Numerical Methods for Solving a System of Nonlinear Parabolic PDEs with Two Spatial Dimensions

David Keffer
Department of Materials Science & Engineering
University of Tennessee, Knoxville
date begun: February 11, 2015
last revised: February 19, 2015

Table of Contents

I. Formulation

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

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

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 have already likewise converted 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 for a system with one spatial dimension. We now extend this process to a system with two spatial dimensions. There is little conceptual development required for this extentions. Rather, there is only the need for methodical bookkeeping.

A comment on notation: we will write $T^{(\ell)}(t_j, x_i, y_k)$ as ${T^{(\ell)}}^j_{i,k}$, where *j* superscripts designate temporal increments, *i* subscripts designate spatial increments along the first spatial dimension, *k* subscripts designate spatial increments along the second spatial dimension, and ℓ 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.

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

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

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

The braces in equation (3) stand for the complete set over both positions *i,k* and function (ℓ).

The second function in equation (2) is given by $K(x_i, y_k, t_{j+1}, T_i^j + \Delta t K_i^j)$

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

where the temperature is approximated with an Euler method

$$T^{j+1}_{i,k} \approx T^j_{i,k} + \Delta t K^j_{i,k} \tag{5}$$

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, an implementation of this algorithm is provided. The code, `parapde_n_anyBC_2d.m`, is a general code. This code is also available for download on the course website.

II. Heat Transfer in a Plate with Radiative Heat Loss

II.A. Dirichlet Boundary Conditions

Consider heat transfer in a square plate. Our intention is to describe the evolution of the temperature in this space subject to a set of initial conditions and boundary conditions. In this case, we focus exclusively on the energy balance, so we only have one partial differential equation. However, we intend to describe the evolution of the temperature in two spatial dimensions.

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

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

The radiative heat loss term involves temperature to the fourth power and is therefore nonlinear. Consider a flat Cu plate, square in shape with a side length of 1.0 m and a thickness of 0.01 m, which is initially at $T(x, y, t = 0) = 1000$ K. The boundary conditions for this plate are given as follows

$$T(x = x_o, y, t) = 800\,K$$
$$T(x = x_f, y, t) = 1000\,K$$
$$T(x, y = y_o, t) = 800\,K$$
$$T(x, y = y_f, t) = 1000\,K$$

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]
- spatial position along material x and y [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 $S = \ell^2 / \ell^2 w = 100$ [m^{-1}] (for the top surface of a square plate with side $\ell = 1$ m and thickness $w = 0.01$ m)

This is a single non-linear parabolic PDE with two spatial dimensions and four Dirichlet boundary conditions. To solve this problem, I will use the code `parapde_n_anyBC_2d.m`.

I modified the input functions in `parapde_n_anyBC_2d.m` as follows.

I set the number of PDEs to one.

```
neq = 1;
```

I assigned the appropriate type of boundary conditions.

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

I set the final time to 10,000 seconds and chose the time interval to be 5 seconds, so I had 2000 temporal intervals.

```
% discretize time
to = 0;
tf = 1.0e+4;
dt = 5.0e+0;
```

I defined the geometry and resolution of both spatial dimensions. The plate spans from 0 to 1 m. The resolution was 0.05 m, so I had 20 intervals in each dimension, or 400 area elements.

```
% discretize space
xo = 0;
xf = 1.0;
dx = 5.0e-2;

yo = 0;
yf = 1.0;
dy = 5.0e-2;
```

I defined the PDE in the following function.

```
%
%  function defining PDE
%
function dTdt_out = pdefunk(x,y,t,Told,dTdx,dTdy,d2Tdx2,d2Tdy2,d2Tdxdy,keq);
%
%  physical properties
%
% 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;
%  Stefan-Boltzmann constant [J/s/m^2/K^4]
sigma = 5.670373e-8;
% gray body permittivity
eps = 0.15; % (for dull Cu)
% plate geometry
side = 1.0; % [m]
thickness = 0.01; % [m]
```

```
area = side*side;
volume = area*thickness;
s = area/volume; % 1/m
% surrounding temperature [K]
Tsurround = 300.0;
% PDE
dTdt_out = alpha*(d2Tdx2(1)+d2Tdy2(1)) - eps*sigma*s/(rho*Cp)*(Told(1)^4 -
Tsurround^4);
```

I defined the IC and BCs in the functions below. Remember the initial condition provides the initial temperature. The code accepts boundary conditions in the form for an x boundary

$$a_{BC}(y,t)T + b_{BC}(y,t)\frac{\partial T}{\partial x} + c_{BC}(y,t) = 0$$

or analogously for a y boundary

$$a_{BC}(x,t)T + b_{BC}(x,t)\frac{\partial T}{\partial y} + c_{BC}(x,t) = 0$$

so three functions must be entered for each of the four boundaries. For Dirichlet BCs, $a_{BC} = 1$, $b_{BC} = 0$, and $c_{BC} = -T_{bound}$, where T_{bound} is the boundary temperature.

```
%
%  function defining initial condition
%

function ic_out = icfunk(x,y,keq);
ic(1) = 1000.0;
ic_out = ic(keq);

%
%  functions defining initial boundary condition in first spatial dimension
%

function fout = aBCxo(y,t,k);
f(1) = 1;
fout = f(k);

function fout = bBCxo(y,t,k);
f(1) = 0;
fout = f(k);

function fout = cBCxo(y,t,k);
f(1) = -800;
fout = f(k);

%
%  functions defining final boundary condition in first spatial dimension
%

function fout = aBCxf(y,t,k);
```

```
f(1) = 1;
fout = f(k);

function fout = bBCxf(y,t,k);
f(1) = 0;
fout = f(k);

function fout = cBCxf(y,t,k);
f(1) = -1000;
fout = f(k);

%
%  functions defining initial boundary condition in second spatial dimension
%

function fout = aBCyo(x,t,k);
f(1) = 1;
fout = f(k);

function fout = bBCyo(x,t,k);
f(1) = 0;
fout = f(k);

function fout = cBCyo(x,t,k);
f(1) = -800;
fout = f(k);

%
%  functions defining final boundary condition in second spatial dimension
%

function fout = aBCyf(x,t,k);
f(1) = 1;
fout = f(k);

function fout = bBCyf(x,t,k);
f(1) = 0;
fout = f(k);

function fout = cBCyf(x,t,k);
f(1) = -1000;
fout = f(k);
```

At the command line prompt, I typed

```
[xvec,yvec,tvec,Tmat] = parapde_n_anyBC_2d_example01;
```

This command generated a movie. Several frames of the movie are shown in Figure 1 (without radiative heat loss, i.e. I set eps = 0.0) and in Figure 2 (with radiative heat loss, i.e. I left eps = 0.15).

In any case, to find the temperature at any point in space and time, I simply need the three indices for the matrix, Tmat, which correspond to the x, y and z values of interest. For example, to find the value of the temperature at x = 0.5 m, y = 0.5 m and t = 10,000 s, I confirmed that I knew the correct spatial and temporal indices.

```
>> xvec(12)

ans =    0.5000

>> yvec(12)

ans =    0.5000

>> tvec(2001)

ans =       10000
```

With these indices, we find that

```
>> Tmat(12,12,2001)

ans =  900.0000
```

Therefore the temperature at x = 0.5 m, y = 0.5 m and t = 10,000 s, is 900 K for the case without radiative heat loss. This is the steady-state (infinite time) solution. From the movie, we would have seen almost no change in the profile after 4,000 s.

In the case where there was radiative heat loss, we find that

```
>> Tmat(12,12,201)

ans =  835.6018
```

Therefore the temperature at x = 0.5 m, y = 0.5 m and t = 10,000 s, is 836 K for the case with radiative heat loss. This is very near the steady-state (infinite time) solution.

If you were interested in the temporal evolution of a particular point in the plate, such information is contained in the solution matrix, Tmat.

For example, the following set of commands, yields the plot shown in Figure 3. We extract all the temperature of the central point at all times and store it in a temporary vector for ease in plotting.

```
>> temp(1:2001) = Tmat(12,12,1:2001);
>> plot(tvec,temp,'k-');
```

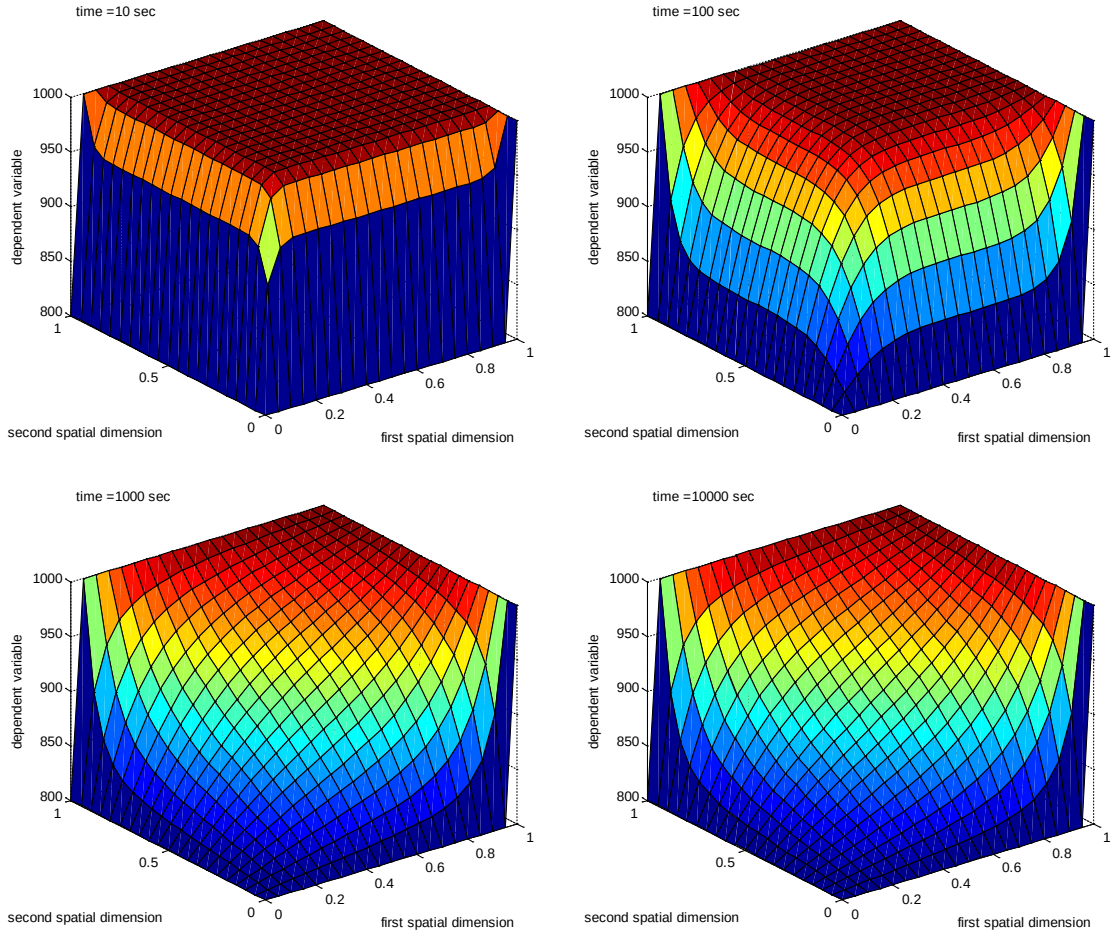

Figure 1. Transient behavior of the temperature in a plate without radiative heat loss.

