Numerical Methods for Solving a System of Nonlinear Parabolic PDEs

David Keffer
Department of Materials Science & Engineering
University of Tennessee, Knoxville
date begun: May, 1999
last revised: March 11, 2014

Table of Contents

I. Formulation

Consider a set of coupled nonlinear parabolic PDEs of the general form,

$$\frac{\partial T^{(\ell)}}{\partial \mathrm{t}} = K^{(\ell)}\left(x,t,\{T\},\{\nabla T\},\left\{\nabla^2 T\right\}\right) \tag{1}$$

where the RHS of each PDE is potentially a function of all variables, their gradients and their Laplacians. Systems of linear PDEs are certainly a subset of this more general form. Single PDEs, either linear or nonlinear, are also certainly a subset of this more general form.

We have already derived and demonstrated a second-order method for a single nonlinear PDE, which is an analog of Heun's method for ODEs. That method converted a single PDE into a system of ODEs, where there was an ODE describing the temporal evolution of the dependent variable at each node. We can likewise convert a system of nonlinear PDEs into a system of ODEs, where there is an ODE describing the temporal evolution of each dependent variable at each node. In truth, there is little intellectual effort required to make this extension. There is only the need for methodical bookkeeping.

A comment on notation: we will write $T^{(\ell)}(t_j, x_i)$ as ${T^{(\ell)}}_i^j$, where j superscripts designate temporal increments, i subscripts designate spatial increments, and ℓ and k superscripts inside parentheses designate different dependent variables.

To recap what we did in the single parabolic PDE case, we first discretized time and space. Second we used the second order Heun's method to solve the time component of the PDE like an ODE.

$${T^{(\ell)}}_i^{j+1} = {T^{(\ell)}}_i^j + \frac{\Delta t}{2}\left[{K^{(\ell)}}_i^{j+1} + {K^{(\ell)}}_i^j \right] \tag{2}$$

where $K^{(\ell)}{}_i^j$ is the time derivative of $T^{(\ell)}{}_i^j$,

$$K^{(\ell)}{}_i^j = K^{(\ell)}\left(x_i, t_j, \left\{T^j\right\}, \left\{\nabla T^j\right\}, \left\{\nabla^2 T^j\right\}\right) \tag{3}$$

The braces in equation (3) now stand for the complete set over both position *i* and function (ℓ). The second function in equation (2) is given by $K(x_i, t_{j+1}, T_i^j + \Delta t K_i^j)$

$$K^{(\ell)}{}_i^{j+1} = K^{(\ell)}\left(x_i, t_{j+1}, \left\{T^{j+1}\right\}, \left\{\nabla T^{j+1}\right\}, \left\{\nabla^2 T^{j+1}\right\}\right) \tag{4}$$

where the temperature is approximated with an Euler method

$$T_i^{j+1} \approx T_i^j + \Delta t K_i^j \tag{5}$$

Note: this temperature is used not only for the explicit temperatures but is also used in the finite difference formulae to obtain the first and second spatial partial derivatives.

The boundary conditions are handled the same way as in the single non-linear parabolic PDE algorithm.

At the end of this file, two implementations of this algorithm are provided. The first code, `parapde_n_anyBC.m`, is a general code. The second code, `parapde_n_anyBC_flow.m`, is a version that is appropriate when a large convection term is present in the model. These codes are also available for download on the course website. As noted for the single PDE case, the only difference between these codes is that the general code uses a centered finite difference formula for the gradients and the flow code uses a backward finite difference formula for the gradients.

II. Plug Flow Reactor Example

Consider a plug flow reactor. (This is a pipe with a reaction taking place in the fluid flowing inside it.) Consider the irreversible reaction

$$A + 2B \rightarrow C + 2D$$

taking place in a non-reactive solvent. The molar balance for each component is given by

$$\frac{\partial C_i}{\partial t} = -v\frac{dC_i}{dz} + D_i\frac{d^2C_i}{dz^2} + \nu_i r$$

where z is the spatial dimension in the axial direction, t is time, C_i is the molar concentration of species i, v is the axial velocity, D_i is the diffusion coefficient of species i, ν_i is the stoichiometric for species i, (namely -1, -2, +1, +2 and 0 for A, B, C, D, and S respectively) and r is the reaction rate. The reaction rate is given by

$$r = kC_AC_B^2$$

where k is the rate constant. Assume the reactor is operated isothermally so we have no need for an energy balance.

The pipe is 10 m long with a diameter of 0.1 m. The velocity is 0.1 m/s. The diffusivities are all $1.0\text{x}10^{-9}$ m^2/s. The rate constant is $k = 1x10^{-7}\frac{m^6}{mol^2 \cdot s}$. Initially, the pipe contains nothing but solvent. At the inlet, the reactants, A and B, are fed in at 1000.0 and 2000.0 mol/m^3 respectively. No C or D is present in the feed stream. At the outlet, assume the concentrations no longer change (i.e. a no flux boundary condition).

(a) Solve the problem. Estimate how long it takes this reactor to get to steady state.
(b) Show the steady state profile.
(c) What fraction of the reactants are used, i.e. what is the fractional yield?
(d) What can be done to the velocity to increase the fractional yield? How does this impact the amount of product made per hour, i.e. the through-put?

