Alek Pensky

MAE 404

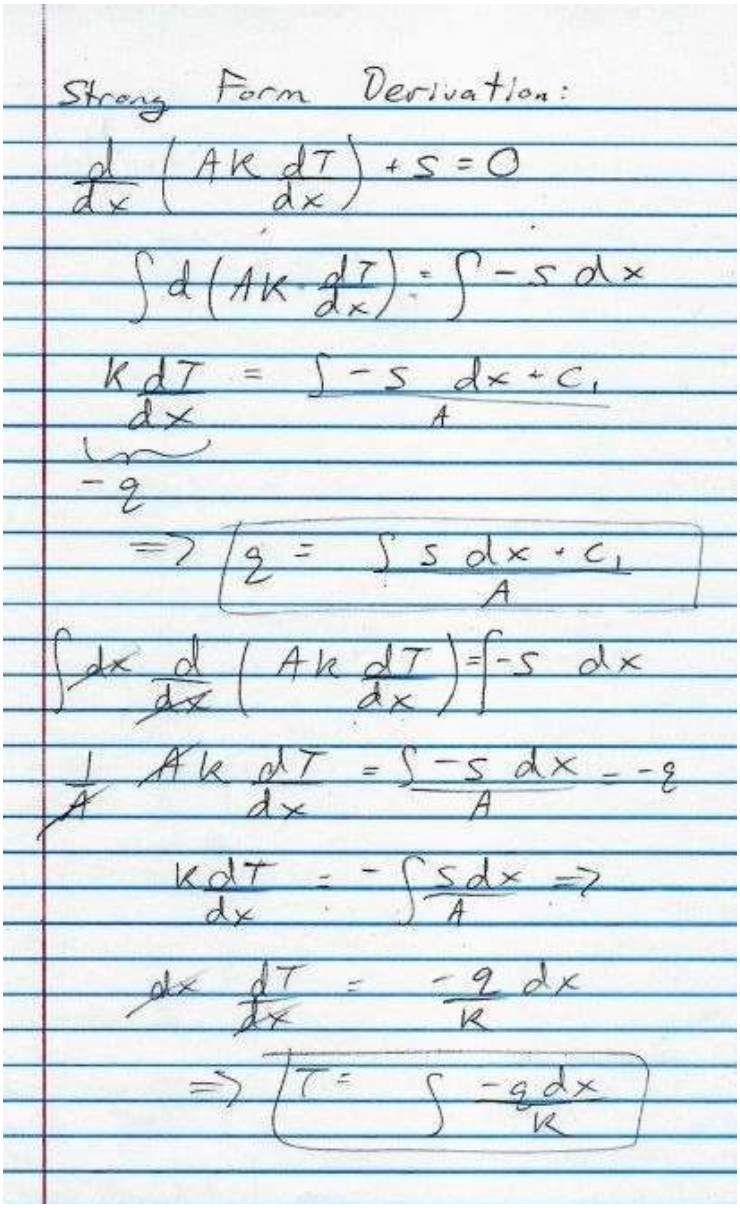
Project 1

3/20/20

Objective

The purpose of this project is to determine the temperature and heat flux fields for a wire of changing cross-section that is under a current load. the material properties of the wire are known, and there are two boundary conditions. Due to symmetry of the model, there is a zero-heat flux condition at the midplane of the wire. Additionally, there is an imposed temperature boundary condition on the outside face at 20 C. The temperature and flux fields will be solved for numerically, analytically, and with commercial FEA software using MATLAB, Mathematica, and Abaqus, respectively.

Analytical Solution

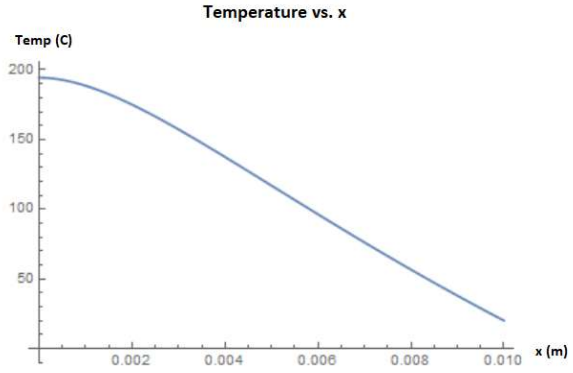


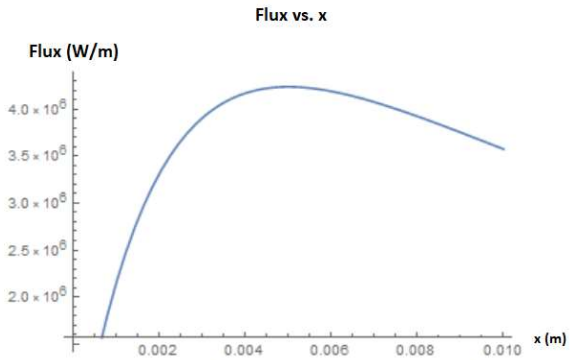
Starting with the strong form, first heat flux and then temperature were derived analytically. This was implemented in Mathematica as below:

```
flux = (Integrate[source, x] + c1)/A

temp = (Integrate[-flux, x] + c2)/k;
```

Using Mathematica, the following solution curves were generated:

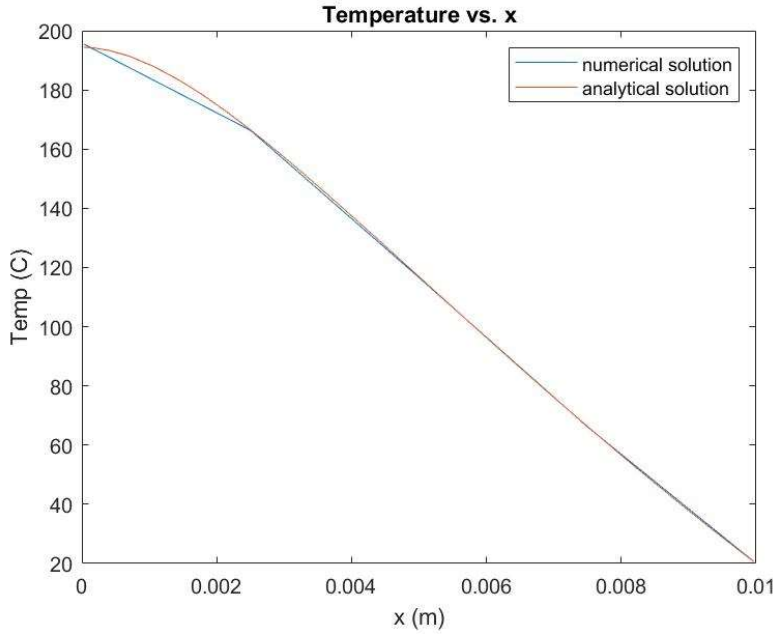




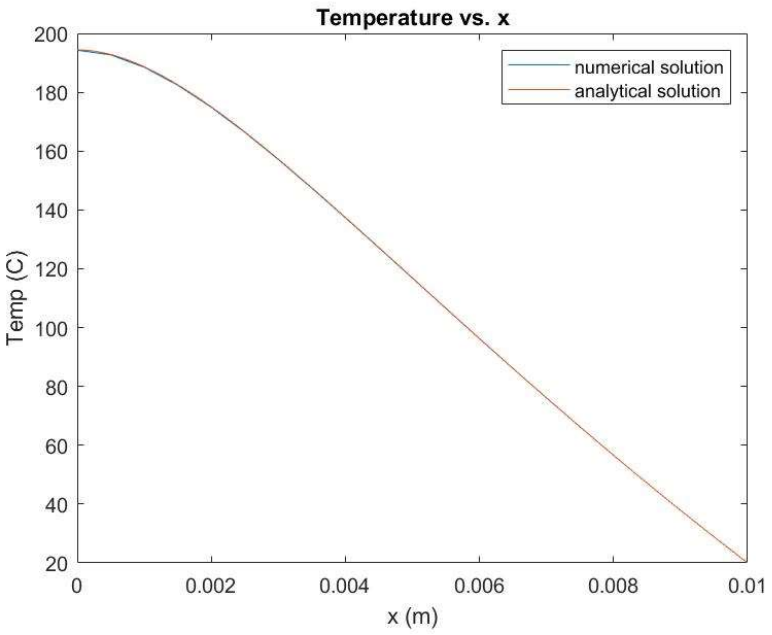
The maximum current is 1916 Amps. This was found by changing the variable for current in Mathematica until the temperature at x=0 was equal to the melting point. This is when the temperature at the midplane reaches the melting point. This is not a good assumption because it does not consider heat loss from the bpdy to the ambient environment or through conduction. If it were perfectly insulated, it would still be a bad assumption because it assumes that the resistivity is a constant, when in fact it varies with temperature.

MATLAB solution

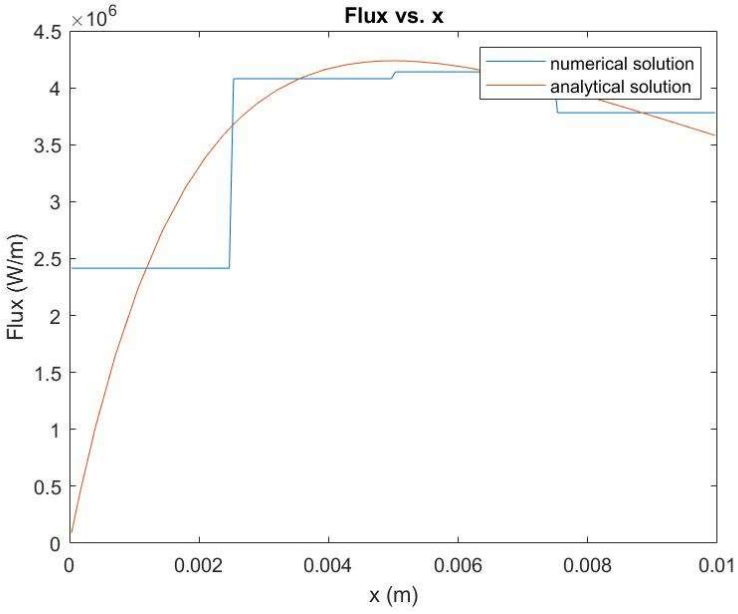
Linear:



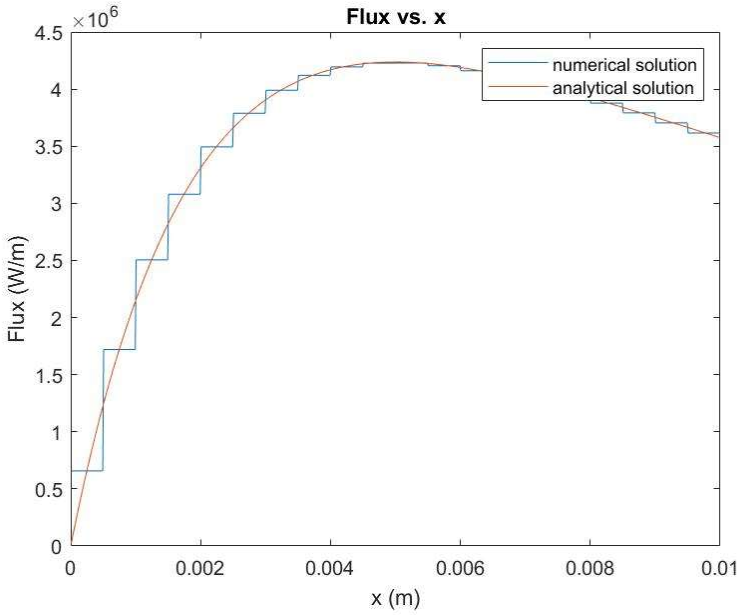
4 elements



20 elements

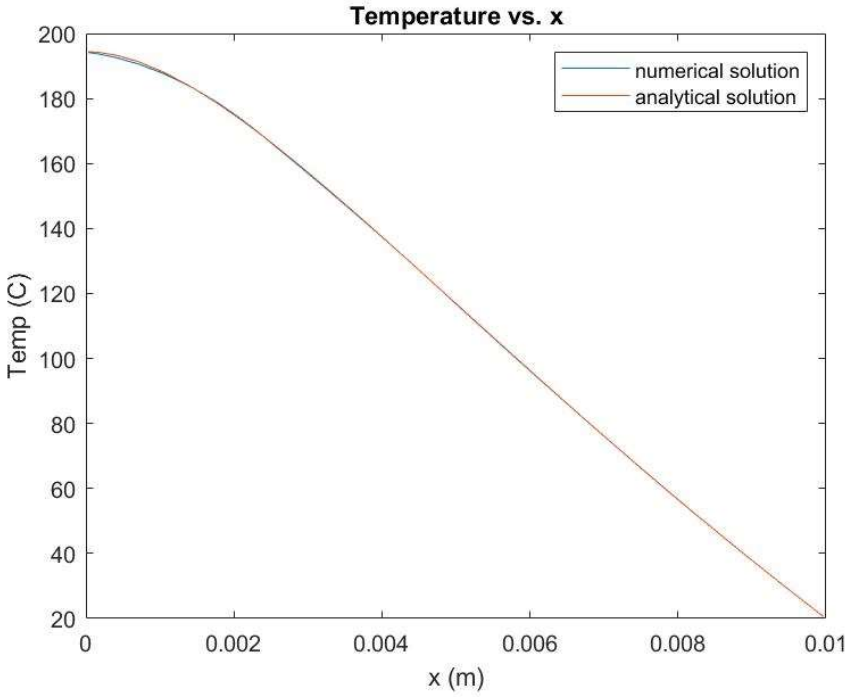


4 elements

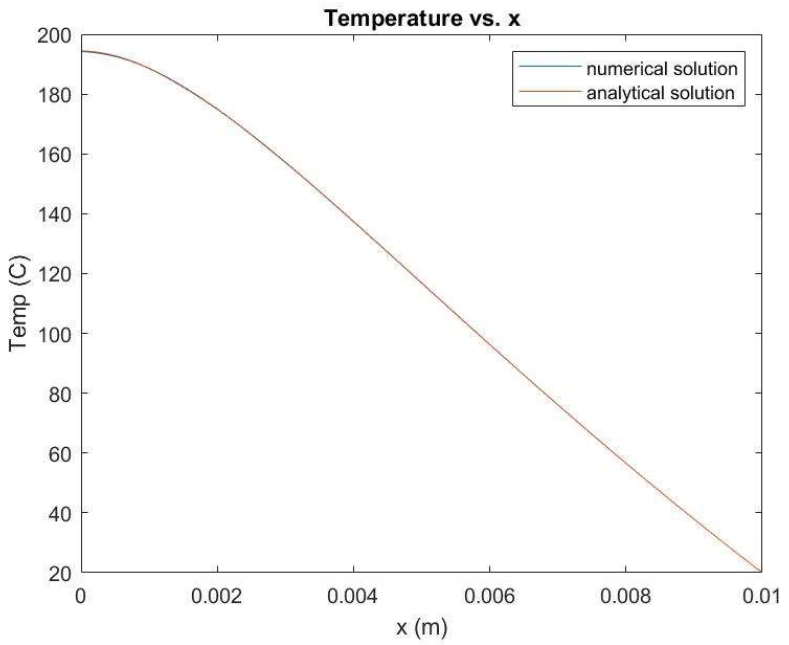


20 elements

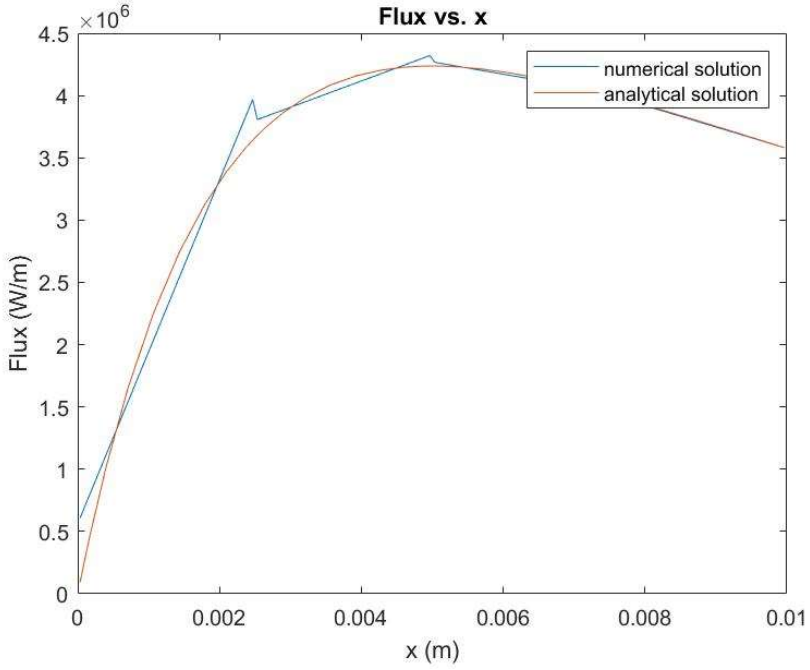
Quadratic:



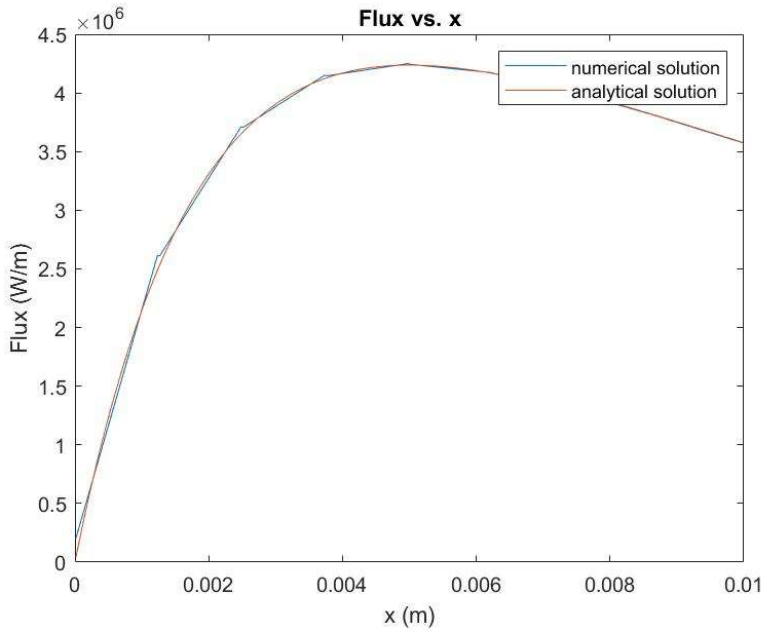
4 elements



8 elements

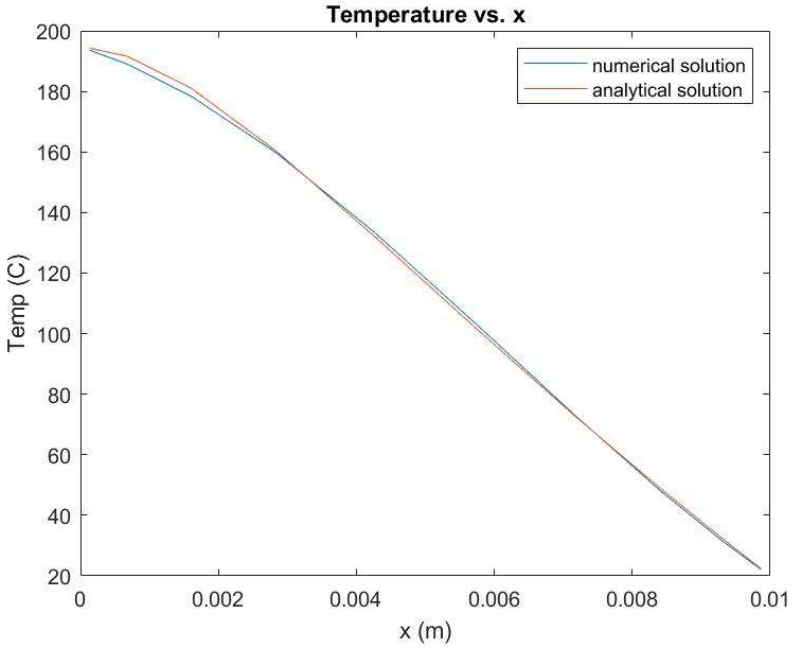


4 elements

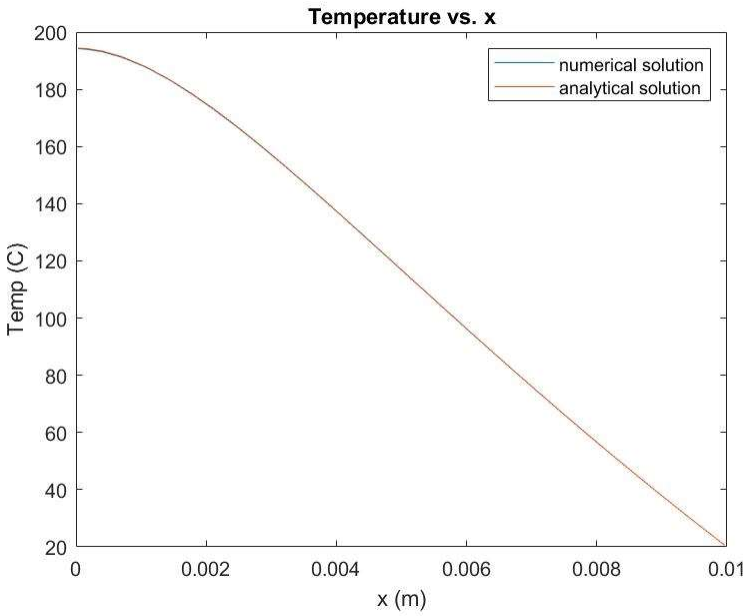


8 elements

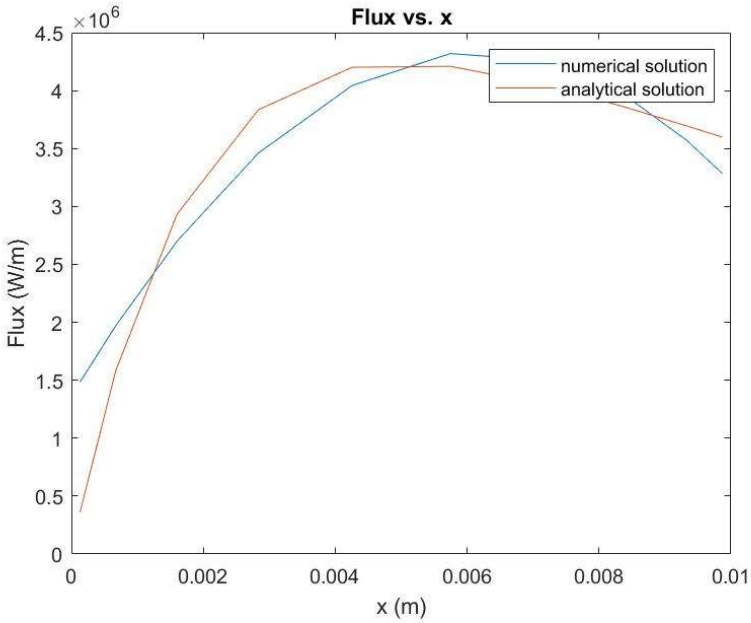
Cubic



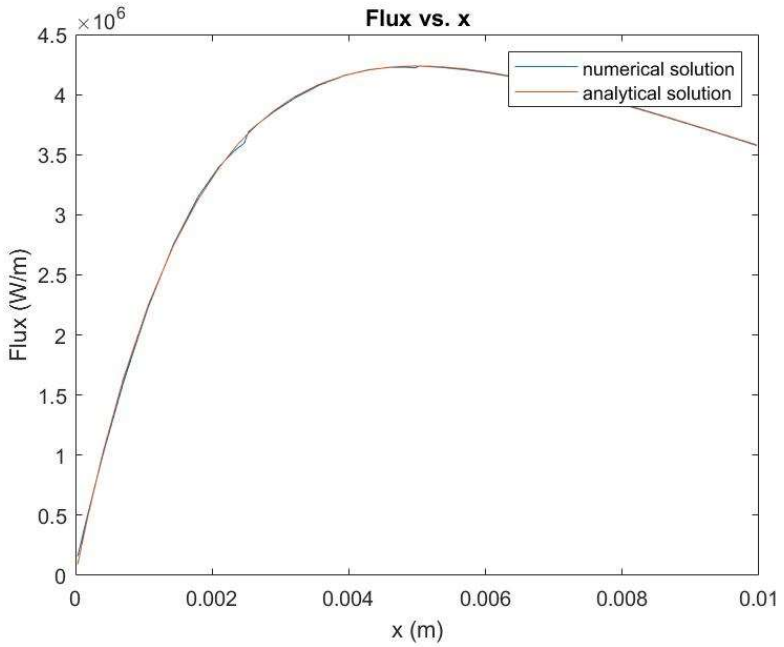
1 elements



4 elements



1 elements



4 elements

As shown by the graphs above, as the number of nodes increases the closer the numerical solution gets to the exact solution.

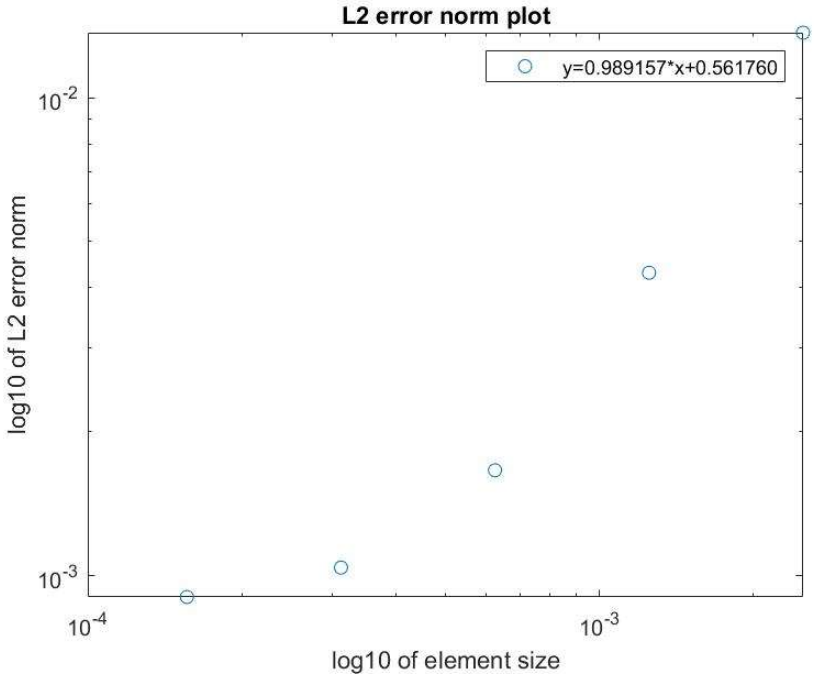
Also, the higher the element order, the less nodes are needed to achieve an accurate result.

Sparse Storage Runtime

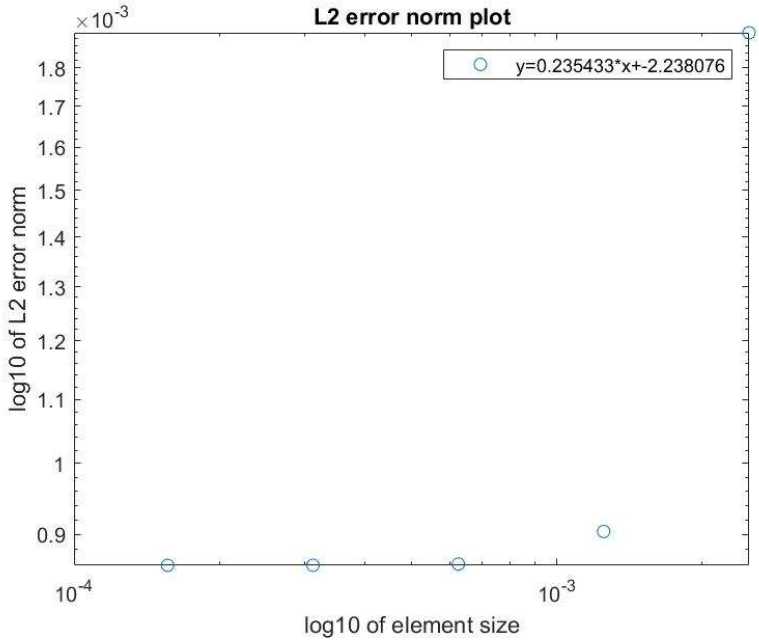
Sparse storage is a way to avoid storing unused values in a matrix, which are set to 0 as default normally. Runtimes were tested for 1000 linear elements. As shown below, there is a significant difference in solving time between the two methods. Assembling does not require as much computation, but there is still a fair improvement percentage-wise. For 1000 linear elements in a 1D problem, the matrix will be smaller than for quadratic or cubic elements. Thus, they will have even bigger improvements in solving and assembling time.