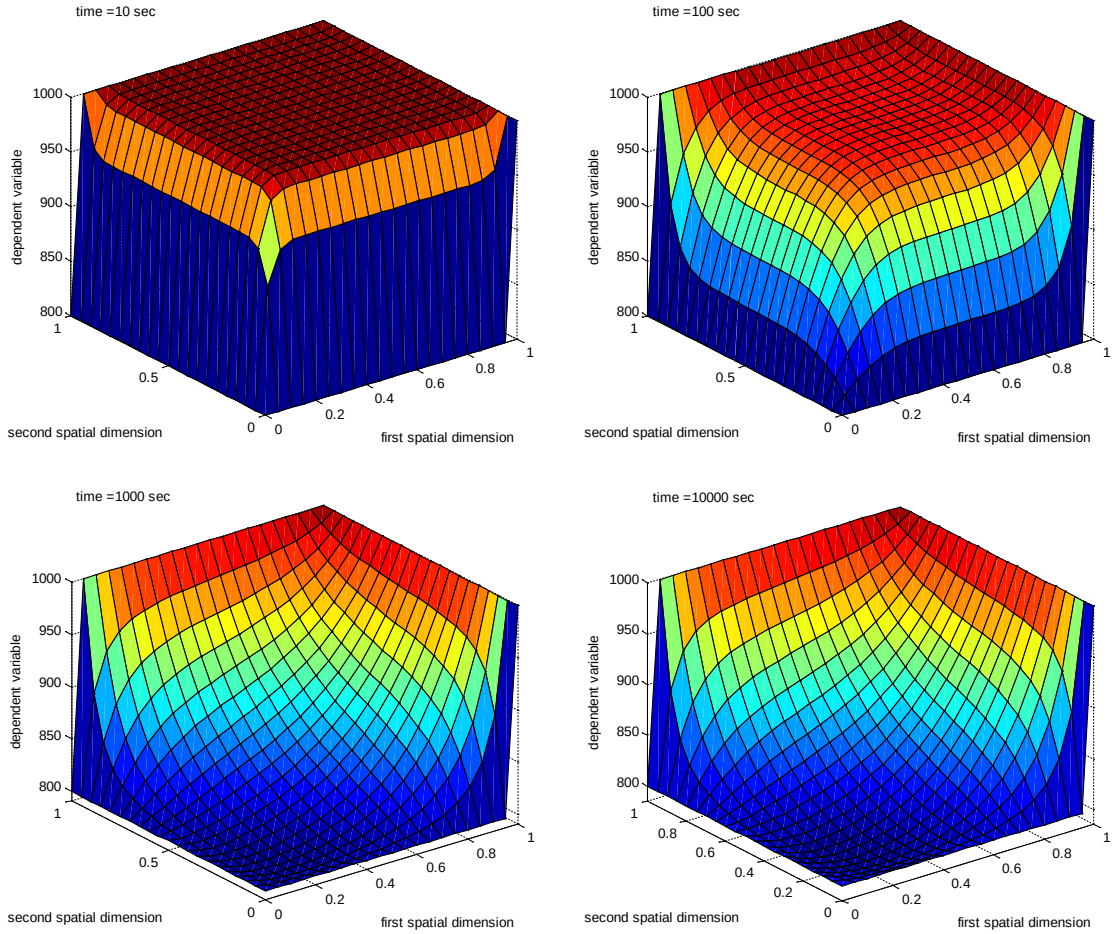

Figure 2. Transient behavior of the temperature in a plate with radiative heat loss.

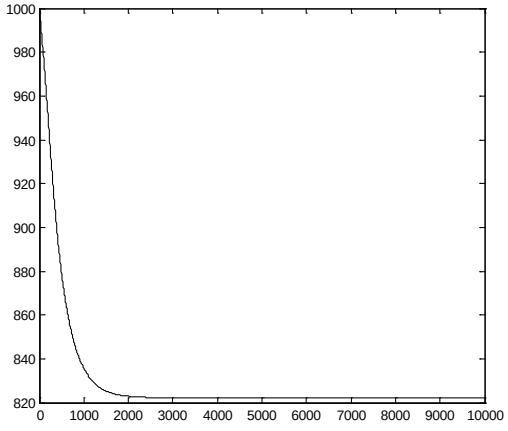

Figure 3. Transient behavior of the temperature at x = 0.5 m, y = 0.5 m with radiative heat loss. The y-axis is temperature (K) and the x-axis is time (s).

II.B. Dirichlet & Neumann Boundary Conditions

Let's rework the previous problem (including radiation loss) where two sides of the plate are insulated, resulting in no heat loss, a no flux boundary condition. The boundary conditions are

$$\left.\frac{\partial T}{\partial x}\right|_{x_o} = 0$$

$$T(x = x_f, y, t) = 1000\,K$$

$$\left.\frac{\partial T}{\partial y}\right|_{y_o} = 0$$

$$T(x, y = y_f, t) = 1000\,K$$

Given the required format for BC input, for Neumann BCs, $a_{BC} = 0$, $b_{BC} = 1$, and $c_{BC} = -\nabla T_{bound}$, where, in this case, the boundary gradient is zero.

The changes to the code involve the following steps.

I assigned the appropriate type of boundary conditions.

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

I accordingly changed two of the boundary conditions.

```
%
%  functions defining initial boundary condition in first spatial dimension
%

function fout = aBCxo(y,t,k);
f(1) = 0;
fout = f(k);

function fout = bBCxo(y,t,k);
f(1) = 1;
fout = f(k);

function fout = cBCxo(y,t,k);
f(1) = 0;
fout = f(k);

%
%  functions defining initial boundary condition in second spatial dimension
%

function fout = aBCyo(x,t,k);
f(1) = 0;
fout = f(k);
```

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

function fout = cBCyo(x,t,k);
f(1) = 0;
fout = f(k);
```

At the command line prompt, I typed

```
[xvec,yvec,tvec,Tmat] = parapde_n_anyBC_2d_example01;
```

This command generated a movie. Several frames of the movie are shown in Figure 4. The midpoint temperature at 10,000 s is 823 K.

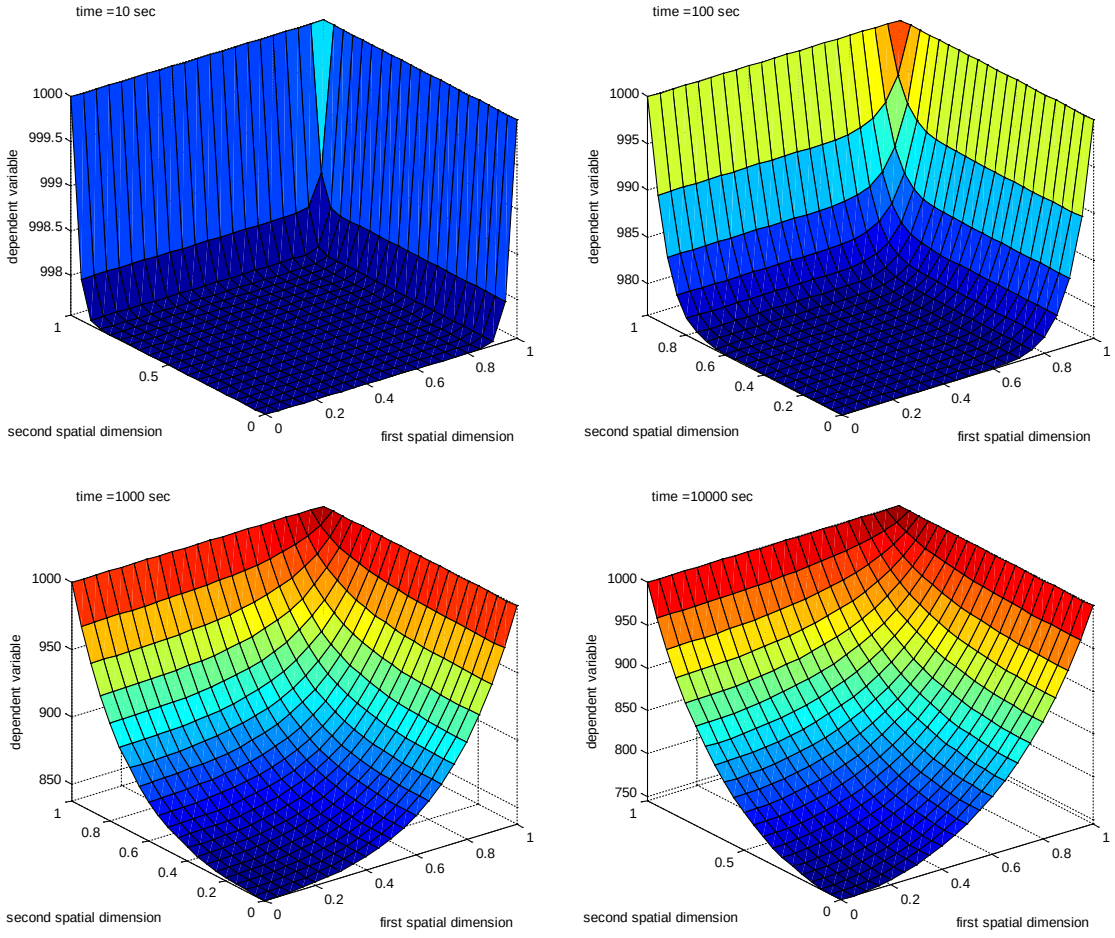

Figure 4. Transient behavior of the temperature in a plate with radiative heat loss and two insulated boundary conditions.

III. Heat Transfer in a Cylindrical Rod with Radiative and Convective Heat Loss

Let's solve the same heat transfer problem as above in a cylindrical rod of finite length. Again, our intention is to describe the evolution of the temperature in this object subject to a set of initial conditions and boundary conditions. In this case, we focus exclusively on the energy balance, so we only have one partial differential equation. However, we intend to describe the evolution of the temperature in two spatial dimensions, an axial and radial direction. We assume there is no variation in the angular dimension.

The two-dimensional heat equation can describe heat transfer in a material with both heat conduction and radiative heat loss. In cylindrical coordinates we have an additional term in the scalar Laplacian.

$$\frac{\partial T}{\partial \mathrm{t}} = \frac{k}{\rho C_p}\left(\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{\partial^2 T}{\partial z^2}\right) - \frac{\varepsilon \sigma S}{\rho C_p}\left(T^4 - T_s{}^4\right)$$

The radiative heat loss term involves temperature to the fourth power and is therefore nonlinear. Consider a cylindrical Cu rod, with an axial length of 1.0 m and a radius of 0.01 m, which is initially at $T(x, y, t = 0) = 1000$ K. The boundary conditions for this rod are given as follows. In the axial dimension, finite temperatures are maintained. In the radial direction, there is the standard symmetry boundary at the interior (r=0) boundary condition. At the radius of the cylinder, the rod is poorly insulated, resulting in convective heat loss.

$$T(z = z_o, r, t) = 800\,K$$
$$T(z = z_f, r, t) = 1000\,K$$
$$\left.\frac{\partial T}{\partial r}\right|_{r_o} = 0$$
$$q = -k\left.\frac{\partial T}{\partial r}\right|_{r_f} = h\left[T(z, r = r_f, t) - T_{surr}\right]$$

This last boundary condition indicates that the heat flux at any point on the surface of the rod is equal in magnitude to a convective heat loss to the surroundings. The variable h is an empirical heat transfer coefficient with units of W/m^2/K. Thus, this BC can be written in a form compatible with the code.

$$hT(z, r = r_f, t) + k\left.\frac{\partial T}{\partial r}\right|_{r_f} - hT_{surr} = 0$$

Remember the initial condition provides the initial temperature. The code accepts boundary conditions in the form for an x boundary

$$a_{BC}(y,t)T + b_{BC}(y,t)\frac{\partial T}{\partial x} + c_{BC}(y,t) = 0$$

So three functions must be entered for each of the four boundaries. For Dirichlet BCs, $a_{BC}=1$, $b_{BC}=0$, and $c_{BC}=-T_{bound}$, where T_{bound} is the boundary temperature. For Neumann no flux BCs, $a_{BC}=0$, $b_{BC}=1$, and $c_{BC}=-\nabla T_{bound}$. For this last boundary, $a_{BC}=h$, $b_{BC}=k$, and $c_{BC}=-hT_{surr}$.

I assigned the appropriate type of boundary conditions.

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

I set the final time to 10,000 seconds and chose the time interval to be 5 seconds, so I had 2000 temporal intervals. You will note that I used a much smaller timestep here, which will be due to the finer resolution in the radial dimension.

