CHAPTER 6

Influenza with Vaccination and Diffusion

6.1 Introduction

The PDE model discussed in this chapter pertains to spatial and temporal effects of vaccination and diffusion in influenza epidemics [1]. The model consists of five 1D partial differential equations (PDEs with the dependent variables

1. $S(x,t)$, susceptibles
2. $V(x,t)$, vaccinated
3. $E(x,t)$, exposed
4. $I(x,t)$, infected
5. $R(x,t)$, recovered

so the model can be classified as SVEIR. x is a spatial variable to account for the spread of influenza and t is time.

The model is represented diagrammatically in Fig. 6.1 ([1], Fig. 1). The intent of the following presentation is to demonstrate:

- A system of five simultaneous (coupled) 1D PDEs programmed in R.
- Specification of no flux boundary conditions (BCs) and Gaussian function initial conditions (ICs).
- Diffusion to model the spatial spread of influenza.

Differential Equation Analysis in Biomedical Science and Engineering: Partial Differential Equation Applications with R, First Edition. William E. Schiesser.

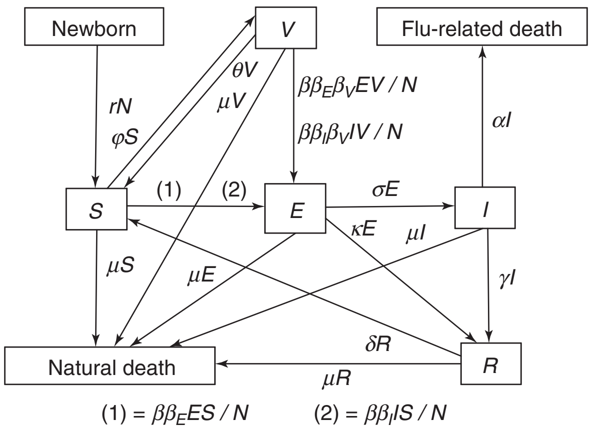

Figure 6.1 Diagram of the influenza model.

- Numerical integration of the model PDEs using the method of lines (MOL).
- A variety of numerical and graphical output formats for the five PDE solutions (dependent variables listed above).
- Rather typical challenges and uncertainties of reproducing numerical solutions to PDE models reported in the open literature.

6.2 Five PDE Model

The five PDEs of the influenza model are listed next (note in particular the five dependent variables $S(x,t), V(x,t), E(x,t), I(x,t), R(x,t)$).

$$\frac{\partial S}{\partial t} = -\beta\beta_E ES - \beta\beta_I IS + \alpha IS - \phi S - rS + \delta R + \theta V + r + d_1\frac{\partial^2 S}{\partial x^2} \tag{6.1a}$$

$$\frac{\partial V}{\partial t} = -\beta\beta_E\beta_V EV - \beta\beta_I\beta_V IV + \alpha IV - rV - \theta V + \phi S + d_2\frac{\partial^2 V}{\partial x^2} \tag{6.1b}$$

$$\frac{\partial E}{\partial t} = \beta\beta_E ES + \beta\beta_I IS + \beta\beta_E\beta_V EV + \beta\beta_I\beta_V IV + \alpha IE - (r+\kappa+\sigma)E + d_3\frac{\partial^2 E}{\partial x^2} \tag{6.1c}$$

$$\frac{\partial I}{\partial t} = \sigma E - (r+\alpha+\gamma)I + \alpha I^2 + d_4\frac{\partial^2 I}{\partial x^2} \tag{6.1d}$$

$$\frac{\partial R}{\partial t} = \kappa E + \gamma I - rR - \delta R + \alpha IR + d_5\frac{\partial^2 R}{\partial x^2} \tag{6.1e}$$

Eqs. (6.1) are second order in x and therefore require two BCs for each PDE. In this case, the following no flux (zero diffusion) BCs are used.

$$\frac{\partial S(x=-3,t)}{\partial x} = \frac{\partial S(x=3,t)}{\partial x} = 0 \tag{6.2a}$$

$$\frac{\partial V(x=-3,t)}{\partial x} = \frac{\partial V(x=3,t)}{\partial x} = 0 \tag{6.2b}$$

$$\frac{\partial E(x=-3,t)}{\partial x} = \frac{\partial E(x=3,t)}{\partial x} = 0 \tag{6.2c}$$

$$\frac{\partial I(x=-3,t)}{\partial x} = \frac{\partial I(x=3,t)}{\partial x} = 0 \tag{6.2d}$$

$$\frac{\partial R(x=-3,t)}{\partial x} = \frac{\partial R(x=3,t)}{\partial x} = 0 \tag{6.2e}$$

The interval in x has been changed from the original $-2 \leq x \leq 2$ in [1] to $-3 \leq x \leq 3$. This increase is based on the observation that the solutions did not have a zero slope at the boundaries in accordance with the BCs (eqs. (6.2)).

Eqs. (6.1) are first order in t and therefore require one IC for each PDE, which is stated subsequently as eqs. (6.3).

The parameters of eqs. (6.1) are listed in Table 6.1. The use of these parameters is elucidated by Fig. 6.1. For example, the RHS terms of eq. (6.1a) for streams coming to the S block are $+\delta R$, $+\theta V$, and $+r$ (the $+$ indicates an increase in the LHS derivative $\partial S/\partial t$). The RHS terms of eq. (6.1a) for streams leaving the S block are $-\beta\beta_E ES$, $-\beta\beta_I IS$, $-\phi S$, and $-\mu S$ (the $-$ indicates a decrease in the LHS derivative $\partial S/\partial t$). The diffusion term $d_1\partial^2 S/\partial x^2$ can either add to or subtract from S depending on the direction of the diffusion and is not included in Fig. 6.1.

TABLE 6.1 Model parameters[a].

Parameter	Description	Value
β	contact rate	0.5140
β_E	ability to cause infection by exposed individuals ($0 \leq \beta_E \leq 1$)	0.2500
β_I	ability to cause infection by infectious individuals ($0 \leq \beta_I \leq 1$)	1
$1-\beta_V$	factor by which the vaccine reduces infection ($0 \leq \beta_V \leq 1$)	Variable
σ^{-1}	mean duration of latency (days)	2.000
γ^{-1}	mean recovery time for clinically ill (days)	5.000
δ^{-1}	duration of immunity loss (days)	365
μ	natural mortality rate	5.500×10^{-8}
r	birth rate	7.140×10^{-5}
κ	recovery rate of latents	1.857×10^{-4}
α	flu-induced mortality rate	9.300×10^{-6}
θ^{-1}	duration of vaccine-induced immunity loss (days)	365
ϕ	rate of vaccination	Variable
d_1	diffusivity, eq. (6.1a)	0.05
d_2	diffusivity, eq. (6.1b)	0.05
d_3	diffusivity, eq. (6.1c)	0.025
d_4	diffusivity, eq. (6.1d)	0.001
d_5	diffusivity, eq. (6.1e)	0

[a]Ref. 1.

In addition, positive terms with α are included in eqs. (6.1), that is, αIS (eq. 6.1a), αIV (eq. 6.1b), αIE (eq. 6.1c), αI^2 (eq. 6.1d), and αIR (eq. 6.1e). The origin and significance of these terms are not explained in the original reference other than with the statement

"After some calculations ..." and are not included in the diagram of the PDE model ([1], Fig. 1, p 124).

In addition to the change in the spatial interval (to $-3 \le x \le 3$), the following changes and/or additions were used in the programming discussed subsequently.

