Numerical Methods for Solving a Single Nonlinear Parabolic PDE

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

Nonlinear parabolic partial differential equations are, in their most general form, given by:

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

An example of this is the general form of the linear parabolic PDE, which we solved in the last section.

$$\frac{\partial T}{\partial \mathrm{t}} = \nabla \cdot \left[c(\nabla T)\right] - aT - \underline{b} \cdot \nabla T + f \tag{2}$$

Another example is

$$\frac{\partial T}{\partial \mathrm{t}} = \nabla \cdot \left[c(\nabla T)\right] - aT^2 - \underline{b} \cdot \nabla T + f \tag{3}$$

Because equation (3) is no longer linear in the temperature and its derivatives, the techniques for solving linear parabolic PDEs, e.g. the Crank-Nicolson method no longer apply. We cannot reduce equation (3) to a discrete system of linear equations.

Instead we shall use an approach that is essentially Heun's method, a second order method, for solving systems of ODEs. The extension to PDEs is straightforward since we will transform the PDE to a system of ODEs through the use of finite-difference formulae for the gradient and Laplacian.

II. Method Formulation for General Systems

This section provides a derivation of the second-order finite difference equations for nonlinear parabolic PDEs

Let *j* superscripts designate temporal increments and let *i* subscripts designate spatial increments. For purposes of brevity only, we will consider the case with variation only in one spatial dimension. Our most general parabolic PDE becomes in one spatial dimension

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

Looking at it in this light, we can obtain a new estimate of the temperature, one increment ahead in time, namely

$$\left(\frac{\partial T}{\partial \mathrm{t}}\right)_i \approx \frac{T_i^{j+1} - T_i^{j}}{t_{j+1} - t_j} = \frac{T_i^{j+1} - T_i^{j}}{\Delta t} \tag{4}$$

This statement is true at any given point *i* in space. It is a can make a forward finite difference formula of the partial derivative with respect to time. In the second-order method, we will approximate the slope with the average of two functional evaluations of the PDE, one at the beginning of the interval and one at the end.

$$\begin{aligned}\left(\frac{\partial T}{\partial \mathrm{t}}\right)_i &\approx \frac{1}{2}\left[K(x_i,t_{j+1},T^{j+1},\nabla T^{j+1},\nabla^2 T^{j+1}) + K(x_i,t_j,T^{j},\nabla T^{j},\nabla^2 T^{j})\right] \\ &= \frac{1}{2}\left[K_i^{j+1} + K_i^{j}\right]\end{aligned} \tag{5}$$

so that:

$$T_i^{j+1} = T_i^{j} + \frac{\Delta t}{2}\left[K_i^{j+1} + K_i^{j}\right] \tag{6}$$

The problem here is that we have no way of determining T_i^{j+1}, which is used as an argument in K_i^{j+1} unless we rely on a costly and inefficient method for finding the roots of a system of non-linear algebraic equations at each time iteration. This method is, of course, an option, but if we only want second order accuracy in our model, there are other, easier ways to get it.

We can estimate (or predict) the value of the new temperature using the Euler method,

$$T_i^{j+1} \approx T_i^{j} + \Delta t K_i^{j} \tag{7}$$

With this prediction, we can evaluate K_i^{j+1}. Once we know, K_i^{j+1}, we can use Heun's method (equation (6)) to correct the new value of the temperature.

It is important to remember that these K_i^j are functions not only of T_i^j but of the temperature at all spatial points $\left\{ T^j \right\}$. So we have to solve a system of ODEs simultaneously, which we have now done many times in this course.

Also, we need to recognize that the gradients and Laplacians inside K_i^j must be evaluated using the same finite difference formulae as were used in the linear case, namely:

$$\left(\frac{\partial T}{\partial \mathrm{x}}\right)_i^j \approx \frac{T_{i+1}^j - T_{i-1}^j}{x_{i+1} - x_{i-1}} = \frac{T_{i+1}^j - T_{i-1}^j}{2\Delta x} \tag{8}$$

and

$$\left(\frac{\partial^2 T}{\partial \mathrm{x}^2}\right)_i^j \approx \frac{\left(\frac{T_{i+1}^j - T_i^j}{\Delta x}\right) - \left(\frac{T_i^j - T_{i-1}^j}{\Delta x}\right)}{\Delta x} = \frac{T_{i+1}^j - 2T_i^j + T_{i-1}^j}{\Delta x^2} \tag{9}$$

A code that implements this routine, `parapde_1_anyBC.m`, is provided in Appendix 1 below.