```
% discretize time
to = 0;
tf = 1.0e+3;
dt = 1.0e-1;
```

I defined the geometry and resolution of both spatial dimensions. The rod runs from 0 to 1 m. The resolution in the axial dimension was 0.05 m, so I had 20 intervals in the axial dimension. The rod has a radius of 0.1 m. The resolution in the radial direction was 0.01, so I had 10 intervals in the radial dimension, or 200 area elements.

Note: There is a trick here when solving in the radial dimension! You make your first radial position (yo) set to the discretization size (dy) rather than 0. This allows the imaginary node required for the Neumann boundary condition to exist at r = 0.

```
% discretize space
xo = 0;
xf = 1.0;
dx = 5.0e-2;

dy = 1.0e-2;
yo = dy;
yf = 0.1;
```

I defined the PDE in the following function.

```
function dTdt_out = pdefunk(z,r,t,Told,dTdz,dTdr,d2Tdz2,d2Tdr2,d2Tdzdr,keq);
%
%  physical properties
%
% 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;
%  Stefan-Boltzmann constant [J/s/m^2/K^4]
sigma = 5.670373e-8;
% gray body permittivity
eps = 0.15; % (for dull Cu)
%eps = 0.0; % no radiative loss
% plate geometry
length = 1.0; % [m]
radius = 0.1; % [m]
pi = 2.0*asin(1.0);
area = 2*pi*radius*length;
volume = pi*radius*radius*length;
s = area/volume; % 1/m
% surrounding temperature [K]
Tsurround = 300.0;
%
% PDE
%
dTdt_out = alpha*(d2Tdz2(1) + 1.0/r*dTdr(1) + d2Tdr2(1)) -
eps*sigma*s/(rho*Cp)*(Told(1)^4 - Tsurround^4);
```

I defined the IC and BCs in the functions below.

```
%
%  function defining initial condition
%

function ic_out = icfunk(x,y,keq);
ic(1) = 1000.0;
ic_out = ic(keq);

%
%  functions defining initial boundary condition in first spatial dimension
%

function fout = aBCxo(y,t,k);
f(1) = 1;
fout = f(k);

function fout = bBCxo(y,t,k);
f(1) = 0;
fout = f(k);

function fout = cBCxo(y,t,k);
f(1) = -800;
fout = f(k);

%
%  functions defining final boundary condition in first spatial dimension
%

function fout = aBCxf(y,t,k);
f(1) = 1;
fout = f(k);

function fout = bBCxf(y,t,k);
```

```
f(1) = 0;
fout = f(k);

function fout = cBCxf(y,t,k);
f(1) = -1000;
fout = f(k);

%
%  functions defining initial boundary condition in second spatial dimension
%

function fout = aBCyo(x,t,k);
f(1) = 0;
fout = f(k);

function fout = bBCyo(x,t,k);
f(1) = 1;
fout = f(k);

function fout = cBCyo(x,t,k);
f(1) = 0;
fout = f(k);

%
%  functions defining final boundary condition in second spatial dimension
%

function fout = aBCyf(x,t,k);
%  heat transfer coefficient in [W/m^2/K]
h = 4.0e+1;
f(1) = h;
fout = f(k);

function fout = bBCyf(x,t,k);
% kc = thermal conductivity [W/m/K]
kc = 401.0;
f(1) = kc;
fout = f(k);

function fout = cBCyf(x,t,k);
%  heat transfer coefficient in [W/m^2/K]
h = 4.0e+1;
% surrounding temperature [K]
Tsurround = 300.0;
f(1) = -h*Tsurround;
fout = f(k);
```

At the command line prompt, I typed

```
[xvec,yvec,tvec,Tmat] = parapde_n_anyBC_2d_cyl;
```

This command generated a movie. Several frames of the movie are shown in Figure 5. Initially the rod is entirely at 1000 K. It cools axially first, with the temperature of one end of the rod dropping down to 800 K. It also cools due to radial heat loss to the surroundings, which is why we see that eventually the temperature in the rod can drop below both the axial boundary temperatures.

The temporal evolution of the temperature at several points is shown in Figure 6. The two figures in Figure 6 were generated with the following script, after the pde was solved.

```
figure(2);
temp(1:10001) = Tmat(12,1,1:10001);
plot(tvec,temp,'k-');
hold on;
temp(1:10001) = Tmat(12,11,1:10001);
plot(tvec,temp,'r-');
hold off;
xlabel('time (s)')
ylabel('temperature (K)');
legend('z=0.5, r=0.0','z=0.5, r=0.1');

