PennState
College of Engineering

NucE 497: Reactor Fuel Performance

Lecture 10: 2D transient solution of heat equation using Matlab

February 1, 2017

Michael R Tonks
Mechanical and Nuclear Engineering

Some material taken from Matlab documentation

Today we will discuss solving for the temperature profile in the fuel using 2D transient FEM

- Module 1: Fuel basics
- Module 2: Heat transport
  - Intro to heat transport and the heat equation
  - Analytical solution of the heat equation
  - Numerical solution of the heat equation
  - **1D solution of the heat equation using Matlab**
  - **2D solution of the heat equation using Matlab**
  - Coolant temperature change, power generation, and melting
- Module 3: Mechanical behavior
- Module 4: Materials issues in the fuel
- Module 5: Materials issues in the cladding
- Module 6: Accidents, used fuel, and fuel cycle

Here is some review from last time

- What kind of solution is done by the PDEPE function in Matlab?
  - 3D
  - Transient 2D axisymmetric, smeared pellets
  - Transient 1D axisymmetric
  - Steady state 1D axisymmetric
- How do you define your PDE in the PDEPE function?
  - You pass in a description of your PDE (‘heat conduction’, ‘wave’)
  - You write out your PDE in weak form
  - You command it with your mind
  - You define coefficient values for their master generic PDE

We make the solution fit the heat equation by correctly setting the parameters

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

$$\rho\, c_p \frac{\partial T}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(rk(T)\frac{\partial T}{\partial r}\right) + Q(r)$$

- x = r, t = t, u = T, dU/dx = dT/dr
- m = 1
- c(x, t, u, du/dx) = ρ c_p
- f(x, t, u, du/dx) = k(T) dT/dr
- s(x, t, u, du/dx) = Q(r)

In order to use PDEPE, we have to provide six inputs

sol = pdepe(m, @PDEfunction, @ICfunction, @BCfunction, r, t);

- The input m sets the coordinate system (m = 0 for Cartesian, m = 1 for cylindrical, and m = 2 for spherical)
- We need to create three functions
  - [c, f, s] = PDEfunction(x, t, u, dudx) – This function defines the three constants in the PDE
  - u = ICfunction(x) – This function defines the initial condition of u
  - [pl, ql, pr, qr] = BCfunction(xl, ul, xr, ur, t) – This function defines boundary conditions on the left and right side of the domain
- We create the mesh using the command r = linspace(a, b, N)
  - This creates r as a vector that goes from a to b with N points
- We create the time domain t = linspace(t_0, t_s, M)
  - This creates t as a vector that goes from t_0 to t_s with M time steps

Here is more detail about the function that creates the three terms of the PDE

[c, f, s] = PDEfunction(x, t, u, dudx)

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

- Four inputs are passed in by the solver
  - x – the current location in space (r in our case)
  - t – the current time
  - u – the current value of the variable (T in our case)
  - dudx – the current value of the derivative of the variable (dT/dr in our case)
- You then define c, f, and s as functions of these variables

```
function [c,f,s] = PDEfunction(x,t,u,DuDx)
    global density cp Q k

    c = density*cp;
    f = k*DuDx;
    s = Q;
end
```

Here is more detail about the function that creates the initial condition

u = ICfunction(x)

- One input is passed in by the solver
  - x – the current location in space (r in our case)
- You then define the value of the variable throughout the mesh at t = 0

```
function u0 = ICfunction(x)
        global Ts;
        %Initial temperature
        T0 = Ts; %K

        %Assign values
        u0 = T0*ones(size(x));
end
```

Here is more detail about the function that creates the boundary condition

[pl, ql , pr, qr] = BCfunction(xl, ul, xr, ur, t)

- Five inputs are passed in by the solver
  - xl – the coordinate location on the left side (r = 0 in our case)
  - ul – the value of u on the left side (T(r=0) in our case)
  - xr – the coordinate location on the right side (r = Rf in our case)
  - ur – the value of u on the right side (T(r=Rf) in our case)
- You define the boundary conditions on the left and right side in this form:
  $p(x,t,u) + q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right) = 0$
  - pl and ql set p and q on the left. These are ignored for cylindrical and spherical coordinates
  - pr and qr set p and q on the right.
  - What are p and q for $T_r = T_s$?
    - pr = ur - T_s