	With Spalloc Function	Using all-zeros matrix
Solution Time (s)	4.534e-04	0.0173252
Assembly Time	0.214846	0.1839807

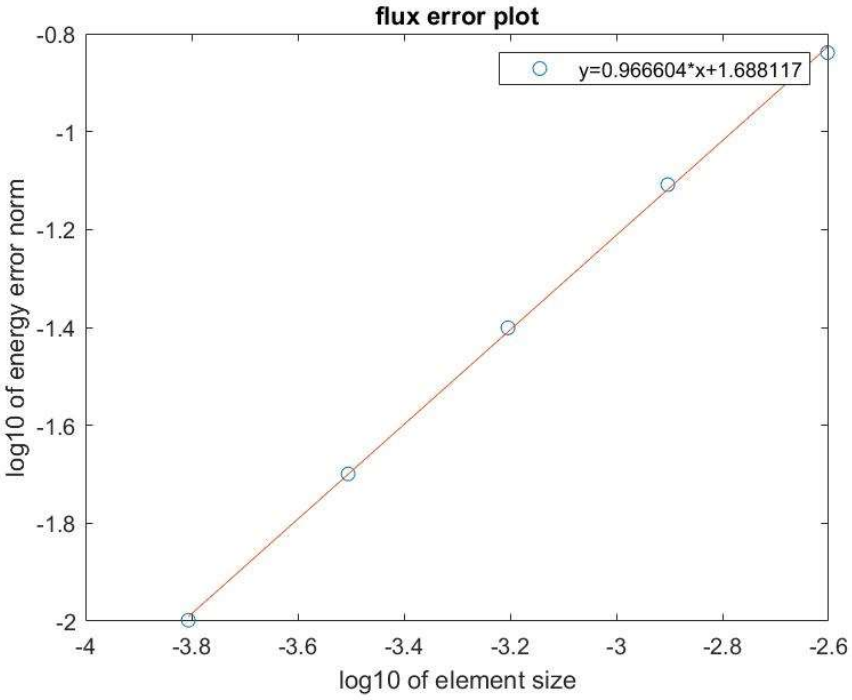
Error Plots



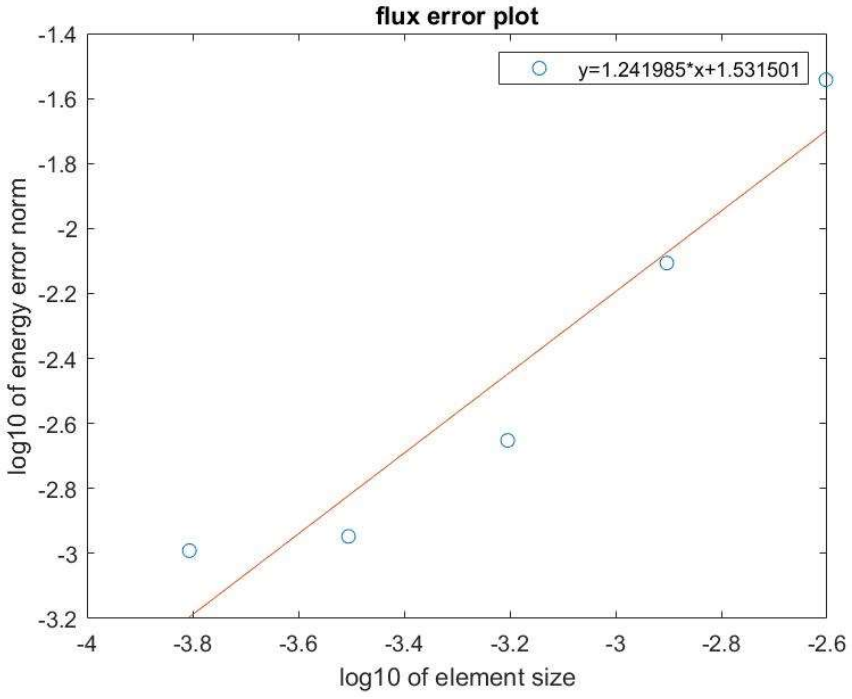
(Linear)



(Quadratic)



(Linear)



(quadratic)

The above plots do not represent the correct relationships. The only correct one is the flux error norm for the linear elements. This has a slope of 1. The post-processing of the matlab code must have some error because the numerical and analytical solutions align very well.

ABAQUS Solution

For the Abaqus solution, only half of the geometry was modeled, so that it could be compared directly to the MATLAB solution. The circle at the midplane was created in a sketch and extruded, with an extrusion angle defined by the geometry of the wire. Thus, the geometry is 3D. There are two boundary conditions. One of them is a body heat flux, which is applied to the entire 3D body, and the other is a fixed temperature condition applied to the external face. The body heat flux varies based on the cross section, so it is defined by the analytical equation below:

pow(1/(3.14159*pow(1+abs(Z/10),2)),2)*2.82*pow(10,-5)*1e6 = body heat flux

where pow(a,b) ← → a^b.

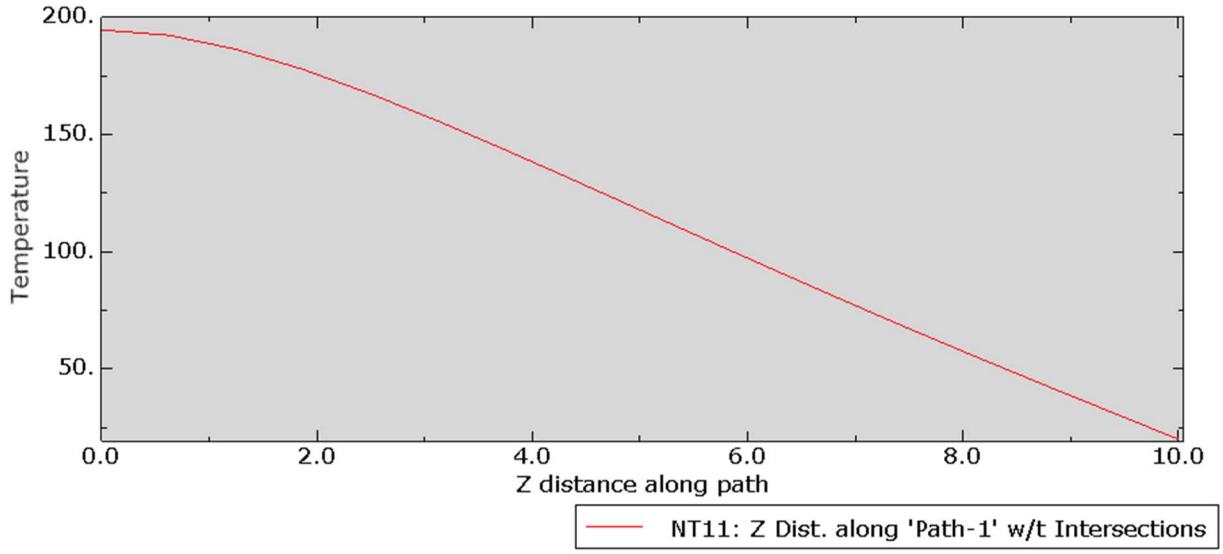
This comes from plugging in the values for the equation

