Final Project CHE 348
Lan Do

My final project was to create a function that could calculate the steady state heat profiles of cartesian, cylindrical, and spherical geometries. The motivation behind this project was not only its relevance in chemical engineering, but I also wanted to develop a project with multiple parts where the user has multiple functionalities in one program.

I used an iterative method to calculate the heat profiles. Analytical methods are possible to calculate heat profiles but can be incredibly cumbersome when large amounts of resistances are applied. This is also in the spirit of numerical methods – using iterative solving methods when analytical methods are too complicated or tedious to do. The formulas to iterate by are listed as follows:

$$Q = \sum_i \frac{\Delta T_i}{R_i}$$

Where:
Q = Heat Flux
T = Temperature at each resistance
R = Resistance value through each resistance

For Cartesian Geometry:

$$R_i = \frac{\Delta x_i}{k_i}$$

Where:
x = x coordinate or radius
k = thermal conductivity

For Cylindrical Geometry:

$$R_i = \frac{\Delta x_i}{k_i * A_{LM}}$$

$$A_{LM} = \frac{2\pi(r_i - r_{i-1})}{\ln\left(\frac{r_i}{r_{i-1}}\right)}$$

Where:
A_{LM} = Log mean area
r = Radius

For Spherical Geometry

$$R_i = \frac{\Delta x_i}{k_i * A_{GM}}$$

$$A_{GM} = \sqrt[2]{A_1 * A_2} = \sqrt[2]{(4\pi r_1^2)(4\pi r_2^2)} = \sqrt[2]{(r_1^2)(r_2^2) * (4\pi)^2}$$

Where:
A_{GM} = Geometric mean area

To calculate the heat profiles with increasing accuracy, the Δx_i terms are brought close to zero and the points are put on a plot. One can also compare the final temperature calculated with the values derived from the iterative calculations to determine the accuracy of calculations.

Some important notes:

The cartesian heat flux values are in power/length^2, the cylindrical heat flux values are in power/length, and the spherical heat flux values are in power. This greatly changes the q values calculated in the script but was done for the simplicity of calculations later in the scripts.

xStep must evenly divide into each resistance length. This is so that resistances calculations do not end in between resistances.

Some next steps would be to calculate the profiles for mass and momentum transfer. The setup probably would not be very difficult, as most of the equations are the same.

Example behavior is given in test_heatprofile.mlx. Please view this, as the formatting is way better than what word can provide.