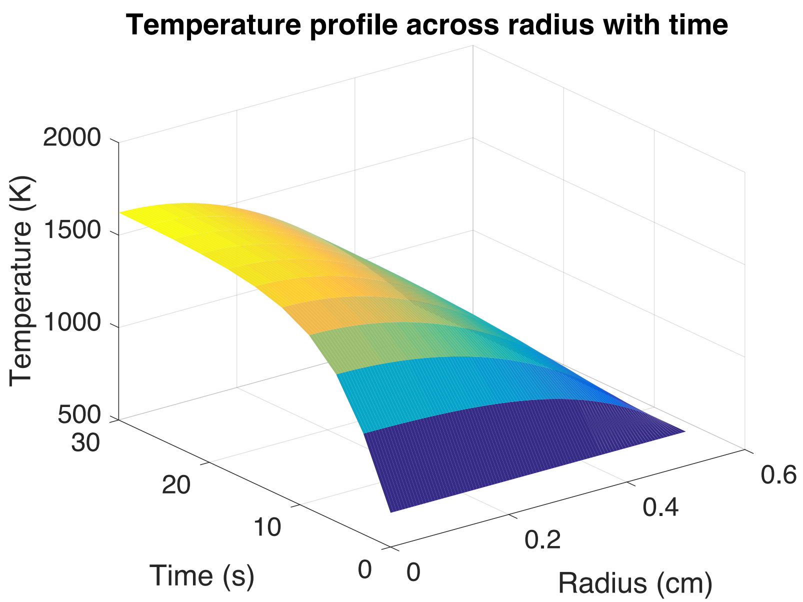
```
function = BCfunction(xl,ul,xr,ur,t)
    global Ts;

    pl = 0; %This gets ignored
    ql = 0; %This gets ignored
    pr = ur - Ts;
    qr = 0;
end
```

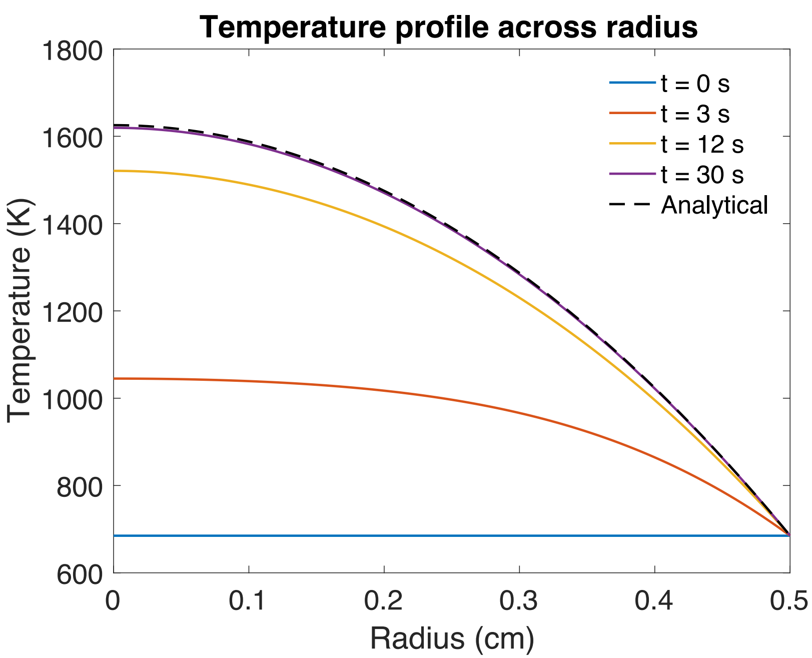
The solution comes out as a 3D array of data

T = pdepe(1,@PDEfunction,@ICfunction,@BCfunction,r,t);

- The 1st dimension is the spatial coordinate, the 2nd is the time coordinate, and the third is for additional variables you may be solving for.
- So, if we solve just for T on a mesh with N nodes and M times steps, T is a N × M matrix
- You could plot it in various ways