Solution:

This is a system of four coupled non-linear parabolic PDEs with one spatial dimension and Dirichlet boundary conditions at z=0 and a Neumann boundary conditions at z=10 m. Moreover, this is a system in which convection is dominant. Therefore, to solve this problem, I will use the code `parapde_n_anyBC_flow.m`.

I modified the input functions in `parapde_n_anyBC_flow.m` as follows.

I assigned the appropriate type of boundary conditions.

```
BC(1,1) = 'D';
BC(2,1) = 'N';
BC(1,2) = 'D';
BC(2,2) = 'N';
BC(1,3) = 'D';
BC(2,3) = 'N';
BC(1,4) = 'D';
BC(2,4) = 'N';
```

I set the final time to 300 seconds and chose dt to be 0.1 seconds, so I had 3000 temporal intervals.

```
% discretize time
to = 0;
tf = 3.0e+2;
dt = 1.0e-1;
```

The pipe spans from 0 to 10.0 meter. I set dx to be 1.0 m, so I had 10 spatial intervals.

```
% discretize space
xo = 0;
xf = 10.0;
dx = 1.0e-0;
```

I defined the PDE in the following function.

```
function dydt_out = pdefunk(x,t,y,dydx,d2ydx2,keq);
% molar concentrations [mol/m^3]
CA = y(1);
CB = y(2);
CC = y(3);
CD = y(4);
% velocity [m/s]
v = 0.1;
% diffusivity [m^2/s]
D = 1.0e-9;
DA = 1.0*D;
DB = 1.0*D;
DC = 1.0*D;
DD = 1.0*D;
%  rate constant [m^6/mol^2/s]
k = 1.0e-7;
% stoichiometric coefficients
nuA = -1.0;
nuB = -2.0;
nuC = 1.0;
nuD = 2.0;
% reaction rate [mol/m^3/s]
rate = k*CA*CB*CB;
dydt(1) = -v*dydx(1) + DA*d2ydx2(1) + nuA*rate;
dydt(2) = -v*dydx(2) + DB*d2ydx2(2) + nuB*rate;
dydt(3) = -v*dydx(3) + DC*d2ydx2(3) + nuC*rate;
dydt(4) = -v*dydx(4) + DD*d2ydx2(4) + nuD*rate;
dydt_out = dydt(keq);
```

I defined the IC and BCs in the functions below.

The tank is initially all solvent. No A, B, C or D is present.

```
%
%  function defining initial condition
%

function ic_out = icfunk(x,keq);
ic(1) = 0.0;
ic(2) = 0.0;
ic(3) = 0.0;
ic(4) = 0.0;
ic_out = ic(keq);
```

Only A and B are fed into the reactor.

```
%
%  functions defining LHS boundary condition
%

function fout = aBCo(t,k);
f(1) = 1;
f(2) = 1;
f(3) = 1;
f(4) = 1;
fout = f(k);

function fout = bBCo(t,k);
f(1) = 0;
f(2) = 0;
f(3) = 0;
f(4) = 0;
fout = f(k);

function fout = cBCo(t,k);
f(1) = -1000;
f(2) = -2000;
f(3) = 0;
f(4) = 0;
fout = f(k);
```

All concentrations gradients are zero as the far end of the reactor.

```
%
%  functions defining RHS boundary condition
%

function fout = aBCf(t,k);
f(1) = 0;
f(2) = 0;
f(3) = 0;
f(4) = 0;
fout = f(k);

function fout = bBCf(t,k);
f(1) = 1;
```

```
f(2) = 1;
f(3) = 1;
f(4) = 1;
fout = f(k);

function fout = cBCf(t,k);
f(1) = 0;
f(2) = 0;
f(3) = 0;
f(4) = 0;
fout = f(k);
```

At the command line prompt, I typed

```
[xvec,tvec,Tmat] = parapde_n_anyBC_flow;
```

This command generated the following final frame of a movie.