III. Heat Transfer Application

The one-dimensional heat equation can describe heat transfer in a material with both heat conduction and radiative heat loss.

$$\frac{\partial T}{\partial \mathrm{t}} = \frac{k}{\rho C_p}\frac{d^2 T}{dz^2} - \frac{\varepsilon \sigma S}{\rho C_p}\left(T^4 - T_s^4\right)$$

The radiative heat loss term involves temperature to the fourth power and is therefore nonlinear. Consider a cylindrical Cu rod of diameter 0.01 m and length 0.1 mm, which is initially at $T(z, t=0) = 1000$ K. One end of the rod is maintained at $T(z=0, t) = 1000$ K, a Dirichlet boundary condition. The other end of the rod is insulated, $\left.\frac{dT}{dz}\right|_{z=0.1} = 0$ K/m, a Neumann boundary condition.

In this problem, we will employ the following units and numerical values for parameters.

- temperature in the material T [K]
- surrounding temperature $T_s = 300$ [K]
- axial position along material z [m]
- thermal conductivity $k = 401$ [J/K/m/s] (for Cu)
- mass density $\rho =$ 8960 [kg/m^3] (for Cu)
- heat capacity $C_p = 384.6$ [J/kg/K] (for Cu)

- Stefan–Boltzmann constant $\sigma = 5.670373x10^{-8}$ [J/s/m^2/K^4]
- gray body permittivity $\varepsilon = 0.15$ (for dull Cu)
- surface area to volume ratio $S = 200$ [m^{-1}] (for a cylindrical rod of diameter 0.01 m)

This is a single non-linear parabolic PDE with one spatial dimension and a Dirichlet boundary condition at z=0 and a Neumann boundary condition at z=0.1. To solve this problem, I will use the code `parapde_1_anyBC.m`.

I modified the input functions in `parapde_1_anyBC.m` as follows.

I assigned the appropriate type of boundary conditions.

```
BC(1) = 'D';
BC(2) = 'N';
```

I set the final time to 100 seconds and chose dt to be 0.1 seconds, so I had 1000 temporal intervals.

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

The rod spans from 0 to 0.1 meter. I set dx to be 0.005 m, so I had 20 spatial intervals.

```
% discretize space
xo = 0;
xf = 0.1;
dx = 5.0e-3;
```

I defined the PDE in the following function.

```
%
%  function defining PDE
%
function k = pdefunk(x,t,y,dydx,d2ydx2);
%
Temp = y;
% rho = density [kg/m^3]
rho = 8960.0;
% Cp = heat capacity [J/kg/K]
Cp = 384.6;
% k = thermal conductivity [W/m/K]
k = 401.0;
%  alpha = thermal diffusivity
alpha = k/rho/Cp;
% length of rod [m]
L = 0.1;
% diameter in [m]
radius = 0.1;
diameter = 2.0*radius;
% surface Area in [m^2]
Area = pi*diameter*L;
```

```
% Volume in [m^3]
Volume = pi/4*diameter^2*L;
% surface area to volume ratio
S = Area/Volume;
%  Temperature of the surroundings [K]
Tsurround = 300.0;
% Stefan-Boltzmann constant [J/s/m^2/K^4]
sigma = 5.670373e-8;
% gray body permittivity [dimensionless]
eps = 0.15;
fac = eps*sigma*S/(rho*Cp);
k = alpha*d2ydx2 - fac*(Temp^4 - Tsurround^4);
```

I defined the IC and BCs in the functions below.

```
%
%  function defining initial condition
%

function ic = icfunk(x);
ic = 1000;

%
%  functions defining LHS boundary condition
%

function f = aBCo(t);
f = 1;

function f = bBCo(t);
f = 0;

function f = cBCo(t);
f = -1000;

%
%  functions defining RHS boundary condition
%

function f = aBCf(t);
f = 0;

function f = bBCf(t);
f = 1;

function f = cBCf(t);
f = 0;
```

At the command line prompt, I typed

```
[xvec,tvec,Tmat] = parapde_1_anyBC;
```

This command generated the plot shown in Figure 1.