For a much easier way to view all my code, I have compiled my code onto a Github repository. You can see it here (https://github.com/utexaslando/Final-Project).

```
% Calculates geometric mean radius for use in spherical heat transfer
% script

function [area] = A_GM(outerRadius, innerRadius)
    area = sqrt(((4*pi)^2)*((outerRadius^2)*(innerRadius^2)));
end
% Calculates logarithmic mean radius for use in cylindrical heat transfer
% script

function [area] = A_LM(outerRadius, innerRadius)
    area = (2*pi*(outerRadius-innerRadius))/log(outerRadius/innerRadius);
end
```

```
function [length] = calculatelength(m, resistance)
    length = 0;
    for i = 1:resistance
        length = length + m(i, 1);
    end
end

% defines a 2D cartesian heat profile for a given number of resistances and
% boundary conditions
%
% Input values:
%   h_2: heat transfer coefficient at the rightmost side (float)
%   bulkTemp2: Temperature of the air on the outside of the plate
%   resistances: a matrix of all resistances which contains individual 1x2
%       vectors with the thermal conductivity of the material and length of
the
%       material. Expected in form: [length, thermal conductivity (k)]
%   Temp1: Starting temperature of leftmost side
%   xStep: specified x step size
%
% Output values:
%   x: x values
%   T: temp at x values
%
% The strategy to solve this will to be to:
% 1. Calculate total resistance.
%       R = (x length)/k
%       Resistance total = Sum of resistances
% 2. Find temperature at the rightmost side.
%       (T1-T2)/(R_T) = q
%       (T2-bulkTemp2)*h_2 = q
%       T2 = (bulkTemp2*h_2*R_T+T_1)/(h_2*R_T+1) - albegra is faster than
%       solver
% 3. Find q by plugging back into above equation
% 4. Loop through each resistance to find temperature gradient by:
%       T2 = T1-q*R_T
%       Where R_T is determined by the x given by the user and the
appropriate k (thermal conductivity) and T1 is numerically derived.
%       This will be done in a nested for loop where each resistance will
have a different resistance value

function [x, T] = cartesian(h2, bulkTemp2, resistanceInfo, Temp1, xStep)

    if xStep>sum(resistanceInfo(:, 1))
        error('x Step is larger than length of resistances')
    end
    if sum(resistanceInfo(:, 1))/xStep ~= floor(sum(resistanceInfo(:,
1))./xStep)
        error('xStep does not divide into resistances')
    end
    if h2<0
        error('h2 is negative')
    elseif bulkTemp2<0
        error('Bulk temp 2 is negative')
    elseif Temp1<0
```

```
        error('Temp 2 is negative')
    elseif xStep<=0
        error('X Step is negative')
    end

    x = linspace(0, sum(resistanceInfo(:, 1)), 1+sum(resistanceInfo(:,
1))/xStep);
    T = zeros(size(x));
    T(1) = Temp1;

    % step 1
    totalResistance = 0;

    for resistance = 2:size(resistanceInfo, 1) % loops thru rows
        totalResistance = totalResistance + resistanceInfo(resistance,
1)/resistanceInfo(resistance, 2);
    end
    disp("Total Resistance:")
    disp(totalResistance)

    % step 2
    % using algebra instead of solver for speed

    Temp2 = (bulkTemp2*h2*totalResistance+Temp1)./(h2*totalResistance+1)

    % step 3
    q = (Temp2-Temp1)./totalResistance

    % step 4

    for resistance = 2:(size(resistanceInfo, 1)) % start on the second
element
    conductivity = resistanceInfo(resistance, 2);
    resistanceValue = xStep/conductivity;
    % this for loop breaks if xstep doesn't evenly divide
        for xval = 2+calculatelength(resistanceInfo, resistance-1)/xStep :
1+calculatelength(resistanceInfo, resistance)/xStep % increment by xStep
            T(xval) = T(xval-1)+q*resistanceValue;
        end
    end
end

% defines a 2D cylindrical heat profile for a given number of resistances and
% boundary conditions
%
% Input values:
%   h_2: heat transfer coefficient at the rightmost side (float)
%   bulkTemp2: Temperature of the air on the outside of the plate
%   resistances: a matrix of all resistances which contains individual 1x2
%       vectors with the thermal conductivity of the material and length of
the
%       material. Expected in form: [length, thermal conductivity (k)]
%   Temp1: Starting temperature of leftmost side
%   xStep: specified x step size
%
```

```
% Output values:
%   x: x values
%   T: temp at x values
%
% Note that this geometry makes input variables different than cartesian, as
a solid
% center does not hold physical reality. As a result, arrays will be
% indexed starting at the third value such that there is empty space inside
% the middle of the sphere, and the boundary condition for temperature
% starts at the first wall.
% The strategy to solve this will to be to:
% 1. Calculate total resistance.
%       R = (x length)/(k*A_LM)
%       A_LM = pi*(outerRadius^2-
innerRadius^2))/ln((outerRadius^2)/(innerRadius^2));
%       Resistance total = Sum of resistances
% 2. Find temperature at the outermost side.
%       (T2-T1)/(R_T) = q
%       (bulkTemp2-T2)*h_2*(A_o) = q
%       T2 = (bulkTemp2*h_2*R_T*A_o+T_1)/(A_o*h_2*R_T+1) - albegra is faster
than
%       solver
%       Note that:
%           A_o = 2*pi*R_i
% 3. Find q by plugging back into above equation
