z/v) = \pi/2$ the derivative is ω. The derivative from the segment $u_1 = 1 + \cos(\omega(t - z/v))$ is

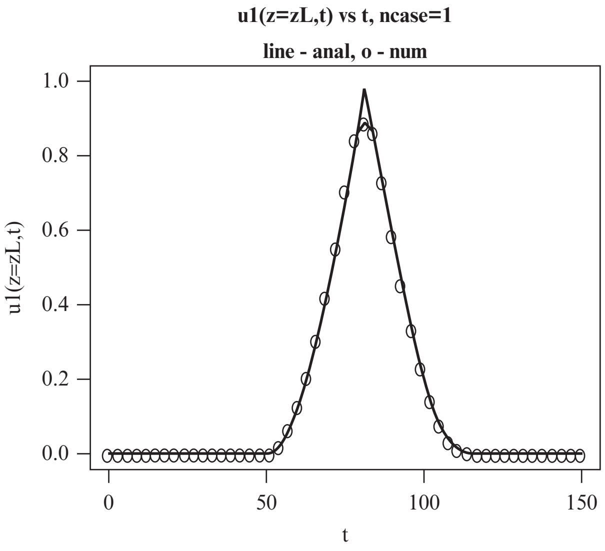

Figure 1.16 Comparison of the numerical and analytical solutions of eqs. (1.1b), (1.2b) `ncase=1`, `ifd=4`, pulse BC

$-\omega \sin(\omega(t - z/v))$ and at the peak, $\omega(t - z/v) = \pi/2$ the derivative is $-\omega$. Therefore, the derivative at the peak is a finite step of magnitude 2ω (as demonstrated in Fig. 1.16), and the calculation of $\dfrac{\partial u_1}{\partial z}$ is difficult numerically.

In summary, for `ncase=1` the comparison of the numerical and analytical solutions demonstrated smaller differences than for the unit step as expected since the cosine pulse of eq. (1.7a) is smoother than the step of eq. (1.5d). But the differences are large enough that some experimentation for `ncase=2` is suggested.

We now compare the numerical solutions for `ifd=3,4` and `ncase=2` as was done previously for the unit step. Again, this is easily accomplished by replacing the step BC with the pulse BC. The graphical output is in Fig. 1.17 (with points) and Fig. 1.18 (without points).

The differences between 5pbu and van Leer are clear. If they are considered excessive, one possibility to improve the agreement would be to use more points in z since $n = 41$ is a rather coarse grid (h refinement). Another possibility would be to use another flux limiter (e.g., from the set in [1], pp 40-42) which is a generalized form of p refinement in which the algorithm is changed.

The preceding discussion is for a single component with concentrations $u_1(z, t)$ (fluid) and $u_2(z, t)$ (adsorbent). The intention is to demonstrate the features and performance of some approximations for the derivative in $\dfrac{\partial u_1}{\partial z}$ in eq. (1.1b). However, movement of fronts through the chromatographic column (Fig. 1.1) for multi component systems is of primary interest in the application of chromatographic separation and analysis. We therefore next consider how the preceding routines can be extended to two components (and thus, for any number of components).