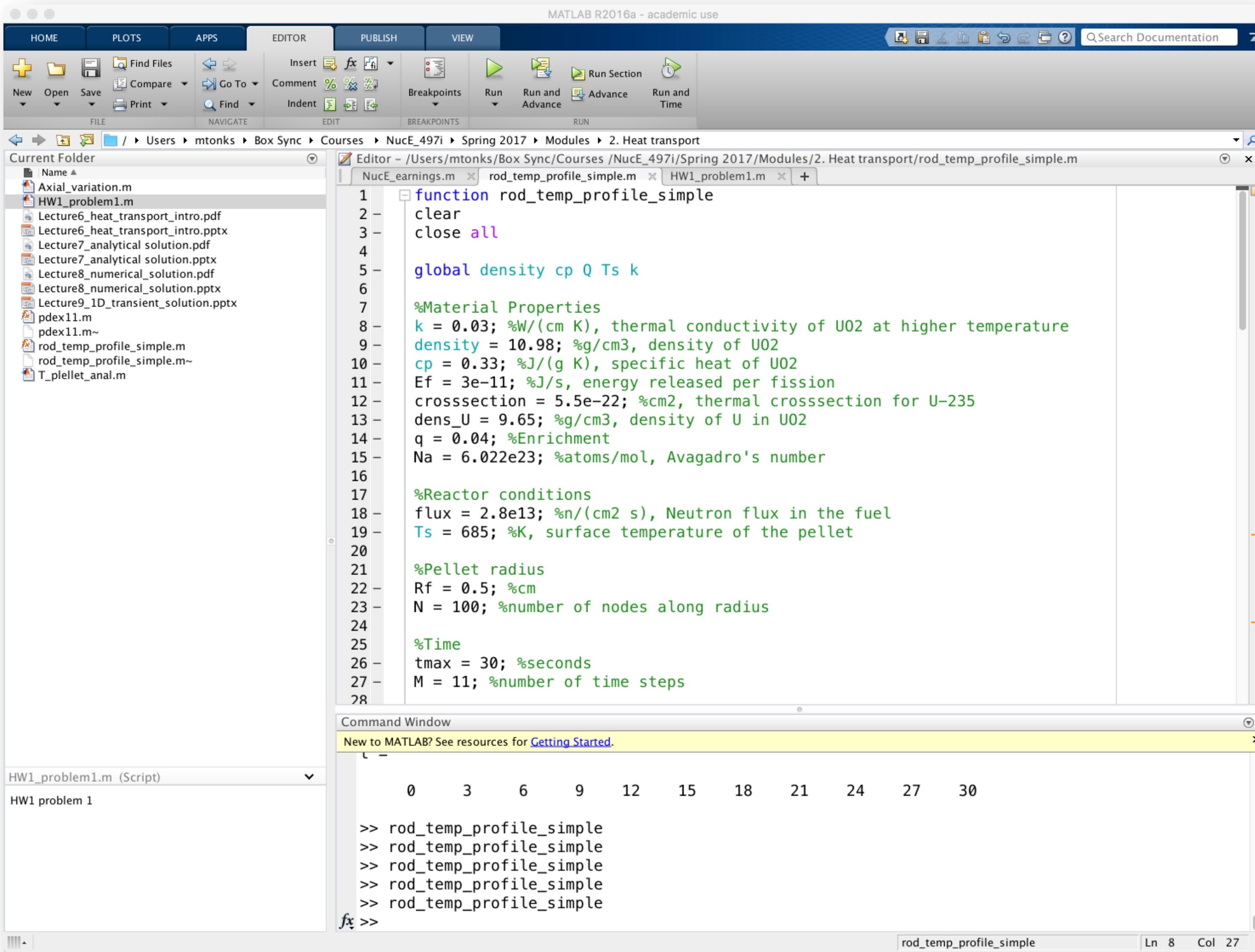


surf(r, t, T, 'edgecolor','none')



plot(r, T([1,2,5,10],:),'linewidth',1.5)

