19

Kuramoto–Sivashinsky Equation

The Kuramoto–Sivashinsky (KS) equation has applications in fluid mechanics, including as a basic description for turbulence [1, 3, 4]. It is also a useful test problem for numerical methods with the following features:

- Higher order (up to $\dfrac{\partial^4 u}{\partial x^4}$ in x)
- Nonlinear (with a convective term $u\dfrac{\partial u}{\partial x}$)
- Dirichlet and Neumann BCs
- Traveling wave solutions
- Available exact (analytical) solutions (to test numerical solutions).

These features are demonstrated in the following discussion.

19.1 PDE Model

The KS is [2]

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - \alpha\frac{\partial^2 u}{\partial x^2} - \beta\frac{\partial^3 u}{\partial x^3} - \gamma\frac{\partial^4 u}{\partial x^4} \tag{19.1}$$

Equation (19.1) is first order in t and fourth order in x. It therefore requires one initial condition (IC) and four boundary conditions (BCs).

$$u(x, t = 0) = f(x) \tag{19.2}$$

$$u(x = x_l, t) = f_1(t); \quad u(x = x_u, t) = f_2(t) \tag{19.3a,b}$$

$$\frac{\partial u(x = x_l, t)}{\partial x} = f_3(t); \quad \frac{\partial u(x = x_u, t)}{\partial x} = f_4(t) \tag{19.3c,d}$$

where x_l, x_u are the boundary points in x, and $f_1(t)$,$f_2(t)$,$f_3(t)$,$f_4(t)$ are functions to be specified.

Two analytical solutions are available to test the spline collocation method of lines (SCMOL) solution.

Spline Collocation Methods for Partial Differential Equations: With Applications in R, First Edition.
William E. Schiesser.

Companion website: www.wiley.com/go/Spline_Collocation

`ncase = 1:`

$$u_a(x,t) = c_0 + c_1 f + c_2 f^2 + c_3 f^3 \tag{19.4a}$$

with

$$f = \frac{k}{1 + e^{-kx-\lambda t}} \tag{19.4b}$$

For Neumann BCs (19.3c,d), the x derivative of the solution is required.

$$\frac{\partial u_a(x,t)}{\partial x} = c_1 \frac{\partial f}{\partial x} + 2c_2 f \frac{\partial f}{\partial x} + 3c_3 f^2 \frac{\partial f}{\partial x} \tag{19.4c}$$

with

$$\frac{\partial f}{\partial x} = \frac{k^2 e^{-kx-\lambda t}}{(1 + e^{-kx-\lambda t})^2} \tag{19.4d}$$

$u_a(x,t)$ of Eqs. (19.4) is a traveling wave solution, that is, a function of the Lagrangian variable $-kx - \lambda t$ (a linear combination of x and t).

`ncase = 2:`

The analytical solution is available ([2], p593) for the parameter values $\beta = 0$, $\alpha = \gamma = 1, k = \pm\sqrt{\frac{11}{19}}$.

$$u_a(x,t) = \frac{15}{19}k(11H^3 - 9H + 2); \quad H = \tanh\left(\frac{1}{2}kx - \frac{15}{19}k^2 t\right) \tag{19.5a}$$

For Neumann BCs (19.3c,d), the x derivative of the solution is required.

$$\frac{\partial u}{\partial x} = \frac{15}{19}k\left(33H^2\frac{\partial H}{\partial x} - 9\frac{\partial H}{\partial x}\right) \tag{19.5b}$$

with

$$\frac{\partial H}{\partial x} = \frac{1}{2}k\left(1 - \tanh^2\left(\frac{1}{2}kx - \frac{15}{19}k^2 t\right)\right) \tag{19.5c}$$

$u_a(x,t)$ of Eqs. (19.5) is a traveling wave solution, that is, a function of the Lagrangian variable $\frac{1}{2}kx - \frac{15}{19}k^2 t$ (a linear combination of x and t).

The two analytical solutions, `ncase=1,2`, are programmed in the R routines that follow, starting with the main program.