figure(3);
temp(1:10001) = Tmat(3,6,1:10001);
plot(tvec,temp,'k-');
hold on;
temp(1:10001) = Tmat(21,6,1:10001);
plot(tvec,temp,'r-');
hold off;
xlabel('time (s)')
ylabel('temperature (K)');
legend('z=0.05, r=0.05','z=0.95, r=0.05');
```

The figure on the left in Figure 6 shows that there is only a small difference between the temperature evolution at the interior of the cylinder and at its difference. The figure on the left shows that there is a large difference between the temperature near different ends of the rod. When there is a large disparity in dimensions, people often assume that there is no significant temperature gradient in the radial direction and then solve the problem with only one spatial dimension! This is the ultimate resolution to dealing with problems of having to discretize different dimensions to different degrees.

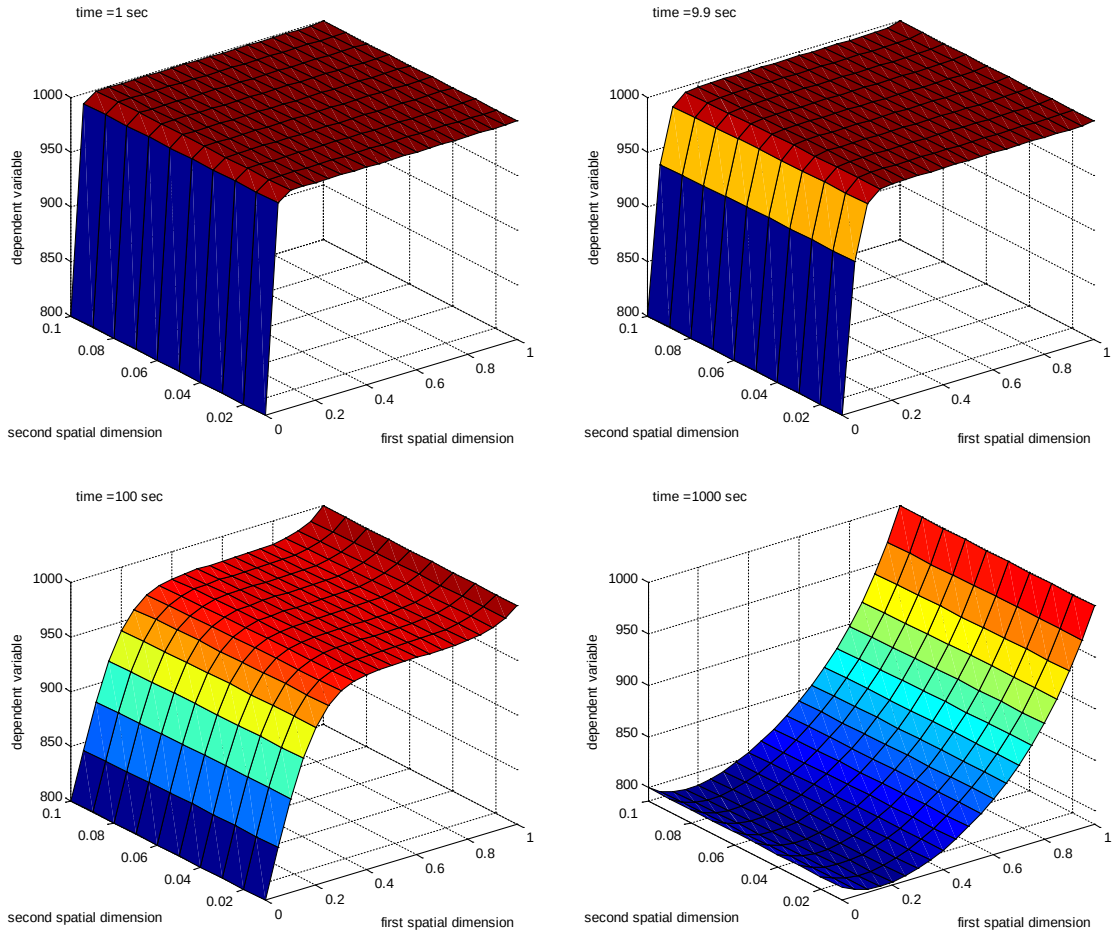

Figure 5. Transient behavior of the temperature in a rod with radiative heat loss.

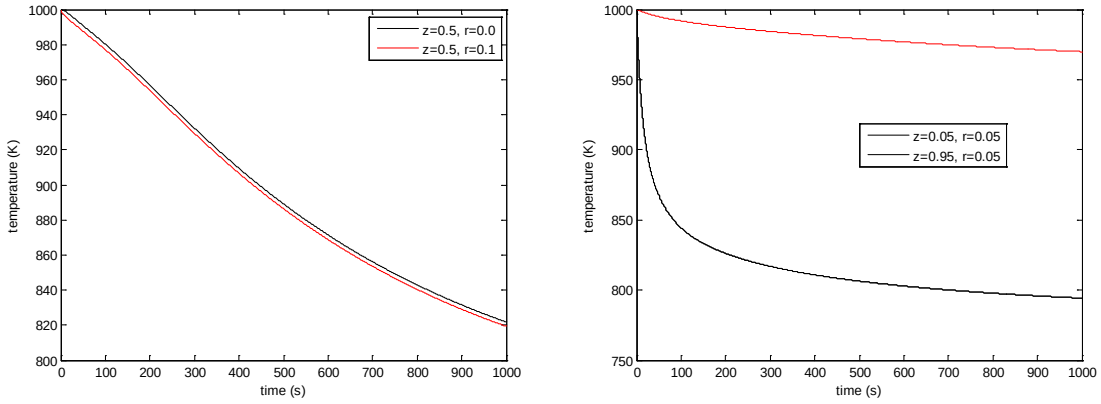

Figure 6. Transient behavior of the temperature at selected points in the rod with radiative heat loss from the interior and convective heat loss at the external radial boundary. The y-axis is temperature (K) and the x-axis is time (s). The first figure shows that there is only a small difference between the temperature evolution at the interior of the cylinder and at its difference. The second figure shows that there is a large difference between the temperature near different ends of the rod.

IV. Simultaneous Solution of a System of parabolic PDEs in two spatial dimensions

In the examples that we looked at above, there was only one PDE. Often we have more than one PDE. It could be a PDE corresponding to material balances for each component and to the energy balance. Here, we will look at a "two temperature model", which describes ionic bombardment of a material, in which there are different temperatures for the electrons and the nuclei of atoms, resulting in two energy balances, one yielding the temperature of the electrons and the other the temperature of the nuclei. We will examine this phenomenon in the two dimensional plate geometry.

The two-dimensional heat equation can describe heat transfer in a material with both heat conduction and radiative heat loss. The first equation provides a description of the electron energy. There are the traditional accumulation and diffusion terms. There is an electron-phonon coupling factor, g, which transfers energy from the electrons to the phonons (nuclei). There is also energy input to the electrons from the ion beam, *A*.

$$\frac{\partial T_e}{\partial \mathrm{t}} = \alpha_e\left(\frac{\partial^2 T_e}{\partial x^2} + \frac{\partial^2 T_e}{\partial y^2}\right) - g\left(T_e - T_n\right) + A$$

The energy balance for the nuclear temperature has a similar form. The sign of the electron-phonon coupling is reversed, since energy lost by the electrons is gained by the neutrons. Also, in this example, the incoming beam is at an energy such that it only interacts with electrons.

$$\frac{\partial T_n}{\partial \mathrm{t}} = \alpha_e\left(\frac{\partial^2 T_n}{\partial x^2} + \frac{\partial^2 T_n}{\partial y^2}\right) + g\left(T_e - T_n\right)$$

These equations happen to be linear, but the tools that we are using, are suitable for both linear and nonlinear PDEs, so this does not present a problem. Consider a flat Cu plate, square in shape with a side length of 10.0 nm, which is initially at $T_e(x, y, t = 0) = 300$ K and $T_n(x, y, t = 0) = 300$. The boundary conditions for this plate are given as follows

$$T_e(x = x_o, y, t) = 300\,K \qquad T_n(x = x_o, y, t) = 300\,K$$
$$T_e(x = x_f, y, t) = 300\,K \qquad T_n(x = x_f, y, t) = 300\,K$$
$$T_e(x, y = y_o, t) = 300\,K \qquad T_n(x, y = y_o, t) = 300\,K$$
$$T_e(x, y = y_f, t) = 300\,K \qquad T_n(x, y = y_f, t) = 300\,K$$

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

- temperature in the material *T* [K]
- spatial position along material *x* and *y* [nm]
- 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)
- nuclear thermal mobility $\alpha_n = \frac{k}{\rho C_p} \cdot 10^{-5}$ [nm^2/fs]
- electron mobility $\alpha_e = 10\alpha_n$ [nm^2/fs]
- electron-phonon coupling factor $g = 0.001$ [1/fs]
- energy input due to beam $A = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{r^2}{2\sigma^2}\right)$ [aJ/fs/nm^3]

where *A* is a Gaussian distribution of energy, centered at $(x, y) = (5,5)$ (the middle of the plate) and where *r* is the distance from the center and σ is the standard deviation of the distribution, set to 1 nm. The ion beam is a pulse that lasts only the first 200.0 fs. Note the nuclear thermal mobility has been artificially slowed down by five orders of magnitude for the purposes of the example problem.

I modified the file as follows. First, I set the number of equations to 2.

```
neq = 2;
```

I set all the boundary conditions to Dirichlet.

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

I discretized time and space.

```
% discretize time
to = 0;
tf = 1.0e+3;
dt = 1.0e+1;

% discretize the first spatial dimension
xo = 0;
xf = 10.0;
dx = 5.0e-1;

% discretize the second spatial dimension
yo = 0;
yf = 10.0;
dy = 5.0e-1;
```

I entered the PDEs.

```
function dTdt_out = pdefunk(x,y,t,Told,dTdx,dTdy,d2Tdx2,d2Tdy2,d2Tdxdy,keq);
% 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 [m^2/s]
alpha_n = k/rho/Cp;
alpha_n = alpha_n*1.0e+0; % [nm^2/fs]
% electron thermal diffusivity
alpha_e = 10.0*alpha_n; % [nm^2/fs]
g = 1.0e-3;
xc = 5.0;
yc = 5.0;
rcut = 2.0;
rcut2 = rcut*rcut;
tcut = 200.0;
r2 = (x-xc)^2 + (y-yc)^2;
pi = 2.0*asin(1.0);
sigma = 1.0;
sigma2 = sigma*sigma;
if (t <= tcut)
   if (r2 <= rcut2)
      r = sqrt(r2);
      A = 1.0/(sqrt(2.0*pi)*sigma)*exp(-r2/(2.0*sigma2));
   else
      A = 0.0;
   end
else
    A = 0.0;
end
dTdt(1) = alpha_e*d2Tdx2(1) + alpha_e*d2Tdy2(1) - g*(Told(1) - Told(2)) + A;
dTdt(2) = alpha_n*d2Tdx2(2) + alpha_n*d2Tdy2(2) + g*(Told(1) - Told(2));
dTdt_out = dTdt(keq);
```

I entered the initial conditions.

```
function ic_out = icfunk(x,y,keq);
ic(1) = 300.0;
ic(2) = 300.0;
ic_out = ic(keq);
```

I entered theboundary conditions. Only the boundary conditions for each PDE at xo are shown below, but the other six BCs are exactly analogous.

```
function fout = aBCxo(y,t,k);
f(1) = 1;
f(2) = 1;
fout = f(k);

function fout = bBCxo(y,t,k);
f(1) = 0;
f(2) = 0;
fout = f(k);

function fout = cBCxo(y,t,k);
f(1) = -300;
f(2) = -300;
fout = f(k);
```

At the command line prompt, I typed.