To find the last value at x = 0.1 m, I confirmed that I knew the correct spatial and temporal indices.

```
>> xvec(22)
ans =    0.1000

>> tvec(1001)
ans =   100

>> Tmat(22,1001)
ans =  998.0308
```

Therefore the temperature at the end at 100 seconds is 998.03 K.

I don't know that this is steady state. I can run the simulation longer. If I change nothing but the final time to 1000 seconds, then I generate the data point

```
>> Tmat(22,10001)
ans =  997.9108
```

Therefore the temperature at the end at 1000 seconds is 997.91 K.
The two answers agree to three digits, so we are pretty close to the steady state solution, but we can run for a longer time. If I change nothing but the final time to 5000 seconds, then I generate the data point

```
>> Tmat(22,50001)
ans =  997.9108
```

Therefore the temperature at the end at 5000 seconds is 997.91 K.

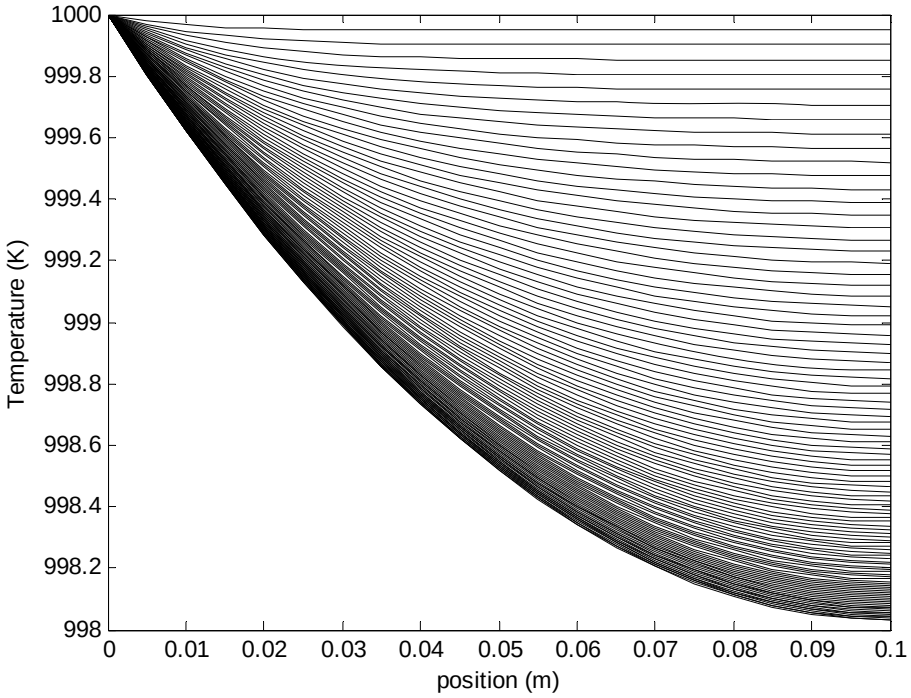

Figure 1. Transient behavior of a rod with radiative heat loss.

IV. Method Formulation for Systems Dominated by Convection (Flow)

It turns out that if you apply this code to a flowing system, that the algorithm is unstable, as will be demonstrated in the following section. A simple modification to the code removes this instability.

If the velocity is positive, then flowing material or energy moves from left to right. This introduces an asymmetry in the problem that is not well handled by the centered-finite difference formula. Therefore, we replace the centered-finite-difference formula with the backward finite difference formula.

$$\left(\frac{\partial T}{\partial \mathrm{x}}\right)_i^j \approx \frac{T_i^j - T_{i-1}^j}{x_i - x_{i-1}} = \frac{T_i^j - T_{i-1}^j}{\Delta x} \tag{10}$$

The formula for the Laplacian is left alone. This change will allow the code to be stable when describing convection-dominated systems.

A code that implements this routine, `parapde_1_anyBC_flow.m`, is provided in Appendix 2 below.

An example of a convection dominated problem using both codes is provided in the next section.

V. Plug Flow Reactor Application

V.A. Using the Appropriate Flow-Dominated Routine

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

$$2A \rightarrow B$$

taking place in a non-reactive solvent.