19.2 Main Program

The following main program is for Eqs. (19.1)–(19.5).

```
#
# Kuramoto-Sivashinsky
#
```

```
# Delete previous workspaces
  rm(list=ls(all=TRUE))
#
# Access ODE integrator, routines
  library(deSolve);
  setwd("f:/collocation/chap19");
  source("pde_1a.R");source("ua_1.R");
  source("uax_1.R");
#
# Select case
  ncase=1;
  if(ncase==1){
    alpha=1;gamma=1;c0=1;
    beta=4*(alpha*gamma)^0.5;
    k=(alpha/gamma)^0.5;
    lambda=-c0*k-(3/2)*beta*k^3;
    c1=(15/76)*(16*alpha-beta^2/gamma)+
        15*beta*k+60*gamma*k^2;
    c2=-(15*beta+180*gamma*k);
    c3=60*gamma;
#
#   Spatial grid
    n=51;xl=0;xu=1;
    x=seq(from=xl,to=xu,by=(xu-xl)/(n-1));
#
#   Independent variable for ODE integration
    t0=0;tf=0.15;nout=6;
    tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
    ncall=0;
  }
#
  if(ncase==2){
    alpha=1;beta=0;gamma=1;c0=0;
    k=(11/19)^0.5;
#
#   Spatial grid
    n=101;xl=-10;xu=20;
    x=seq(from=xl,to=xu,by=(xu-xl)/(n-1));
#
#   Independent variable for ODE integration
    t0=0;tf=10;nout=6;
    tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
    ncall=0;
  }
#
# Initial condition (IC)
  u0=rep(0,n);
  for(i in 1:n){
    u0[i]=ua_1(x[i],0);
  }
#
# ODE integration
  out=lsodes(y=u0,times=tout,func=pde_1a,
      sparsetype="sparseint",rtol=1e-6,atol=1e-6,
      maxord=5);
  nrow(out)
  ncol(out)
#
# Store numerical, analytical solutions for plotting
   u=matrix(0,nrow=n,ncol=nout);
  ua=matrix(0,nrow=n,ncol=nout);
   t=rep(0,nout);
  for(it in 1:nout){
    t[it]=out[it,1];
  for(i in 1:n){
    u[i,it]=out[it,i+1];
   ua[i,it]=ua_1(x[i],t[it]);
  }
   u[1,it]=ua_1(x[1],t[it]);
   u[n,it]=ua_1(x[n],t[it]);
  }
#
# Numerical solution
  for(it in 1:nout){
  cat(sprintf("\n      t        x   u(x,t)
              ua(x,t)     diff"));
  if(ncase==1){iv=seq(from=1,to=n,by=5);}
  if(ncase==2){iv=seq(from=1,to=n,by=10);}
  for(i in iv){
    diff=u[i,it]-ua[i,it];
    cat(sprintf("\n %6.3f%8.3f%9.4f%9.4f%9.4f",
                t[it],x[i],u[i,it],ua[i,it],diff));
  }
    cat(sprintf("\n"));
  }
  cat(sprintf("\n ncall = %4d\n",ncall));
```

```
#
# Plot numerical solution
  matplot(x,u,type="l",lwd=2,col="black",
          lty=1,xlab="x",ylab="u(x,t)",
          main="Kuramoto-Sivashinsky");
   matpoints(x,ua,pch="o",col="black");
#
# Plot 3D numerical solution
    persp(x,t,u,theta=55,phi=45,xlim=c(xl,xu),
          ylim=c(t0,t[nout]),xlab="x",ylab="t",
          zlab="u(x,t)");
```

Listing 19.1 Main program for Eqs. (19.1)–(19.5).

We can note the following details about Listing 19.1:

- Previous workspaces are removed. Then the ODE integrator library `deSolve` is accessed. Note that the `setwd` (set working directory) uses / rather than the usual \. The `setwd` requires editing for the local computer (to specify the directory with the R routines).

  ```
  #
  # Kuramoto-Sivashinsky
  #
  # Delete previous workspaces
    rm(list=ls(all=TRUE))
  #
  # Access ODE integrator, routines
    library(deSolve);
    setwd("f:/collocation/chap19");
    source("pde_1a.R");source("ua_1.R");
    source("uax_1.R");
  ```

  `pde_1a` is the routine for the MOL/ODEs (discussed subsequently). `ua_1, uax_1.R` are functions for the analytical solutions, Eqs. (19.4) and (19.5).
- Two cases are programmed. For `ncase=1`, Eqs. (19.4) are programmed.

  ```
  #
  # Select case
    ncase=1;
    if(ncase==1){
      alpha=1;gamma=1;c0=1;
      beta=4*(alpha*gamma)^0.5;
      k=(alpha/gamma)^0.5;
  ```

```
    lambda=-c0*k-(3/2)*beta*k^3;
    c1=(15/76)*(16*alpha-beta^2/gamma)+
         15*beta*k+60*gamma*k^2;
    c2=-(15*beta+180*gamma*k);
    c3=60*gamma;
#
#   Spatial grid
    n=51;xl=0;xu=1;
    x=seq(from=xl,to=xu,by=(xu-xl)/(n-1));
#
#   Independent variable for ODE integration
    t0=0;tf=0.15;nout=6;
    tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
    ncall=0;
  }
```