```
>> [xvec,yvec,tvec,Tmat] = parapde_n_anyBC_2d_example03;
```

This command generated two movies, one for each dependent variable. Several frames of the movies are shown in Figure 7. The electronic temperature is shown on the left and the nuclear temperature on the right. A peak in the electronic temperature grows for the first 200 fs, as energy is input into the system. The peak in the electronic temperature then falls as energy diffuses spatially away and is transferred to the nuclear degrees of freedom. The nuclear temperature, always much lower than the electronic temperature, rises for approximately 900 fs then gradually falls. Note that the scale of the temperature axis is different for the electronic and nuclear temperatures and is also different from frame to frame.

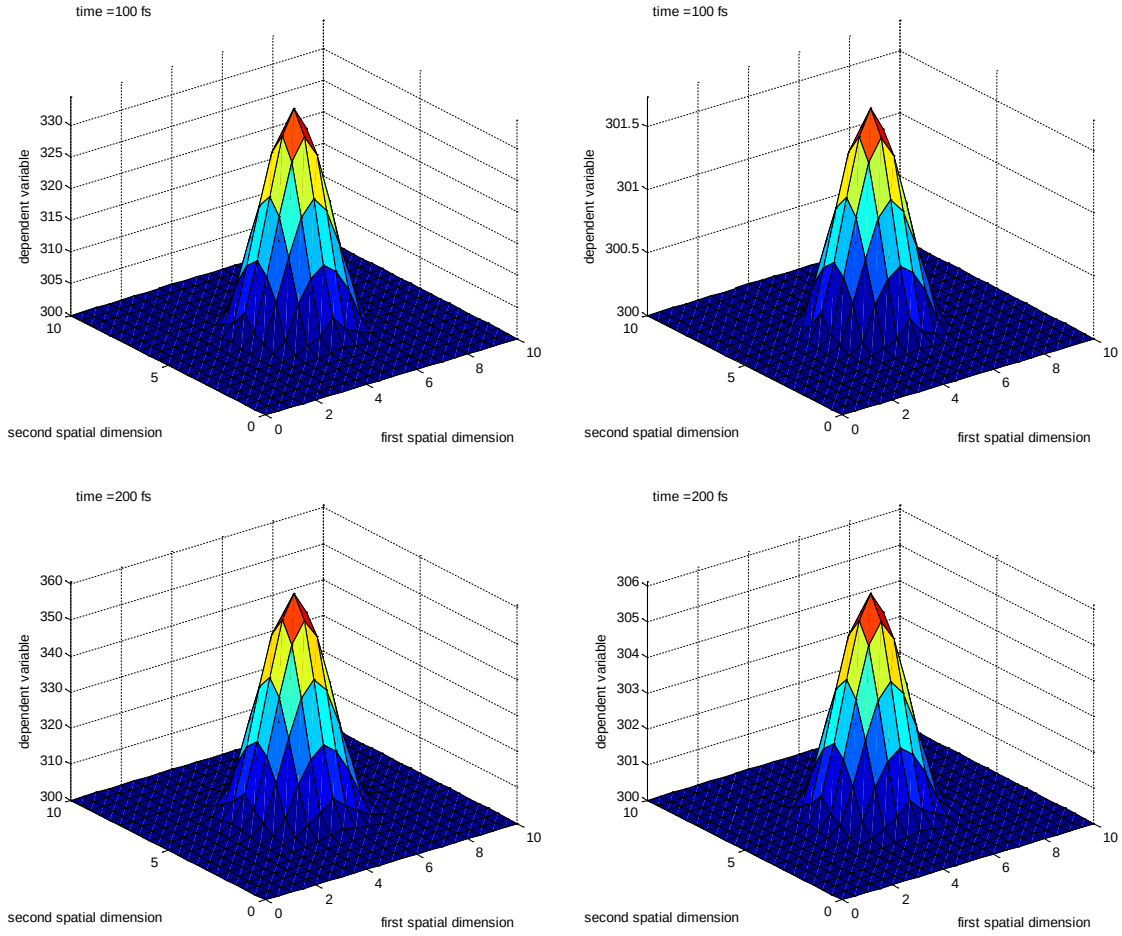

Figure 7. (Frames at 100 and 200 fs.) Description of transient behavior of the electronic temperature (left) and nuclear temperature (right) in a plate for a single pulse of ion beam lasting 200 fs.

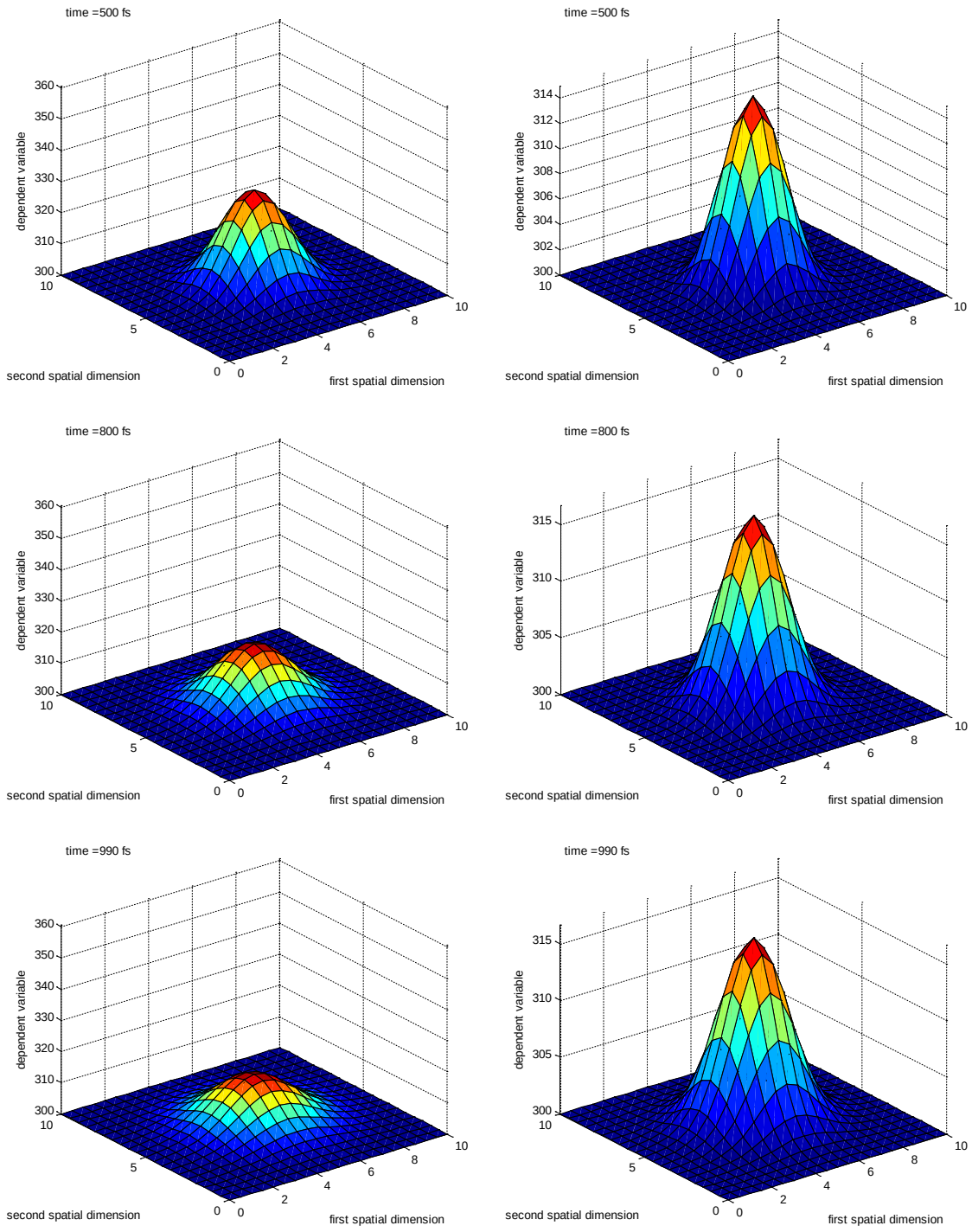

Figure 7. (continued) (Frames at 500, 800 and 1000 fs.) Description of transient behavior of the electronic temperature (left) and nuclear temperature (right) in a plate for a single pulse of ion beam lasting 200 fs.

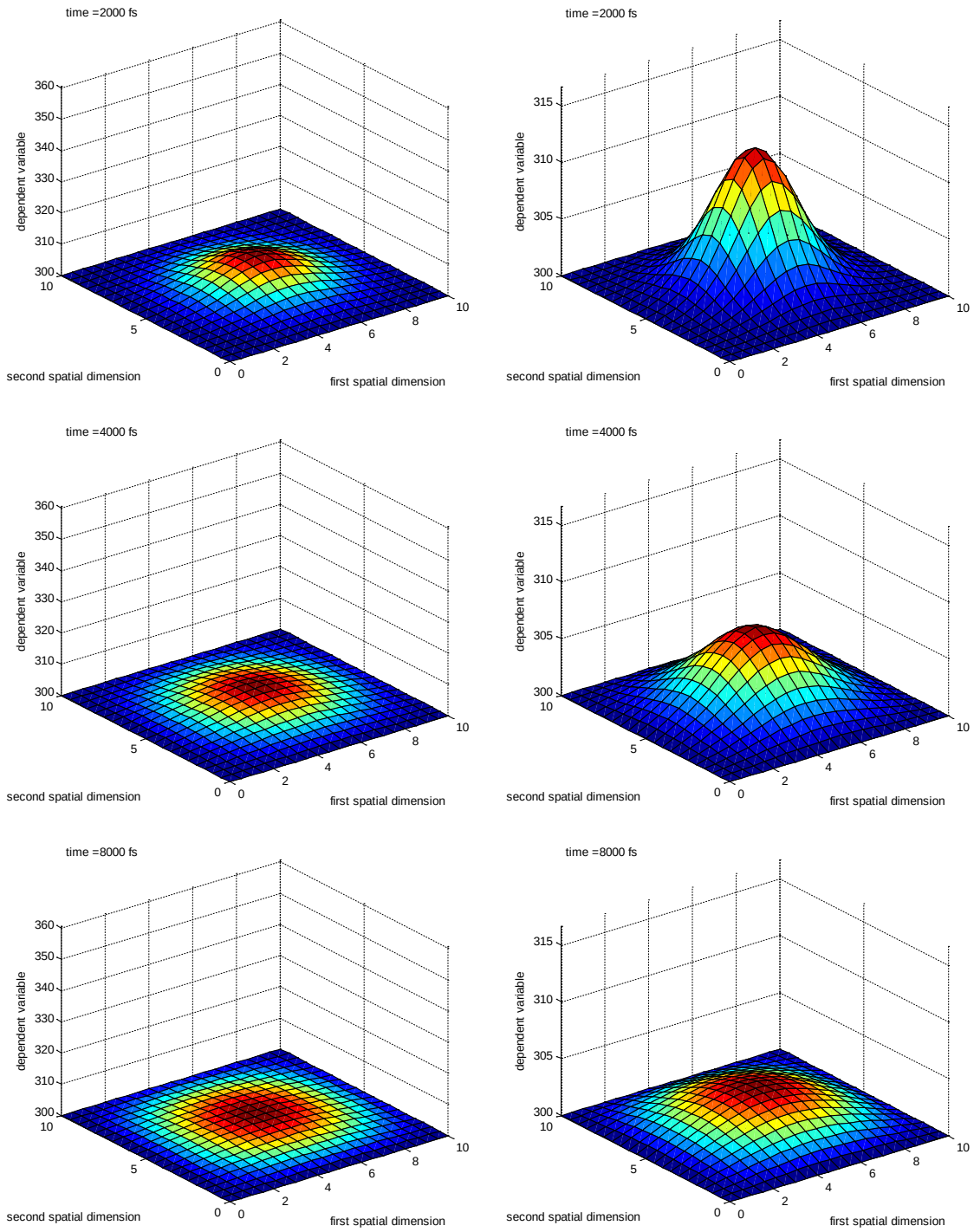

Figure 7. (continued) (Frames at 2000, 4000 and 8000 fs.) Description of transient behavior of the electronic temperature (left) and nuclear temperature (right) in a plate for a single pulse of ion beam lasting 200 fs.

Appendix I. parapde_n_anyBC_2d.m

```
function [xvec,yvec,tvec,Tmat] = parapde_n_anyBC_2d
%
%  The routine parapde_n_anyBC will solve
%  a system of non-linear 2-D parabolic PDEs of the form
%
%  dT/dt = f(x,y,t,T,dTdx,dTdy,d2Tdx2,d2Ydy2,d2Tdxdy)
%
%  using a second order method
%
%  The initial conditions must have the form
%  IC:  T(x,y,to,k_eq) = To(x,y,k_eq);
%
%  The boundary conditions can be
%  Dirichlet, Neumann or Mixed of the form
%  x BC 1:  aBCxo(y,t,k)*T(xo,y,t) + bBCxo(y,t,k)*dTdx(xo,y,t) + cBCxo(y,t,k) = 0;
%  x BC 2:  aBCxf(y,t,k)*T(xf,y,t) + bBCxf(y,t,k)*dTdx(xf,y,t) + cBCxf(y,t,k) = 0;
%  y BC 3:  aBCyo(x,t,k)*T(x,yo,t) + bBCyo(x,t,k)*dTdx(x,yo,t) + cBCyo(x,t,k) = 0;
%  y BC 4:  aBCyf(x,t,k)*T(x,yf,t) + bBCyf(x,t,k)*dTdx(x,yf,t) + cBCyf(x,t,k) = 0;
%
%  The necessary functions:
%  pdefunkf(x,y,t,T,dTdx,dTdy,d2Tdx2,d2Ydy2,d2Tdxdy),
%  To(x,y,k),
%  aBCxo(y,t,k), bBCxo(y,t,k), cBCxo(y,t,k), aBCxf(y,t,k), bBCxf(y,t,k), cBCxf(y,t,k)
%  aBCyo(x,t,k), bBCyo(x,t,k), cBCyo(x,t,k), aBCyf(x,t,k), bBCyf(x,t,k), cBCyf(x,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 initial value, final value and discretization in space, y,
%  yo, yf, and dy 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,yvec,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 = 2;

%
%  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 dimension is the first index 1 = x, 2 = y
%  The boundary is the second index  1 = o, 2 = f
%  The equation is the third index.  from 1 no neq
BC(1,1,1) = 'D';
BC(1,2,1) = 'D';
BC(2,1,1) = 'D';
BC(2,2,1) = 'D';
BC(1,1,2) = 'N';
BC(1,2,2) = 'D';
BC(2,1,2) = 'N';
BC(2,2,2) = 'D';

% discretize time
to = 0;
tf = 4.0e+3;
dt = 1.0e+1;
tvec = [to:dt:tf];
nt = length(tvec);