$$s = (I/A)^2 \rho.$$

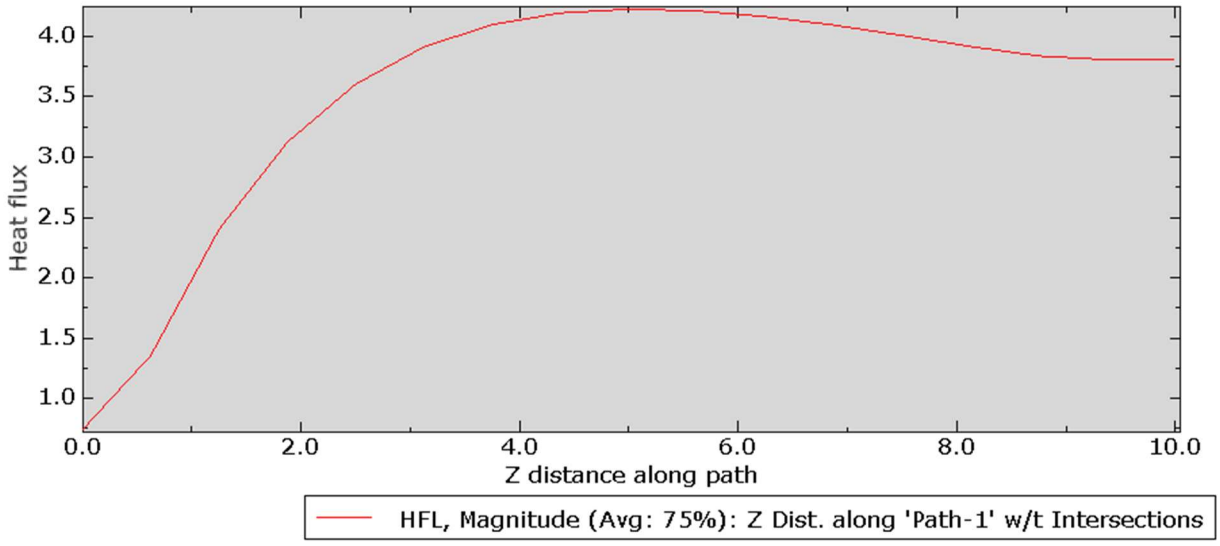
Where A changes based on the Z axis (axis of extrusion) and is given in terms of radius which is a function of x.

The body flux is in units of W/m^3, since it is in 3D. For meshing, the number of nodes was a limitation. This is discussed after the Abaqus plots are displayed. The window that shows element type also shows the type of analysis one is doing. Here, it is important to select the option for Thermal Analysis.

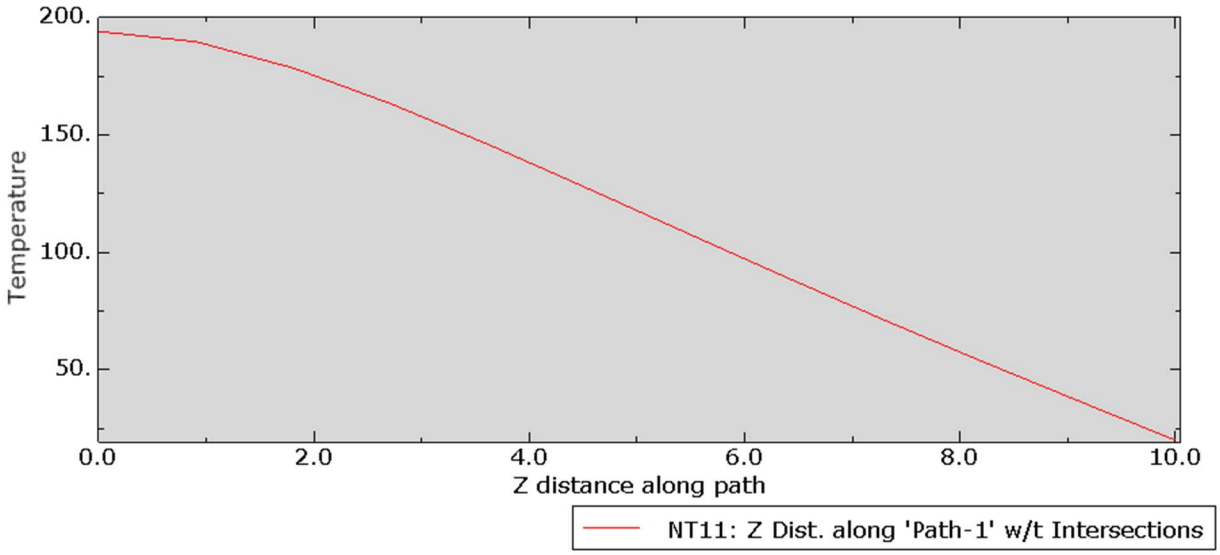
Temperature vs. Z-distance: Linear Elements



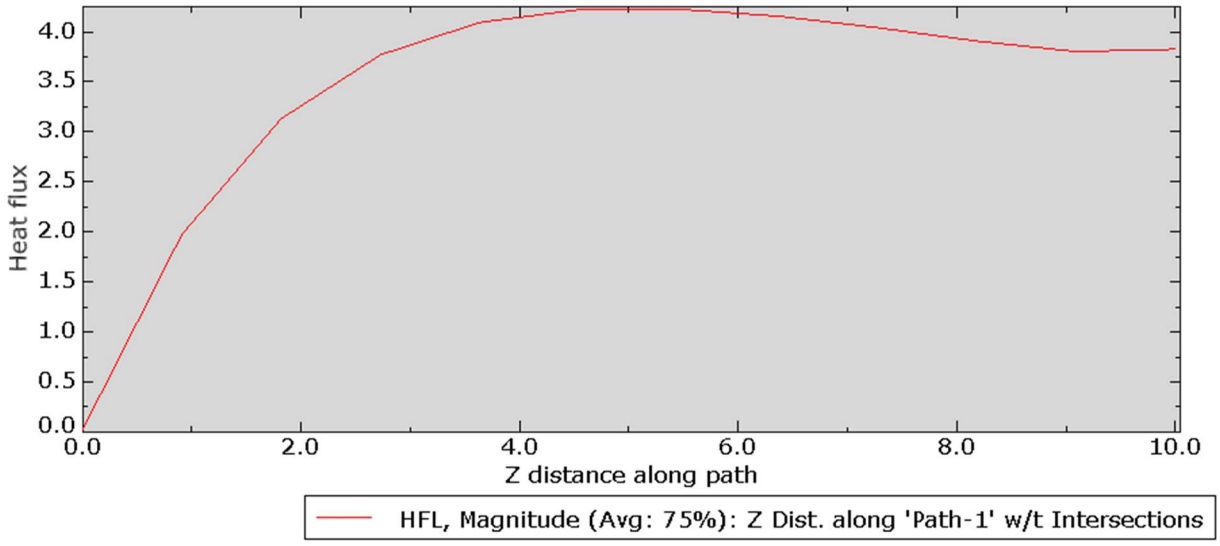
Flux vs. Z-distance: Linear Elements



Temperature vs. Z-distance: Quad Elements



Flux vs. Z-distance: Quad Elements



The solutions for linear and quadratic elements do not differ much in "smoothness" because of the limitations in the student version of Abaqus. The maximum number of nodes was used for both element types, which means they had approximately the same number of nodes in the mesh. Thus, although the linear elements are only able to approximate linearly, the curve looks very similar to the quadratic solution because there are just as many nodes. If the number of elements was kept the same, then we would expect the quadratic curves to be smoother.