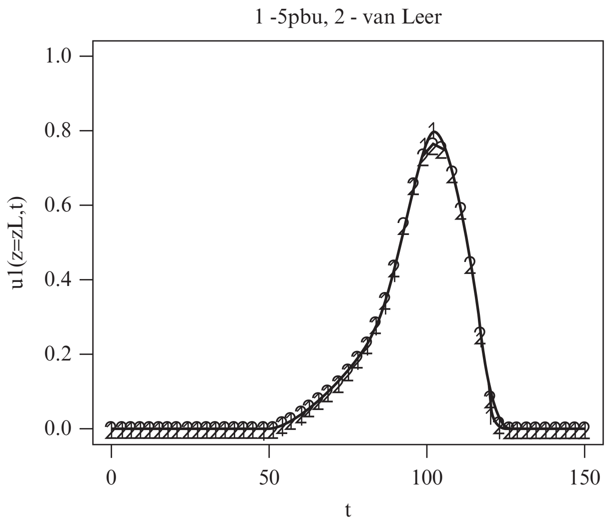

Figure 1.17 Comparison of 5pbu (`ifd=3`) and van Leer (`ifd=4`), `ncase=2`

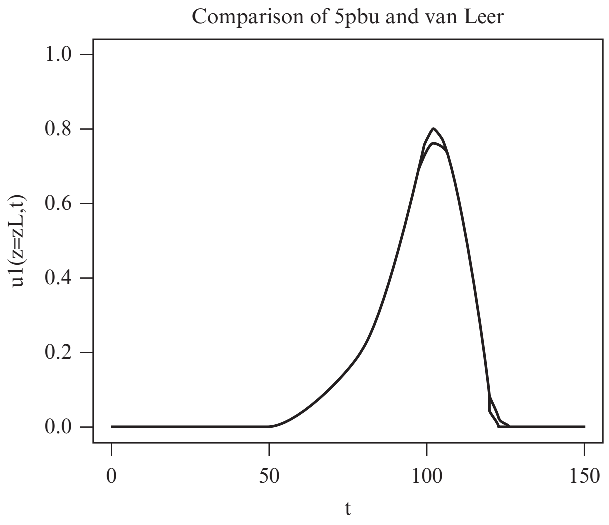

Figure 1.18 Comparison of 5pbu (`ifd=3`) and van Leer (`ifd=4`), `ncase=2`

(1.4) Multi component model

The interest in chromatography is primarily within the context of analysis and separation of multi component mixtures. Here we consider how the previous single component model can be extended to multi component applications.

Eqs. (1.1b), (1.2b) can be extended to a system of two components, with dependent variables u_1, u_2, u_3, u_4[14]

$$\frac{\partial u_1}{\partial t} = -\frac{\partial (vu_1)}{\partial z} - \frac{(1-\epsilon)}{\epsilon}(k_f u_1(u_2^e - u_2) - k_r u_2) \tag{1.8a}$$

$$\frac{\partial u_2}{\partial t} = k_f u_1(u_2^e - u_2) - k_r u_2 \tag{1.8b}$$

$$\frac{\partial u_3}{\partial t} = -\frac{\partial (vu_3)}{\partial z} - \frac{(1-\epsilon)}{\epsilon}(k_f u_3(u_4^e - u_4) - k_r u_4) \tag{1.8c}$$

$$\frac{\partial u_4}{\partial t} = k_f u_3(u_4^e - u_4) - k_r u_4 \tag{1.8d}$$

The adsorbent equilibrium concentrations, u_2^e, u_4^e are given by the isotherms

$$u_2^e = \frac{c_1 u_1}{1 + c_2 u_1};\ u_4^e = \frac{c_3 u_3}{1 + c_4 u_3} \tag{1.9a,b}$$

These isotherms are single component, but they could easily be extended to the multi component case, e.g., $u_2^e = f_2(u_1, u_3), u_4^e = f_4(u_1, u_3)$.

The ICs for eqs. (1.8) are

$$u_1(z, t=0) = f_1(z),\ u_2(z, t=0) = f_2(z),\ u_3(z, t=0) = f_3(z),$$
$$u_1(z, t=0) = f_4(z) \tag{1.10a,b,c,d}$$

The BCs for eqs. (1.8) are

$$u_1(z=0, t) = g_1(t),\ u_3(z=0, t) = g_3(t) \tag{1.11a,b}$$

We will next consider eqs. (1.8) to (1.11) for homogeneous ICs, $f_1(z) = f_2(z) = f_3(z) = f_4(z) = 0$, and pulse function BCs, $g_1(t) = g_3(t) =$ pulse(t).

(1.5) MOL routines

The main program and subordinate routines for the multi component model are next. A main program for eqs. (1.8) to (1.11) is in Listing 1.6.

(1.5.1) Main program

```
#
# Delete previous workspaces
  rm(list=ls(all=TRUE))
#
#   1D, two component, chromatography model
#
```