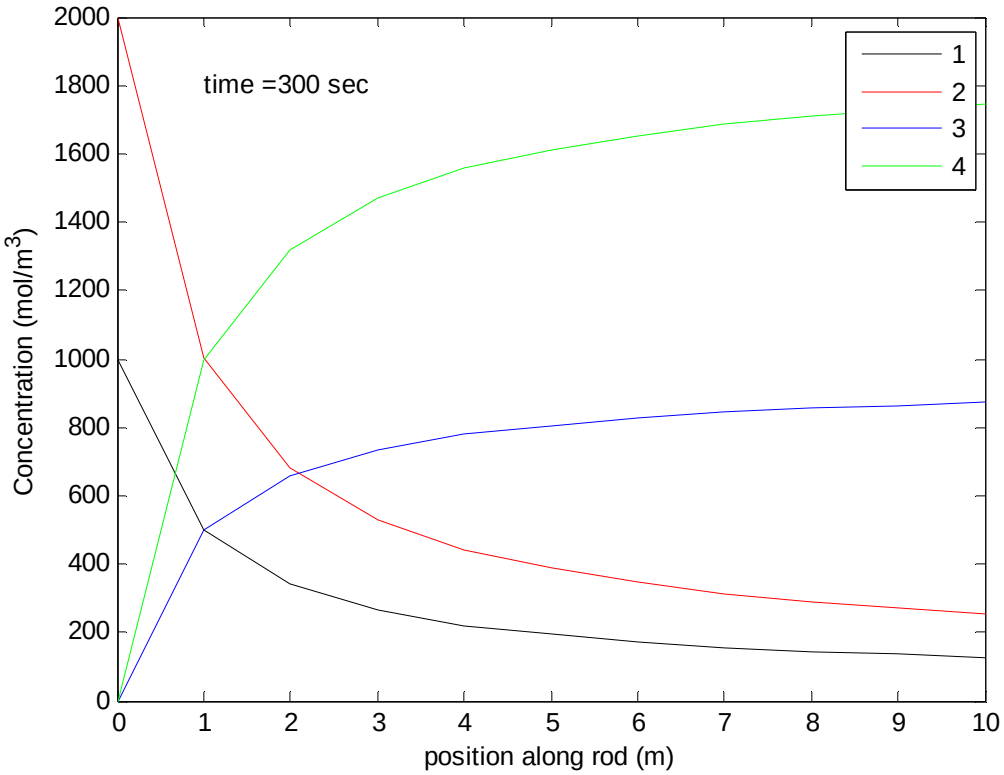

Figure 1. Concentration profiles of A, B, C and D at 300 s.

We can estimate how long it takes this reactor to get to steady state. First, make sure you are plotting the correct variables. The twelfth spatial index is the end of the pipe.

```
>> xvec(12)
ans =    10
```

The solution matrix, Tmat, has three indices, space, time, and equation respectively.

```
>> whos Tmat
  Name        Size                    Bytes  Class     Attributes
  Tmat       13x3001x4              1248416  double
```

This command plots the first concentration at the twelfth spatial node for all times.

```
>> plot(tvec,Tmat(12,:,1),'k-')
```

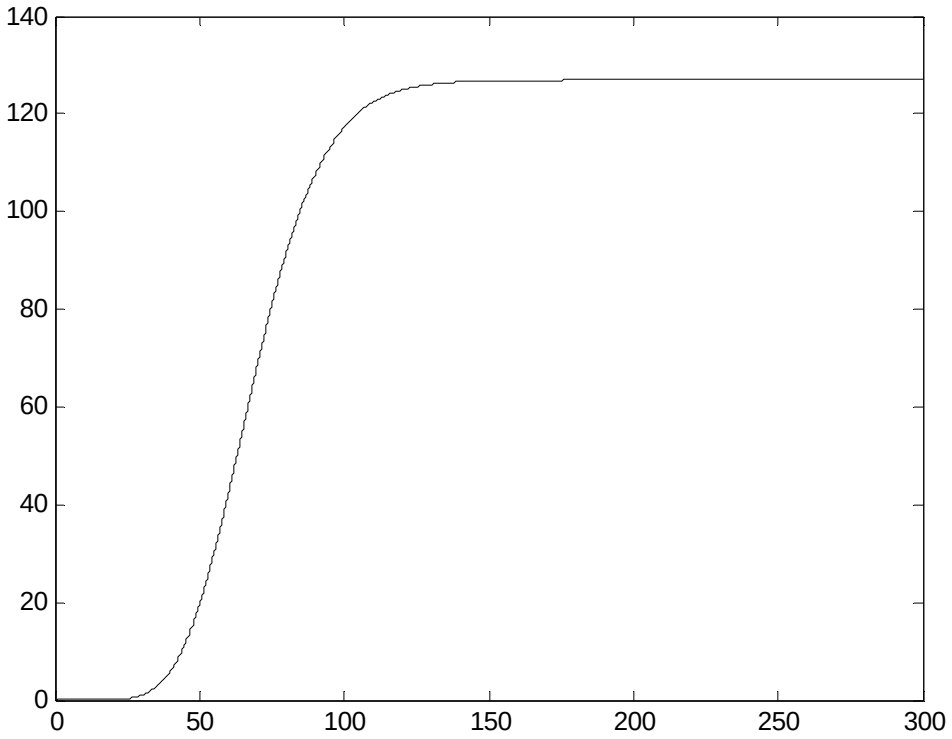

Figure 2. Concentration of A at the exit of the reactor as a function of time.

The system has pretty clearly reach steady state by 200 sec.

Since the reactor is at steady state by 200 sec, the profiles shown in Figure 1, obtained at 300 seconds, represent steady state profiles.

Any additional questions about fractional yield or through-put can be obtained by analysis of the results of this model.