We can note the following details about this programming:

- The parameters in Eq. (19.1) are defined numerically ($\alpha, \beta, \gamma, \lambda, k$). The constants c_0, c_1, c_2, c_3 in Eqs. (19.4) are also computed. These parameters are available to subordinate routines `pde_1a, ua_1, uax_1` (discussed subsequently) without any special designation (a feature of R).
- A uniform grid in x of `51` points is defined with the `seq` utility for the interval $x_l = 0 \leq x \leq x_u = 1$. Therefore, the vector `x` has the values $x = 0, 0.02, \ldots, 1$.

```
#
#   Spatial grid
    n=51;xl=0;xu=1;
    x=seq(from=xl,to=xu,by=(xu-xl)/(n-1));
```

- A uniform grid in t of `6` output points is defined with the `seq` utility for the interval $t_0 = 0 \leq t \leq t_f = 0.15$. Therefore, the vector `tout` has the values $t = 0, 0.03, \ldots, 0.15$.

```
#
#   Independent variable for ODE integration
    t0=0;tf=0.15;nout=6;
    tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
    ncall=0;
  }
```

At this point, the intervals of x and t in Eqs. (19.1)–(19.4) are defined. Also, the counter for the calls to the MOL/ODE routine `pde_1a` is initialized (and passed to `pde_1a` without a special designation).

- The programming is similar for `ncase=2` (Eqs. (19.5) are programmed). A uniform grid in x of `101` points is defined with the `seq` utility for the interval $x_l = -10 \le x \le x_u = 20$ so that the vector `x` has the values $x = -10, -9.70, \dots, 20$.
- A uniform grid in t of `6` output points is defined with the `seq` utility for the interval $t_0 = 0 \le t \le t_f = 10$ so the vector `tout` has the values $t = 0, 2, \dots, 10$.

```
#
  if(ncase==2){
    alpha=1;beta=0;gamma=1;c0=0;
    k=(11/19)^0.5;
#
#   Spatial grid
    n=101;xl=-10;xu=20;
    x=seq(from=xl,to=xu,by=(xu-xl)/(n-1));
#
#   Independent variable for ODE integration
    t0=0;tf=10;nout=6;
    tout=seq(from=t0,to=tf,by=(tf-t0)/(nout-1));
    ncall=0;
  }
```

The number of MOL/ODEs is increased from 51 (`ncase=1`) to 101 (`ncase=2`) to improve the spatial resolution in x (this will be clear when the graphical output in Figures 19.1a and 19.2a is discussed). Also, the time intervals are different for `ncase=1,2` because of the difference in the solutions for `ncase=1,2`. At this point, the intervals of x and t in Eqs. (19.1)–(19.3) and (19.5) are defined as $-10 \le x \le 20, 0 \le t \le 10$.

- The IC, Eq. (19.2), is taken as the analytical solution of Eqs. (19.4) or (19.5) (depending on `ncase`) with $t = 0$.

```
#
# Initial condition (IC)
  u0=rep(0,n);
  for(i in 1:n){
    u0[i]=ua_1(x[i],0);
  }
```

- The system of MOL/ODEs is integrated by the library integrator `lsodes` (available in `deSolve`) with the sparse matrix option specified. As expected, the inputs to `lsodes` are the ODE function, `pde_1a`, the IC vector `u0`, and the vector of output values of t, `tout`. The length of `u0` (51 for `ncase=1`, 101 for `ncase=2`) informs `lsodes` how many ODEs are to be integrated. `func,y,times` are reserved names.

```
#
# ODE integration
  out=lsodes(y=u0,times=tout,func=pde_1a,
      sparsetype="sparseint",rtol=1e-6,atol=1e-6,
      maxord=5);
  nrow(out)
  ncol(out)
```

The numerical solution to the ODEs is returned in matrix out. In this case, out has the dimensions $nout \times (n+1) = 6 \times 52$ for ncase=1 or $nout \times (n+1) = 6 \times 102$ for ncase=2. The offset $n+1$ is required since the first element of each column has the output t (also in tout), and the $2, \dots, n+1 = 2, \dots, 52$ or 102 column elements have the ODE solutions. This indexing of out is used next.