- The interval in t was changed from $0 \le t \le 2000$ days to $0 \le t \le 60$ days. This reduction in the interval in t was made to gain a better resolution of the changes in the numerical solutions with x and t as discussed subsequently.
- The unassigned parameter in [1], β_V, was given the value $\beta_V = 0.9$ which is within the stated limits $0 \le \beta_V \le 1$.
- The unassigned parameter in [1], ϕ, was given the value $\phi = 0.05$. This value was selected to produce a significant variation in the solution of eqs. (6.1) with x and t. As indicated in Fig. 6.1, ϕ determines the transition of S to V (susceptibles to vaccinated) and is therefore a key parameter in determining the effect of vaccination. As the value of ϕ used in [1] was unspecified, the differences in the solutions from the present study and in [1] as discussed subsequently may be due to the difference in ϕ values.

The following details are unexplained in [1].

- The terms reflecting natural deaths, $\mu S, \mu V, \mu E, \mu I, \mu R$, included in Fig. 1 of [1], were replaced with rS, rV, rE, rI, rR in the PDEs in accordance with the original [1], which gives no explanation other than "After some calculations ..."
- The origin of the numerical values of the diffusivities d_1, d_2, d_3, d_4, d_5 used in the diffusion terms of eqs. (6.1) is unspecified.

The Gaussian IC of [1] is used in the following coding. The other ICs in [1] were not implemented to limit the volume of numerical and graphical outputs.

The five ICs in Table 6.2 are referenced as eqs. (6.3a)–(6.3e) in the subsequent discussion of the R code.

TABLE 6.2 Initial conditions for eqs. (6.1).

IC Function	Interval in x	Equation
$S(x,t=0)=0.86\exp(-(x/1.4)^2)$	$-3 \le x \le 3$	eq. (6.3a)
$V(x,t=0)=0.10\exp(-(x/1.4)^2)$	$-3 \le x \le 3$	eq. (6.3b)
$E(x,t=0)=0$	$-3 \le x \le 3$	eq. (6.3c)
$I(x,t=0)=0.04\exp(-x^2)$	$-3 \le x \le 3$	eq. (6.3d)
$R(x,t=0)=0$	$-3 \le x \le 3$	eq. (6.3e)

6.2.1 ODE Routine

The programming of eqs. (6.1) and (6.2) as a system of approximating ODEs is in `flu_1` of Listing 6.1.

```
  flu_1=function(t,u,parms){
#
# Function flu_1 computes the t derivative vector
# of the S,V,E,I,R vectors
#
# One vector to five vectors
  S=rep(0,nx);V=rep(0,nx);
  E=rep(0,nx);I=rep(0,nx);
  R=rep(0,nx);
  for(i in 1:nx){
    S[i]=u[i];
    V[i]=u[i+nx];
    E[i]=u[i+2*nx];
    I[i]=u[i+3*nx];
    R[i]=u[i+4*nx];
  }
#
# Boundary conditions
  Sx=dss004(xl,xu,nx,S);
  Vx=dss004(xl,xu,nx,V);
  Ex=dss004(xl,xu,nx,E);
  Ix=dss004(xl,xu,nx,I);
  Rx=dss004(xl,xu,nx,R);
  Sx[1]=0;Sx[nx]=0;
  Vx[1]=0;Vx[nx]=0;
  Ex[1]=0;Ex[nx]=0;
  Ix[1]=0;Ix[nx]=0;
  Rx[1]=0;Rx[nx]=0;
  nl=2;nu=2;
#
# Sxx to Rxx
  Sxx=dss044(xl,xu,nx,S,Sx,nl,nu);
  Vxx=dss044(xl,xu,nx,V,Vx,nl,nu);
  Exx=dss044(xl,xu,nx,E,Ex,nl,nu);
  Ixx=dss044(xl,xu,nx,I,Ix,nl,nu);
  Rxx=dss044(xl,xu,nx,R,Rx,nl,nu);
#
# PDEs
  b=beta;be=betae;bi=betai;bv=betav;
  a=alpha;p=phi;d=delta;t=theta;k=kappa;
  s=sigma;g=gamma;
  St=rep(0,nx);Vt=rep(0,nx);
  Et=rep(0,nx);It=rep(0,nx);
  Rt=rep(0,nx);
  for(i in 1:nx){
    ES=E[i]*S[i];
    IS=I[i]*S[i];
    EV=E[i]*V[i];
    IV=I[i]*V[i];
    IE=I[i]*E[i];
    IR=I[i]*R[i];
    St[i]=-b*be*ES-b*bi*IS+a*IS-p*S[i]-r*S[i]
          +d*R[i]+t*V[i]+r+d1*Sxx[i];
    Vt[i]=-b*be*bv*EV-b*bi*bv*IV+a*IV-r*V[i]
          -t*V[i]+p*S[i]+d2*Vxx[i];
    Et[i]=b*be*ES+b*bi*IS+b*be*bv*EV+b*bi*bv*IV
         +a*IE-(r+k+s)*E[i]+d3*Exx[i];
    It[i]=s*E[i]-(r+a+g)*I[i]+a*I[i]^2+d4*Ixx[i];
    Rt[i]=k*E[i]+g*I[i]-r*R[i]-d*R[i]+a*IR+d5*Rxx[i];
  }
#
# Five vectors to one vector
  ut=rep(0,5*nx);
  for(i in 1:nx){
    ut[i]     =St[i];
    ut[i+nx]  =Vt[i];
    ut[i+2*nx]=Et[i];
    ut[i+3*nx]=It[i];
    ut[i+4*nx]=Rt[i];
  }
#
# Increment calls to flu_1
  ncall <<- ncall+1;
#
# Return derivative vector
  return(list(c(ut)));
}
```

Listing 6.1 ODE routine `flu_1`.

We can note the following details about Listing 6.1.

- The function is defined.

  ```
    flu_1=function(t,u,parms){
  #
  # Function flu_1 computes the t derivative vector
  # of the S,V,E,I,R vectors
  ```

  Concerning the RHS (input) arguments of `flu_1`, `t` is the current value of t in eqs. (6.1). `u` is a $5 * 61 = 305$-vector of ODE dependent variables for the five PDE dependent variables of `S,V,E,I,R` of eqs. (6.1). `parms` for passing parameters to `flu_1` is unused.
- `u` is placed in five 61-vectors (`nx=61` is set in the main program to follow).

  ```
  #
  # One vector to five vectors
    S=rep(0,nx);V=rep(0,nx);
    E=rep(0,nx);I=rep(0,nx);
    R=rep(0,nx);
    for(i in 1:nx){
      S[i]=u[i];
      V[i]=u[i+nx];
      E[i]=u[i+2*nx];
      I[i]=u[i+3*nx];
      R[i]=u[i+4*nx];
    }
  ```

  The five vectors are declared (preallocated) with the `rep` utility.

- The first derivatives in x, $\partial S/\partial x$ = Sx to $\partial R/\partial x$ = Rx, are computed by the library differentiator `dss004` [2]. The input arguments to `dss004` include `xl` = -3, `xu = 3` (set in the main program).

```
#
# Boundary conditions
  Sx=dss004(xl,xu,nx,S);
  Vx=dss004(xl,xu,nx,V);
  Ex=dss004(xl,xu,nx,E);
  Ix=dss004(xl,xu,nx,I);
  Rx=dss004(xl,xu,nx,R);
  Sx[1]=0;Sx[nx]=0;
  Vx[1]=0;Vx[nx]=0;
  Ex[1]=0;Ex[nx]=0;
  Ix[1]=0;Ix[nx]=0;
  Rx[1]=0;Rx[nx]=0;
  nl=2;nu=2;
```

BCs (6.2) are then programmed with subscript `1` corresponding to $x = -3$ (the left boundary in x) and subscript `nx` corresponding to $x = 3$ (the right boundary in x). These are Neumann BCs (because the first derivatives are specified at the boundaries) which are designated with `nl=nu=2`.