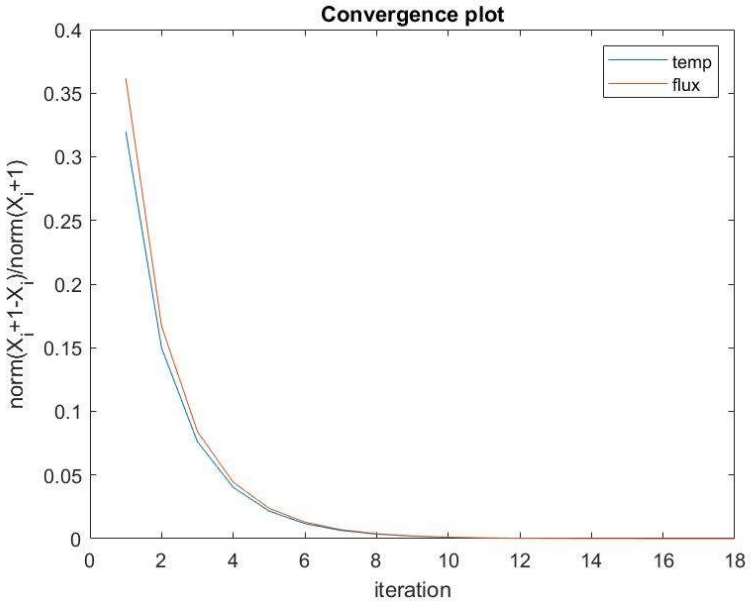
Extra project

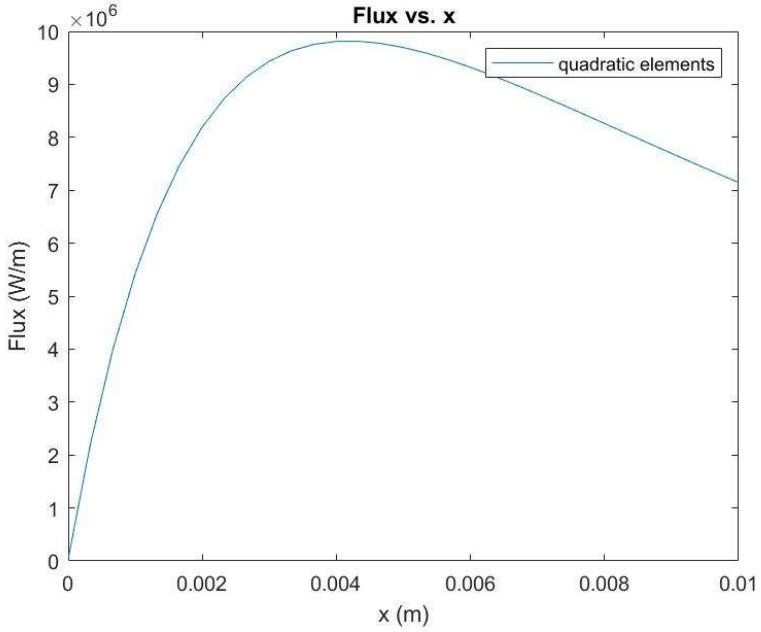
Most of the implementation for the extra credit portion of the project was the same as for the regular project objective. Below is a walkthrough of the key additions to the code:

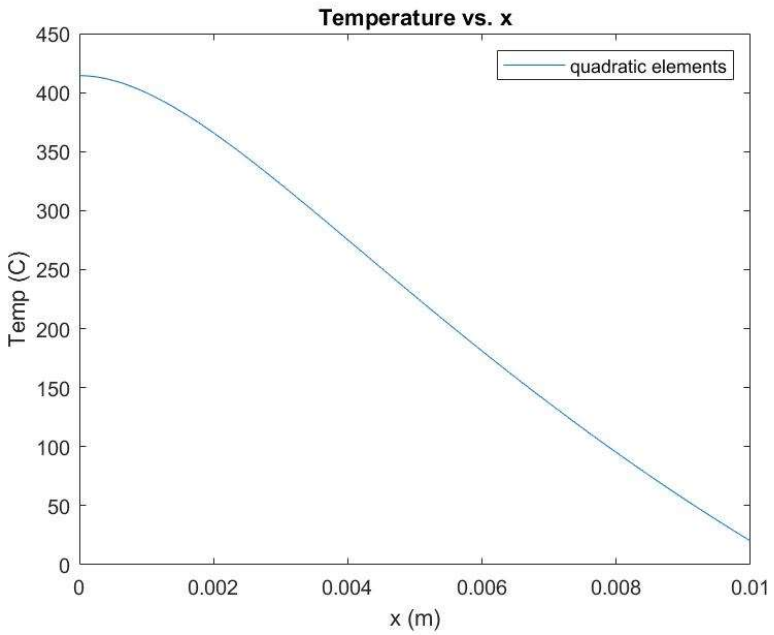
All of the code for solving the temperature and flux fields is placed inside of a while loop. This is to solve iteratively until the solution fields converge. The condition to terminate the while loop is when norm(X_i+1 – X_i)/ norm(X_i+1) is below 1e-5, where this equation is evaluated for X = Temperature field AND X = heat flux field. The plots will show that both of these metrics converge at the solution, as expected.

Unlike the original project, the heat source term is now dependent on temperature because resistivity is dependent on temperature. Thus, a counter variable is used to create the heat source term differently after the first iteration of the while loop. Once the counter is greater than one, the heat source term will use a new definition of rho based on temperature, as defined in the problem statement. The temperature used to evaluate rho is temp_old, which is discussed below. It is the previous iteration's temperature. This heat source is then used to create the element flux, fe, which is scattered into the global flux vector, f, just like in the original problem.

After solving for temperature with the same conductivity matrix, the post-processing is very similar. Flux values for all of the quadrature points within each element are calculated in another for loop, that iterates through all elements. This is the same as the original project. After the first iteration, there is an extra method (in an if statement) that will calculate the norm metrics discussed above to evaluate the while loop the next time. this requires information of the last iteration's flux and temperature fields. Thus, there is a new variable called temp_old and flux_old that will store the current iterations solutions in order to use them on the next iteration. This, of course, is updated every iteration. Once the solution has converged to a satisfactory amount, the while loop will end and the solutions will be plotted. Also, the norm metrics will be plotted versus the number of iterations. This is what the iteration counter is used for. Looking at the code, the length of iteration counter is counter-2 because the first iteration does not have a norm and the counter variable updated one extra time before the termination of the while loop.


Convergence plot
temp
flux
norm(X_i+1-X_i)/norm(X_i+1)
iteration
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
2
4
6
8
10
12
14
16
18


Flux vs. x
×10^6
quadratic elements
Flux (W/m)
x (m)
10
9
8
7
6
5
4
3
2
1
0
0
0.002
0.004
0.006
0.008
0.01



We expect the temperature to be higher because resistivity is higher with the new equation for rho.