The molar balance for component A is given by

$$\frac{\partial C_A}{\partial \mathrm{t}} = -v\frac{dC_A}{dz} + D\frac{d^2C_A}{dz^2} + \nu_A r$$

where z is the spatial dimension in the axial direction, t is time, C_A is the molar concentration of species A, v is the axial velocity, D is the diffusion coefficient, ν_A is the stochiometric for species A, (namely -2) and r is the reaction rate. The reaction rate is given by

$$r = kC_A^2$$

where k is the rate constant. Assume the reactor is operated isothermally so we have no need for an energy balance. Since the reaction is second order, the PDE is nonlinear.

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

In this case, we can examine the reactor, strictly through the molar balance on A, since the PDE is not coupled to any other material or energy balances. This is a single 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_1_anyBC_flow.m`.

I modified the input functions in `parapde_1_anyBC.m` as follows.

I assigned the appropriate type of boundary conditions.

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

I set the final time to 2000 seconds and chose dt to be 1 seconds, so I had 2000 temporal intervals.

```
% discretize time
to = 0;
tf = 2.0e+3;
dt = 1.0e+0;
```

The rod spans from 0 to 10 meters. I set dx to be 1 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 = pdefunk(x,t,y,dydx,d2ydx2);
% molar concentrations [mol/m^3]
CA = y(1);
% velocity [m/s]
v = 0.1;
% diffusivity [m^2/s]
D = 1.0e-9;
%  rate constant [m^6/mol^2/s]
k = 1.0e-5;
```

```
% stoichiometric coefficients
nuA = -2.0;
% reaction rate [mol/m^3/s]
rate = k*CA^2;
dydt = -v*dydx + D*d2ydx2 + nuA*rate;
```

I defined the IC and BCs in the functions below.

```
%
%  function defining initial condition
%

function ic = icfunk(x);
ic = 0;

%
%  functions defining LHS boundary condition
%

function f = aBCo(t);
f = 1;

function f = bBCo(t);
f = 0;

function f = cBCo(t);
f = -1000;

%
%  functions defining RHS boundary condition
%

function f = aBCf(t);
f = 0;


function f = bBCf(t);
f = 1;

function f = cBCf(t);
f = 0;
```

At the command line prompt, I typed

```
[xvec,tvec,Tmat] = parapde_1_anyBC_flow;
```

This command generated the plot shown in Figure 2.

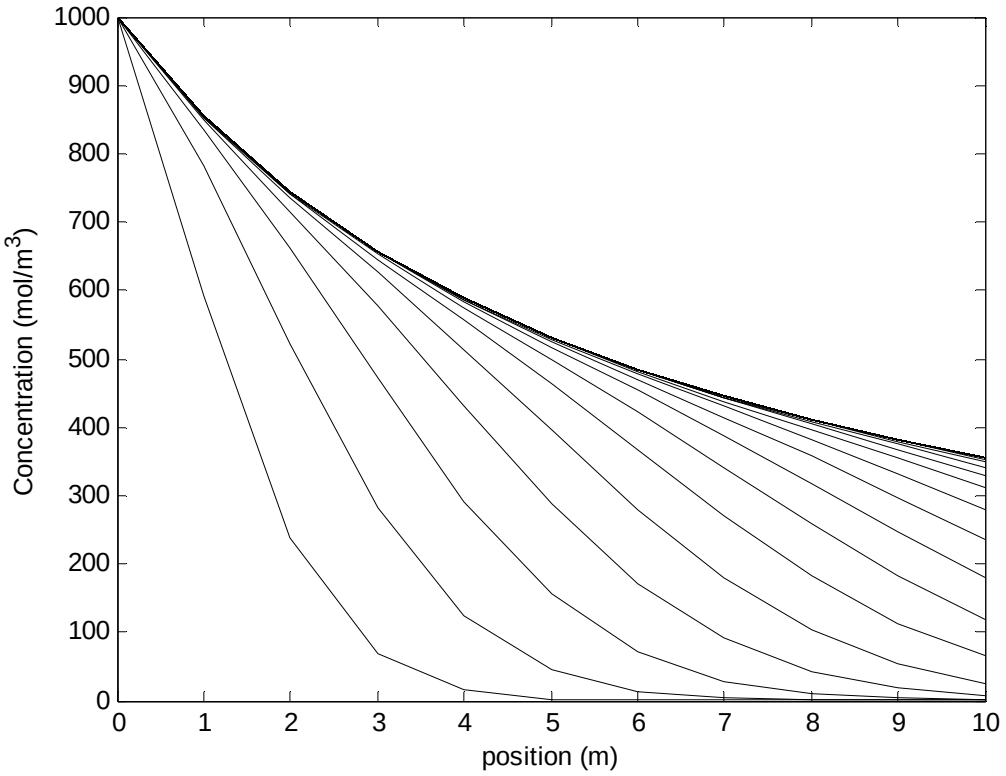

Figure 2. Transient behavior of concentration of A in a plug flow reactor.

Interesting problems relating to the domain science of reaction engineering can be answered using this model. For example, we can estimate how long it takes this reactor to get to steady state. If a pipe (plug flow reactor) is L = 10 m long and the velocity is 0.1 m/s, then the "residence time of the reactor" is given by $\tau = \frac{L}{v}$, (100 s in this example) which sets a lower bound for reaching steady state. Maybe it takes two residence times to reach steady state. We can examine the concentration of A at the outlet as a function of time. To do so, first, we make sure that we are plotting the correct variables.

The twelfth spatial index is the end of the pipe.

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

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