[14]We have followed the usual convention of naming PDE dependent variables with u and a number. The alternative is to use variables names that are more closely identified with physical variables, e.g., $c_{f,1}, c_{a,1}, c_{f,2}, c_{a,2}$ where f, a denote fluid and adsorbent, respectively.

```
#   The ODE/PDE system is
#
#   u1_t = -v*u1_z - (1 - eps)/eps*rate1
#
#   u2_t = rate1
#
#   u3_t = -v*u3_z - (1 - eps)/eps*rate3
#
#   u4_t = rate3
#
#   Boundary conditions
#
#     u1(z=0,t) = pulse(t)
#
#     u3(z=0,t) = pulse(t)
#
#   Initial conditions
#
#     u1(z,t=0) = 0
#
#     u2(z,t=0) = 0
#
#     u3(z,t=0) = 0
#
#     u4(z,t=0) = 0
#
#   The method of lines (MOL) solution is coded below.
#   Specifically, the spatial derivatives in the fluid
#   balances, u1_z, u3_z, are replaced by one of four
#   approximations as selected by the variable ifd.
#
# Access ODE integrator
  library("deSolve");
#
# Access files
  setwd("G:/chap1");
  source("pde_1.R") ;source("pulse.R") ;
  source("dss004.R");source("dss012.R");
  source("dss020.R");source("vanl.R")  ;
#
# Step through cases
  for(ncase in 1:2){
#
#   Model parameters
      v=1; eps=0.4;
    u10=0;   u20=0; u30=0; u40=0;
```

```
     c1=1;    c2=1;  c3=1;  c4=1;
    zL=50;    n=41;
    if(ncase==1){ kf1=0; kr2=0; kf3=0; kr4=0;}
    if(ncase==2){ kf1=1; kr2=1; kf3=0; kr4=0;}
#
# Select an approximation for the convective derivative u1z
#
#   ifd = 1: Two point upwind approximation
#
#   ifd = 2: Centered approximation
#
#   ifd = 3: Five point, biased upwind approximation
#
#   ifd = 4: van Leer flux limiter
#
  ifd=3;
#
# Level of output
#
#   Detailed output   - ip = 1
#
#   Brief (IC) output - ip = 2
#
  ip=1;
#
# Initial condition
  u0=rep(0,4*n);
  for(i in 1:n){
    u0[i]=    u10;
    u0[i+n]=  u20;
    u0[i+2*n]=u30;
    u0[i+3*n]=u40;
  }
  t0=0;tf=150;nout=51;
  tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
  ncall=0;
#
# ODE integration
  out=ode(func=pde_1,times=tout,y=u0);
#
# Store solution
  u1=matrix(0,nrow=nout,ncol=n);
  u2=matrix(0,nrow=nout,ncol=n);
  u3=matrix(0,nrow=nout,ncol=n);
  u4=matrix(0,nrow=nout,ncol=n);
  t=rep(0,nout);
  for(it in 1:nout){
  for(iz in 1:n){
    u1[it,iz]=out[it,iz+1];
    u2[it,iz]=out[it,iz+1+n];
    u3[it,iz]=out[it,iz+1+2*n];
    u4[it,iz]=out[it,iz+1+3*n];
  }
    t[it]=out[it,1];
  }
#
# Display ifd, ncase
  cat(sprintf("\n ifd = %2d   ncase = %2d",ifd,ncase));
#
# Display numerical solution
  if(ip==1){
  cat(sprintf(
    "\n\n     t     u1(z=zL,t)  rate1(z=zL,t)\n"));
  cat(sprintf(
    "\n\n     t     u3(z=zL,t)  rate3(z=zL,t)\n"));
  u2eq=rep(0,nout);  u4eq=rep(0,nout);
  rate1=rep(0,nout);rate3=rep(0,nout);
  for(it in 1:nout){
    u2eq[it]=c1*u1[it,n]/(1+c2*u1[it,n]);
    u4eq[it]=c3*u3[it,n]/(1+c4*u3[it,n]);
    rate1[it]=kf1*u1[it,n]*(u2eq[it]-u2[it,n])-kr2*u2[it,n];
    rate3[it]=kf3*u3[it,n]*(u4eq[it]-u4[it,n])-kr4*u4[it,n];
    cat(sprintf(
      "%7.2f%12.4f%12.4f\n"  ,t[it],u1[it,n],rate1[it]));
    cat(sprintf(
      "%7.2f%12.4f%12.4f\n\n",t[it],u3[it,n],rate3[it]));
  }
  }
#
# Store solution for plotting
  u1plot=rep(0,nout);u3plot=rep(0,nout);
   tplot=rep(0,nout);
  for(it in 1:nout){
    u1plot[it]=u1[it,n];
    u3plot[it]=u3[it,n];
     tplot[it]=t[it];
  }
#
# Calls to ODE routine
  cat(sprintf("\n ncall = %4d\n",ncall));
#
# Plot for u1(z=zL,t)
# ncase = 1
  if(ncase==1){
  par(mfrow=c(1,1))
  plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t),u3(z=zL,t)",
    lwd=2,main="u1(z=zL,t),u3(z=zL,t) vs t, ncase=1\n
    line - anal, o - num",type="l",xlim=c(0,tplot[nout]),
    ylim=c(-0.1,1.1));
  points(tplot,u1plot,pch="1",lwd=2);
   lines(tplot,u3plot,lwd=2,type="l");
  points(tplot,u3plot,pch="3",lwd=2);
  }
#
# Analytical solution, ncase=1
  if(ncase==1){
  u1expl=rep(0,nout);u3expl=rep(0,nout);
  for(it in 1:nout){
    u1expl[it]=pulse(tplot[it],zL,v);
    u3expl[it]=pulse(tplot[it],zL,v);
  }
  lines(tplot,u1expl,lwd=2,type="l");
  lines(tplot,u3expl,lwd=2,type="l");
  }
#
# ncase = 2
  if(ncase==2){
  par(mfrow=c(1,1))
  plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t),u3(z=zL,t)",
    lwd=2,main="u1(z=zL,t),u3(z=zL,t) vs t, ncase=2",
    type="l",xlim=c(0,tplot[nout]),ylim=c(-0.1,1.1));
  points(tplot,u1plot,pch="1",lwd=2);
   lines(tplot,u3plot,lwd=2,type="l");
  points(tplot,u3plot,pch="3",lwd=2);
  }
#
# Next case
  }
```