Now, let's look at the code



Now, we will do a 2D transient, smeared pellet simulation

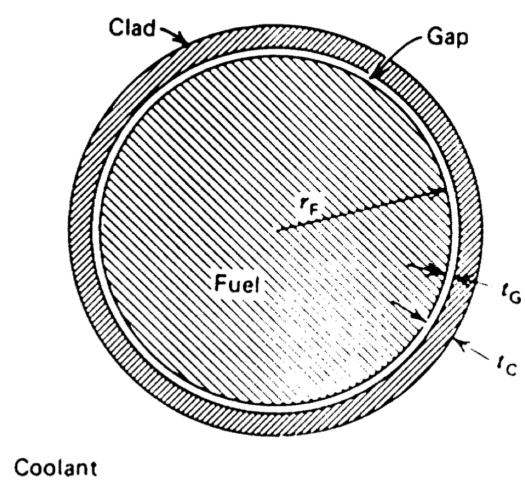
$$\rho c_p \frac{\partial T}{\partial t} = \nabla \cdot (k \nabla T) + Q$$

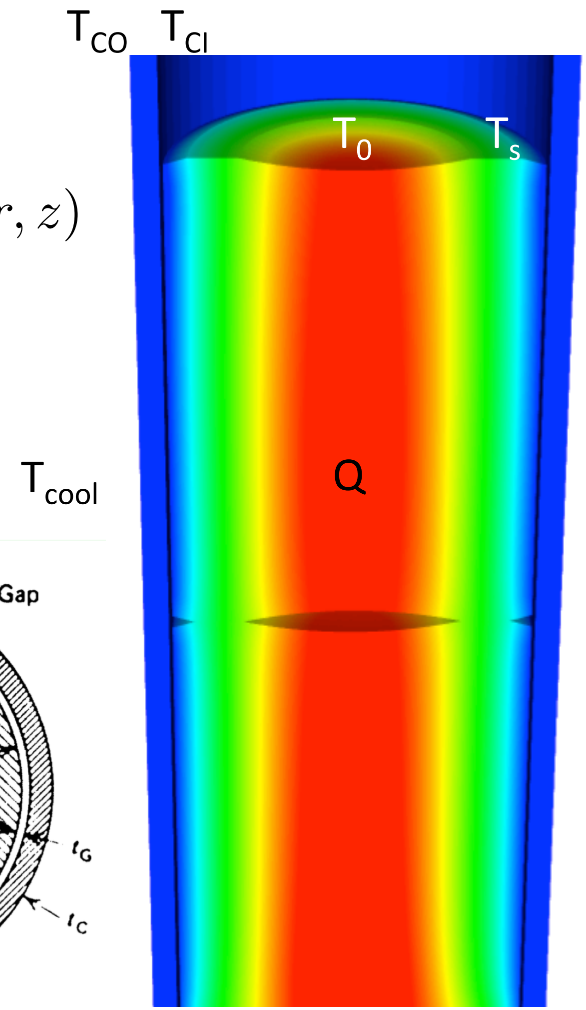
- *Assumption 1:* The behavior is axisymmetric

$$\rho c_p \frac{\partial T}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(r k(T) \frac{\partial T}{\partial r} \right) + \frac{\partial}{\partial z}\left(k(T) \frac{\partial T}{\partial z} \right) + Q(r,z)$$

- *Assumption 2: Pellets are smeared*

- **We will solve this using FEM and implicit time integration in Matlab**





Matlab’s PDE solver can solve PDEs in 2D and 3D

- The function used to set up a PDE in the PDE solver is **specifyCoefficients**. It has a general PDE form
$$m\frac{\partial^2 u}{\partial t^2}+d\frac{\partial u}{\partial t}-\nabla\cdot(c\nabla u)+au=f$$
    - u is the variable we are solving for, it is a function of y, z, and t (in 2D)
    - y and z are the space variable, defined by the mesh
    - t is the time variable, $t_0 \leq t \leq t_f$
    - m, d, c, a, and f are coefficients that can be functions of y, z, t, and u
- How can we make this equation match our heat equation?
$$\rho c_p\frac{\partial T}{\partial t}=\frac{1}{r}\frac{\partial}{\partial r}\left(rk(T)\frac{\partial T}{\partial r}\right)+\frac{\partial}{\partial z}\left(k(T)\frac{\partial T}{\partial z}\right)+Q(r,z)$$
- To make the forms match, we need to multiple by r
$$\rho\, c_p\, r\frac{\partial T}{\partial t}=\frac{\partial}{\partial r}\left(rk(T)\frac{\partial T}{\partial r}\right)+\frac{\partial}{\partial z}\left(rk(T)\frac{\partial T}{\partial z}\right)+r\,Q(r,z)$$