- The second derivatives $\partial^2 S/\partial x^2$ = Sxx to $\partial^2 R/\partial x^2$ = Rxx are computed by `dss044`. Note the vector of first derivatives is an input to `dss044` (e.g., `Sx`) to include the BCs as well as the values of `nl,nu` to specify the Neumann BCs.

```
#
# Sxx to u5xx
  Sxx=dss044(xl,xu,nx,S,Sx,nl,nu);
  Vxx=dss044(xl,xu,nx,V,Vx,nl,nu);
  Exx=dss044(xl,xu,nx,E,Ex,nl,nu);
  Ixx=dss044(xl,xu,nx,I,Ix,nl,nu);
  Rxx=dss044(xl,xu,nx,R,Rx,nl,nu);
```

- Eqs. (6.1) are programmed. First, the parameters are redefined as single characters (e.g., `b=beta`) to simplify the programming of the PDEs. Also, vectors for the derivatives in t, for example, $\partial S/\partial t$ = St, are declared with `rep`. Then the product nonlinear terms are computed for repeated use in the programming of the PDEs, for example, $E(x,t)S(x,t)$ = ES=E[i]*S[i].

```
#
# PDEs
  b=beta;be=betae;bi=betai;bv=betav;
  a=alpha;p=phi;d=delta;t=theta;k=kappa;
  s=sigma;g=gamma;
  St=rep(0,nx);Vt=rep(0,nx);
  Et=rep(0,nx);It=rep(0,nx);
  Rt=rep(0,nx);
  for(i in 1:nx){
    ES=E[i]*S[i];
    IS=I[i]*S[i];
    EV=E[i]*V[i];
    IV=I[i]*V[i];
    IE=I[i]*E[i];
    IR=I[i]*R[i];
    St[i]=-b*be*ES-b*bi*IS+a*IS-p*S[i]-r*S[i]
          +d*R[i]+t*V[i]+r+d1*Sxx[i];
    Vt[i]=-b*be*bv*EV-b*bi*bv*IV+a*IV-r*V[i]
          -t*V[i]+p*S[i]+d2*Vxx[i];
    Et[i]=b*be*ES+b*bi*IS+b*be*bv*EV+b*bi*bv*IV
         +a*IE-(r+k+s)*E[i]+d3*Exx[i];
    It[i]=s*E[i]-(r+a+g)*I[i]+a*I[i]^2+d4*Ixx[i];
    Rt[i]=k*E[i]+g*I[i]-r*R[i]-d*R[i]+a*IR+d5*Rxx[i];
  }
```

The `for` steps along the values in x from $x = -3$ to $x = 3$. The final `}` concludes the `for`. The programming of the PDEs is essentially self-explanatory by comparison with eqs. (6.1) and demonstrates the ease of including nonlinear terms numerically.

- The five dependent derivative vectors are placed in a single derivative vector `ut` with a `for` in x for return to the ODE integrator `lsoda` (called in the main program to follow).

```
#
# Five vectors to one vector
```

```
  ut=rep(0,5*nx);
  for(i in 1:nx){
    ut[i]     =St[i];
    ut[i+nx]  =Vt[i];
    ut[i+2*nx]=Et[i];
    ut[i+3*nx]=It[i];
    ut[i+4*nx]=Rt[i];
  }
```

Note that `ut` is of length `5*nx = 5*61 = 305` that matches the length of `u` as expected. The final `}` concludes the `for`.

- The number of calls to `flu_1` is incremented and returned to the calling (main) program with `<<-`.

```
#
# Increment calls to flu_1
  ncall <<- ncall+1;
```

- The derivative vector `ut` is returned to `lsoda` as a list that is required by the R ODE integrators in the library `deSolve` (discussed next in the main program).

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

The final `}` concludes `flu_1`.

This concludes the programming of the 305 ODEs that approximate eqs. (6.1). The main program that calls `flu_1` is in Listing 6.2.

6.2.2 Main Program

The main program for eqs. (6.1)–(6.3) follows.

```
#
# Access ODE integrator
  library("deSolve");
```

```
#
# Access functions for analytical solutions
  setwd("c:/R/bme_pde/chap6");
  source("flu_1.R");
  source("dss004.R");
  source("dss044.R");
#
# Format of output
#
#   ip = 1 - graphical (plotted) solutions vs x with
#            t as a parameter
#
#   ip = 2 - graphical solutions vs t at specific x
#
  ip=1;
#
# Grid in x
  nx=61;xl=-3;xu=3;
  xg=seq(from=xl,to=xu,by=(xu-xl)/(nx-1));
#
# Grid in t
  if(ip==1){nout=11;t0=0;tf=60;}
  if(ip==2){nout=61;t0=0;tf=60;}
# if(ip==2){nout=61;t0=0;tf=2000;}
  tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
#
# Parameters
  beta=0.5140; betae=0.250;     betai=1;     betav=0.9;
  sigma=1/2;   gamma=1/5;       delta=1/365;
                                             mu=5.50e-08;
  r=1.140e-05; kappa=1.857e-04; alpha=9.30e-06;
                                             theta=1/365;
  phi=1/20;    d1=0.05;         d2=0.05;     d3=0.025;
  d4=0.001;    d5=0;
#
# Display selected parameters
  cat(sprintf(
    "\n\n    betav = %6.3f   phi = %6.3f\n",betav,phi));
#
# ICs
  u0=rep(0,5*nx);
  for(ix in 1:nx){
    u0[ix]     =0.86*exp(-(xg[ix]/1.4)^2);
    u0[ix+nx]  =0.10*exp(-(xg[ix]/1.4)^2);
    u0[ix+2*nx]=0;
    u0[ix+3*nx]=0.04*exp(-xg[ix]^2);
    u0[ix+4*nx]=0;
  }
  ncall=0;
#
# ODE integration
  out=ode(y=u0,times=tout,func=flu_1,parms=NULL);
  nrow(out)
  ncol(out)
#
# Arrays for plotting numerical solutions
  S_xplot=matrix(0,nrow=nx,ncol=nout);
  V_xplot=matrix(0,nrow=nx,ncol=nout);
  E_xplot=matrix(0,nrow=nx,ncol=nout);
  I_xplot=matrix(0,nrow=nx,ncol=nout);
  R_xplot=matrix(0,nrow=nx,ncol=nout);
  for(it in 1:nout){
    for(ix in 1:nx){
       S_xplot[ix,it]=out[it,ix+1];
       V_xplot[ix,it]=out[it,ix+1+nx];
       E_xplot[ix,it]=out[it,ix+1+2*nx];
       I_xplot[ix,it]=out[it,ix+1+3*nx];
       R_xplot[ix,it]=out[it,ix+1+4*nx];
    }
  }
#
# Display numerical solutions (for t = 0, 60)
  if(ip==1){
  for(it in 1:nout){
    if((it-1)*(it-11)==0){
    cat(sprintf("\n\n      t      x      S(x,t)
       V(x,t)"));
    cat(sprintf("\n        E(x,t)      I(x,t)
       R(x,t)"));
      for(ix in 1:nx){
        cat(sprintf("\n %6.1f%7.2f%12.5f%12.5f",
        tout[it],xg[ix],S_xplot[ix,it],V_xplot[ix,it]));
        cat(sprintf("\n%14.5f%12.5f%12.5f",
        E_xplot[ix,it],I_xplot[ix,it],R_xplot[ix,it]));
      }
    }
    }
  }
  if(ip==2){
  for(it in 1:nout){
    if((it-1)*(it-61)==0){
    cat(sprintf("\n\n      t      x      S(x,t)
       V(x,t)"));
    cat(sprintf("\n        E(x,t)      I(x,t)
       R(x,t)"));
      for(ix in 1:nx){
        cat(sprintf("\n %6.1f%7.2f%12.5f%12.5f",
        tout[it],xg[ix],S_xplot[ix,it],V_xplot[ix,it]));
        cat(sprintf("\n%14.5f%12.5f%12.5f",
        E_xplot[ix,it],I_xplot[ix,it],R_xplot[ix,it]));
      }
    }
    }
  }
#
# Calls to ODE routine
  cat(sprintf("\n\n   ncall = %5d\n\n",ncall));
#
# Plot S,V,E,I,R numerical solutions
#
# vs x with t as a parameter, t = 0,6,...,60
  if(ip==1){
    par(mfrow=c(1,1));
    matplot(x=xg,y=S_xplot,type="l",xlab="x",
            ylab="S(x,t), t=0,6,...,60",xlim=c(xl,xu),
               lty=1,main="S(x,t); t=0,6,...,60;",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=V_xplot,type="l",xlab="x",
            ylab="V(x,t), t=0,6,...,60",xlim=c(xl,xu),
               lty=1,main="V(x,t); t=0,6,...,60;",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=E_xplot,type="l",xlab="x",
            ylab="E(x,t), t=0,6,...,60",xlim=c(xl,xu),
              lty=1,main="E(x,t); t=0,6,...,60;",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=I_xplot,type="l",xlab="x",
            ylab="I(x,t), t=0,6,...,60",xlim=c(xl,xu),
              lty=1,main="I(x,t); t=0,6,...,60;",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=R_xplot,type="l",xlab="x",
            ylab="R(x,t), t=0,6,...,60",xlim=c(xl,xu),
              lty=1,main="R(x,t); t=0,6,...,60;",lwd=2);
  }
#
# vs t at x = 0, t = 0,1,...,60
  if(ip==2){
    S_tplot=rep(0,nout);V_tplot=rep(0,nout);
    E_tplot=rep(0,nout);I_tplot=rep(0,nout);
    R_tplot=rep(0,nout);
    for(it in 1:nout){
      S_tplot[it]=S_xplot[31,it];
      V_tplot[it]=V_xplot[31,it];
      E_tplot[it]=E_xplot[31,it];
      I_tplot[it]=I_xplot[31,it];
      R_tplot[it]=R_xplot[31,it];
    }
    par(mfrow=c(1,1));
    matplot(x=tout,y=S_tplot,type="l",xlab="t",
            ylab="S(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="S(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=V_tplot,type="l",xlab="t",
            ylab="V(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="V(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=E_tplot,type="l",xlab="t",
            ylab="E(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="E(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=I_tplot,type="l",xlab="t",
            ylab="I(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="I(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=R_tplot,type="l",xlab="t",
            ylab="R(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="R(x,t); x = 0",lwd=2);
}
```