- The numerical $u(x,t)$ is taken from the solution matrix out returned by lsodes. The intervals in t and x are covered by two nested fors (with indices it and i for t and x, respectively). The offset i+1 accounts for t placed in the first column position of out (t[it]=out[it,1]).

```
#
# Store numerical, analytical solutions for plotting
   u=matrix(0,nrow=n,ncol=nout);
  ua=matrix(0,nrow=n,ncol=nout);
   t=rep(0,nout);
  for(it in 1:nout){
    t[it]=out[it,1];
  for(i in 1:n){
    u[i,it]=out[it,i+1];
   ua[i,it]=ua_1(x[i],t[it]);
  }
   u[1,it]=ua_1(x[1],t[it]);
   u[n,it]=ua_1(x[n],t[it]);
  }
```

The analytical solution from ua_1 is placed in array ua for comparison with the numerical solution in u. The numerical solution at $x = x_l, x = x_u$ is placed in u with ua_1 since these boundary values are not returned from lsodes in out (only dependent variables from the integration of ODEs in pde_1a are returned, but the boundary values are set with ua_1 in pde_1a).

- The numerical and analytical solutions, and their difference, are displayed.

```
#
# Numerical solution
  for(it in 1:nout){
  cat(sprintf("\n      t       x   u(x,t)
              ua(x,t)     diff"));
  if(ncase==1){iv=seq(from=1,to=n,by=5);}
  if(ncase==2){iv=seq(from=1,to=n,by=10);}
  for(i in iv){
    diff=u[i,it]-ua[i,it];
    cat(sprintf("\n %6.3f%8.3f%9.4f%9.4f%9.4f",
                t[it],x[i],u[i,it],ua[i,it],diff));
  }
    cat(sprintf("\n"));
  }
  cat(sprintf("\n ncall = %4d\n",ncall));
```

Every fifth or tenth value is displayed for ncase=1,2, respectively. Finally, the number of calls to the ODE/MOL routine pde_1a (considered next) is displayed as a measure of the computational effort required for computing the numerical solution.

- The numerical and analytical solutions are plotted in 2D with matplot, matpoints, and in 3D with persp.

```
  #
  # Plot numerical solution
    matplot(x,u,type="l",lwd=2,col="black",
            lty=1,xlab="x",ylab="u(x,t)",
            main="Kuramoto-Sivashinsky");
      matpoints(x,ua,pch="o",col="black");
  #
  # Plot 3D numerical solution
      persp(x,t,u,theta=55,phi=45,xlim=c(xl,xu),
            ylim=c(t0,t[nout]),xlab="x",ylab="t",
            zlab="u(x,t)");
```

This concludes the discussion of the main program in Listing 19.1. The ODE/MOL routine pde_1a called by lsodes in the main program of Listing 19.1 is considered next.

19.3 ODE Routine

pde_1a for Eq. (19.1) follows.

```
  pde_1a=function(t,u,parms){
#
# Function pde_1a computes the t derivative vector
# for u(x,t)
#
# Dirichlet BCs at x = xl,xu
  u[1]=ua_1(x[1],t);
  u[n]=ua_1(x[n],t);
#
# ux
  tablex=splinefun(x,u);
  ux=tablex(x,deriv=1);
#
# Neumann BCs at x = xl,xu
  ux[1]=uax_1(x[1],t);
  ux[n]=uax_1(x[n],t);
#
# uxx
  tablexx=splinefun(x,ux);
  uxx=tablexx(x,deriv=1);
#
# uxxx
  tablexxx=splinefun(x,uxx);
  uxxx=tablexxx(x,deriv=1);
#
# uxxxx
  tablexxxx=splinefun(x,uxxx);
  uxxxx=tablexxxx(x,deriv=1);
#
# PDE
  ut=rep(0,n);
  for(i in 2:(n-1)){
    ut[i]=-u[i]*ux[i]-alpha*uxx[i]-
           beta*uxxx[i]-gamma*uxxxx[i];
  }
  ut[1]=0;ut[n]=0;
#
# Increment calls to pde_1a
  ncall <<- ncall+1;
#
# Return derivative vector
  return(list(c(ut)))
}
```

Listing 19.2 ODE/MOL routine `pde_1a` for Eq. (19.1).