```
>> whos Tmat
  Name        Size                Bytes  Class     Attributes
  Tmat       13x2001             208104  double
```

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

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

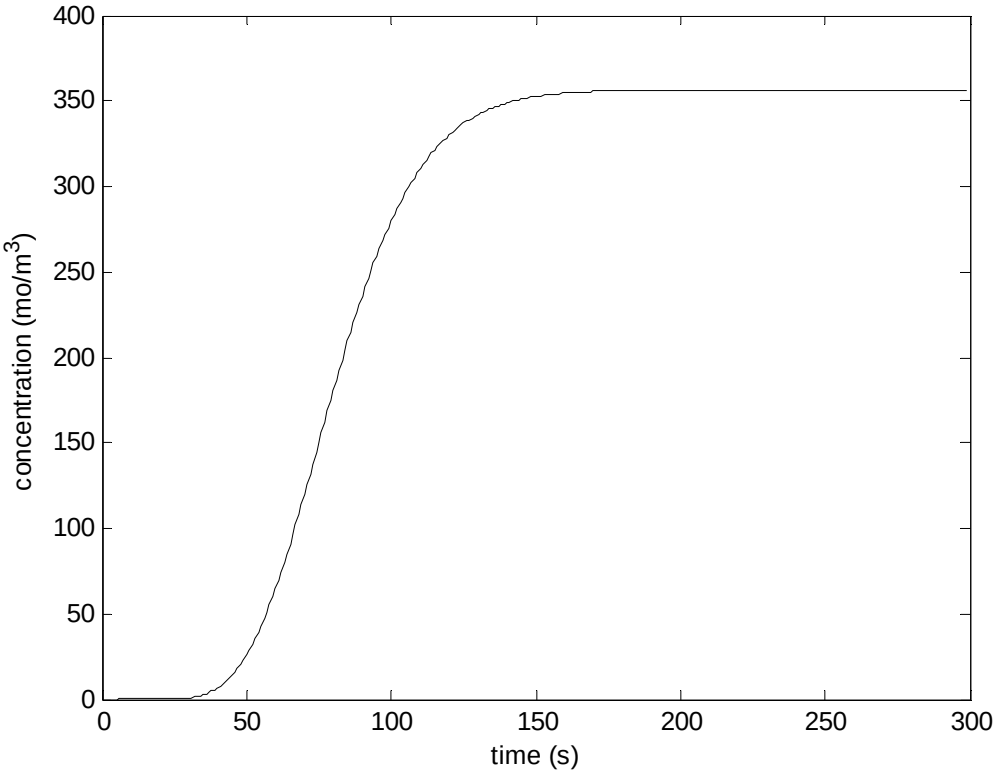

Figure 3. Plot of outlet concentration of A as a function of time.

From Figure 3, we can observe that it takes aobut 200 s for the reactor to reach steady state.

Since this is the case, the steady state profile of the concentration of A in the reactor is the last curve (corresponding to a time of 2000 s) shown in Figure 2.

We can also examine the fractional yield, which is defined as

$$Y_A = \frac{C_{A,in} - C_{A,out}}{C_{A,in}} = \frac{1000 - 356.5}{1000} = 64.4\%$$

where the inlet concentration of A was specified in a boundary condition as 1000 mol/m^3 and the outlet concentration of A was determined by examining the solution, as follows