Listing 6.2 Main program for eqs. (6.1)–(6.3).

We can note the following details about this main program.

- The library of R ODE integrators, `deSolve`, is accessed (which includes `lsoda` called subsequently). Also, `flu_1` of Listing 6.1 and the two spatial differentiators `dss004`, `dss044` called in `flu_1` are accessed.

```
#
# Access ODE integrator
  library("deSolve");
#
# Access functions for analytical solutions
  setwd("c:/R/bme_pde/chap6");
  source("flu_1.R");
  source("dss004.R");
  source("dss044.R");
```

  Note the use of a forward slash / in the `setwd` (set working directory) rather than the usual backslash \.

- The format of the numerical and graphical outputs is selected.

```
#
# Format of output
#
#   ip = 1 - graphical (plotted) solutions vs x with
#            t as a parameter
#
#   ip = 2 - graphical solutions vs t at specific x
#
  ip=1;
```

  For `ip=1`, the solutions $S(x,t)$ to $R(x,t)$ are displayed as a function of x with t as a parameter with the values $t = 0, 6, \ldots, 60$

(11 values including $t = 0$). For ip=2, the solutions $S(x = 0, t)$ to $R(x = 0, t)$ are displayed for $t = 0, 1, \ldots, 60$ (61 values including $t = 0$).

- A grid in x is defined with 61 points for $-3 \le x \le 3$ so that xg has the spacing (3-(-3))/(61-1) = 0.1, that is, $x = -3$, $-2.9, \ldots, 3$.

```
#
# Grid in x
  nx=61;xl=-3;xu=3;
  xg=seq(from=xl,to=xu,by=(xu-xl)/(nx-1));
```

- The output values of t are defined. For ip=1, they are $t = 0, 6, \ldots, 60$, and for ip=2, $t = 0, 1, \ldots, 60$. The smaller number of points for ip=1 is used because t is a parameter in the plots of the solutions against x. The additional points for ip=2 are used to produce smooth plots in t (at $x = 0$).

```
#
# Grid in t
  if(ip==1){nout=11;t0=0;tf=60;}
  if(ip==2){nout=61;t0=0;tf=60;}
# if(ip==2){nout=61;t0=0;tf=2000;}
  tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
```

Also, the interval $0 \le t \le 2000$ is programmed for comparison of the graphical output with [1]. This case is used by activating the comment (removing #).

- The model parameters are defined numerically (and are available to flu_1 in Listing 6.1 as a consequence of a basic property of R, i.e., variables defined in a superior routine such as the main program in Listing 6.2 are available to subordinate routines such as flu_1).

```
#
# Parameters
  beta=0.5140; betae=0.250;     betai=1;  betav=0.9;
  sigma=1/2;   gamma=1/5;       delta=1/365;
                                         mu=5.50e-08;
  r=1.140e-05; kappa=1.857e-04; alpha=9.30e-06;
                                         theta=1/365;
  phi=1/20;    d1=0.05;         d2=0.05; d3=0.025;
  d4=0.001;    d5=0;
```

Note in particular the values `betav=0.9`, phi=1/20 that are unspecified in [1] (for β_V, ϕ in eqs. (6.1)).

- β_V, ϕ are displayed to emphasize their values.

```
#
# Display selected parameters
  cat(sprintf(
    "\n\n    betav = %6.3f   phi = %6.3f\n",betav,
         phi));
```

A line break is used here to limit the length of this coded line for printing.

- An IC vector `u0` is defined with `5*nx = 5*61 = 305*` elements by a `for` over the interval $-3 \leq x \leq 3$. This length is used to notify `lsoda` of the total number of ODEs to be integrated numerically.

```
#
# ICs
  u0=rep(0,5*nx);
  for(ix in 1:nx){
    u0[ix]     =0.86*exp(-(xg[ix]/1.4)^2);
    u0[ix+nx]  =0.10*exp(-(xg[ix]/1.4)^2);
    u0[ix+2*nx]=0;
    u0[ix+3*nx]=0.04*exp(-xg[ix]^2);
    u0[ix+4*nx]=0;
  }
  ncall=0;
```

The ICs for $S(x,t), V(x,t), I(x,t)$ are Gaussian functions of x specified in [1]. The counter for the calls to `flu_1` is also initialized (and is available to `flu_1` as a global variable).

- The 305 ODEs are integrated by `lsoda` (the default of the ODE integrator `ode`).

```
#
# ODE integration
  out=ode(y=u0,times=tout,func=flu_1,parms=NULL);
  nrow(out)
  ncol(out)
```

Note the use of the IC vector, `u0`, the vector of output values of t, `tout`, and the ODE routine `flu_1` of Listing 6.1. `y,times,func` are reserved names of `ode`. The argument for passing parameters to `flu_1`, `parms`, is unused. The number of rows and columns of the output (solution) matrix `out` are displayed to confirm the expected values. Actually, the number of columns is 306 rather than 305 because `lsoda` also returns the values of t (in addition to the 305 values of the ODE dependent variables).