% discretize the first spatial dimension
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;

% discretize the second spatial dimension
yo = 0;
yf = 1.0;
dy = 1.0e-1;
% include imaginary boundary nodes
yvec = [yo-dy:dy:yf+dy];
ny = length(yvec);
dyi = 1.0/dy;

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


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

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



%
% apply Neumann/Mixed BCs as ICs at imaginary nodes
%
for k = 1:1:neq
%  along initial boundary of first spatial dimension
   if (BC(1,1,k) ~= 'D')
      for m = 2:1:ny-1
         y = yvec(m);
```

```
         Tmat(1,m,i,k)  = 2.0*dx/bBCxo(y,t,k)*( aBCxo(y,t,k)*Tmat(2,m,i,k)    +
bBCxo(y,t,k)/(2.0*dx)*Tmat(3,m,i,k)    + cBCxo(y,t,k));
      end
   end
%  along second boundary of first spatial dimension
   if (BC(1,2,k) ~= 'D')
      for m = 2:1:ny-1
         y = yvec(m);
         Tmat(nx,m,i,k) = 2.0*dx/bBCxf(y,t,k)*(-aBCxf(y,t,k)*Tmat(nx-1,m,i,k) +
bBCxf(y,t,k)/(2.0*dx)*Tmat(nx-2,m,i,k) - cBCxf(y,t,k));
      end
   end
%  along initial boundary of second spatial dimension
   if (BC(2,1,k) ~= 'D')
      for j = 2:1:nx-1
         x = xvec(j);
         Tmat(j,1,i,k)  = 2.0*dy/bBCyo(x,t,k)*( aBCyo(x,t,k)*Tmat(j,2,i,k)    +
bBCyo(x,t,k)/(2.0*dy)*Tmat(j,3,i,k)    + cBCyo(x,t,k));
      end
   end
%  along second boundary of second spatial dimension
   if (BC(2,2,k) ~= 'D')
      for j = 2:1:nx-1
         x = xvec(j);
         Tmat(j,ny,i,k) = 2.0*dy/bBCyf(x,t,k)*(-aBCyf(x,t,k)*Tmat(j,ny-1,i,k) +
bBCyf(x,t,k)/(2.0*dy)*Tmat(j,ny-2,i,k) - cBCyf(x,t,k));
      end
   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
%
%  first index of these three variables is spatial dimension
%  second index is equation
ivaro = zeros(2,neq);
ivarf = zeros(2,neq);
nvar = zeros(2,neq);
for k = 1:1:neq
%  initial boundary of first spatial dimension
   if (BC(1,1,k) == 'D')
      ivaro(1,k) = 3;
```

```
   else
      ivaro(1,k) = 2;
   end
%  final boundary of first spatial dimension
   if (BC(1,2,k) == 'D')
      ivarf(1,k) = nx-2;
   else
      ivarf(1,k) = nx-1;
   end
%  number of nodes along first spatial dimension
   nvar(1,k) = ivarf(1,k) - ivaro(1,k) + 1;
%  initial boundary of second spatial dimension
   if (BC(2,1,k) == 'D')
      ivaro(2,k) = 3;
   else
      ivaro(2,k) = 2;
   end
%  final boundary of second spatial dimension
   if (BC(2,2,k) == 'D')
      ivarf(2,k) = ny-2;
   else
      ivarf(2,k) = ny-1;
   end
%  number of nodes along second spatial dimension
   nvar(2,k) = ivarf(2,k) - ivaro(2,k) + 1;
end


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

%
%  Prediction Step
%

% allocate memory for spatial derivatives
dTdx = zeros(nx,ny,neq);
dTdy = zeros(nx,ny,neq);
d2Tdx2 = zeros(nx,ny,neq);
d2Tdy2 = zeros(nx,ny,neq);
```

```
d2Tdxdy = zeros(nx,ny,neq);

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            dTdx(j,m,k) = 0.5*( Told(j+1,m,k) - Told(j-1,m,k) )*dxi;
            dTdy(j,m,k) = 0.5*( Told(j,m+1,k) - Told(j,m-1,k) )*dyi;
            d2Tdx2(j,m,k) = ( Told(j+1,m,k) - 2.0*Told(j,m,k) + Told(j-1,m,k) )*dxi^2;
            d2Tdy2(j,m,k) = ( Told(j,m+1,k) - 2.0*Told(j,m,k) + Told(j,m-1,k) )*dyi^2;
            d2Tdxdy(j,m,k) = ( Told(j+1,m+1,k) - Told(j+1,m-1,k) - Told(j-1,m+1,k) + Told(j-1,m-1,k)
)*dxi*dyi*0.25;
         end
      end
   end

k1 = zeros(nx,ny,neq);
% estimate slope  at beginning of temporal interval
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            k1(j,m,k) =
pdefunk(xvec(j),yvec(m),tvec(i),Told(j,m,1:neq),dTdx(j,m,1:neq),dTdy(j,m,1:neq),d2Tdx2(j,m,1:neq),d2Tdy2(j,m
,1:neq),d2Tdxdy(j,m,1:neq),k);
         end
      end
   end

% apply Euler method for the prediction step
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            Ttem(j,m,k) = Told(j,m,k) + dt*k1(j,m,k);
         end
      end
   end

%  apply BCs at the prediction step
   for k = 1:1:neq
%  along initial boundary of first spatial dimension
      if (BC(1,1,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
```

```
            Ttem(2,m,k) = -cBCxo(y,t,k)/aBCxo(y,t,k);
         end
      else
         for m = 2:1:ny-1
            y = yvec(m);
            Ttem(1,m,k)  = 2.0*dx/bBCxo(y,t,k)*( aBCxo(y,t,k)*Ttem(2,m,k)    +
bBCxo(y,t,k)/(2.0*dx)*Ttem(3,m,k)    + cBCxo(y,t,k));
         end
      end
%  along second boundary of first spatial dimension
      if (BC(1,2,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
            Ttem(nx-1,m,k) = -cBCxf(y,t,k)/aBCxf(y,t,k);
         end
      else
         for m = 2:1:ny-1
            y = yvec(m);
            Ttem(nx,m,k) = 2.0*dx/bBCxf(y,t,k)*(-aBCxf(y,t,k)*Ttem(nx-1,m,k) +
bBCxf(y,t,k)/(2.0*dx)*Ttem(nx-2,m,k) - cBCxf(y,t,k));
         end
      end
%  along initial boundary of second spatial dimension
     if (BC(2,1,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
            Ttem(j,2,k) = -cBCyo(x,t,k)/aBCyo(x,t,k);
         end
     else
         for j = 2:1:nx-1
            x = xvec(j);
            Ttem(j,1,k)  = 2.0*dy/bBCyo(x,t,k)*( aBCyo(x,t,k)*Ttem(j,2,k)    +
bBCyo(x,t,k)/(2.0*dy)*Ttem(j,3,k)    + cBCyo(x,t,k));
         end
      end
%  along second boundary of second spatial dimension
      if (BC(2,2,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
            Ttem(j,ny-1,k) = -cBCyf(x,t,k)/aBCyf(x,t,k);
         end
     else
         for j = 2:1:nx-1
            x = xvec(j);
```

```
            Ttem(j,ny,k) = 2.0*dy/bBCyf(x,t,k)*(-aBCyf(x,t,k)*Ttem(j,ny-1,k) +
bBCyf(x,t,k)/(2.0*dy)*Ttem(j,ny-2,k) - cBCyf(x,t,k));
         end
      end
   end

%
%  Correction Step
%

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            dTdx(j,m,k) = 0.5*( Ttem(j+1,m,k) - Ttem(j-1,m,k) )*dxi;
            dTdy(j,m,k) = 0.5*( Ttem(j,m+1,k) - Ttem(j,m-1,k) )*dyi;
            d2Tdx2(j,m,k) = ( Ttem(j+1,m,k) - 2.0*Ttem(j,m,k) + Ttem(j-1,m,k) )*dxi^2;
            d2Tdy2(j,m,k) = ( Ttem(j,m+1,k) - 2.0*Ttem(j,m,k) + Ttem(j,m-1,k) )*dyi^2;
            d2Tdxdy(j,m,k) = ( Ttem(j+1,m+1,k) - Ttem(j+1,m-1,k) - Ttem(j-1,m+1,k) + Ttem(j-1,m-1,k)
)*dxi*dyi*0.25;
         end
      end
   end

k2 = zeros(nx,ny,neq);
% estimate slope  at end of temporal interval
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            k2(j,m,k) =
pdefunk(xvec(j),yvec(m),tvec(i),Ttem(j,m,1:neq),dTdx(j,m,1:neq),dTdy(j,m,1:neq),d2Tdx2(j,m,1:neq),d2Tdy2(j,m
,1:neq),d2Tdxdy(j,m,1:neq),k);
         end
      end
   end

% apply second-order method for the correction step
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            Tnew(j,m,k) = Told(j,m,k) + 0.50*dt*(k1(j,m,k)+k2(j,m,k));
         end
      end
```

```
   end


%  apply BCs at the correction step
   for k = 1:1:neq
%  along initial boundary of first spatial dimension
      if (BC(1,1,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(2,m,k) = -cBCxo(y,t,k)/aBCxo(y,t,k);
         end
      else
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(1,m,k)  = 2.0*dx/bBCxo(y,t,k)*( aBCxo(y,t,k)*Tnew(2,m,k)    +
bBCxo(y,t,k)/(2.0*dx)*Tnew(3,m,k)    + cBCxo(y,t,k));
         end
      end
%  along second boundary of first spatial dimension
      if (BC(1,2,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(nx-1,m,k) = -cBCxf(y,t,k)/aBCxf(y,t,k);
         end
      else
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(nx,m,k) = 2.0*dx/bBCxf(y,t,k)*(-aBCxf(y,t,k)*Tnew(nx-1,m,k) +
bBCxf(y,t,k)/(2.0*dx)*Tnew(nx-2,m,k) - cBCxf(y,t,k));
         end
      end
%  along initial boundary of second spatial dimension
     if (BC(2,1,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
            Tnew(j,2,k) = -cBCyo(x,t,k)/aBCyo(x,t,k);
         end
     else
         for j = 2:1:nx-1
            x = xvec(j);
            Tnew(j,1,k)  = 2.0*dy/bBCyo(x,t,k)*( aBCyo(x,t,k)*Tnew(j,2,k)    +
bBCyo(x,t,k)/(2.0*dy)*Tnew(j,3,k)    + cBCyo(x,t,k));
         end
```

```
      end
%  along second boundary of second spatial dimension
      if (BC(2,2,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
            Tnew(j,ny-1,k) = -cBCyf(x,t,k)/aBCyf(x,t,k);
         end
     else
         for j = 2:1:nx-1
            x = xvec(j);
            Tnew(j,ny,k) = 2.0*dy/bBCyf(x,t,k)*(-aBCyf(x,t,k)*Tnew(j,ny-1,k) +
bBCyf(x,t,k)/(2.0*dy)*Tnew(j,ny-2,k) - cBCyf(x,t,k));
         end
      end
   end