Appendix: Matlab code

(Extra credit portion)

```
clc

ne=30; %number of elements
% ne=20;
L=0.01; %length
r0=.002;
r1=.001;
k_matl=205;
current= 1000;

% mesh with linear elements
% a = (1:1:ne);
% b = (2:1:ne+1);
% nn= ne+1; %number of nodes
% conn = [a;b]; %linear

% mesh with quadratic elements
nn= 2*ne+1;
conn = [1:2:nn-2; 2:2:nn-1; 3:2:nn];

%mesh with cubic elements
% nn= 3*ne+1;
% conn = [1:3:nn-3; 2:3:nn-2; 3:3:nn-1; 4:3:nn];

mesh = linspace(0,L,nn); % x coordinates

if size(conn,1)==2 %determines what order of
element ad what shape function to use
    shape=@shape2;
end
if size(conn,1)==3
    shape=@shape3;
end
if size(conn,1)==4
    shape=@shape4;
end
```

```
counter=1; %for intializing things first time thru
while loop
test=[];

e_L2 = [10]; %placeholder
e_en = [10];


%start of recursive solution
while e_L2(end)>1e-5 | e_en(end)>1e-5
clearvars -except counter L r0 r1 k_matl current nn
conn mesh...
    ne temp_old flux_old shape e_L2 e_en

k=zeros(nn);
f=zeros(nn,1); %resetting for next run

count=1; %needed to go thru all temps from old_temp

%for generating stiffness matrix
for c = conn %goes thru all elements
    xe=mesh(:,c); %gets node locations for specific
element
    qpts=quadrature(10); %quadrature point
    ke=zeros(length(c)); %k matrix for element
    fe=zeros(length(c),1); %f matrix for element
    for q=qpts %do the following for each
quadrature point (and its correpsoning weight)
        %constructing ke
        [N, dNdp]=shape(q(1));
        J=xe*dNdp; % dx/dkesi eval at 1st qpoint
        B=dNdp/J; %dN/dx = dN/dkesi * dkesi/dx
        x=xe*N; % interpolates to find proper x
given xe and N, with zeta of given qpt as input for
N.
        ke = ke + B *k_matl*area(x)*B'*J*q(2); %
construct k matrix for that element using
        % gauss quadrature integration
```

```
        %constructing fe
        s = heat_source(x,current); % evaluates
heat source at current and x given
        %
        % if not the first time, s= new heat source
= [x, I , T]
        if counter>1
            %need a T
            T = temp_old(count);
            s=new_heat_source(x,current,T);
            test(count) =  T; %see what is going on
with T value
            count=count+1;
        end

        fe=fe+N*s*J*q(2); %integration of heat
source using gauss quadrature gives flux for node
    end
    %now there is ke and fe for specific element,
needs to be
    %scattered to global k and f matrix/vector
    sctr = c; %gets the nodal values for specific
element
    k(sctr, sctr) = k(sctr, sctr) +ke; %scatters to
the node's spot.
    f(sctr) = f(sctr) +fe; %scatters the heat
source to correct nodes

end


%ex cubic elements have 4 nodes. --> 4x4 ke and 4x1
fe

%boundary conditions in celsius
Tbar=20;
```

```
fixed_temp = nn; %last node in body. Used for
associated locations in k matrix

k(fixed_temp, :)=0; %setting the associated row(s)
to zero
k(fixed_temp,fixed_temp) = eye(length(fixed_temp));
%adding 1's for fixed temps
f(fixed_temp)=Tbar; %assign T to fixed node

% how to impose 0 flux condition? omit flux term

%solve for temp (d)
d=k\f;


%evaulate flux and temp at integration points
(reset if after first iter)

flux = [];
temp=[];
xs = []; % x at quadrature points

%for error norms (reset if after first iter)
top_temp=0;
bot_temp=0;
top_flux=0;
bot_flux=0;

if counter==1 %initialize the temperature storage
variables
temp_old=[];
flux_old=[];
end

for c = conn
    xe=mesh(:,c);
    de=d(c); %gathers the temp field for specific
element's nodes
    for q=qpts
```

```
        %constructing ke
        [N, dNdp]=shape(q(1));
        J=xe*dNdp;
        dNdx= dNdp/J;
        dudx = de'*dNdx; %dT/dx (temp gradient)
        temp(end+1) = de'*N; %interpolates primary
unknown to quad pt.
        flux(end+1)=-k_matl*dudx; % de x dN/dx x k
= q (not in that order necessarily)
        xs(end+1)=xe*N; %finds corresponding x for
quad pt.

        %error norm terms IF beyond first iteration
        if counter>1
        top_temp = top_temp + (temp(end)-
temp_old(length(temp)))^2*J*q(2); %error norm
equations
        bot_temp=  bot_temp + temp(end)^2*J*q(2);
        top_flux=top_flux + (flux(end)-
flux_old(length(flux)))^2*J*q(2);
        bot_flux=bot_flux + flux(end)^2*J*q(2);

        end
    end

end

%error norms for the new vs old
if counter>1 %evaluate the error norms

e_L2(counter-1) = (top_temp/bot_temp)^.5;
e_en(counter-1) = (top_flux/bot_flux)^.5;

end

%assign new soln as old one for next iteration
temp_old=temp;
flux_old=flux;
%end of while loop
```

```
counter=counter+1;

end

figure()
plot(xs, temp);
xlabel('x (m)');
ylabel('Temp (C)');
title('Temperature vs. x')
legend('quadratic elements');

figure()
plot(xs,flux);

xlabel('x (m)');
ylabel('Flux (W/m)');
title('Flux vs. x')
legend('quadratic elements');

iteration_counter=1:counter-2;

figure()
plot(iteration_counter, e_L2)
hold on
plot(iteration_counter, e_en)
legend('temp','flux')
xlabel('iteration');
ylabel('norm(X_i+1-X_i)/norm(X_i+1)');
title('Convergence plot')



%heat source term at given x value
function [s] = heat_source(x, current)
   rho = 2.82e-8;  % in Ohm-m
   s = current^2  / area(x) * rho;
end

function [N, dNdp] = shape2(p)
```

```
% Define linear shape functions and derivatives
zeta=p(1);
N = 1/2 *[1-zeta; 1+zeta];
dNdp=[-.5;.5];
end

function [N, dNdp] = shape3(p)
% Define quadratic shape functions and derivatives

zeta=p(1);

N=[zeta*(zeta-1)/2; (1+zeta)*(1-zeta);
zeta*(zeta+1)/2;];
dNdp=[zeta-.5; -2*zeta; zeta+.5];

end

function [N, dNdp] = shape4(p)
% Define cubic shape functions and derivatives
zeta=p(1);

N=-9/16*[(zeta+3^-1)*(zeta-3^-1)*(zeta-1); ...
    -3*(zeta+1)*(zeta-1)*(zeta-3^-1); ...
    3*(zeta+1)*(zeta-1)*(zeta+3^-1); ...
    -(zeta+3^-1)*(zeta-3^-1)*(zeta+1)];

dNdp=-9/16*[3*zeta^2-2*zeta+1/3-4/9; ...
    -9*zeta^2+2*zeta+3; ...
    9*zeta^2+2*zeta-3; ...
    -3*zeta^2-2*zeta+1/3-2/9];
end

function [A] = area(x)
L=.01;
r0=.002;
r1=.001;
  r = r1+(r0-r1)*abs(x/L);
  A = pi * r^2;
end
```

```
function [rho] = resistivity(T)
rho=2.6e-8+1.1e-10*T; %ohm-m?

end

function [s]=new_heat_source(x,current,temp)
rho = resistivity(temp);
s = current^2  / area(x) * rho;
end
```