```
>> Tmat(12,2001)
ans =  356.5422
```

We can also examine the through-put, the amount of product made per hour, i.e. the through-put?

$$Q_B = vA_X C_{B,out} = v\left(\pi r^2\right)\frac{\nu_B}{\nu_A}\left(C_{A,in} - C_{A,out}\right)$$

In the latter equality, we used the stoichiometry of the reaction to relate the amount of B produced based on the amount of A consumed. For $v = 0.1$ m/s, $C_{A,out} = 356.5$ mol/m^3 and $Q_B = 0.253$ mol/s.

If we rerun the code with half the velocity, $v = 0.05$ m/s, then $C_{A,out} = 224.7$ mol/m^3 and the fractional yield is $Y_A = 77.5\%$ and the through-out is $Q_B = 0.152$ mol/s. So we observe that slowing the velocity increases the fractional yield but decreases the overall the overall rate of production of B. There must be an optimal velocity for through-put. One could directly find this by nesting this PDE solver inside the Newton-Raphson method.

V.B. Using the Inappropriate General Routine

One can use the general code, `parapde_1_anyBC.m`, which implements the centered-finite difference formula for the gradients to solve the above plug flow reactor problem, rather than the version of the code, `parapde_1_anyBC_flow.m`, in which the backward finite difference formula is used for the gradients. Doing so results in numerical instabilities, as shown in Figure 4 below.

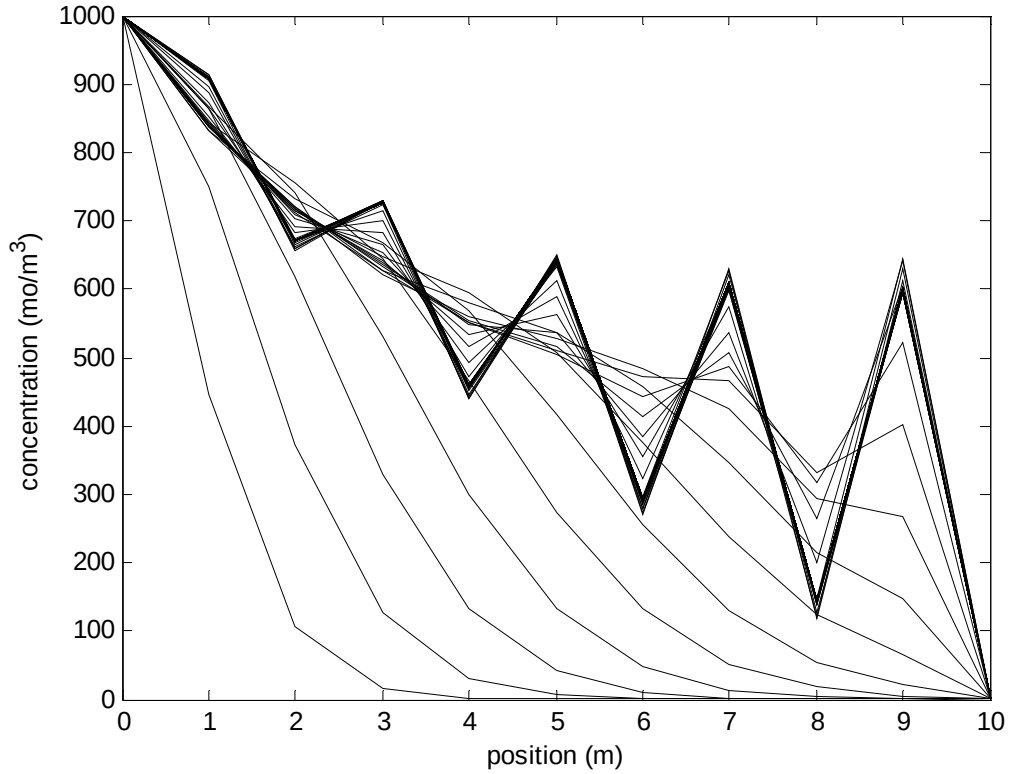

Figure 4. Concentration of A along the reactor. Using a centered-finite difference formula in a flow dominated regime leads to clear numerical instabilities. These instabilities can be reduced by reductions in time step but cannot be entirely eliminated, except by switching to a backward finite difference formula.

Appendix I. parapde_1_anyBC.m