Listing 1.6: Main program for the multi component model, eqs. (1.8) to (1.11)

Listing 1.6 is similar to Listing 1.1. Therefore, the differences will be emphasized next.

- A block of documentation comments for the two component model, followed by access to the library of ODE solvers `deSolve` and the routines specific to the coding are at the beginning as before (e.g., in Listings 1.1 and 1.4).
- Two cases are programmed in a `for`

```
#
# Step through cases
  for(ncase in 1:2){
#
#   Model parameters
      v=1; eps=0.4;
    u10=0;   u20=0; u30=0; u40=0;
     c1=1;    c2=1;  c3=1;  c4=1;
    zL=50;    n=41;
    if(ncase==1){ kf1=0; kr2=0; kf3=0; kr4=0;}
    if(ncase==2){ kf1=1; kr2=1; kf3=0; kr4=0;}
```

For `ncase=1`, no transfer of the two components to the adsorbent occurs, so the linear advection equation (1.5a) applies to both components. For `ncase=2`, component `1` is adsorbed while component `2` is not. Thus, we would expect some separation of the two components, that is, differences in $u_1(z = z_L, t)$ and $u_3(z = z_L, t)$. This selective adsorption will be observed in the solutions reported next.

- Four spatial differentiators are again programmed corresponding to `ifd=1,2,3,4`. The 5pbu FD `ifd=3` will be used primarily since it gives little numerical diffusion and oscillation, and is computationally efficient, as observed previously for the one component case.
- Homogeneous (zero) ICs are programmed for $u_1(z, t = 0), u_2(z, t = 0), u_3(z, t = 0), u_4(z, t = 0)$.

```
#
# Initial condition
  u0=rep(0,4*n);
  for(i in 1:n){
    u0[i]=    u10;
    u0[i+n]=  u20;
    u0[i+2*n]=u30;
    u0[i+3*n]=u40;
  }
```

Again `n=41` so the number of ODEs in the MOL analysis (of eqs. (1.8a), (1.8b)) is now $4(41) = 164$.

- The variation in t is again $0 \leq t \leq 150$ with 51 output points (including $t = 0$).

```
  t0=0;tf=150;nout=51;
  tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
  ncall=0;
```

- Integrator `ode` is used to compute the numerical solution in array `out`. The four dependent variables are then placed in arrays `u1,u2,u3,u4` that are 2D for the variations in z and t.

```
#
# ODE integration
  out=ode(func=pde_1,times=tout,y=u0);
#
# Store solution
  u1=matrix(0,nrow=nout,ncol=n);
  u2=matrix(0,nrow=nout,ncol=n);
  u3=matrix(0,nrow=nout,ncol=n);
  u4=matrix(0,nrow=nout,ncol=n);
  t=rep(0,nout);
  for(it in 1:nout){
  for(iz in 1:n){
    u1[it,iz]=out[it,iz+1];
    u2[it,iz]=out[it,iz+1+n];
    u3[it,iz]=out[it,iz+1+2*n];
    u4[it,iz]=out[it,iz+1+3*n];
  }
    t[it]=out[it,1];
  }
```

Also, t (`out[it,1]`) is placed in vector `t` as before.