We make the solution fit the heat equation by correctly setting the parameters

$$m\frac{\partial^2 u}{\partial t^2} + d\frac{\partial u}{\partial t} - \frac{\partial}{\partial y}\left(c\frac{\partial u}{\partial y}\right) - \frac{\partial}{\partial z}\left(c\frac{\partial u}{\partial z}\right) + au = f$$

$$\rho\, c_p\, r\frac{\partial T}{\partial t} = \frac{\partial}{\partial r}\left(rk(T)\frac{\partial T}{\partial r}\right) + \frac{\partial}{\partial z}\left(rk(T)\frac{\partial T}{\partial z}\right) + r\,Q(r,z)$$

- m = 0
- d = ρ c_p r
- c = r k(T)
- a = 0
- f = r Q(r, z, t)

There are seven steps required to set up a solve using the PDE toolbox (or any other FEM code)

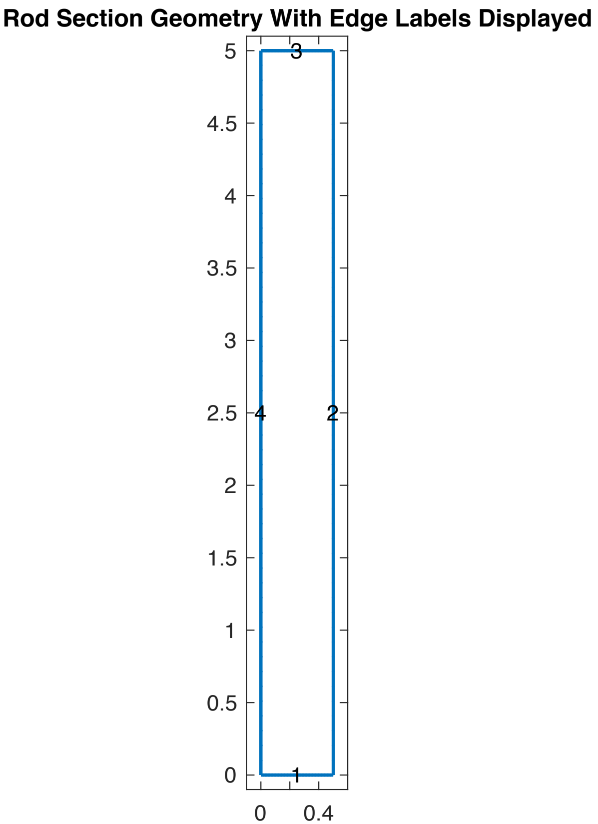
1. Define how many and what variables you are solving for
2. Define your geometry
3. Create a mesh
4. Define your PDE and material properties
5. Set up boundary conditions
6. Set up the initial condition and time steps (for a transient solve)
7. Details about executing the solve

The first step is to define the problem you are trying to solve

- In the PDE toolbox in Matlab, you define an object that contains all the information about your solve.
- You create it with the createpde command that takes one input
  - Input is the number of PDEs you wish to solve
  - model = createpde(numberOfPDE);