We can note the following details about `pde_1a`:

- The function is defined.

```
  pde_1a=function(t,u,parms){
#
# Function pde_1a computes the t derivative vector
# for u(x,t)
```

  `t` is the current value of t in Eq. (19.1). `u` is the 51 (`ncase=1`) or 101 (`ncase=2`) vector of ODE dependent variables. `parm` is an argument to pass parameters to `pde_1a` (unused, but required in the argument list). The arguments must be listed in the order stated to properly interface with `lsodes` called in the main program.
- BCs (19.3a,b) are implemented with `ua_1` (discussed subsequently).

```
#
# Dirichlet BCs at x = xl,xu
  u[1]=ua_1(x[1],t);
  u[n]=ua_1(x[n],t);
```

  The subscripts `1,n` correspond to $x = x_l, x_u$ at the boundaries.
- $\frac{\partial u}{\partial x}$ in Eq. (19.1) is computed with `splinefun`.

```
#
# ux
  tablex=splinefun(x,u);
  ux=tablex(x,deriv=1);
```

- BCs (19.3c,d) are implemented with `uax_1` (discussed subsequently).

```
#
# Neumann BCs at x = xl,xu
  ux[1]=uax_1(x[1],t);
  ux[n]=uax_1(x[n],t);
```

  The subscripts `1,n` correspond to $x = x_l, x_u$ at the boundaries.
- $\frac{\partial^2 u}{\partial x^2}, \frac{\partial^3 u}{\partial x^3}, \frac{\partial^4 u}{\partial x^4}$ in Eq. (19.1) are computed with `splinefun` by successive (stagewise) differentiation.

```
#
# uxx
  tablexx=splinefun(x,ux);
  uxx=tablexx(x,deriv=1);
#
# uxxx
```

```
  tablexxx=splinefun(x,uxx);
  uxxx=tablexxx(x,deriv=1);
#
# uxxxx
  tablexxxx=splinefun(x,uxxx);
  uxxxx=tablexxxx(x,deriv=1);
```

- Equation (19.1) is programmed at the interior points (using `2:(n-1)`).

```
#
# PDE
  ut=rep(0,n);
  for(i in 2:(n-1)){
    ut[i]=-u[i]*ux[i]-alpha*uxx[i]-
           beta*uxxx[i]-gamma*uxxxx[i];
  }
  ut[1]=0;ut[n]=0;
```

  The derivatives in t are set to zero at the boundaries to avoid having `lsodes` move them away from the values prescribed by BCs (19.3a,b).
- The counter for the calls to `pde_1a` is incremented and its value is returned to the calling program (of Listing 19.1) with the `<<-` operator.

```
#
# Increment calls to pde_1a
  ncall <<- ncall+1;
```

- The derivative vector `ut` is returned to `lsodes`, which requires a list. `c` is the R vector utility. The combination of `return, list, c` gives `lsodes` the required derivative vector for the next step along the solution.

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

  The final } concludes `pde_1a`.

This completes the programming of Eqs. (19.1)–(19.3). The subordinate routines for Eqs. (19.4) and (19.5) follow.

19.4 Subordinate Routines

The following routines are straightforward implementations of the corresponding equations noted in the titles (captions) of the listings (with switching based on ncase). The various parameters are defined in the main program of Listing 19.1. The arguments (x,t) are the independent variables (x, t) in Eq. (19.1). The analytical solutions are returned numerically to the calling routines with return, c (a list is not required as in Listing 19.2).

```
  ua_1=function(x,t){
#
# Function ua_1 computes two analytical solutions
# of the Kuramoto-Sivashinsky equation
#
# Analytical solution
  if(ncase==1){
    F=k/(1+exp(-k*x-lambda*t));
    ua=c0+c1*F+c2*F^2+c3*F^3;
  }
  if(ncase==2){
    H=tanh(0.5*k*x-(15/19)*k^2*t);
    ua=(15/19)*k*(11*H^3-9*H+2);
  }
#
# Return analytical solution
  return(c(ua))
}
```

Listing 19.3a ua_1 for Eqs. (19.4a,b) and (19.5a).

```
  uax_1=function(x,t){
#
# Function uax_1 computes the derivative of two
# analytical solutions of the Kuramoto-Sivashinsky
# equation
#
# Derivative of analytical solution
  if(ncase==1){
    F=k/(1+exp(-k*x-lambda*t));
    Fx=(k^2)*(1+exp(-k*x-lambda*t))^(-2)*
        exp(-k*x-lambda*t);
    uax=c1*Fx+2*c2*F*Fx+3*c3*F^2*Fx;
  }
  if(ncase==2){
    arg=0.5*k*x-(15/19)*k^2*t;
    H=tanh(arg);
    Hx=(1-tanh(arg)^2)*0.5*k;
    uax=(15/19)*k*(33*H^2*Hx-9*Hx);
  }
#
# Return analytical solution derivative
  return(c(uax))
}
```

Listing 19.3b `uax_1` for Eqs. (19.4c,d) and (19.5b,c).