```
function [xvec,tvec,Tmat] = parapde_1_anyBC_hw5p1
%
%  The routine parapde_1_anyBC will solve
%  a single non-linear 1-D parabolic PDE of the form
%
%  dT/dt = f(x,t,T,dTdx,d2Tdx2)
%
%  using a second order method
%
%  The initial condition must have the form
%  IC:  T(x,to) = To(x);
%
%  The boundary conditions can be
%  Dirichlet, Neumann or Mixed of the form
%  BC 1:  aBCo(t)*T(xo,t) + bBCo(t)*dTdx(xo,t) + cBCo(t) = 0;
%  BC 2:  aBCf(t)*T(xf,t) + bBCf(t)*dTdx(xf,t) + cBCf(t) = 0;
%
%  The necessary functions
%  pdefunk(x,t,T,dTdx,d2Tdx2), To(x),
%  aBCo(t), bBCo(t), cBCo(t), aBCf(t), bBCf(t), cBCf(t)
%  are entered at the bottom 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_1_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 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
%
BC(1) = 'D';
BC(2) = 'N';

% discretize time
to = 0;
tf = 5.0e+3;
dt = 1.0e-1;
tvec = [to:dt:tf];
nt = length(tvec);
```

```
% discretize space
xo = 0;
xf = 0.1;
dx = 5.0e-3;
% include imaginary boundary nodes
xvec = [xo-dx:dx:xf+dx];
nx = length(xvec);
dxi = 1.0/dx;

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


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

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

% apply Neumann/Mixed BCs as ICs at imaginary nodes
if (BC(1) ~= 'D')
   Tmat(1,i) =  2.0*dx/bBCo(t)*( aBCo(t)*Tmat(2,i)    +
bBCo(t)/(2.0*dx)*Tmat(3,i)    + cBCo(t));
end
if (BC(2) ~= 'D')
   Tmat(nx,i) = 2.0*dx/bBCf(t)*(-aBCf(t)*Tmat(nx-1,i) +
bBCf(t)/(2.0*dx)*Tmat(nx-2,i) - cBCf(t));
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
%
if (BC(1) == 'D')
   ivaro = 3;
else
   ivaro = 2;
end
if (BC(2) == 'D')
   ivarf = nx-2;
else
   ivarf = nx-1;
end
nvar = ivarf - ivaro + 1;

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

```
%
%  Prediction Step
%

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

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

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

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

%
%  Correction Step
%

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

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

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

%  apply BCs at the correction step
   if (BC(1) == 'D')
      Tnew(2) = -cBCo(t)/aBCo(t);
   else
```

```
      Tnew(1) =  2.0*dx/bBCo(t)*( aBCo(t)*Tnew(2)    +
bBCo(t)/(2.0*dx)*Tnew(3)    + cBCo(t));
   end
   if (BC(2) == 'D')
      Tnew(nx-1) = -cBCf(t)/aBCf(t);
   else
      Tnew(nx) = 2.0*dx/bBCf(t)*(-aBCf(t)*Tnew(nx-1) +
bBCf(t)/(2.0*dx)*Tnew(nx-2) - cBCf(t));
   end

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

% plot
figure(1);
nskip = 10;
for i = 1:nskip:nt
   plot(xvec(2:nx-1),Tmat(2:nx-1,i),'k-');
   %pause(1);
   hold on;
end
xlabel('position (m)')
ylabel('Temperature (K)');


%
%  functions defining PDE
%

function k = pdefunk(x,t,y,dydx,d2ydx2);
%
Temp = y;
% rho = density [kg/m^3]
rho = 8960.0;
% Cp = heat capacity [J/kg/K]
Cp = 384.6;
% k = thermal conductivity [W/m/K]
k = 401.0;
%  alpha = thermal diffusivity
alpha = k/rho/Cp;
% length of rod [m]
L = 0.1;
% diameter in [m]
radius = 0.1;
diameter = 2.0*radius;
% surface Area in [m^2]
Area = pi*diameter*L;
% Volume in [m^3]
Volume = pi/4*diameter^2*L;
% surface area to volume ratio
S = Area/Volume;
%  Temperature of the surroundings [K]
Tsurround = 300.0;
% Stefan-Boltzmann constant [J/s/m^2/K^4]
sigma = 5.670373e-8;
% gray body permittivity [dimensionless]
eps = 0.15;
fac = eps*sigma*S/(rho*Cp);
```

```
k = alpha*d2ydx2 - fac*(Temp^4 - Tsurround^4);


%
%  function defining initial condition
%

function ic = icfunk(x);
ic = 1000;

%
%  functions defining LHS boundary condition
%

function f = aBCo(t);
f = 1;

function f = bBCo(t);
f = 0;

function f = cBCo(t);
f = -1000;

%
%  functions defining RHS boundary condition
%

function f = aBCf(t);
f = 0;

function f = bBCf(t);
f = 1;

function f = cBCf(t);
f = 0;
```