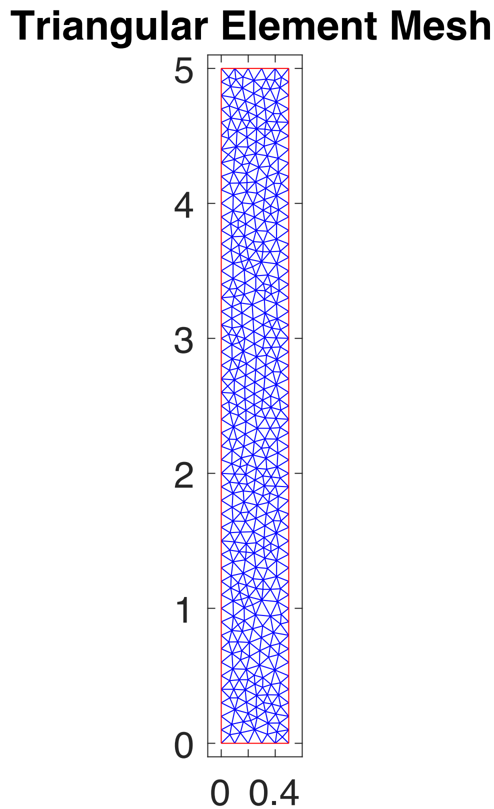
The first step is to create the geometry

- We will model geometry using symmetry around the center axis in the r direction and about the half plane in the z direction
- g = decsg([3 4 0 Rf Rf 0 0 0 length/2 length/2]');
  - First number tells it type of geometry, 3 = rectangle
  - Second is how many points define the geometry
  - The next four are the y coordinates of the points
  - The last four are the z coordinates
- Then, convert the geometry to the correct form and append it to the pde model
- geometryFromEdges(model,g);
  - We add the geometry to our model, called “model”



Now we need to create a mesh over the geometry

- The PDE toolbox uses triangle elements in 2D
- We control the mesh size by giving the target max element size Hmax
  - generateMesh(model,'Hmax',0.1);



Next, we define the coefficients

$$m\frac{\partial^2 u}{\partial t^2} + d\frac{\partial u}{\partial t} - \frac{\partial}{\partial y}\left(c\frac{\partial u}{\partial y}\right) - \frac{\partial}{\partial z}\left(c\frac{\partial u}{\partial z}\right) + au = f$$

- specifyCoefficients(model,'m',0,'d',@dFunc,'c',@cFunc,'a',0,'f',@fFunc);
  - We are defining the coefficients for "model"
  - The coefficients are defined in pairs. The first is the coefficient name and the second is the corresponding function (or 0)
- For each function, you pass in region and state and pass out the coefficient
  - Region contains the values of y, z, and t: region.y
  - State contains the variable values: state.u

```
function c = cFunc(region, state)
global k
c = 2*k*region.y;
end
```

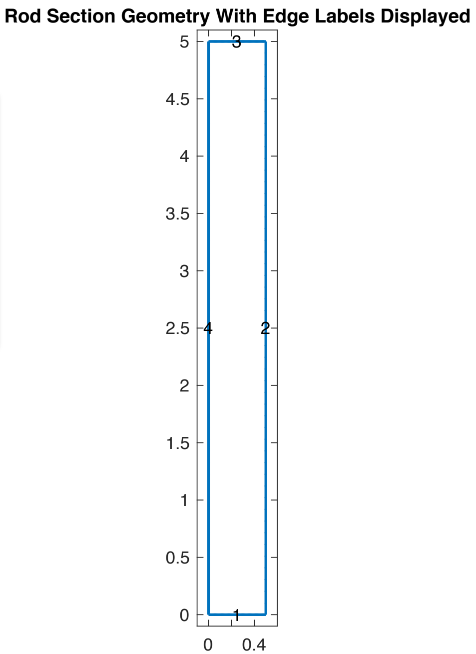
```
function d = dFunc(region,state)
global density cp;
d = density*cp*region.y;
end
```

```
function f = fFunc(region,state)
global Q;
f = Q*region.y;
end
```

Next, we define the boundary conditions for each boundary

- For each boundary we have to either specify the variable value or its derivative.
  - Dirichlet condition: Set the value of the variable
    bbottom = applyBoundaryCondition(model,'Edge',1,'u',Ts);
  - Neumann condition: Set the values of the expression $c\nabla u + q\,u = g$
    bmid = applyBoundaryCondition(model,'Edge',3,'g',0.0);

```
bbottom = applyBoundaryCondition(model,'Edge',1,'u',Ts);
bouter = applyBoundaryCondition(model,'Edge',2,'u',Ts);
bmid = applyBoundaryCondition(model,'Edge',3,'g',0.0);
bcenter = applyBoundaryCondition(model,'Edge',4,'g',0.0);
```