This completes the programming of Eqs. (19.1)–(19.5). The numerical and graphical output from the R routines is considered next.

19.5 Model Output

Abbreviated numerical output for `ncase=1` is in Table 19.1.

Table 19.1 Abbreviated numerical output for `ncase=1`.

```
 [1] 6

 [1] 52

       t        x   u(x,t)  ua(x,t)     diff
   0.000    0.000   8.5000   8.5000   0.0000
   0.000    0.100   6.5339   6.5339   0.0000
   0.000    0.200   4.3974   4.3974   0.0000
   0.000    0.300   2.1103   2.1103   0.0000
   0.000    0.400  -0.3048  -0.3048   0.0000
   0.000    0.500  -2.8237  -2.8237   0.0000
   0.000    0.600  -5.4212  -5.4212   0.0000
   0.000    0.700  -8.0717  -8.0717   0.0000
   0.000    0.800 -10.7503 -10.7503   0.0000
   0.000    0.900 -13.4329 -13.4329   0.0000
   0.000    1.000 -16.0975 -16.0975   0.0000
              .                 .
              .                 .
              .                 .
```

Table 19.1 (Continued)

```
     Output for t = 0.03 to 0.120 removed
               .                 .
               .                 .
               .                 .
       t         x    u(x,t)   ua(x,t)      diff
   0.150     0.000   17.0247   17.0247    0.0000
   0.150     0.100   17.1082   17.1012    0.0070
   0.150     0.200   17.0943   17.0244    0.0699
   0.150     0.300   16.8925   16.7775    0.1150
   0.150     0.400   16.4935   16.3457    0.1478
   0.150     0.500   15.8574   15.7165    0.1409
   0.150     0.600   15.0006   14.8805    0.1201
   0.150     0.700   13.9043   13.8319    0.0723
   0.150     0.800   12.6011   12.5689    0.0322
   0.150     0.900   11.0847   11.0939   -0.0093
   0.150     1.000    9.4138    9.4138    0.0000

   ncall =  462
```

We can note the following details about this output:

- The solution array `out` has the dimensions `6 x 52` for 6 output points and 51 ODEs, with t included as an additional element, that is, `51 + 1 = 52`.
- The interval in x is $((1-(-0))/(51-1) = 0.02$ (with every fifth value in Table 19.1).
- The output interval in t is 0.03 as specified in the main program of Listing 19.1.
- The ICs for the numerical and analytical solutions are the same since `ua_1` was used in Listing 19.1 to define the numerical IC (with `ncase = 1`).
- The numerical solution is in acceptable agreement with the analytical solution (from the values of `diff` at $t = 0.15$). Thus, the succession of four first-order differentiations in `pde_1` of Listing 19.2 is valid.
- The computational effort is modest, `ncall = 462`.

The agreement between the numerical and analytical solutions is clear in Figure 19.1a (with numerical as solid lines and analytical as points).

The 3D graphical output from `persp` is in Figure 19.1b.