- The numerical ODE solutions are placed in five 2D arrays for subsequent plotting by using the `matrix` utility.

```
#
# Arrays for plotting numerical solutions
  S_xplot=matrix(0,nrow=nx,ncol=nout);
  V_xplot=matrix(0,nrow=nx,ncol=nout);
  E_xplot=matrix(0,nrow=nx,ncol=nout);
  I_xplot=matrix(0,nrow=nx,ncol=nout);
  R_xplot=matrix(0,nrow=nx,ncol=nout);
  for(it in 1:nout){
    for(ix in 1:nx){
       S_xplot[ix,it]=out[it,ix+1];
       V_xplot[ix,it]=out[it,ix+1+nx];
       E_xplot[ix,it]=out[it,ix+1+2*nx];
       I_xplot[ix,it]=out[it,ix+1+3*nx];
       R_xplot[ix,it]=out[it,ix+1+4*nx];
    }
  }
```

Note the `for` in t with index `it` and the `for` in x with index `ix`. Also, the second argument of `out` is offset by `1` to accommodate t that is also returned by `lsoda`, for example, `ix+1`. Thus, the second index of out has the range 1 to ix+1+4*nx $= 61+1+4*61 = 306$ (not 305) as noted previously.

- For ip=1, the solution is displayed as a function of x (with the for in nx) and t (with the for in it).

```
#
# Display numerical solutions (for t = 0, 60)
  if(ip==1){
  for(it in 1:nout){
    if((it-1)*(it-11)==0){
    cat(sprintf("\n\n      t      x      S(x,t)
       V(x,t)"));
    cat(sprintf("\n        E(x,t)      I(x,t)
       R(x,t)"));
      for(ix in 1:nx){
        cat(sprintf("\n %6.1f%7.2f%12.5f%12.5f",
        tout[it],xg[ix],S_xplot[ix,it],V_xplot[ix,
           it]));
        cat(sprintf("\n%14.5f%12.5f%12.5f",
        E_xplot[ix,it],I_xplot[ix,it],R_xplot[ix,
           it]));
      }
    }
    }
  }
```

To limit the output to a manageable size, the solution at only $t = 0, 60$ is displayed by using if((it-1)*(it-11)==0). Of course, any of the output values of t can be selected in this way.

- The output for ip=2 is displayed in essentially the same way as for ip=1. Note again that this output is only for $t = 0, 60$ by using if((it-1)*(it-61)==0) (because there are now 61 output points in t rather than the 11 for ip=1).

```
  if(ip==2){
  for(it in 1:nout){
    if((it-1)*(it-61)==0){
    cat(sprintf("\n\n      t      x      S(x,t)
       V(x,t)"));
    cat(sprintf("\n        E(x,t)      I(x,t)
       R(x,t)"));
      for(ix in 1:nx){
        cat(sprintf("\n %6.1f%7.2f%12.5f%12.5f",
        tout[it],xg[ix],S_xplot[ix,it],V_xplot[ix,
           it]));
        cat(sprintf("\n%14.5f%12.5f%12.5f",
        E_xplot[ix,it],I_xplot[ix,it],R_xplot[ix,
           it]));
      }
    }
    }
  }
```

- The counter for the calls to `flu_1` is displayed as a measure of the effort required to compute the numerical solution.

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

- For `ip=1`, five individual (separate) plots for $S(x,t)$ to $R(x,t)$ are produced with `par(mfrow=c(1,1))` (a 1×1 matrix of individual plots, i.e., a single plot). Note the use of `xg` as the horizontal `x` variable and the 2D arrays with the solutions to eqs. (6.1) as the vertical `y` variable.

```
#
# Plot S,V,E,I,R numerical solutions
#
# vs x with t as a parameter, t = 0,6,...,60
  if(ip==1){
    par(mfrow=c(1,1));
    matplot(x=xg,y=S_xplot,type="l",xlab="x",
            ylab="S(x,t), t=0,6,...,60",xlim=c(xl,xu),
               lty=1,main="S(x,t); t=0,6,...,60;",
                  lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=V_xplot,type="l",xlab="x",
            ylab="V(x,t), t=0,6,...,60",xlim=c(xl,xu),
               lty=1,main="V(x,t); t=0,6,...,60;",
                  lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=E_xplot,type="l",xlab="x",
            ylab="E(x,t), t=0,6,...,60",xlim=c(xl,xu),
               lty=1,main="E(x,t); t=0,6,...,60;",
                  lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=I_xplot,type="l",xlab="x",
            ylab="I(x,t), t=0,6,...,60",xlim=c(xl,xu),
               lty=1,main="I(x,t); t=0,6,...,60;",
                  lwd=2);
    par(mfrow=c(1,1));
    matplot(x=xg,y=R_xplot,type="l",xlab="x",
            ylab="R(x,t), t=0,6,...,60",xlim=c(xl,xu),
               lty=1,main="R(x,t); t=0,6,...,60;",
                  lwd=2);
  }
```

`matplot` requires that the number of rows of `x` equals the number of rows of `y`, `61` in this case. The number of columns of the `y` arrays is `11` (refer to the definition of the 2D `y` arrays discussed previously) so that t is plotted parametrically (as `11` separate curves in each plot). The other arguments of `matplot` pertain to labeling and line types for the plots (as reflected in Figs. 6.2–6.6).

- For `ip=2`, five individual plots for $S(x = 0, t)$ to $R(x = 0, t)$ are produced, again with `par(mfrow=c(1,1))`. The 1D arrays (vectors) for the plotting as a function of t are defined with the `rep` utility. The solutions for $x = 0$ are then placed in the 1D arrays (from the previous 2D arrays using subscript `31` for $x = 0$). In the calls to `matplot`, note the use of `tout` as the horizontal `x` variable and the 1D arrays with the solutions at $x = 0$ as the vertical `y` variables. The net result is the plotting of the five dependent variables from eq. (6.1) for $x = 0$ as a function of t for $t = 0, 1, \ldots, 60$. The resulting plots are in Figs. 6.7–6.11.

```
#
# vs t at x = 0, t = 0,1,...,60
  if(ip==2){
    S_tplot=rep(0,nout);V_tplot=rep(0,nout);
```

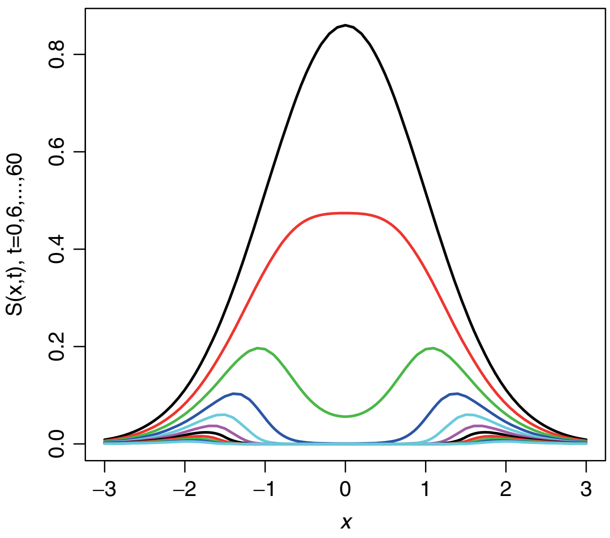

Figure 6.2 $S(x,t)$ versus x, $t = 0, 6, \ldots, 60$.

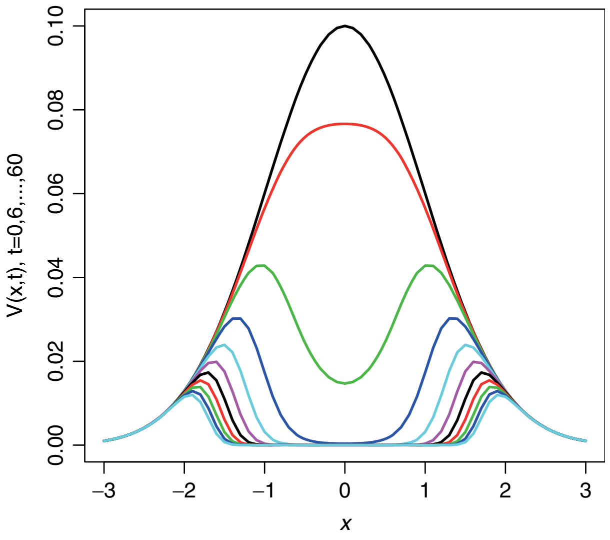

Figure 6.3 $V(x,t)$ versus x, $t = 0, 6, \ldots, 60$.