Now, because our problem is transient, we need to define our time behavior and initial condition

- We create a vector with our times we wish to simulate the solution at
  - tlist = linspace(0, tmax, M);
- Then we set the initial condition
  - setInitialConditions(model, Ts);

Once we have all the pieces set up, we can solve the system

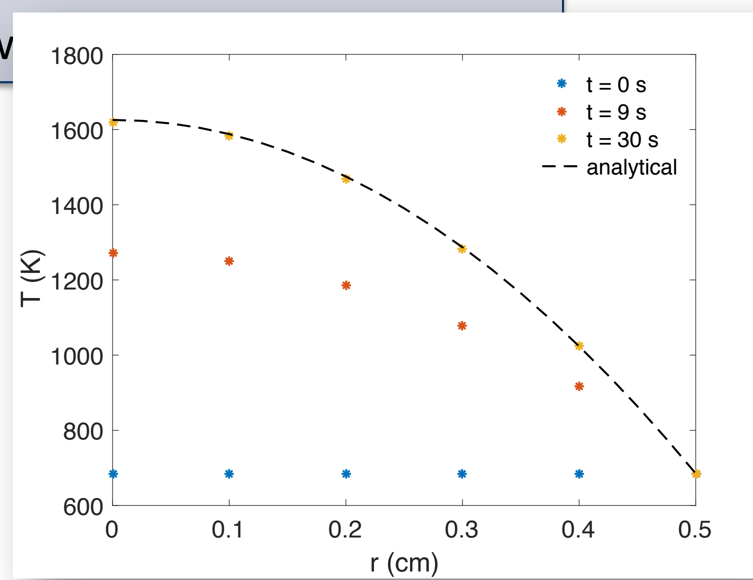
- For the solve, you pass in the model object and the time vector
  - result = solvepde(model,tlist);
  - result is an object containing the coordinate, the times, and the solution
- When the solve is complete, you can extract the solution
  - u = result.NodalSolution;
  - u is a list of the temperature at every node at all the solution times

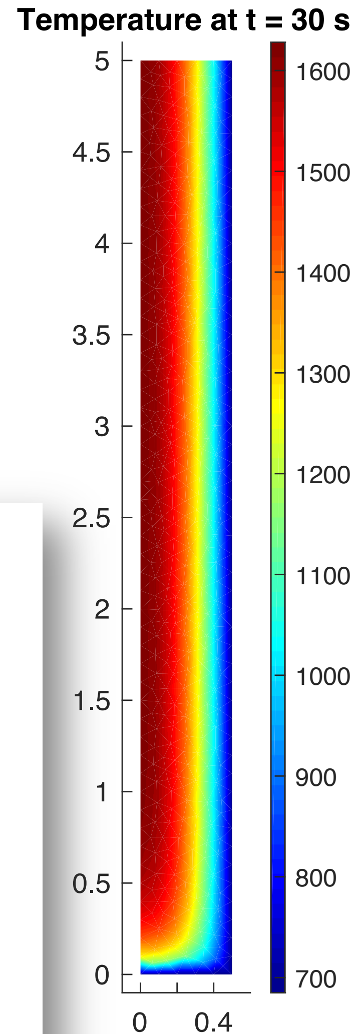
Once the solve is complete, you can plot the solution

- You can plot the solution with this command
  - pdeplot(model,'XYData',u(:,end),'Contour','on');
- You can make line plots like this