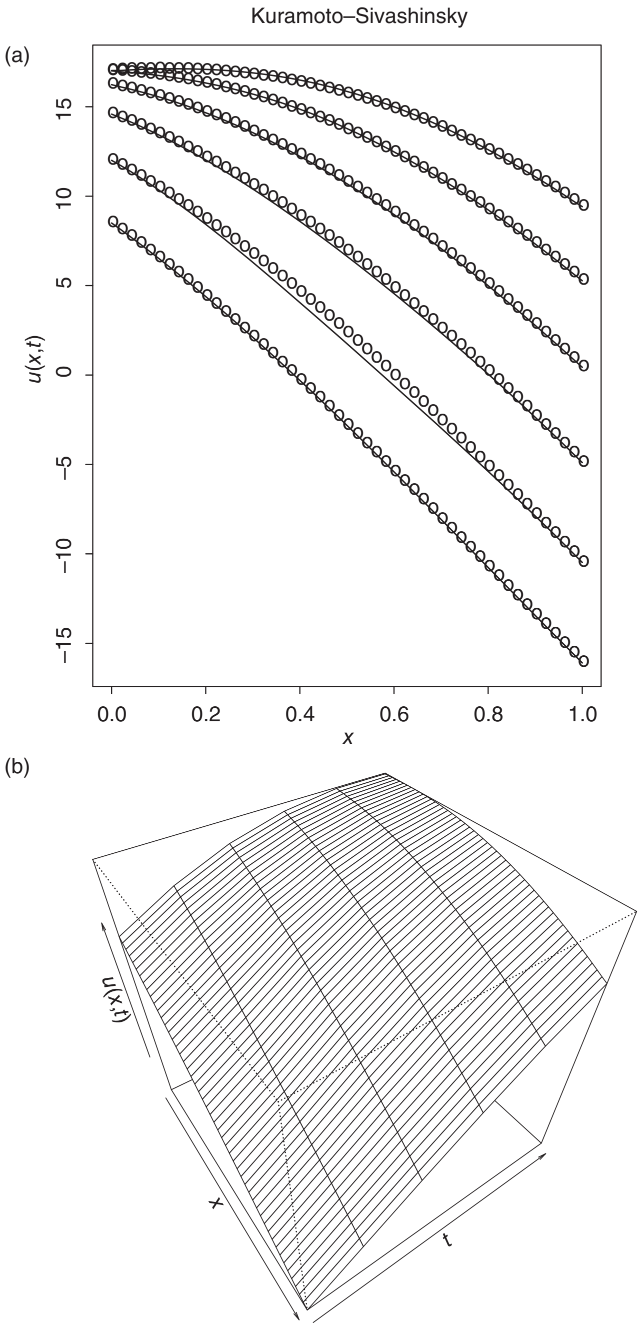

Figure 19.1 (a) Comparison of the numerical and analytical solutions of Eq. (19.1) `ncase = 1`, solid – num, points – anal. (b) Numerical solution to Eq. (19.1), `ncase=1`.

Abbreviated numerical output for `ncase=2` is in Table 19.2.

Table 19.2 Abbreviated numerical output for `ncase=2`.

```
[1] 6

[1] 102

      t        x   u(x,t)  ua(x,t)     diff
  0.000 -10.000   0.0143   0.0143   0.0000
  0.000  -7.000   0.1377   0.1377   0.0000
  0.000  -4.000   1.1527   1.1527   0.0000
  0.000  -1.000   2.8481   2.8481   0.0000
  0.000   2.000  -0.5221  -0.5221   0.0000
  0.000   5.000   1.8117   1.8117   0.0000
  0.000   8.000   2.3378   2.3378   0.0000
  0.000  11.000   2.3961   2.3961   0.0000
  0.000  14.000   2.4021   2.4021   0.0000
  0.000  17.000   2.4027   2.4027   0.0000
  0.000  20.000   2.4028   2.4028   0.0000
            .                 .
            .                 .
            .                 .
    Output for t = 2 to t = 8 removed
            .                 .
            .                 .
            .                 .
      t        x   u(x,t)  ua(x,t)     diff
 10.000 -10.000   0.0000   0.0000   0.0000
 10.000  -7.000  -0.0002   0.0000  -0.0002
 10.000  -4.000   0.0002   0.0001   0.0000
 10.000  -1.000   0.0018   0.0014   0.0003
 10.000   2.000   0.0144   0.0141   0.0002
 10.000   5.000   0.1365   0.1362   0.0003
 10.000   8.000   1.1424   1.1426  -0.0002
 10.000  11.000   2.8616   2.8608   0.0008
 10.000  14.000  -0.5302  -0.5306   0.0004
 10.000  17.000   1.8054   1.8059  -0.0005
 10.000  20.000   2.3372   2.3372   0.0000

 ncall =  652
```

We can note the following details about this output:

- The solution array `out` has the dimensions `6 x 102` for 6 output points and 101 ODEs, with t included as an additional element, that is, `101 + 1 = 102`.
- The interval in x is $((20-(-10))/(101-1)=0.3$ (with every tenth value in Table 19.1).
- The output interval in t is 2 as specified in the main program of Listing 19.1.
- The ICs for the numerical and analytical solutions are the same since `ua_1` was used in Listing 19.1 to define the numerical IC (with `ncase = 2`).
- The numerical solution is in good agreement with the analytical solution (from the values of `diff` at $t = 10$). Thus, the succession of four first-order differentiations in `pde_1` of Listing 19.2 is valid.
- The computational effort is modest, `ncall = 652`.

The agreement between the numerical and analytical solutions is clear in Figure 19.2a (with numerical as solid lines and analytical as points).