Appendix I. parapde_n_anyBC.m

```
function [xvec,tvec,Tmat] = parapde_n_anyBC
%
%  The routine parapde_n_anyBC will solve
%  a system of non-linear 1-D parabolic PDEs of the form
%
%  dT/dt = f(x,t,T,dTdx,d2Tdx2)
%
%  using a second order method
%
%  The initial conditions must have the form
%  IC:  T(x,to,k_eq) = To(x,k_eq);
%
%  The boundary conditions can be
%  Dirichlet, Neumann or Mixed of the form
%  BC 1:  aBCo(t,k)*T(xo,t) + bBCo(t,k)*dTdx(xo,t) + cBCo(t,k) = 0;
%  BC 2:  aBCf(t,k)*T(xf,t) + bBCf(t,k)*dTdx(xf,t) + cBCf(t,k) = 0;
%
%  The necessary functions
%  pdefunk(x,t,T,dTdx,d2Tdx2,k_eq), To(x,k_eq),
%  aBCo(t,k), bBCo(t,k), cBCo(t,k), aBCf(t,k), bBCf(t,k), cBCf(t,k)
%  are entered at the bottom of the file
%
%  The number of equations, neq, must be entered at the top of the file
%
%  The initial value, final value and discretization in time, t,
%  to, tf, and dt must be entered at the top of the file.
%
%  The initial value, final value and discretization in space, x,
%  xo, xf, and dx must be entered at the top of the file.
%
%  The type of boundary conditions 'D', 'N' or 'M'
%  for each boundary must be entered at the top of the file.
%
%  sample execution:
%  [xvec,tvec,Tmat] = parapde_n_anyBC;
%
%  code written by:  David J. Keffer
%  University of Tennessee, dkeffer@utk.edu
%  code written:  October 21, 2001
%  comments last updated:  February 26, 2014
%
clear all;
close all;
%  define number of PDEs
neq = 4;

%
%  define type of boundary condition
%  Choices are 'D' = Dirichlet, i.e. bBC(t) = 0
%  Choices are 'N' = Neumann, i.e. aBC(t) = 0
%  Choices are 'M' = Mixed, bBC(t) ~= 0 & aBC(t) ~= 0
%
%  The boundary is the first index.  The equation is second.
BC(1,1) = 'D';
BC(2,1) = 'D';
BC(1,2) = 'D';
BC(2,2) = 'N';
```

```
BC(1,3) = 'D';
BC(2,3) = 'N';
BC(1,4) = 'D';
BC(2,4) = 'D';

% discretize time
to = 0;
tf = 4.0e+3;
dt = 4.0e+1;
tvec = [to:dt:tf];
nt = length(tvec);

% discretize space
xo = 0;
xf = 1.0;
dx = 1.0e-1;
% include imaginary boundary nodes
xvec = [xo-dx:dx:xf+dx];
nx = length(xvec);
dxi = 1.0/dx;

% dimension solution
Tmat = zeros(nx,nt,neq);


% dimension temporary vectors
Told = zeros(nx,1,neq);
Ttem = zeros(nx,1,neq);
Tnew = zeros(nx,1,neq);

% apply initial conditions to all real nodes
i = 1;
t = tvec(i);
for k = 1:1:neq
   for j = 2:1:nx-1
      Tmat(j,i,k) = icfunk(xvec(j),k);
   end
end


% apply Neumann/Mixed BCs as ICs at imaginary nodes
for k = 1:1:neq
   if (BC(1,k) ~= 'D')
      Tmat(1,i,k) =  2.0*dx/bBCo(t,k)*( aBCo(t,k)*Tmat(2,i,k)    +
bBCo(t,k)/(2.0*dx)*Tmat(3,i,k)    + cBCo(t,k));
   end
   if (BC(2,k) ~= 'D')
      Tmat(nx,i,k) = 2.0*dx/bBCf(t,k)*(-aBCf(t,k)*Tmat(nx-1,i,k) +
bBCf(t,k)/(2.0*dx)*Tmat(nx-2,i,k) - cBCf(t,k));
   end
end
%
%  Determine, based on type of BC, how to define which nodes are determined by
PDE vs BCs
%  ivaro = index of first node to be solved by PDE
%  ivarf = index of last node to be solved by PDE
%  nvar = number of nodes to be solved by PDE
%
for k = 1:1:neq
   if (BC(1,k) == 'D')
```

```
      ivaro(k) = 3;
   else
      ivaro(k) = 2;
   end
   if (BC(2,k) == 'D')
      ivarf(k) = nx-2;
   else
      ivarf(k) = nx-1;
   end
   nvar(k) = ivarf(k) - ivaro(k) + 1;
end


% loop over times
Told(1:nx,1:neq) = Tmat(1:nx,i,1:neq);
for i = 2:1:nt
%  update time
   t = tvec(i);

%
%  Prediction Step
%

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         dTdx(j,k) = 0.5*( Told(j+1,k) - Told(j-1,k) )*dxi;
         d2Tdx2(j,k) = ( Told(j+1,k) - 2.0*Told(j,k) + Told(j-1,k) )*dxi^2;
      end
   end

% estimate slope  at beginning of temporal interval
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         k1(j,k) =
pdefunk(xvec(j),tvec(i),Told(j,1:neq),dTdx(j,1:neq),d2Tdx2(j,1:neq),k);
      end
   end

% apply Euler method for the prediction step
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         Ttem(j,k) = Told(j,k) + dt*k1(j,k);
      end
   end

%  apply BCs at the prediction step
   for k = 1:1:neq
      if (BC(1,k) == 'D')
         Ttem(2,k) = -cBCo(t,k)/aBCo(t,k);
      else
         Ttem(1,k) =  2.0*dx/bBCo(t,k)*( aBCo(t,k)*Ttem(2,k)    +
bBCo(t,k)/(2.0*dx)*Ttem(3,k)    + cBCo(t,k));
      end
      if (BC(2,k) == 'D')
         Ttem(nx-1,k) = -cBCf(t,k)/aBCf(t,k);
      else
         Ttem(nx,k) = 2.0*dx/bBCf(t,k)*(-aBCf(t,k)*Ttem(nx-1,k) +
bBCf(t,k)/(2.0*dx)*Ttem(nx-2,k) - cBCf(t,k));
      end
```

```
   end

%
%  Correction Step
%

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         dTdx(j,k) = 0.5*( Ttem(j+1,k) - Ttem(j-1,k) )*dxi;
         d2Tdx2(j,k) = ( Ttem(j+1,k) - 2.0*Ttem(j,k) + Ttem(j-1,k) )*dxi^2;
      end
   end

% estimate slope  at end of temporal interval