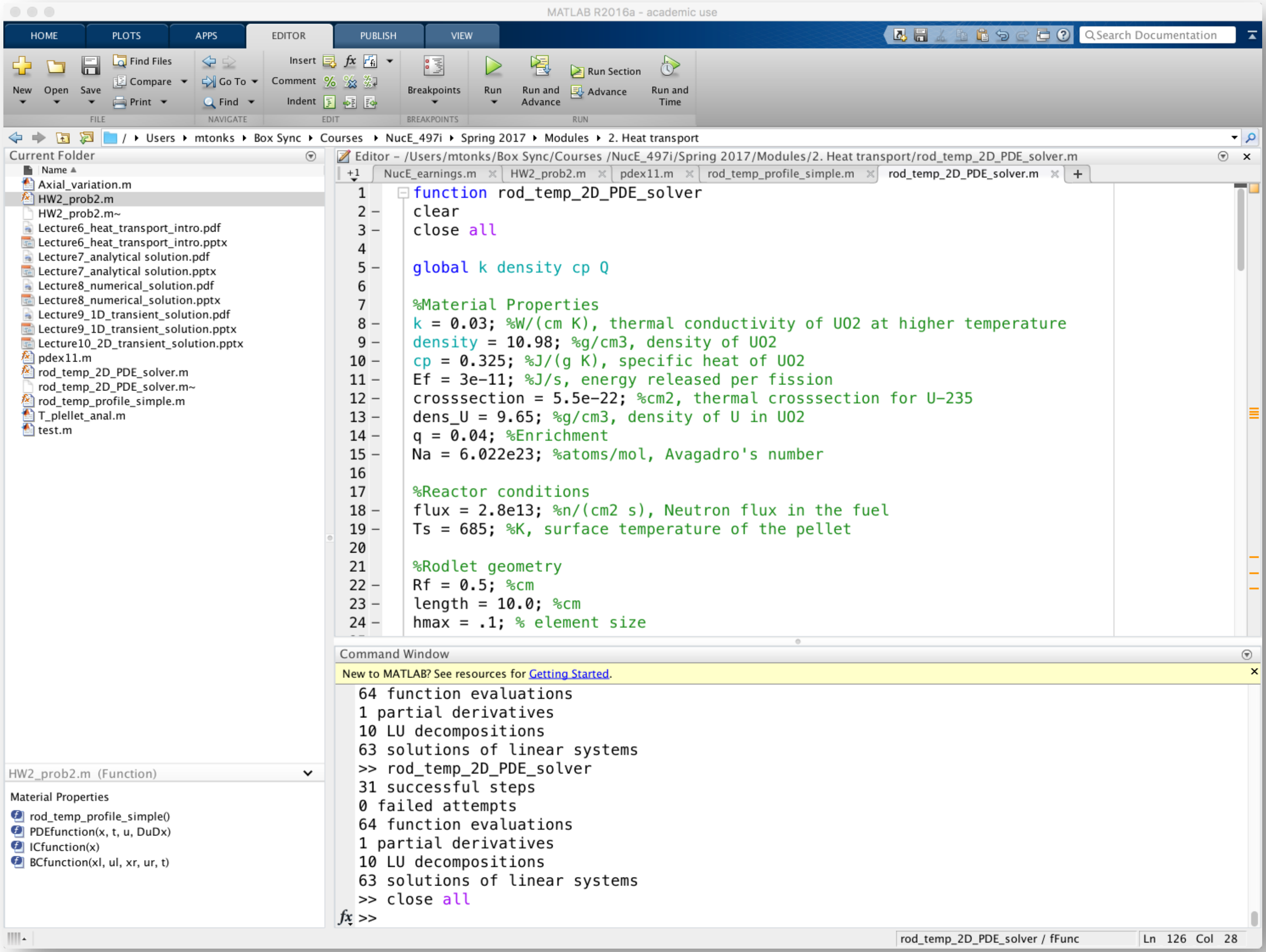
```
p = model.Mesh.Nodes;
top_nodes = find(p(2,:) == 5.0);
top_radius = p(1,top_nodes);

plot(top_radius, top_T,‘*‘,'linew
```





Now, we will look at the code



Summary

- The Matlab function PDE toolbox allows you to solve PDEs in 2D or 3D, transient or steady-state.
- You define your PDE by setting coefficient values
- There are seven steps to set up your problem
    1. Define how many variables you are solving for
    2. Define your geometry
    3. Create a mesh
    4. Define your PDE and material properties
    5. Set up boundary conditions
    6. Set up the initial condition and time steps (for a transient solve)
    7. Details about executing the solve