% 4. Loop through each resistance to find temperature gradient by:
%       T2 = T1-q*R
%       Where R is determined by the x given by the user and the appropriate
k (thermal conductivity) and T1 is numerically derived.
%       This will be done in a nested for loop where each resistance will
have a different resistance value

function [x, T] = cylindrical(h2, bulkTemp2, resistanceInfo, Temp1, xStep)

    x = linspace(0, sum(resistanceInfo(:, 1)), 1+sum(resistanceInfo(:,
1))/xStep);
    T = zeros(size(x));
    T(1:(2+resistanceInfo(2, 1)/xStep)) = Temp1;
    T(1) = Temp1;

    % step 1
    totalResistance = 0;

    for resistance = 3:size(resistanceInfo, 1) % loops thru rows starting at
the third element
        r_o = calculatelength(resistanceInfo, resistance);
        r_i = calculatelength(resistanceInfo, resistance-1);
        totalResistance = totalResistance + resistanceInfo(resistance,
1)/(A_LM(r_o, r_i) * resistanceInfo(resistance, 2));
    end
    disp("Total Resistance:")
    disp(totalResistance)

    % step 2
    % using algebra instead of solver for speed
```

```
    A_O = 2*pi*sum(resistanceInfo(:, 1));
    Temp2 =
(bulkTemp2*h2*totalResistance*A_O+Temp1)./(h2*totalResistance*A_O+1);

    % step 3
    q = (Temp2-Temp1)./totalResistance

    % step 4

    for resistance = 3:(size(resistanceInfo, 1)) % start on the third
element, second element k should be zero
    conductivity = resistanceInfo(resistance, 2);
    % this for loop breaks if xstep doesn't evenly divide
        for xval = 2+calculatelength(resistanceInfo, resistance-1)/xStep :
1+calculatelength(resistanceInfo, resistance)/xStep % increment by xStep
            r_o = xval*xStep;
            r_i = (xval-1)*xStep;
            resistanceValue = xStep/(conductivity*A_LM(r_o, r_i));
            T(xval) = T(xval-1)+q*resistanceValue;
        end
    end
end

% defines a 2D spherical heat profile for a given number of resistances and
% boundary conditions. Notably, the q is given in pure power - note that this
is
% different than cartesian (power/dist^2) and cylindrical (power/dist)
%
% Input values:
%   h_2: heat transfer coefficient at the rightmost side (float)
%   bulkTemp2: Temperature of the air on the outside of the plate
%   resistances: a matrix of all resistances which contains individual 1x2
%       vectors with the thermal conductivity of the material and length of
the
%       material. Expected in form: [length, thermal conductivity (k)]
%   Temp1: Starting temperature of leftmost side
%   xStep: specified x step size
%
% Output values:
%   x: x values
%   T: temp at x values
%
% Note that this geometry makes input variables different than cartesian, as
a solid
% center does not hold physical reality. As a result, arrays will be
% indexed starting at the third value such that there is empty space inside
% the middle of the sphere, and the boundary condition for temperature
% starts at the first wall.
% The strategy to solve this will to be to:
% 1. Calculate total resistance.
%       R = (x length)/(k*A_GM)
%       A_GM = sqrt(((4*pi)^2)*((r_o^2)(r_i^2)))
%       Resistance total = Sum of resistances
% 2. Find temperature at the outermost side.
%       (T2-T1)/(R_T) = q
```

```
%       (bulkTemp2-T2)*h_2*(A_o) = q
%       T2 = (bulkTemp2*h_2*R_T*A_o+T_1)/(A_o*h_2*R_T+1) - albegra is faster
than
%       solver
%       Note that:
%           A_o = 4*pi*R_i^2
% 3. Find q by plugging back into above equation
% 4. Loop through each resistance to find temperature gradient by:
%       T2 = T1-q*R
%       Where R is determined by the x given by the user and the appropriate
k (thermal conductivity) and T1 is numerically derived.
%       This will be done in a nested for loop where each resistance will
have a different resistance value

function [x, T] = spherical(h2, bulkTemp2, resistanceInfo, Temp1, xStep)

    x = linspace(0, sum(resistanceInfo(:, 1)), 1+sum(resistanceInfo(:,
1))/xStep);
    T = zeros(size(x));
    T(1:(2+resistanceInfo(2, 1)/xStep)) = Temp1;
    T(1) = Temp1;

    % step 1
    totalResistance = 0;

    for resistance = 3:size(resistanceInfo, 1) % loops thru rows starting at
the third element
        r_o = calculatelength(resistanceInfo, resistance);
        r_i = calculatelength(resistanceInfo, resistance-1);
        totalResistance = totalResistance + resistanceInfo(resistance,
1)/(A_GM(r_o, r_i) * resistanceInfo(resistance, 2));
    end
    disp("Total Resistance:")
    disp(totalResistance)

    % step 2
    % using algebra instead of solver for speed
    A_O = 4*pi*sum(resistanceInfo(:, 1))^2;
    Temp2 =
(bulkTemp2*h2*totalResistance*A_O+Temp1)./(h2*totalResistance*A_O+1)

    % step 3
    q = (Temp2-Temp1)./totalResistance

    % step 4

    for resistance = 3:(size(resistanceInfo, 1)) % start on the third
element, second element k should be zero
    conductivity = resistanceInfo(resistance, 2);
    % this for loop breaks if xstep doesn't evenly divide
        for xval = 2+calculatelength(resistanceInfo, resistance-1)/xStep :
1+calculatelength(resistanceInfo, resistance)/xStep % increment by xStep
            r_o = xval*xStep;
            r_i = (xval-1)*xStep;
            resistanceValue = xStep/(conductivity*A_GM(r_o, r_i));
```