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         k2(j,k) =
pdefunk(xvec(j),tvec(i),Ttem(j,1:neq),dTdx(j,1:neq),d2Tdx2(j,1:neq),k);
      end
   end

% apply second-order method for the correction step
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         Tnew(j,k) = Told(j,k) + 0.50*dt*(k1(j,k)+k2(j,k));
      end
   end


%  apply BCs at the correction step
   for k = 1:1:neq
      if (BC(1,k) == 'D')
         Tnew(2,k) = -cBCo(t,k)/aBCo(t,k);
      else
         Tnew(1,k) =  2.0*dx/bBCo(t,k)*( aBCo(t,k)*Tnew(2,k)    +
bBCo(t,k)/(2.0*dx)*Tnew(3,k)    + cBCo(t,k));
      end
      if (BC(2,k) == 'D')
         Tnew(nx-1,k) = -cBCf(t,k)/aBCf(t,k);
      else
         Tnew(nx,k) = 2.0*dx/bBCf(t,k)*(-aBCf(t,k)*Tnew(nx-1,k) +
bBCf(t,k)/(2.0*dx)*Tnew(nx-2,k) - cBCf(t,k));
      end
   end

% store new temperatures
   Tmat(1:nx,i,1:neq) = Tnew(1:nx,1:neq);
   Told(1:nx,1:neq) = Tnew(1:nx,1:neq);
end

% plot via movie
figure(1);
% number of frames in movie
nframe = 40;
% frames per second
fps = 3;
if (nt > nframe)
    nskip = round(nt/nframe);
else
```

```
   nskip = 1;
end
lplot = 1;
if (lplot == 1)
   fpsinv = 1.0/fps;
   newplot;
   Tmax=max(max(max(Tmat(2:nx-1,1:nt,1:neq))));
   Tmin=min(min(min(Tmat(2:nx-1,1:nt,1:neq))));
   xtext = xo + 0.1*(xf-xo);
   ytext = Tmin +0.9*(Tmax-Tmin);
   for i=1:nskip:nt
      for k = 1:1:neq
         %for j = 2:1:nx-1
         %   vector_plot(i_x) = T(k_eq,j_t,i_x);
         %end
         color_index = get_plot_color(k);
         plot(xvec(2:nx-1) , Tmat(2:nx-1,i,k),color_index);
         hold on
      end
      axis([xo xf Tmin Tmax]);
      xlabel('position along rod (cm)')
      ylabel('Temperature (K)')
      legend (int2str([1:neq]'));
      temps = char(strcat('time = ', num2str(tvec(i)), ' sec '));
      text(xtext, ytext,temps)
      hold off
      pause(fpsinv);
    end
end


%
%  functions defining PDE
%

function dydt_out = pdefunk(x,t,y,dydx,d2ydx2,keq);
% rho = density [kg/m^3]
rho = 2700.0;
% Cp = heat capacity [J/kg/K]
Cp_molar = 24.2; % J/mol/K
MW_gpm = 26.9815385; % g/mol
MW_kgpm = MW_gpm/1000.0; % kg/mol
Cp = 24.2/MW_kgpm;
% k = thermal conductivity [W/m/K]
k = 237.0;
%  alpha = thermal diffusivity
alpha = k/rho/Cp;
% length of rod [m]
L = 1.0;
%  heat transfer coefficient in [W/m^2/K]
h_tran_coeff = 40;
h_tran_coeff_23 = 80;
% diameter in [m]
radius = 0.1;
diameter = 2.0*radius;
% Area in [m^2]
Area = pi*diameter*L;
% Volume in [m^3]
Volume = pi/4*diameter^2*L;
% diameter in [m]
```

```
radius_3 = 0.2;
diameter_3 = 2.0*radius;
% Area in [m^2]
Area_3 = pi*diameter_3*L;
% Volume in [m^3]
Volume_3 = pi/4*diameter^2*L - Volume;
%  Temperature of the surroundings [K]
Tsurround = 200.0;
asurr = h_tran_coeff*Area/(rho*Cp*Volume);
a23 = h_tran_coeff_23*Area/(rho*Cp*Volume);
asurr_3 = h_tran_coeff_23*Area_3/(rho*Cp*Volume_3);
%
% convective heat loss in rod 1
%
dydt(1) = alpha*d2ydx2(1) + asurr*(Tsurround-y(1));
% convective heat loss in rod 3
% rod 3 is concentric around rod 2
% heat lost by rod 2 is gained by rod 3
dydt(2) = alpha*d2ydx2(2) + a23*(y(3)-y(2));
dydt(3) = alpha*d2ydx2(3) - a23*(y(3)-y(2)) + asurr*(Tsurround-y(3));
%
%  radiative heat loss in rod 4
%
sigma = 6.6404e-8; % J/s/m^2/K^4
S = Area/Volume; % 1/m
eps = 0.05;
factor = eps*sigma*S/k;
dydt(4) = alpha*d2ydx2(4) + factor*(Tsurround^4 - y(4)^4);
dydt_out = dydt(keq);
%
%  function defining initial condition
%

function ic_out = icfunk(x,keq);
ic(1) = 300.0;
ic(2) = 350.0 + 10.0*sin(x*2.0*pi);
ic(3) = 300.0 + 100.0*x;
ic(4) = 300.0;
ic_out = ic(keq);

%
%  functions defining LHS boundary condition
%

function fout = aBCo(t,k);
f(1) = 1;
f(2) = 1;
f(3) = 1;
f(4) = 1;
fout = f(k);

function fout = bBCo(t,k);
f(1) = 0;
f(2) = 0;
f(3) = 0;
f(4) = 0;
fout = f(k);

function fout = cBCo(t,k);
f(1) = -300;
```

```
f(2) = -300;
f(3) = -300;
f(4) = -300;
fout = f(k);

%
%  functions defining RHS boundary condition
%

function fout = aBCf(t,k);
f(1) = 1;
f(2) = 0;
f(3) = 0;
f(4) = 1;
fout = f(k);

function fout = bBCf(t,k);
f(1) = 0;
f(2) = 1;
f(3) = 1;
f(4) = 0;
fout = f(k);

function fout = cBCf(t,k);
f(1) = -400;
f(2) = 0;
f(3) = 0;
f(4) = -400;
fout = f(k);


%
% this little function sets colors for curves in plot
%
function color_index = get_plot_color(i);
if (i == 1)
   color_index = 'k-';
elseif (i == 2)
   color_index = 'r-';
elseif (i == 3)
   color_index = 'b-';
elseif (i == 4)
   color_index = 'g-';
elseif (i == 5)
   color_index = 'm-';
elseif (i == 6)
   color_index = 'k:';
elseif (i == 7)
   color_index = 'r:';
elseif (i == 8)
   color_index = 'b:';
elseif (i == 9)
   color_index = 'g:';
elseif (i == 10)
   color_index = 'm:';
else
   color_index = 'k-';
end
```