- `ifd` and `ncase` are displayed at the beginning of the output. Then, for `ip=1` (detailed numerical output), the equilibrium concentrations u_2^e, u_4^e are computed from eqs. (1.9), and the adsorption rates in eqs. (1.8) are computed and placed in vectors and displayed.

```
#
# Display ifd, ncase
  cat(sprintf("\n ifd = %2d   ncase = %2d",ifd,ncase));
#
# Display numerical solution
  if(ip==1){
  cat(sprintf(
    "\n\n     t     u1(z=zL,t)  rate1(z=zL,t)\n"));
  cat(sprintf(
    "\n\n     t     u3(z=zL,t)  rate3(z=zL,t)\n"));
  u2eq=rep(0,nout);  u4eq=rep(0,nout);
  rate1=rep(0,nout);rate3=rep(0,nout);
  for(it in 1:nout){
    u2eq[it]=c1*u1[it,n]/(1+c2*u1[it,n]);
    u4eq[it]=c3*u3[it,n]/(1+c4*u3[it,n]);
    rate1[it]=kf1*u1[it,n]*(u2eq[it]-u2[it,n])-kr2*u2[it,n];
    rate3[it]=kf3*u3[it,n]*(u4eq[it]-u4[it,n])-kr4*u4[it,n];
    cat(sprintf(
      "%7.2f%12.4f%12.4f\n"  ,t[it],u1[it,n],rate1[it]));
    cat(sprintf(
      "%7.2f%12.4f%12.4f\n\n",t[it],u3[it,n],rate3[it]));
  }
  }
```

- The exiting concentrations $u_1(z = z_L, t), u_3(z = z_L, t)$ are placed in vectors for subsequent plotting (note the use of `n` corresponding to $z = z_L$).

```
#
# Store solution for plotting
  u1plot=rep(0,nout);u3plot=rep(0,nout);
   tplot=rep(0,nout);
  for(it in 1:nout){
    u1plot[it]=u1[it,n];
    u3plot[it]=u3[it,n];
     tplot[it]=t[it];
  }
```

- The number of calls to the MOL/ODE routine `pde_1` (discussed next) is displayed. Then $u_1(z = z_L, t), u_3(z = z_L, t)$ are plotted against t and identified with the characters `1,3` in the plot using the utilities `plot, points, lines`.

```
#
# Calls to ODE routine
  cat(sprintf("\n ncall = %4d\n",ncall));
#
# Plot for u1(z=zL,t)
# ncase = 1
  if(ncase==1){
  par(mfrow=c(1,1))
  plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t),u3(z=zL,t)",
    lwd=2,main="u1(z=zL,t),u3(z=zL,t) vs t, ncase=1\n
    line - anal, o - num",type="l",xlim=c(0,tplot[nout]),
    ylim=c(-0.1,1.1));
  points(tplot,u1plot,pch="1",lwd=2);
   lines(tplot,u3plot,lwd=2,type="l");
  points(tplot,u3plot,pch="3",lwd=2);
  }
```

- For `ncase=1` (no adsorption), the analytical solution of eqs. (1.8) is placed in two arrays by a call to `pulse` of Listing 1.5.

```
#
# Analytical solution, ncase=1
  if(ncase==1){
  u1expl=rep(0,nout);u3expl=rep(0,nout);
  for(it in 1:nout){
    u1expl[it]=pulse(tplot[it],zL,v);
    u3expl[it]=pulse(tplot[it],zL,v);
  }
  lines(tplot,u1expl,lwd=2,type="l");
  lines(tplot,u3expl,lwd=2,type="l");
  }
```

The analytical solutions are then superimposed on the `ncase=1` plot with the `lines` utility. The superposition takes place because the `par(mfrow=c(1,1))` is not repeated (for a separate plot).