Appendix II. parapde_1_anyBC_flow.m

```
function [xvec,tvec,Tmat] = parapde_1_anyBC_flow
%
%  The routine parapde_1_anyBC will solve
%  a single non-linear 1-D parabolic PDE of the form
%
%  dT/dt = f(x,t,T,dTdx,d2Tdx2)
%
%  where the convection term dominates,
%  using a second order method.
%
%  The initial condition must have the form
%  IC:  T(x,to) = To(x);
%
%  The boundary conditions can be
%  Dirichlet, Neumann or Mixed of the form
%  BC 1:  aBCo(t)*T(xo,t) + bBCo(t)*dTdx(xo,t) + cBCo(t) = 0;
%  BC 2:  aBCf(t)*T(xf,t) + bBCf(t)*dTdx(xf,t) + cBCf(t) = 0;
%
%  The necessary functions
%  pdefunk(x,t,T,dTdx,d2Tdx2), To(x),
%  aBCo(t), bBCo(t), cBCo(t), aBCf(t), bBCf(t), cBCf(t)
%  are entered at the bottom 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_1_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 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
%
BC(1) = 'D';
BC(2) = 'N';

% discretize time
to = 0;
tf = 2.0e+3;
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);


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

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

% apply Neumann/Mixed BCs as ICs at imaginary nodes
if (BC(1) ~= 'D')
   %Tmat(1,i) =  2.0*dx/bBCo(t)*( aBCo(t)*Tmat(2,i)    +
bBCo(t)/(2.0*dx)*Tmat(3,i)    + cBCo(t));
   Tmat(1,i) =       dx/bBCo(t)*( aBCo(t)*Tmat(2,i)    + bBCo(t)/(
dx)*Tmat(2,i)    + cBCo(t));
end
if (BC(2) ~= 'D')
   %Tmat(nx,i) = 2.0*dx/bBCf(t)*(-aBCf(t)*Tmat(nx-1,i) +
bBCf(t)/(2.0*dx)*Tmat(nx-2,i) - cBCf(t));
    Tmat(nx,i) =     dx/bBCf(t)*(-aBCf(t)*Tmat(nx-1,i) + bBCf(t)/(
dx)*Tmat(nx-1,i) - cBCf(t));
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
%
if (BC(1) == 'D')
   ivaro = 3;
else
   ivaro = 2;
end
if (BC(2) == 'D')
   ivarf = nx-2;
else
   ivarf = nx-1;
end
nvar = ivarf - ivaro + 1;
```

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

%
%  Prediction Step
%

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

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

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

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

%
%  Correction Step
%

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

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

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

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

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

% plot
figure(1);
nskip = 100;
for i = 1:nskip:nt
   plot(xvec(2:nx-1),Tmat(2:nx-1,i),'k-');
   pause(1/5);
   %hold on;
end
xlabel('position (m)')
ylabel('Temperature (K)');


%
%  functions defining PDE
%

function dydt = pdefunk(x,t,y,dydx,d2ydx2);
% molar concentrations [mol/m^3]
CA = y(1);
% velocity [m/s]
v = 0.1;
% diffusivity [m^2/s]
D = 1.0e-9;
%  rate constant [m^6/mol^2/s]
k = 1.0e-5;
```

```
% stoichiometric coefficients
nuA = -2.0;
% reaction rate [mol/m^3/s]
rate = k*CA^2;
dydt = -v*dydx + D*d2ydx2 + nuA*rate;

%
%  function defining initial condition
%

function ic = icfunk(x);
ic = 0;

%
%  functions defining LHS boundary condition
%

function f = aBCo(t);
f = 1;

function f = bBCo(t);
f = 0;

function f = cBCo(t);
f = -1000;

%
%  functions defining RHS boundary condition
%

function f = aBCf(t);
f = 0;


function f = bBCf(t);
f = 1;

function f = cBCf(t);
f = 0;
```