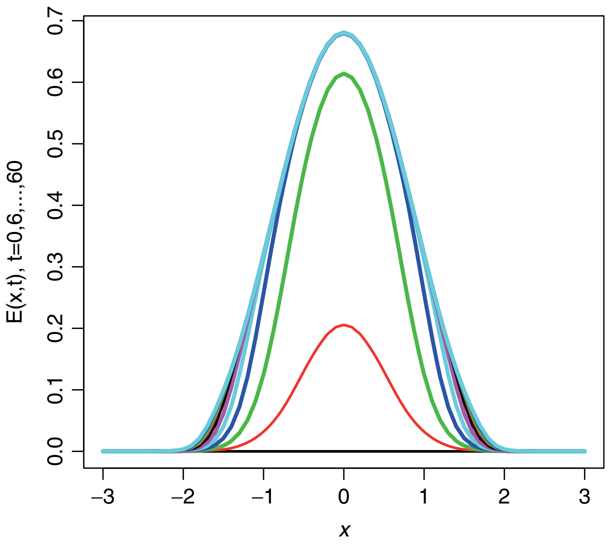

Figure 6.4 $E(x,t)$ versus x, $t = 0, 6, \ldots, 60$.

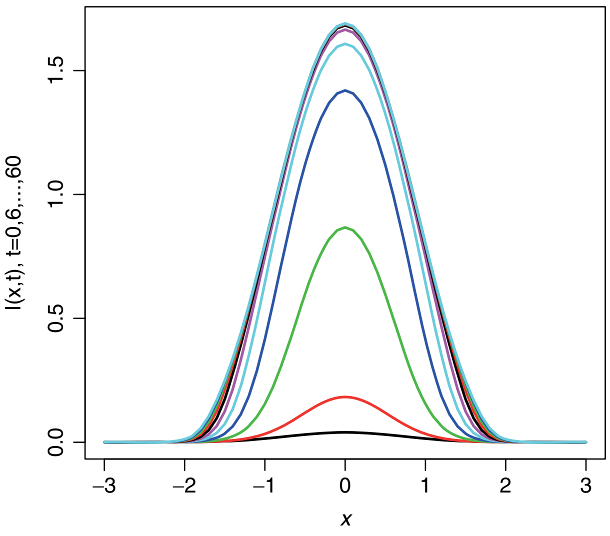

Figure 6.5 $I(x,t)$ versus x, $t = 0, 6, \ldots, 60$.

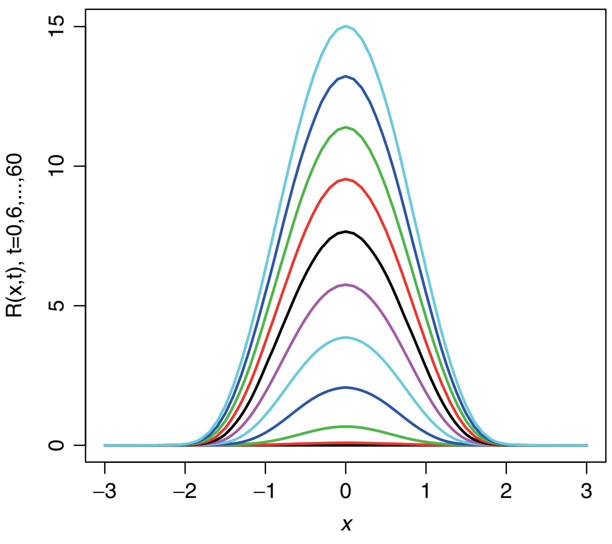

Figure 6.6 $R(x,t)$ versus x, $t = 0, 6, \ldots, 60$.

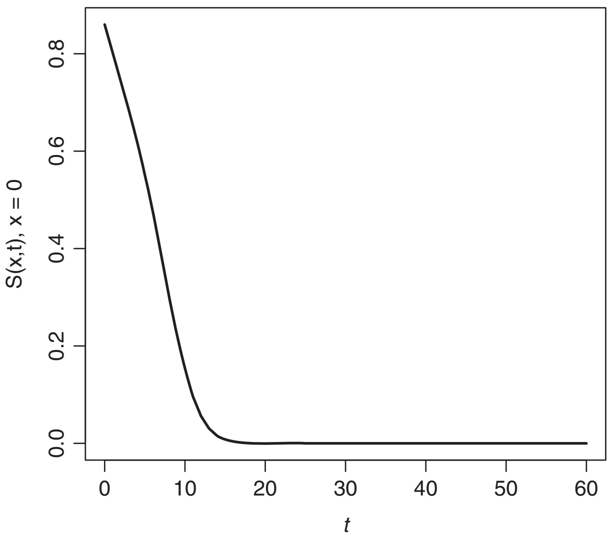

Figure 6.7 $S(x,t)$ versus t, $x = 0$.

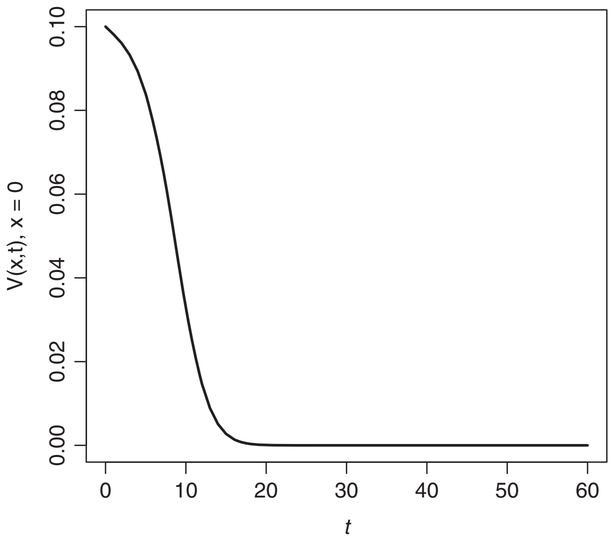

Figure 6.8 $V(x,t)$ versus t, $x = 0$.

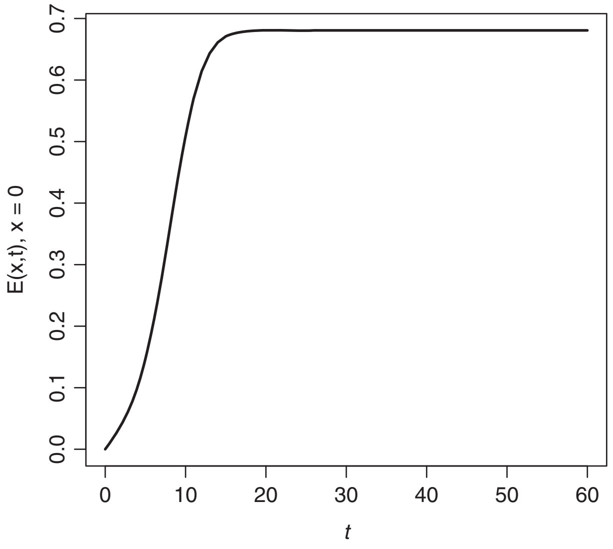

Figure 6.9 $E(x,t)$ versus t, $x = 0$.