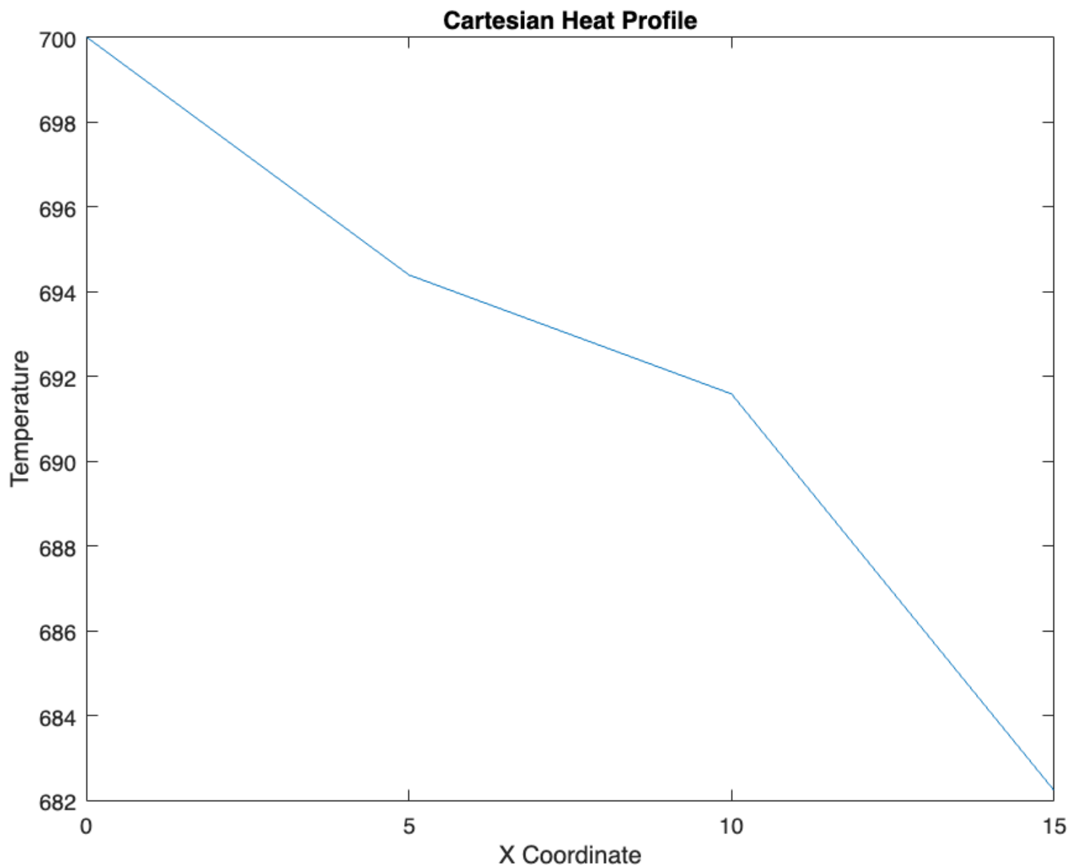
```
            T(xval) = T(xval-1)+q*resistanceValue;
        end
    end
end



% This script defines a function called heatprofile, where one can input a
% number or resistances and boundary conditions along with a desired
% geometry to get a steady state 2D heat profile.
%
% Input values:
%   h_2: heat transfer coefficient at the rightmost side (float)
%   bulkTemp2: Temperature of the air on the outside of the plate
%   resistances: a matrix of all resistances which contains individual 1x2
%       vectors with the thermal conductivity of the material and length of
the
%       material. Expected in form: [length, thermal conductivity (k)]
%   Temp1: Starting temperature of leftmost side
%   xStep: specified x step size
%   geometry: expects a string with desired geometry. Can handle cartesian,
spherical, and
%   cylindrical.

function [] = heatprofile(h2, bulkTemp2, resistanceInfo, Temp1, xStep,
geometry)

    if xStep>sum(resistanceInfo(:, 1))
        error('x Step is larger than length of resistances')
    end
    if h2<0
        error('h2 is negative')
    elseif bulkTemp2<0
        error('Bulk temp 2 is negative')
    elseif Temp1<0
        error('Temp 2 is negative')
    elseif xStep<=0
        error('X Step is negative')
    end

    for resistance = 2:size(resistanceInfo, 1)
        if mod(resistanceInfo(resistance, 1), xStep) ~= 0
            error('xStep must divide into every resistance!')
        end
    end

    figure

    if strcmp(geometry, 'cartesian')
        [x, T] = cartesian(h2, bulkTemp2, resistanceInfo, Temp1, xStep);
        plot(x, T)
        title('Cartesian Heat Profile')
    elseif strcmp(geometry, 'spherical')
        [x, T] = spherical(h2, bulkTemp2, resistanceInfo, Temp1, xStep);
        plot(x, T)
        title('Spherical Heat Profile')
```

```
    elseif strcmp(geometry, 'cylindrical')
        [x, T] = cylindrical(h2, bulkTemp2, resistanceInfo, Temp1, xStep);
        plot(x, T)
        title('Spherical Heat Profile')
    else
        error('Invalid geometry! Please input "cartesian", "spherical", or
"cylindrical"')
    end


    xlabel('X Coordinate')
    ylabel('Temperature')

end
```

```
% Declaring resistance info in the form [length, thermal conductivity]
resistanceInfo = [0,0;...
                  5, 5;...
                  5, 10;...
                  5, 3
                  ];
```