Appendix II. parapde_n_anyBC_flow.m

```
function [xvec,tvec,Tmat] = parapde_n_anyBC_flow
%
%  The routine parapde_n_anyBC will solve
%  a system of non-linear 1-D parabolic PDEs of the form
%
%  dT/dt = f(x,t,T,dTdx,d2Tdx2)
%
%  using a second order method
%
%  The initial conditions must have the form
%  IC:  T(x,to,k_eq) = To(x,k_eq);
%
%  The boundary conditions can be
%  Dirichlet, Neumann or Mixed of the form
%  BC 1:  aBCo(t,k)*T(xo,t) + bBCo(t,k)*dTdx(xo,t) + cBCo(t,k) = 0;
%  BC 2:  aBCf(t,k)*T(xf,t) + bBCf(t,k)*dTdx(xf,t) + cBCf(t,k) = 0;
%
%  The necessary functions
%  pdefunk(x,t,T,dTdx,d2Tdx2,k_eq), To(x,k_eq),
%  aBCo(t,k), bBCo(t,k), cBCo(t,k), aBCf(t,k), bBCf(t,k), cBCf(t,k)
%  are entered at the bottom of the file
%
%  The number of equations, neq, must be entered at the top of the file
%
%  The initial value, final value and discretization in time, t,
%  to, tf, and dt must be entered at the top of the file.
%
%  The initial value, final value and discretization in space, x,
%  xo, xf, and dx must be entered at the top of the file.
%
%  The type of boundary conditions 'D', 'N' or 'M'
%  for each boundary must be entered at the top of the file.
%
%  sample execution:
%  [xvec,tvec,Tmat] = parapde_n_anyBC;
%
%  code written by:  David J. Keffer
%  University of Tennessee, dkeffer@utk.edu
%  code written:  October 21, 2001
%  comments last updated:  February 26, 2014
%
clear all;
close all;
%  define number of PDEs
neq = 4;

%
%  define type of boundary condition
%  Choices are 'D' = Dirichlet, i.e. bBC(t) = 0
%  Choices are 'N' = Neumann, i.e. aBC(t) = 0
%  Choices are 'M' = Mixed, bBC(t) ~= 0 & aBC(t) ~= 0
%
%  The boundary is the first index.  The equation is second.
BC(1,1) = 'D';
BC(2,1) = 'N';
BC(1,2) = 'D';
BC(2,2) = 'N';
```

```
BC(1,3) = 'D';
BC(2,3) = 'N';
BC(1,4) = 'D';
BC(2,4) = 'N';

% discretize time
to = 0;
tf = 3.0e+2;
dt = 1.0e-1;
tvec = [to:dt:tf];
nt = length(tvec);

% discretize space
xo = 0;
xf = 10.0;
dx = 1.0e-0;
% include imaginary boundary nodes
xvec = [xo-dx:dx:xf+dx];
nx = length(xvec);
dxi = 1.0/dx;

% dimension solution
Tmat = zeros(nx,nt,neq);


% dimension temporary vectors
Told = zeros(nx,1,neq);
Ttem = zeros(nx,1,neq);
Tnew = zeros(nx,1,neq);

% apply initial conditions to all real nodes
i = 1;
t = tvec(i);
for k = 1:1:neq
   for j = 2:1:nx-1
      Tmat(j,i,k) = icfunk(xvec(j),k);
   end
end


% apply Neumann/Mixed BCs as ICs at imaginary nodes
for k = 1:1:neq
   if (BC(1,k) ~= 'D')
      %Tmat(1,i,k) =  2.0*dx/bBCo(t,k)*( aBCo(t,k)*Tmat(2,i,k)    +
bBCo(t,k)/(2.0*dx)*Tmat(3,i,k)    + cBCo(t,k));
      Tmat(1,i,k) =       dx/bBCo(t,k)*( aBCo(t,k)*Tmat(2,i,k)    +
bBCo(t,k)/(    dx)*Tmat(2,i,k)    + cBCo(t,k));
   end
   if (BC(2,k) ~= 'D')
      %Tmat(nx,i,k) = 2.0*dx/bBCf(t,k)*(-aBCf(t,k)*Tmat(nx-1,i,k) +
bBCf(t,k)/(2.0*dx)*Tmat(nx-2,i,k) - cBCf(t,k));
      Tmat(nx,i,k) =     dx/bBCf(t,k)*(-aBCf(t,k)*Tmat(nx-1,i,k) + bBCf(t,k)/(
dx)*Tmat(nx-1,i,k) - cBCf(t,k));
   end
end
%
%  Determine, based on type of BC, how to define which nodes are determined by
PDE vs BCs
%  ivaro = index of first node to be solved by PDE