- For `ncase=2` (with adsorption of component 1), the solutions $u_1(z = z_L, t), u_3(z = z_L, t)$ are plotted as lines with points.

```
#
# ncase = 2
  if(ncase==2){
  par(mfrow=c(1,1))
  plot(tplot,u1plot,xlab="t",ylab="u1(z=zL,t),u3(z=zL,t)",
    lwd=2,main="u1(z=zL,t),u3(z=zL,t) vs t, ncase=2",
    type="l",xlim=c(0,tplot[nout]),ylim=c(-0.1,1.1));
  points(tplot,u1plot,pch="1",lwd=2);
   lines(tplot,u3plot,lwd=2,type="l");
  points(tplot,u3plot,pch="3",lwd=2);
  }
#
# Next case
  }
```

The final } concludes the `for` in `ncase`.

(1.5.2) MOL/ODE routine

The MOL/PDE routine `pde_1` called by `ode` in Listing 1.7 is next.

```
  pde_1=function(t,u,parms) {
#
# Function pde_1 computes the t derivative vector of the u vector
#
# One vector to four PDEs
  u1=rep(0,n);u2=rep(0,n);
  u3=rep(0,n);u4=rep(0,n);
  for (i in 1:n){
    u1[i]=u[i];
    u2[i]=u[i+n];
    u3[i]=u[i+2*n];
    u4[i]=u[i+3*n];
  }
#
# Boundary condition
  u1[1]=pulse(t,0,v);
  u3[1]=pulse(t,0,v);
#
# First order spatial derivative
#
#   ifd = 1: Two point upwind finite difference (2pu)
    if(ifd==1){ u1z=dss012(0,zL,n,u1,v); }
    if(ifd==1){ u3z=dss012(0,zL,n,u3,v); }
#
#   ifd = 2: Five point center finite difference (5pc)
    if(ifd==2){ u1z=dss004(0,zL,n,u1); }
    if(ifd==2){ u3z=dss004(0,zL,n,u3); }
#
#   ifd = 3: Five point biased upwind approximation (5pbu)
    if(ifd==3){ u1z=dss020(0,zL,n,u1,v); }
    if(ifd==3){ u3z=dss020(0,zL,n,u3,v); }
#
#   ifd = 4: van Leer flux limiter
    if(ifd==4){ u1z=vanl(0,zL,n,u1,v); }
    if(ifd==4){ u3z=vanl(0,zL,n,u3,v); }
#
# Temporal derivatives, mass transfer rate
    u1t=rep(0,n);  u2t=rep(0,n);
    u3t=rep(0,n);  u4t=rep(0,n);
   u2eq=rep(0,n);rate1=rep(0,n);
   u4eq=rep(0,n);rate3=rep(0,n);
#
#    u1t, u2t
     for(i in 1:n){
       u2eq[i]=c1*u1[i]/(1+c2*u1[i]);
       rate1[i]=kf1*u1[i]*(u2eq[i]-u2[i])-kr2*u2[i];
       if(i==1){
         u1t[i]=0;
       }else{
         u1t[i]=-v*u1z[i]-(1-eps)/eps*rate1[i];
       }
         u2t[i]=rate1[i];
     }
```

```
#
#    u3t, u4t
     for(i in 1:n){
       u4eq[i]=c3*u3[i]/(1+c4*u3[i]);
       rate3[i]=kf3*u3[i]*(u4eq[i]-u4[i])-kr4*u4[i];
       if(i==1){
         u3t[i]=0;
       }else{
         u3t[i]=-v*u3z[i]-(1-eps)/eps*rate3[i];
       }
         u4t[i]=rate3[i];
     }
#
# Four PDEs to one vector
  ut=rep(0,4*n);
  for(i in 1:n){
    ut[i]    =u1t[i];
    ut[i+n]  =u2t[i];
    ut[i+2*n]=u3t[i];
    ut[i+3*n]=u4t[i];
  }
#
# Increment calls to pde_1
  ncall<<-ncall+1;
#
# Return derivative vector
  return(list(c(ut)));
}
```

Listing 1.7: MOL/ODE routine pde_1 for eqs. (1.8), (1.9) and (1.11)

Listing 1.7 is similar to Listing 1.2 so only the differences are emphasized next.

- The function is first defined. Then the input vector u of length $4(41) = 164$ is placed in four vectors of length 41.

```
  pde_1=function(t,u,parms) {
#
# Function pde_1 computes the t derivative vector
# of the u vector
#
# One vector to four PDEs
  u1=rep(0,n);u2=rep(0,n);
  u3=rep(0,n);u4=rep(0,n);
  for (i in 1:n){
    u1[i]=u[i];
    u2[i]=u[i+n];
    u3[i]=u[i+2*n];
    u4[i]=u[i+3*n];
  }
```

This use of four arrays facilitates the programming of eqs. (1.8) (four PDEs).