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

% plot via movie

% 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;
   [xp,yp] = meshgrid(xvec(2:1:nx-1),yvec(2:1:ny-1));
   for k = 1:1:neq
      Tmax(k)=max(max(max(Tmat(2:nx-1,2:ny-1,1:nt,k))));
      Tmin(k)=min(min(min(Tmat(2:nx-1,2:ny-1,1:nt,k))));
      ztext(k) = Tmin(k) + 1.4*(Tmax(k)-Tmin(k));
   end
```

```
   xtext = xo + 0.01*(xf-xo);
   ytext = yo + 0.99*(yf-yo);


   for i=1:nskip:nt
       temps = char(strcat('time = ', num2str(tvec(i)), ' sec '));
       for k = 1:1:neq
          fig = figure(k);
          set(fig, 'Units','normalized');
          set(fig,'Position', [.08+(k-1)*.44 .2 .4 .6]);
          %color_index = get_plot_color(k);
          surf(xp, yp, Tmat(2:1:nx-1,2:1:ny-1,i,k))
          axis([xo xf yo yf Tmin(k) Tmax(k)]);
          xlabel('first spatial dimension');
          ylabel('second spatial dimension');
          zlabel('dependent variable');
        %legend (int2str([1:neq]'));
          text(xtext, ytext,ztext(k), temps);
          hold off
       end
       pause(fpsinv);
    end
end


%
%  functions defining PDE
%

function dTdt_out = pdefunk(x,y,t,Told,dTdx,dTdy,d2Tdx2,d2Tdy2,d2Tdxdy,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;
%
% convective heat loss in rod 1
```

```
%
dTdt(1) = alpha*d2Tdx2(1) + alpha*d2Tdy2(1);
% convective heat loss in rod 3
% rod 3 is concentric around rod 2
% heat lost by rod 2 is gained by rod 3
dTdt(2) = alpha*d2Tdx2(2) + alpha*d2Tdy2(2);
dTdt_out = dTdt(keq);
%
%  function defining initial condition
%

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

%
%  functions defining initial boundary condition in first spatial dimension
%

function fout = aBCxo(y,t,k);
f(1) = 1;
f(2) = 0;
fout = f(k);

function fout = bBCxo(y,t,k);
f(1) = 0;
f(2) = 1;
fout = f(k);

function fout = cBCxo(y,t,k);
f(1) = -300;
f(2) = 0;
fout = f(k);

%
%  functions defining final boundary condition in first spatial dimension
%

function fout = aBCxf(y,t,k);
f(1) = 1;
f(2) = 1;
```

```
fout = f(k);

function fout = bBCxf(y,t,k);
f(1) = 0;
f(2) = 0;
fout = f(k);

function fout = cBCxf(y,t,k);
f(1) = -400;
f(2) = -400;
fout = f(k);

%
%  functions defining initial boundary condition in second spatial dimension
%

function fout = aBCyo(x,t,k);
f(1) = 1;
f(2) = 0;
fout = f(k);

function fout = bBCyo(x,t,k);
f(1) = 0;
f(2) = 1;
fout = f(k);

function fout = cBCyo(x,t,k);
f(1) = -300;
f(2) = 0;
fout = f(k);

%
%  functions defining final boundary condition in second spatial dimension
%

function fout = aBCyf(x,t,k);
f(1) = 1;
f(2) = 1;
fout = f(k);

function fout = bBCyf(x,t,k);
f(1) = 0;
```

```
f(2) = 0;
fout = f(k);

function fout = cBCyf(x,t,k);
f(1) = -400;
f(2) = -400;
fout = f(k);
```

Appendix II. parapde_n_anyBC_2d_cyl.m

```
function [xvec,yvec,tvec,Tmat] = parapde_n_anyBC_2d
%
%  The routine parapde_n_anyBC will solve
%  a system of non-linear 2-D parabolic PDEs of the form
%
%  dT/dt = f(x,y,t,T,dTdx,dTdy,d2Tdx2,d2Ydy2,d2Tdxdy)
%
%  using a second order method
%
%  The initial conditions must have the form
%  IC:  T(x,y,to,k_eq) = To(x,y,k_eq);
%
%  The boundary conditions can be
%  Dirichlet, Neumann or Mixed of the form
%  x BC 1:  aBCxo(y,t,k)*T(xo,y,t) + bBCxo(y,t,k)*dTdx(xo,y,t) + cBCxo(y,t,k) = 0;
%  x BC 2:  aBCxf(y,t,k)*T(xf,y,t) + bBCxf(y,t,k)*dTdx(xf,y,t) + cBCxf(y,t,k) = 0;
%  y BC 3:  aBCyo(x,t,k)*T(x,yo,t) + bBCyo(x,t,k)*dTdx(x,yo,t) + cBCyo(x,t,k) = 0;
%  y BC 4:  aBCyf(x,t,k)*T(x,yf,t) + bBCyf(x,t,k)*dTdx(x,yf,t) + cBCyf(x,t,k) = 0;
%
%  The necessary functions:
%  pdefunkf(x,y,t,T,dTdx,dTdy,d2Tdx2,d2Ydy2,d2Tdxdy),
%  To(x,y,k),
%  aBCxo(y,t,k), bBCxo(y,t,k), cBCxo(y,t,k), aBCxf(y,t,k), bBCxf(y,t,k), cBCxf(y,t,k)
%  aBCyo(x,t,k), bBCyo(x,t,k), cBCyo(x,t,k), aBCyf(x,t,k), bBCyf(x,t,k), cBCyf(x,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 initial value, final value and discretization in space, y,
%  yo, yf, and dy 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,yvec,tvec,Tmat] = parapde_n_anyBC_2d_example01;
%
%  code written by:  David J. Keffer
%  University of Tennessee, dkeffer@utk.edu
%  code written:  October 21, 2001
%  comments last updated:  February 12, 2015
%
clear all;
close all;
%  define number of PDEs
neq = 1;

%
%  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 dimension is the first index 1 = z, 2 = r
%  The boundary is the second index  1 = o, 2 = f
%  The equation is the third index.  from 1 no neq
BC(1,1,1) = 'D';
BC(1,2,1) = 'D';
BC(2,1,1) = 'N';
BC(2,2,1) = 'N';


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

% discretize the first spatial dimension
xo = 0;
xf = 1.0;
dx = 5.0e-2;
% include imaginary boundary nodes
xvec = [xo-dx:dx:xf+dx];
nx = length(xvec);
dxi = 1.0/dx;
```

```
% discretize the second spatial dimension
dy = 1.0e-2;
yo = dy;
yf = 0.1;
% include imaginary boundary nodes
yvec = [yo-dy:dy:yf+dy];
ny = length(yvec);
dyi = 1.0/dy;

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


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

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



%
% apply Neumann/Mixed BCs as ICs at imaginary nodes
%
for k = 1:1:neq
%  along initial boundary of first spatial dimension
   if (BC(1,1,k) ~= 'D')
      for m = 2:1:ny-1
         y = yvec(m);
         Tmat(1,m,i,k)  = 2.0*dx/bBCxo(y,t,k)*( aBCxo(y,t,k)*Tmat(2,m,i,k)    +
bBCxo(y,t,k)/(2.0*dx)*Tmat(3,m,i,k)    + cBCxo(y,t,k));
      end
   end
%  along second boundary of first spatial dimension
```

```
   if (BC(1,2,k) ~= 'D')
      for m = 2:1:ny-1
         y = yvec(m);
         Tmat(nx,m,i,k) = 2.0*dx/bBCxf(y,t,k)*(-aBCxf(y,t,k)*Tmat(nx-1,m,i,k) +
bBCxf(y,t,k)/(2.0*dx)*Tmat(nx-2,m,i,k) - cBCxf(y,t,k));
      end
   end
%  along initial boundary of second spatial dimension
   if (BC(2,1,k) ~= 'D')
      for j = 2:1:nx-1
         x = xvec(j);
         Tmat(j,1,i,k)  = 2.0*dy/bBCyo(x,t,k)*( aBCyo(x,t,k)*Tmat(j,2,i,k)    +
bBCyo(x,t,k)/(2.0*dy)*Tmat(j,3,i,k)    + cBCyo(x,t,k));
      end
   end
%  along second boundary of second spatial dimension
   if (BC(2,2,k) ~= 'D')
      for j = 2:1:nx-1
         x = xvec(j);
         Tmat(j,ny,i,k) = 2.0*dy/bBCyf(x,t,k)*(-aBCyf(x,t,k)*Tmat(j,ny-1,i,k) +
bBCyf(x,t,k)/(2.0*dy)*Tmat(j,ny-2,i,k) - cBCyf(x,t,k));
      end
   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
%
%  first index of these three variables is spatial dimension
%  second index is equation
ivaro = zeros(2,neq);
ivarf = zeros(2,neq);
nvar = zeros(2,neq);
for k = 1:1:neq
%  initial boundary of first spatial dimension
   if (BC(1,1,k) == 'D')
      ivaro(1,k) = 3;
   else
      ivaro(1,k) = 2;
   end
%  final boundary of first spatial dimension
   if (BC(1,2,k) == 'D')
```

```
      ivarf(1,k) = nx-2;
   else
      ivarf(1,k) = nx-1;
   end
%  number of nodes along first spatial dimension
   nvar(1,k) = ivarf(1,k) - ivaro(1,k) + 1;
%  initial boundary of second spatial dimension
   if (BC(2,1,k) == 'D')
      ivaro(2,k) = 3;
   else
      ivaro(2,k) = 2;
   end
%  final boundary of second spatial dimension
   if (BC(2,2,k) == 'D')
      ivarf(2,k) = ny-2;
   else
      ivarf(2,k) = ny-1;
   end
%  number of nodes along second spatial dimension
   nvar(2,k) = ivarf(2,k) - ivaro(2,k) + 1;
end


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

%
%  Prediction Step
%

% allocate memory for spatial derivatives
dTdx = zeros(nx,ny,neq);
dTdy = zeros(nx,ny,neq);
d2Tdx2 = zeros(nx,ny,neq);
d2Tdy2 = zeros(nx,ny,neq);
d2Tdxdy = zeros(nx,ny,neq);

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
```

```
         for m = ivaro(2,k):1:ivarf(2,k)
            dTdx(j,m,k) = 0.5*( Told(j+1,m,k) - Told(j-1,m,k) )*dxi;
            dTdy(j,m,k) = 0.5*( Told(j,m+1,k) - Told(j,m-1,k) )*dyi;
            d2Tdx2(j,m,k) = ( Told(j+1,m,k) - 2.0*Told(j,m,k) + Told(j-1,m,k) )*dxi^2;
            d2Tdy2(j,m,k) = ( Told(j,m+1,k) - 2.0*Told(j,m,k) + Told(j,m-1,k) )*dyi^2;
            d2Tdxdy(j,m,k) = ( Told(j+1,m+1,k) - Told(j+1,m-1,k) - Told(j-1,m+1,k) + Told(j-1,m-1,k)
)*dxi*dyi*0.25;
         end
      end
   end

k1 = zeros(nx,ny,neq);
% estimate slope  at beginning of temporal interval
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
%             fprintf(1,'main %e %e %e %e %e\n', k, j, m, ivaro(2,k), ivarf(2,k));
            k1(j,m,k) =
pdefunk(xvec(j),yvec(m),tvec(i),Told(j,m,1:neq),dTdx(j,m,1:neq),dTdy(j,m,1:neq),d2Tdx2(j,m,1:neq),d2Tdy2(j,m
,1:neq),d2Tdxdy(j,m,1:neq),k);
         end
      end
   end