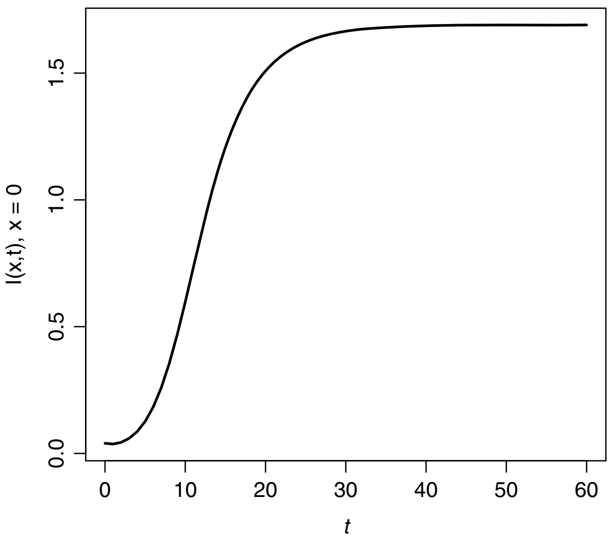

Figure 6.10 $I(x,t)$ versus t, $x = 0$.

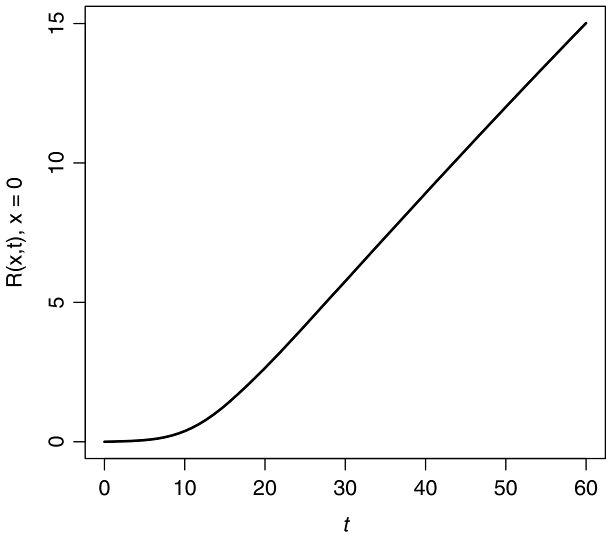

Figure 6.11 $R(x,t)$ versus t, $x = 0$.

```
    E_tplot=rep(0,nout);I_tplot=rep(0,nout);
    R_tplot=rep(0,nout);
    for(it in 1:nout){
      S_tplot[it]=S_xplot[31,it];
      V_tplot[it]=V_xplot[31,it];
      E_tplot[it]=E_xplot[31,it];
      I_tplot[it]=I_xplot[31,it];
      R_tplot[it]=R_xplot[31,it];
    }
    par(mfrow=c(1,1));
    matplot(x=tout,y=S_tplot,type="l",xlab="t",
            ylab="S(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="S(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=V_tplot,type="l",xlab="t",
            ylab="V(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="V(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=E_tplot,type="l",xlab="t",
            ylab="E(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="E(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=I_tplot,type="l",xlab="t",
            ylab="I(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="I(x,t); x = 0",lwd=2);
    par(mfrow=c(1,1));
    matplot(x=tout,y=R_tplot,type="l",xlab="t",
            ylab="R(x,t), x = 0",xlim=c(t0,tf),lty=1,
            main="R(x,t); x = 0",lwd=2);
  }
```

Listings 6.1 and 6.2 produce the numerical and graphical outputs discussed in Section 6.2.3.

6.2.3 Model Output

The abbreviated numerical output from Listing 6.1 is in Table 6.3.

We can note the following details about this numerical output.

- The dimensions of the output array `out` (from the ODE integrator `ode`) are

TABLE 6.3 Abbreviated numerical output from Listing 6.2 for `ip=1`.

```
   betav =  0.900   phi =  0.050

 [1] 11

 [1] 306

     t      x      S(x,t)      V(x,t)
       E(x,t)      I(x,t)      R(x,t)
   0.0  -3.00     0.00872     0.00101
      0.00000     0.00000     0.00000
   0.0  -2.90     0.01178     0.00137
      0.00000     0.00001     0.00000
   0.0  -2.80     0.01575     0.00183
      0.00000     0.00002     0.00000
   0.0  -2.70     0.02085     0.00242
      0.00000     0.00003     0.00000
   0.0  -2.60     0.02733     0.00318
      0.00000     0.00005     0.00000
        .                       .
        .                       .
        .                       .
 Output from x = -2.50 to -0.30 removed
        .                       .
        .                       .
        .                       .
   0.0  -0.20     0.84263     0.09798
      0.00000     0.03843     0.00000
   0.0  -0.10     0.85562     0.09949
      0.00000     0.03960     0.00000
   0.0   0.00     0.86000     0.10000
      0.00000     0.04000     0.00000
   0.0   0.10     0.85562     0.09949
      0.00000     0.03960     0.00000
   0.0   0.20     0.84263     0.09798
      0.00000     0.03843     0.00000
        .                       .
```

(*continued*)

TABLE 6.3 (***Continued***)

```
       .                       .
       .                       .
  Output from x = 0.30 to 2.50 removed
       .                       .
       .                       .
       .                       .
    0.0   2.60     0.02733     0.00318
       0.00000     0.00005     0.00000
    0.0   2.70     0.02085     0.00242
       0.00000     0.00003     0.00000
    0.0   2.80     0.01575     0.00183
       0.00000     0.00002     0.00000
    0.0   2.90     0.01178     0.00137
       0.00000     0.00001     0.00000
    0.0   3.00     0.00872     0.00101
       0.00000     0.00000     0.00000

      t      x      S(x,t)      V(x,t)
        E(x,t)      I(x,t)      R(x,t)
   60.0  -3.00     0.00043     0.00101
       0.00000     0.00000     0.00002
   60.0  -2.90     0.00059     0.00137
       0.00000     0.00000     0.00002
   60.0  -2.80     0.00078     0.00183
       0.00000     0.00000     0.00005
   60.0  -2.70     0.00104     0.00242
       0.00000     0.00001     0.00011
   60.0  -2.60     0.00136     0.00317
       0.00001     0.00003     0.00024
       .                       .
       .                       .
       .                       .
  Output from x = -2.50 to -0.30 removed
       .                       .
       .                       .
       .                       .
   60.0  -0.20    -0.00000    -0.00000
       0.66186     1.64362    14.55656
```

TABLE 6.3 (*Continued*)

```
  60.0  -0.10    -0.00000    -0.00000
         0.67593   1.67822   14.90177
  60.0   0.00    -0.00000    -0.00000
         0.68067   1.68988   15.01821
  60.0   0.10    -0.00000    -0.00000
         0.67593   1.67822   14.90177
  60.0   0.20    -0.00000    -0.00000
         0.66186   1.64362   14.55656
           .                     .
           .                     .
           .                     .
 Output from x = 0.30 to 2.50 removed
           .                     .
           .                     .
           .                     .
  60.0   2.60     0.00136     0.00317
         0.00001   0.00003     0.00024
  60.0   2.70     0.00104     0.00242
         0.00000   0.00001     0.00011
  60.0   2.80     0.00078     0.00183
         0.00000   0.00000     0.00005
  60.0   2.90     0.00059     0.00137
         0.00000   0.00000     0.00002
  60.0   3.00     0.00043     0.00101
         0.00000   0.00000     0.00002

  ncall =   188
```

```
[1] 11

[1] 306
```

as expected (including `5*61 + 1` $= 306$ as discussed previously).