%  ivarf = index of last node to be solved by PDE
```

```
%  nvar = number of nodes to be solved by PDE
%
for k = 1:1:neq
   if (BC(1,k) == 'D')
      ivaro(k) = 3;
   else
      ivaro(k) = 2;
   end
   if (BC(2,k) == 'D')
      ivarf(k) = nx-2;
   else
      ivarf(k) = nx-1;
   end
   nvar(k) = ivarf(k) - ivaro(k) + 1;
end


% loop over times
Told(1:nx,1:neq) = Tmat(1:nx,i,1:neq);
for i = 2:1:nt
%  update time
   t = tvec(i);

%
%  Prediction Step
%

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
%         dTdx(j,k) = 0.5*( Told(j+1,k) - Told(j-1,k) )*dxi;
         dTdx(j,k) =      ( Told(j  ,k) - Told(j-1,k) )*dxi;
         d2Tdx2(j,k) = ( Told(j+1,k) - 2.0*Told(j,k) + Told(j-1,k) )*dxi^2;
      end
   end

% estimate slope  at beginning of temporal interval
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         k1(j,k) =
pdefunk(xvec(j),tvec(i),Told(j,1:neq),dTdx(j,1:neq),d2Tdx2(j,1:neq),k);
      end
   end

% apply Euler method for the prediction step
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         Ttem(j,k) = Told(j,k) + dt*k1(j,k);
      end
   end

%  apply BCs at the prediction step
   for k = 1:1:neq
      if (BC(1,k) == 'D')
         Ttem(2,k) = -cBCo(t,k)/aBCo(t,k);
      else
         %Ttem(1,k) =  2.0*dx/bBCo(t,k)*( aBCo(t,k)*Ttem(2,k)    +
bBCo(t,k)/(2.0*dx)*Ttem(3,k)    + cBCo(t,k));
         Ttem(1,k) =       dx/bBCo(t,k)*( aBCo(t,k)*Ttem(2,k)    + bBCo(t,k)/(
dx)*Ttem(2,k)    + cBCo(t,k));
```

```
      end
      if (BC(2,k) == 'D')
         Ttem(nx-1,k) = -cBCf(t,k)/aBCf(t,k);
      else
         %Ttem(nx,k) = 2.0*dx/bBCf(t,k)*(-aBCf(t,k)*Ttem(nx-1,k) +
bBCf(t,k)/(2.0*dx)*Ttem(nx-2,k) - cBCf(t,k));
         Ttem(nx,k) =     dx/bBCf(t,k)*(-aBCf(t,k)*Ttem(nx-1,k) + bBCf(t,k)/(
dx)*Ttem(nx-1,k) - cBCf(t,k));
      end
   end

%
%  Correction Step
%

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         %dTdx(j,k) = 0.5*( Ttem(j+1,k) - Ttem(j-1,k) )*dxi;
         dTdx(j,k) =      ( Ttem(j,  k) - Ttem(j-1,k) )*dxi;
         d2Tdx2(j,k) = ( Ttem(j+1,k) - 2.0*Ttem(j,k) + Ttem(j-1,k) )*dxi^2;
      end
   end

% estimate slope  at end of temporal interval
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         k2(j,k) =
pdefunk(xvec(j),tvec(i),Ttem(j,1:neq),dTdx(j,1:neq),d2Tdx2(j,1:neq),k);
      end
   end

% apply second-order method for the correction step
   for k = 1:1:neq
      for j = ivaro(k):1:ivarf(k)
         Tnew(j,k) = Told(j,k) + 0.50*dt*(k1(j,k)+k2(j,k));
      end
   end


%  apply BCs at the correction step
   for k = 1:1:neq
      if (BC(1,k) == 'D')
         Tnew(2,k) = -cBCo(t,k)/aBCo(t,k);
      else
%         Tnew(1,k) =  2.0*dx/bBCo(t,k)*( aBCo(t,k)*Tnew(2,k)    +
bBCo(t,k)/(2.0*dx)*Tnew(3,k)    + cBCo(t,k));
         Tnew(1,k) =       dx/bBCo(t,k)*( aBCo(t,k)*Tnew(2,k)    + bBCo(t,k)/(
dx)*Tnew(2,k)    + cBCo(t,k));
      end
      if (BC(2,k) == 'D')
         Tnew(nx-1,k) = -cBCf(t,k)/aBCf(t,k);
      else
         %Tnew(nx,k) = 2.0*dx/bBCf(t,k)*(-aBCf(t,k)*Tnew(nx-1,k) +
bBCf(t,k)/(2.0*dx)*Tnew(nx-2,k) - cBCf(t,k));