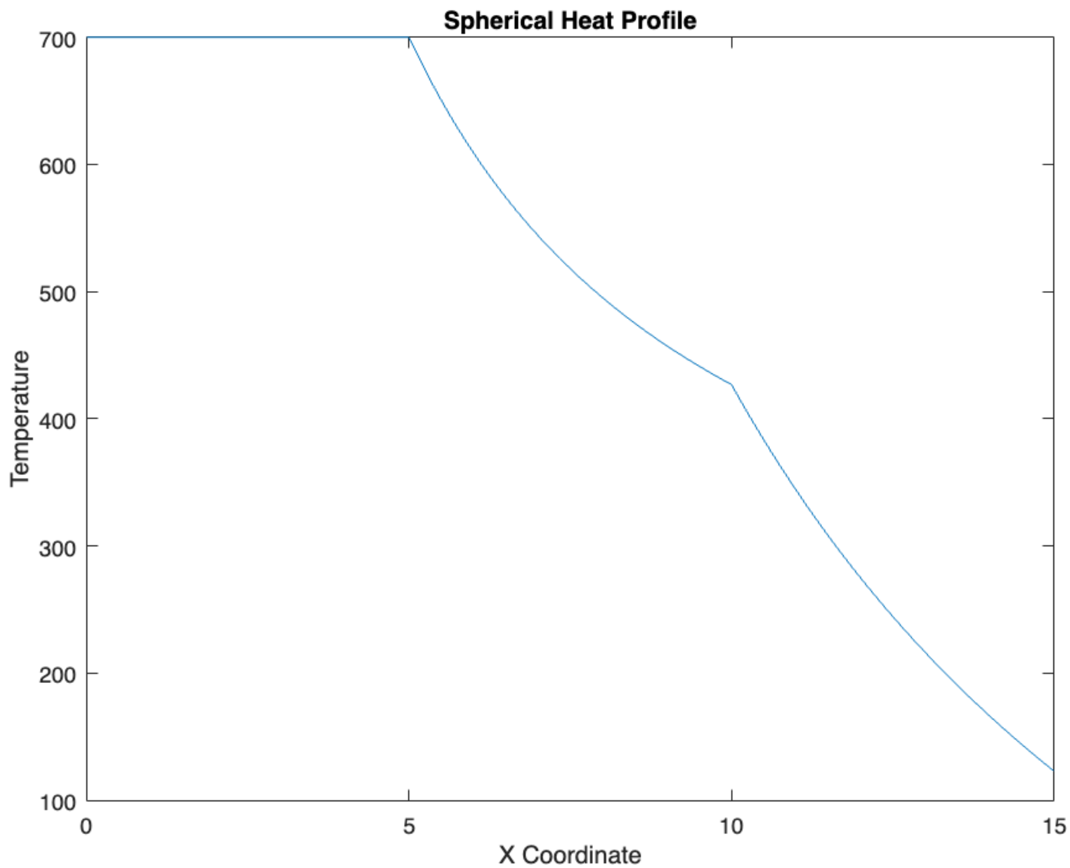
```
geometry = 'cartesian';
h2 = .01;
xStep = .01;
T1 = 700;
BulkT2 = 121.45;


heatprofile(h2, BulkT2, resistanceInfo, T1, xStep, geometry)
```

```
Total Resistance:
    3.1667
Temp2 = 682.2416
q = -5.6079
```