- A pulse BC is used at $z = 0$ for both eqs. (1.8a) and (1.8c) for `ncase=1,2`.

```
#
# Boundary condition
  u1[1]=pulse(t,0,v);
  u3[1]=pulse(t,0,v);
```

In other words, this programming is for BCs (1.11) with a pulse. Since the programming is the same for eqs. (1.8a) and (1.8c), the numerical solutions should be the same (for `ncase=1`), which serves as a check on the coding in `pde_1` of Listing (1.6).

- The derivatives in z in eqs. (1.8a) and (1.8c) are computed by one of the four spatial differentiators considered previously (in Listing 1.2). For the model of eqs. (1.8), `ifd=3` is used, for which the programming is

```
#
#   ifd = 3: Five point biased upwind approximation (5pbu)
    if(ifd==3){ u1z=dss020(0,zL,n,u1,v); }
    if(ifd==3){ u3z=dss020(0,zL,n,u3,v); }
```

The two derivatives $\frac{\partial u_1}{\partial z}, \frac{\partial u_3}{\partial z}$ in eqs. (1.8a) and (1.8c) are calculated in this way.

- Arrays are declared for the derivatives in t in eqs. (1.8a), (1.8c), the equilibrium concentrations of eqs. (1.9), and the adsorption rates in eqs. (1.8a) and (1.8c).

```
#
# Temporal derivatives, equilibrium concentrations,
# mass transfer rates
    u1t=rep(0,n);  u2t=rep(0,n);
    u3t=rep(0,n);  u4t=rep(0,n);
   u2eq=rep(0,n);rate1=rep(0,n);
   u4eq=rep(0,n);rate3=rep(0,n);
```

- Eq. (1.8a) is programmed as

```
#
#    u1t, u2t
     for(i in 1:n){
       u2eq[i]=c1*u1[i]/(1+c2*u1[i]);
       rate1[i]=kf1*u1[i]*(u2eq[i]-u2[i])-kr2*u2[i];
       if(i==1){
         u1t[i]=0;
       }else{
         u1t[i]=-v*u1z[i]-(1-eps)/eps*rate1[i];
       }
         u2t[i]=rate1[i];
     }
```

A for is used for the interval $0 \le z \le z_L$. The equilibrium concentration u_2^e, and adsorption rate are calculated and placed in vectors. For $z = 0$, since the entering concentration is defined by BC (1.11a), the derivative in t is set to zero to retain this boundary value. For $z > 0$, eqs. (1.8a), (1.8b) are used to compute the derivatives $\dfrac{\partial u_1}{\partial t}, \dfrac{\partial u_2}{\partial t}$.

- In the same way, the derivatives $\dfrac{\partial u_3}{\partial t}, \dfrac{\partial u_4}{\partial t}$ in eqs. (1.8c), (1.8d) are computed.

```
#
#    u3t, u4t
     for(i in 1:n){
       u4eq[i]=c3*u3[i]/(1+c4*u3[i]);
       rate3[i]=kf3*u3[i]*(u4eq[i]-u4[i])-kr4*u4[i];
       if(i==1){
         u3t[i]=0;
       }else{
         u3t[i]=-v*u3z[i]-(1-eps)/eps*rate3[i];
       }
         u4t[i]=rate3[i];
     }
```

- The four derivatives in t are placed in a single vector, ut, to return to the ODE integrator, ode (called in Listing 1.6).

```
#
# Four PDEs to one vector
  ut=rep(0,4*n);
  for(i in 1:n){
    ut[i]    =u1t[i];
    ut[i+n]  =u2t[i];
    ut[i+2*n]=u3t[i];
    ut[i+3*n]=u4t[i];
  }
```

- The counter for the calls to pde_1 is incremented and returned to the main program of Listing 1.6 with <<-.

```
#
# Increment calls to pde_1
  ncall<<-ncall+1;
```

- The vector `ut` is returned to `ode` with a combination of `c` (the R vector operator), `list` (required by `ode`), and `return`.

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

The final } concludes the function `pde_1`.

The only other required function is for the BCs of eqs. (1.11), `pulse` in this case in Listing 1.5. The output from the routines in Listings 1.6, 1.7 and 1.5 is considered next.