% apply Euler method for the prediction step
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            Ttem(j,m,k) = Told(j,m,k) + dt*k1(j,m,k);
         end
      end
   end

%  apply BCs at the prediction step
   for k = 1:1:neq
%  along initial boundary of first spatial dimension
      if (BC(1,1,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
            Ttem(2,m,k) = -cBCxo(y,t,k)/aBCxo(y,t,k);
         end
      else
         for m = 2:1:ny-1
```

```
            y = yvec(m);
            Ttem(1,m,k)  = 2.0*dx/bBCxo(y,t,k)*( aBCxo(y,t,k)*Ttem(2,m,k)    +
bBCxo(y,t,k)/(2.0*dx)*Ttem(3,m,k)    + cBCxo(y,t,k));
         end
      end
%  along second boundary of first spatial dimension
      if (BC(1,2,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
            Ttem(nx-1,m,k) = -cBCxf(y,t,k)/aBCxf(y,t,k);
         end
      else
         for m = 2:1:ny-1
            y = yvec(m);
            Ttem(nx,m,k) = 2.0*dx/bBCxf(y,t,k)*(-aBCxf(y,t,k)*Ttem(nx-1,m,k) +
bBCxf(y,t,k)/(2.0*dx)*Ttem(nx-2,m,k) - cBCxf(y,t,k));
         end
      end
%  along initial boundary of second spatial dimension
     if (BC(2,1,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
            Ttem(j,2,k) = -cBCyo(x,t,k)/aBCyo(x,t,k);
         end
     else
         for j = 2:1:nx-1
            x = xvec(j);
            Ttem(j,1,k)  = 2.0*dy/bBCyo(x,t,k)*( aBCyo(x,t,k)*Ttem(j,2,k)    +
bBCyo(x,t,k)/(2.0*dy)*Ttem(j,3,k)    + cBCyo(x,t,k));
         end
      end
%  along second boundary of second spatial dimension
      if (BC(2,2,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
            Ttem(j,ny-1,k) = -cBCyf(x,t,k)/aBCyf(x,t,k);
         end
     else
         for j = 2:1:nx-1
            x = xvec(j);
            Ttem(j,ny,k) = 2.0*dy/bBCyf(x,t,k)*(-aBCyf(x,t,k)*Ttem(j,ny-1,k) +
bBCyf(x,t,k)/(2.0*dy)*Ttem(j,ny-2,k) - cBCyf(x,t,k));
         end
      end
```

```
   end

%
%  Correction Step
%

%  compute first and second spatial derivatives
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            dTdx(j,m,k) = 0.5*( Ttem(j+1,m,k) - Ttem(j-1,m,k) )*dxi;
            dTdy(j,m,k) = 0.5*( Ttem(j,m+1,k) - Ttem(j,m-1,k) )*dyi;
            d2Tdx2(j,m,k) = ( Ttem(j+1,m,k) - 2.0*Ttem(j,m,k) + Ttem(j-1,m,k) )*dxi^2;
            d2Tdy2(j,m,k) = ( Ttem(j,m+1,k) - 2.0*Ttem(j,m,k) + Ttem(j,m-1,k) )*dyi^2;
            d2Tdxdy(j,m,k) = ( Ttem(j+1,m+1,k) - Ttem(j+1,m-1,k) - Ttem(j-1,m+1,k) + Ttem(j-1,m-1,k)
)*dxi*dyi*0.25;
         end
      end
   end

k2 = zeros(nx,ny,neq);
% estimate slope  at end of temporal interval
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            k2(j,m,k) =
pdefunk(xvec(j),yvec(m),tvec(i),Ttem(j,m,1:neq),dTdx(j,m,1:neq),dTdy(j,m,1:neq),d2Tdx2(j,m,1:neq),d2Tdy2(j,m
,1:neq),d2Tdxdy(j,m,1:neq),k);
         end
      end
   end

% apply second-order method for the correction step
   for k = 1:1:neq
      for j = ivaro(1,k):1:ivarf(1,k)
         for m = ivaro(2,k):1:ivarf(2,k)
            Tnew(j,m,k) = Told(j,m,k) + 0.50*dt*(k1(j,m,k)+k2(j,m,k));
         end
      end
   end


%  apply BCs at the correction step
```

```
   for k = 1:1:neq
%  along initial boundary of first spatial dimension
      if (BC(1,1,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(2,m,k) = -cBCxo(y,t,k)/aBCxo(y,t,k);
         end
      else
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(1,m,k)  = 2.0*dx/bBCxo(y,t,k)*( aBCxo(y,t,k)*Tnew(2,m,k)    +
bBCxo(y,t,k)/(2.0*dx)*Tnew(3,m,k)    + cBCxo(y,t,k));
         end
      end
%  along second boundary of first spatial dimension
      if (BC(1,2,k) == 'D')
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(nx-1,m,k) = -cBCxf(y,t,k)/aBCxf(y,t,k);
         end
      else
         for m = 2:1:ny-1
            y = yvec(m);
            Tnew(nx,m,k) = 2.0*dx/bBCxf(y,t,k)*(-aBCxf(y,t,k)*Tnew(nx-1,m,k) +
bBCxf(y,t,k)/(2.0*dx)*Tnew(nx-2,m,k) - cBCxf(y,t,k));
         end
      end
%  along initial boundary of second spatial dimension
     if (BC(2,1,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
            Tnew(j,2,k) = -cBCyo(x,t,k)/aBCyo(x,t,k);
         end
     else
         for j = 2:1:nx-1
            x = xvec(j);
            Tnew(j,1,k)  = 2.0*dy/bBCyo(x,t,k)*( aBCyo(x,t,k)*Tnew(j,2,k)    +
bBCyo(x,t,k)/(2.0*dy)*Tnew(j,3,k)    + cBCyo(x,t,k));
         end
      end
%  along second boundary of second spatial dimension
      if (BC(2,2,k) == 'D')
         for j = 2:1:nx-1
            x = xvec(j);
```

```
            Tnew(j,ny-1,k) = -cBCyf(x,t,k)/aBCyf(x,t,k);
         end
      else
         for j = 2:1:nx-1
            x = xvec(j);
            Tnew(j,ny,k) = 2.0*dy/bBCyf(x,t,k)*(-aBCyf(x,t,k)*Tnew(j,ny-1,k) +
bBCyf(x,t,k)/(2.0*dy)*Tnew(j,ny-2,k) - cBCyf(x,t,k));
         end
       end
   end


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

% plot via movie

% 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;
   [xp,yp] = meshgrid(xvec(2:1:nx-1),yvec(2:1:ny-1));
   for k = 1:1:neq
      Tmax(k)=max(max(max(Tmat(2:nx-1,2:ny-1,1:nt,k))));
      Tmin(k)=min(min(min(Tmat(2:nx-1,2:ny-1,1:nt,k))));
      ztext(k) = Tmin(k) + 1.4*(Tmax(k)-Tmin(k));
   end
   xtext = xo + 0.01*(xf-xo);
   ytext = yo + 0.99*(yf-yo);
   for i=1:nskip:nt
       temps = char(strcat('time = ', num2str(tvec(i)), ' sec '));
       for k = 1:1:neq
```

```
          fig = figure(k);
          set(fig, 'Units','normalized');
          set(fig,'Position', [.08+(k-1)*.44 .2 .4 .6]);
          %color_index = get_plot_color(k);
          surf(xp, yp, Tmat(2:1:nx-1,2:1:ny-1,i,k)')
          axis([xo xf yo yf Tmin(k) Tmax(k)]);
          xlabel('first spatial dimension');
          ylabel('second spatial dimension');
          zlabel('dependent variable');
        %legend (int2str([1:neq]'));
          text(xtext, ytext,ztext(k), temps);
          hold off
       end
       pause(fpsinv);
    end
end


%
%  functions defining PDE
%

function dTdt_out = pdefunk(z,r,t,Told,dTdz,dTdr,d2Tdz2,d2Tdr2,d2Tdzdr,keq);
%
%  physical properties
%
% 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;
%  Stefan-Boltzmann constant [J/s/m^2/K^4]
sigma = 5.670373e-8;
% gray body permittivity
eps = 0.15; % (for dull Cu)
%eps = 0.0; % no radiative loss
% plate geometry
length = 1.0; % [m]
radius = 0.1; % [m]
pi = 2.0*asin(1.0);
```

```
area = 2*pi*radius*length;
volume = pi*radius*radius*length;
s = area/volume; % 1/m
% surrounding temperature [K]
Tsurround = 300.0;
%
% PDE
%
dTdt_out = alpha*(d2Tdz2(1) + 1.0/r*dTdr(1) + d2Tdr2(1)) - eps*sigma*s/(rho*Cp)*(Told(1)^4 - Tsurround^4);
%fprintf(1,'%e %e %e %e \n',z, r, t, dTdt_out);



%
%  function defining initial condition
%

function ic_out = icfunk(x,y,keq);
ic(1) = 1000.0;
ic_out = ic(keq);

%
%  functions defining initial boundary condition in first spatial dimension
%

function fout = aBCxo(y,t,k);
f(1) = 1;
fout = f(k);

function fout = bBCxo(y,t,k);
f(1) = 0;
fout = f(k);

function fout = cBCxo(y,t,k);
f(1) = -800;
fout = f(k);

%
%  functions defining final boundary condition in first spatial dimension
%

function fout = aBCxf(y,t,k);
```

```
f(1) = 1;
fout = f(k);

function fout = bBCxf(y,t,k);
f(1) = 0;
fout = f(k);

function fout = cBCxf(y,t,k);
f(1) = -1000;
fout = f(k);

%
%  functions defining initial boundary condition in second spatial dimension
%

function fout = aBCyo(x,t,k);
f(1) = 0;
fout = f(k);

function fout = bBCyo(x,t,k);
f(1) = 1;
fout = f(k);

function fout = cBCyo(x,t,k);
f(1) = 0;
fout = f(k);

%
%  functions defining final boundary condition in second spatial dimension
%

function fout = aBCyf(x,t,k);
%  heat transfer coefficient in [W/m^2/K]
h = 4.0e+1;
f(1) = h;
fout = f(k);

function fout = bBCyf(x,t,k);
% kc = thermal conductivity [W/m/K]
kc = 401.0;
f(1) = kc;
fout = f(k);
```

```
function fout = cBCyf(x,t,k);
%  heat transfer coefficient in [W/m^2/K]
h = 4.0e+1;
% surrounding temperature [K]
Tsurround = 300.0;
f(1) = -h*Tsurround;
fout = f(k);
```