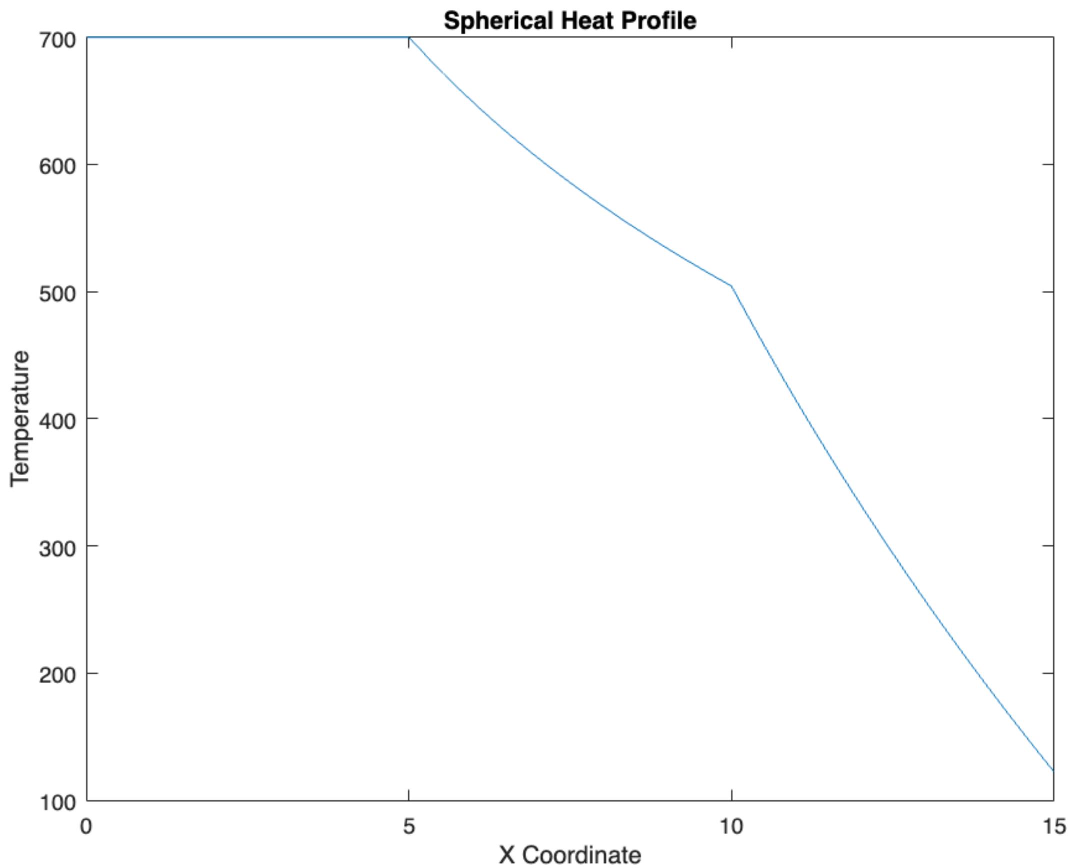



```
geometry = 'spherical';
h2 = 300;
xStep = .01;
T1 = 700;
BulkT2 = 121.45;


heatprofile(h2, BulkT2, resistanceInfo, T1, xStep, geometry)
```

```
Total Resistance:
    0.0017
Temp2 = 121.8557
q = -3.4414e+05
```




```
geometry = 'cylindrical';
h2 = 300;
xStep = .01;
T1 = 700;
BulkT2 = 121.45;


heatprofile(h2, BulkT2, resistanceInfo, T1, xStep, geometry)
Total Resistance:
    0.0325
q = -1.7759e+04
```


Spherical Heat Profile
700
600
500
400
300
200
100
Temperature
0
5
10
15
X Coordinate