         Tnew(nx,k) =     dx/bBCf(t,k)*(-aBCf(t,k)*Tnew(nx-1,k) + bBCf(t,k)/(
dx)*Tnew(nx-1,k) - cBCf(t,k));
      end
   end
```

```
% store new temperatures
   Tmat(1:nx,i,1:neq) = Tnew(1:nx,1:neq);
   Told(1:nx,1:neq) = Tnew(1:nx,1:neq);
end

% plot via movie
figure(1);
% number of frames in movie
nframe = 40;
% frames per second
fps = 3;
if (nt > nframe)
    nskip = round(nt/nframe);
else
   nskip = 1;
end
lplot = 1;
if (lplot == 1)
   fpsinv = 1.0/fps;
   newplot;
   Tmax=max(max(max(Tmat(2:nx-1,1:nt,1:neq))));
   Tmin=min(min(min(Tmat(2:nx-1,1:nt,1:neq))));
   xtext = xo + 0.1*(xf-xo);
   ytext = Tmin +0.9*(Tmax-Tmin);
   for i=1:nskip:nt
      for k = 1:1:neq
         %for j = 2:1:nx-1
         %   vector_plot(i_x) = T(k_eq,j_t,i_x);
         %end
         color_index = get_plot_color(k);
         plot(xvec(2:nx-1) , Tmat(2:nx-1,i,k),color_index);
         hold on
      end
      axis([xo xf Tmin Tmax]);
      xlabel('position along rod (m)')
      ylabel('Concentration (mol/m^3)')
      legend (int2str([1:neq]'));
      temps = char(strcat('time = ', num2str(tvec(i)), ' sec '));
      text(xtext, ytext,temps)
      hold off
      pause(fpsinv);
    end
end


%
%  functions defining PDE
%

function dydt_out = pdefunk(x,t,y,dydx,d2ydx2,keq);
% molar concentrations [mol/m^3]
CA = y(1);
CB = y(2);
CC = y(3);
CD = y(4);
% velocity [m/s]
v = 0.1;
% diffusivity [m^2/s]
D = 1.0e-9;
DA = 1.0*D;
```

```
DB = 1.0*D;
DC = 1.0*D;
DD = 1.0*D;
%  rate constant [m^6/mol^2/s]
k = 1.0e-7;
% stoichiometric coefficients
nuA = -1.0;
nuB = -2.0;
nuC = 1.0;
nuD = 2.0;
% reaction rate [mol/m^3/s]
rate = k*CA*CB*CB;
dydt(1) = -v*dydx(1) + DA*d2ydx2(1) + nuA*rate;
dydt(2) = -v*dydx(2) + DB*d2ydx2(2) + nuB*rate;
dydt(3) = -v*dydx(3) + DC*d2ydx2(3) + nuC*rate;
dydt(4) = -v*dydx(4) + DD*d2ydx2(4) + nuD*rate;
dydt_out = dydt(keq);
%
%  function defining initial condition
%

function ic_out = icfunk(x,keq);
ic(1) = 0.0;
ic(2) = 0.0;
ic(3) = 0.0;
ic(4) = 0.0;
ic_out = ic(keq);

%
%  functions defining LHS boundary condition
%

function fout = aBCo(t,k);
f(1) = 1;
f(2) = 1;
f(3) = 1;
f(4) = 1;
fout = f(k);

function fout = bBCo(t,k);
f(1) = 0;
f(2) = 0;
f(3) = 0;
f(4) = 0;
fout = f(k);

function fout = cBCo(t,k);
f(1) = -1000;
f(2) = -2000;
f(3) = 0;
f(4) = 0;
fout = f(k);

%
%  functions defining RHS boundary condition
%

function fout = aBCf(t,k);
f(1) = 0;
f(2) = 0;
```

```
f(3) = 0;
f(4) = 0;
fout = f(k);

function fout = bBCf(t,k);
f(1) = 1;
f(2) = 1;
f(3) = 1;
f(4) = 1;
fout = f(k);

function fout = cBCf(t,k);
f(1) = 0;
f(2) = 0;
f(3) = 0;
f(4) = 0;
fout = f(k);


%
% this little function sets colors for curves in plot
%
function color_index = get_plot_color(i);
if (i == 1)
   color_index = 'k-';
elseif (i == 2)
   color_index = 'r-';
elseif (i == 3)
   color_index = 'b-';
elseif (i == 4)
   color_index = 'g-';
elseif (i == 5)
   color_index = 'm-';
elseif (i == 6)
   color_index = 'k:';
elseif (i == 7)
   color_index = 'r:';
elseif (i == 8)
   color_index = 'b:';
elseif (i == 9)
   color_index = 'g:';
elseif (i == 10)
   color_index = 'm:';
else
   color_index = 'k-';
end
```