(a)

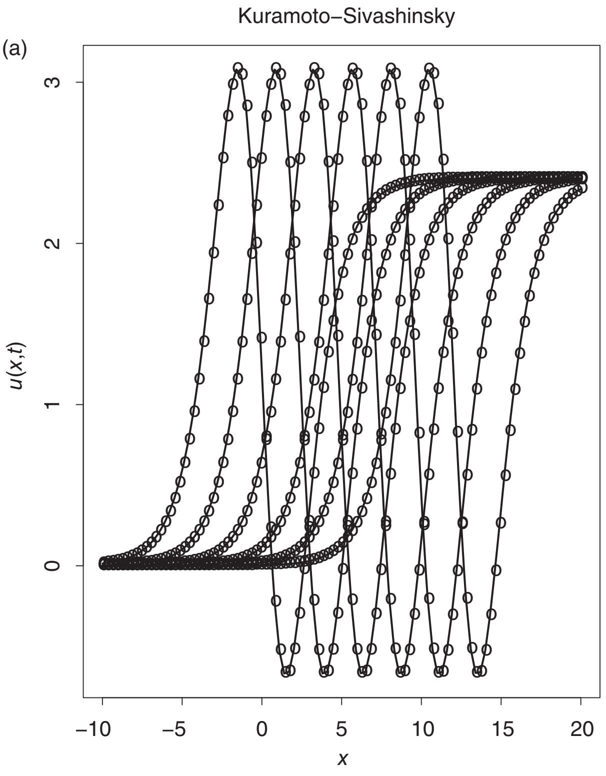

Figure 19.2 (a) Comparison of the numerical and analytical solutions of Eq. (19.1) `ncase = 2`, solid – num, points – anal. (b) Numerical solution to Eq. (19.1), `ncase=2`.

(b)

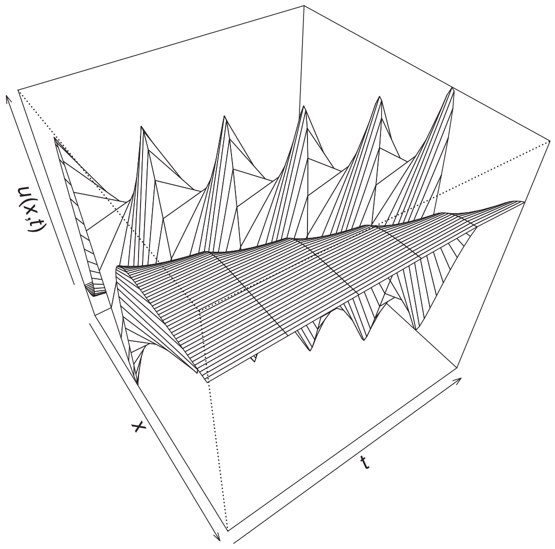

Figure 19.2 (*Continued*)

In particular, the maximum and minimum (peak) values of $u(x, t)$ are essentially constant with increasing t. This is a stringent test of the use of `splinefun` in `pde_1a`. These peak values could be studied in detail by considering the numerical output.

The 3D graphical output from `persp` is in Figure 19.2b.

19.6 Summary and Conclusions

The SCMOL can accommodate the higher order derivatives in x in the Kuramoto–Sivashinsky equation (19.1). For `ncase=2`, the traveling wave solution of Eqs. (19.5) is clear in Figure 19.2a (the solution for `ncase=1` is also a traveling wave, but this is not as apparent in Figure 19.1a). In both cases, the solutions are a function of a Lagrangian variable (a linear combination of x and t), as defined in Eqs. (19.4) and (19.5).

The choice of $n = 51{,}101$ for `ncase=1,2` gives acceptable spatial resolution in x, but these grid specifications could be changed to observe the effect on the accuracy of the numerical solution (the difference between the numerical and analytical solutions in the tabular numerical output). This is an example of h refinement to study the errors in numerical solutions.

However, the analytical solution is not actually required, but rather, the effect of changes in the number of spatial grid points on the numerical solution can be observed (by comparing numerical solutions for differing numbers of grid points). This is an important method for assessing the accuracy of a numerical solution when an analytical solution is not available (the usual case in most applications).

References

1 Kuramoto, Y. (1978), Diffusion-induced chaos in reaction systems, *Supplement of the Progress of Theoretical Physics*, **64**, 346–367.
2 Polyanin, A.D., and V.F. Zaitsev (2004), *Handbook of Nonlinear Partial Differential Equations*, Chapman & Hall/CRC, Boca Raton, FL.
3 Sivashinsky, G.I. (1977), Nonlinear analysis of hydrodynamic instability in laminar flames - I. Derivation of basic equations, *Acta Astronautica*, **4**, 1177–1206.
4 Sivashinsky, G.I. (1983), Instabilities, pattern formation, and turbulence in flames, *Annual Reviews of Fluid Mechanics*, **15**, 179–99.