- The ICs of eqs. (6.3) (Gaussian functions in Table 6.2) can be checked approximately. For example, at $x = 0, t = 0$, the ICs are from Table 6.2 $S(x = 0, t = 0) = 0.86e^{-0^2} = 0.86$, $V(x = 0, t = 0) = 0.10$, $E(x = 0, t = 0) = 0$, $I(x = 0, t = 0) = 0.04$, $R(x = 0, t = 0) = 0$, which are confirmed in Table 6.3 as

```
      t       x       S(x,t)       V(x,t)
         E(x,t)       I(x,t)       R(x,t)

    0.0    0.00      0.86000      0.10000
        0.00000      0.04000      0.00000
```

- The ICs are symmetrical with respect to $x = 0$ as they should be from the Gaussian functions of Table 6.2 (which are even functions in x). For example,

```
      t       x       S(x,t)       V(x,t)
         E(x,t)       I(x,t)       R(x,t)

    0.0   -0.10      0.85562      0.09949
        0.00000      0.03960      0.00000
    0.0    0.00      0.86000      0.10000
        0.00000      0.04000      0.00000
    0.0    0.10      0.85562      0.09949
        0.00000      0.03960      0.00000
```

- The peak values of $S(x = 0, t = 0) = 0.86$, $V(x = 0, t = 0) = 0.10$ decrease and $E(x = 0, t)$, $I(x = 0, t = 0) = 0.04$, $R(x = 0, t)$ increase as the influenza progresses and then subsides. At $t = 60$, the values are

```
      t       x       S(x,t)       V(x,t)
         E(x,t)       I(x,t)       R(x,t)

   60.0    0.00     -0.00000     -0.00000
        0.68067      1.68988     15.01821
```

so that $R(x, t)$ (recovered) experiences a strong increase with a significant $I(x, t)$ (infected) remaining at $t = 60$.

- The symmetry of the solutions remains as expected (there is no preferred direction in x). For example, at $t = 60$,

```
      t       x       S(x,t)       V(x,t)
         E(x,t)       I(x,t)       R(x,t)

   60.0   -0.10     -0.00000     -0.00000
```

```
         0.67593     1.67822    14.90177
  60.0    0.00      -0.00000    -0.00000
         0.68067     1.68988    15.01821
  60.0    0.10      -0.00000    -0.00000
         0.67593     1.67822    14.90177
```

which can be considered to be a check of the numerical integration of `lsoda` and the coding of the FDs in the spatial differentiators `dss004,dss044`.

The symmetry of the solutions around x is also evident in Figs. 6.2–6.6. This also suggests that the interval in x could be reduced to $0 \leq x \leq 3$ with zero derivative BCs imposed at $x = 0$ (to reflect the symmetry) rather than at $x = -3$ (to reflect zero diffusion). In other words, there is no diffusion across $x = 0$ as well as at $x = -3$.

For `ip=2`, the solutions at $x = 0$ are in Figs. 6.7–6.11. Figures 6.7–6.10 indicate that $S(x = 0, t)$ to $I(x = 0, t)$ reach steady state well before $t = 60$ while $R(x, t)$ continues to increase beyond $t = 60$. This is further confirmed by the solutions for $0 \leq t \leq 2000$ (using again `ip=2` in Listing 6.2). In Fig. 6.12, $S(x = 0, t)$ has reached steady state for $t < 60$, whereas $R(x, t)$ has reach steady state for $t < 2000$ so that the influenza has passed by $t = 2000$. In all cases, the solutions are monotonic (the LHS derivatives of eqs. (6.1) do not change sign) rather than oscillatory as reported in [1].

Figures 6.12 and 6.13 also indicate that the 305 ODE system is stiff in the sense that some of the ODEs have a fast solution, for example, $S(x = 0, t)$ in Fig. 6.12, while others have a slow solution, for example, $R(x = 0, t)$ in Fig. 6.13. It is this wide difference in the timescales that causes excessive computation when using an explicit ODE integrator. For example, $S(x = 0, t)$ requires small integration steps to maintain stability (because of the stability limit or constraint on the integration step of explicit integrators such as the explicit Euler method), while $R(x = 0, t)$ requires many steps to compute a complete solution in t.

In other words, the requirement for many small steps leads to a lengthy computation. In the case of the 305 ODE model, this

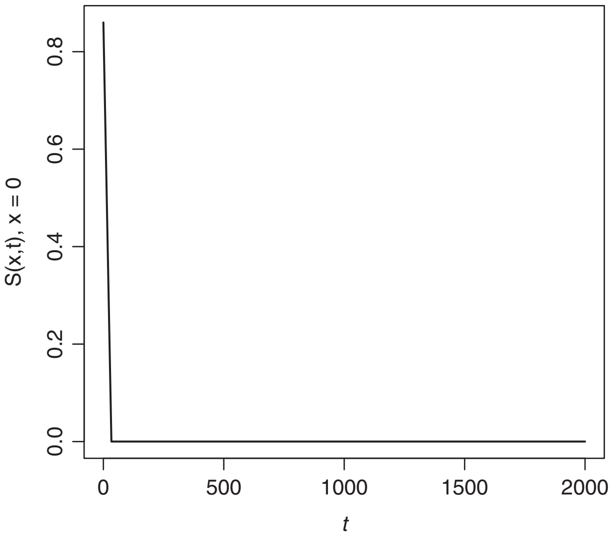

Figure 6.12 $S(x,t)$ versus t, $x = 0$.

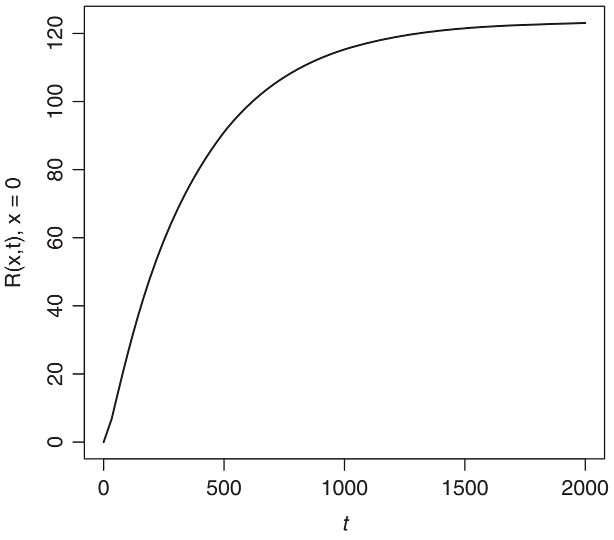

Figure 6.13 $R(x,t)$ versus t, $x = 0$.

limitation of explicit methods was circumvented by using `lsoda` in `ode` (v. Listing 6.2) that automatically switches between stiff and nonstiff methods based on an eigenvalue analysis (the "a" in `lsoda` designates the automatic switching). The effectiveness of the stiff option in `lsoda` is demonstrated by the modest number of calls to the ODE routine `fhn_1` of Listing 6.1 such as `ncall` = 188 in Table 6.3.

6.3 Summary

The preceding discussion of the SVEIR model of eqs. (6.1)–(6.3) illustrates the straightforward numerical integration of a system of five 1D nonlinear PDEs. Also, this discussion illustrates the challenge of understanding the equations and reproducing the reported numerical solutions. This uncertainty in interpreting ODE/PDE models taken from the literature is not uncommon and is generally due to incomplete information such as

- Incomplete equation, IC, BC, and parameter sets.
- Reported equations with little or no indication of their origin, for example, a derivation.
- Little or no discussion of the algorithms used to compute reported numerical solutions.
- Computer codes used to compute reported numerical solutions are unavailable.

For these reasons, reproducing reported solutions is often nearly impossible. In summary, this situation of uncertainty is not unusual when accessing ODE/PDE models reported in the open literature so some additional effort and care is generally required in documenting and using the models.

References

[1] Samsuzzoha, Md., M. Singh, and D. Lucy (2012), A numerical study on an influenza epidemic model with vaccination and diffusion, *Appl. Math. Comput.*, **219**, 122–141.

[2] Schiesser, W.E., and G.W. Griffiths (2009), *A Compendium of Partial Differential Equation Models*, Cambridge University Press, Cambridge, UK.