(1.6) Model output, multi component chromatography

The numerical output from Listing 1.6 is not reproduced here to conserve space. The two plots produced by the main program of Listing 1.6 are in Figs. 1.19 and 1.20.

In Fig. 1.19, all four solutions are essentially identical (for 5pbu, $u_1(t, z = z_L)$, $u_3(t, z = z_L)$, numerical and analytical) as expected. This is a worthwhile check on the coding since any errors might produce different solutions.

In Fig. 1.20, the numerical solutions reflect the difference of adsorption for the two components, i.e., component 1 is adsorbed (from the values `kf1=1,kr1=1` in Listing 1.6), while component 2 is not adsorbed (from the values `kf2=0,kr2=0`). As expected, component 2 leaves the column first and a partial separation is effected. While the output

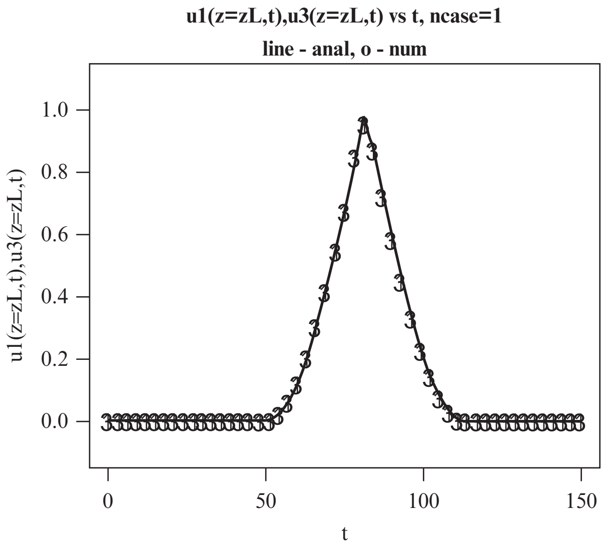

Figure 1.19 Comparison of the numerical and analytical solutions of eqs. (1.8) `ncase=1`, `ifd=3`, pulse BC

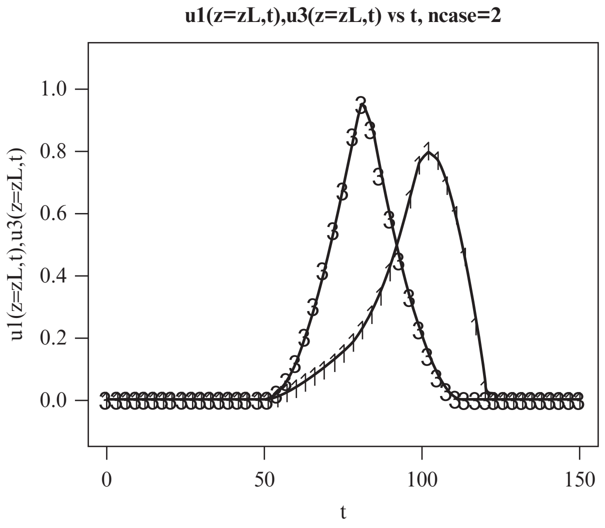

Figure 1.20 Numerical solutions of eqs. (1.8), `ncase=2`, `ifd=3`, pulse BC

stream is not very pure in either component, the purity of the output stream could be substantially enhanced by, for example, increasing the length of the column, changing the velocity v, using multiple columns in sequence, using an adsorbent with different selective properties, etc.

These various options can be studied with the model which would be time consuming and expensive experimentally. Also, additional components can easily be added to the coding in Listings 1.6 and 1.7. This type of design study and possible optimization illustrates the inherent value of a mathematical model.

We now consider a series of PDE models for BMSE applications in subsequent chapters. When considering the models, we will introduce diffusion modeled by second derivatives in the spatial derivatives (*parabolic* PDEs), that is, the use of Fick's first and second laws, including nonlinear extensions. However, the intent in this chapter is to focus on the inherent difficulties of solving PDE models numerically for strongly convective (hyperbolic) systems. The addition of diffusion eases the numerical requirements since steep fronts and discontinuities are smoothed by diffusion.

Reference

[1] Griffiths, G.W., and W.E. Schiesser (2012), *Traveling Wave Analysis of Partial Differential Equations*, Elsevier/Academic Press, Boston, MA