MAE 503 Project 1

Steven Van Overmeiren

March 2025

1 Introduction

The objective of this project is to approximate the steady-state temperature field of a given Nichrome wire geometry using the finite element method in both a MATLAB coding environment and an ABAQUS simulation environment. One of the main goals is to compare and contrast the solutions of both types of finite element analysis. Key insights of this project lie in identifying the differences in each model approximation, exact solutions, and rates of convergence. The geometry of interest and problem setup are described as follows:

An electrical current of 10 Amps passes through a Nichrome wire (an alloy of nickel and chromium). The wire has a square cross-section that tapers from an edge length of h1 = 4 mm at the ends to an edge length of h2 = 1 mm at the center. The length of the wire is 100 mm (L = 50 mm).

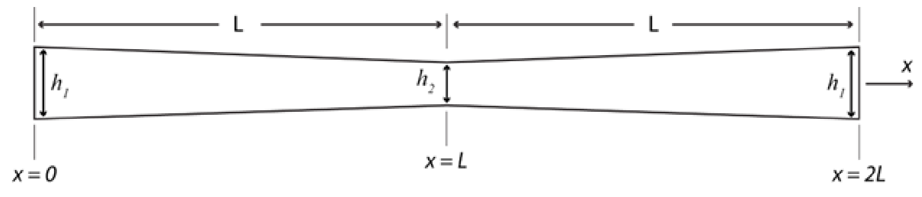


Figure 1: Nichrome Wire Geometry

The temperature at both ends of the rod is fixed at 23°C. Write a finite element program in MATLAB to approximate the steady-state temperature field and compare it with analytical solutions and an approximate solution from ABAQUS. The project tasks are listed below:

A. Describe how you model this problem in ABAQUS including the geometry approximation (1D, 2D, 3D), symmetry conditions, analysis type, element type used, boundary conditions applied, and heat loads specified. You should provide enough detail that would allow a knowledgeable user to duplicate your result.

B. Plot the steady state solutions of the temperature field from your MATLAB code and ABAQUS on the same plot. Use any element order, but make sure that both solutions have enough elements (at least 30) to verify whether both solutions are the same. If the solutions differ, explain why (e.g., are the model approximations different?)

C. Using your MATLAB code, plot the temperature field and heat flux for the following cases listed below.

- Linear (2-node) elements: 4, 8, 16 elements
- Quadratic or cubic elements: 1, 2, 4 elements

D. Compute the exact solution to this problem using the symbolic math package in MATLAB. Plot the exact solution and the FEM approximation of the temperature field on the same plot, using enough elements (at least 30) to show they are the same.

E. Compute the rates of convergence of the temperature field of both linear and higher order (quadratic or cubic) element types by plotting the logarithm of the L2 error norm against the logarithm of the element size for solutions with 4, 8, 16, 32, 64, ... elements. Comment on whether the expected rates of convergence are achieved.

A. I modeled this problem in ABAQUS by choosing a model database with a standard/explicit model at launch. Within the database, I navigated to the model tree on the left-hand side and created a new part, ensuring that it was a three-dimensional and deformable solid in the Create Part menu. A decision was made to consider only half of the model geometry and to assume symmetry with its properties in an effort to simplify the modeling process. Within the Create Part menu, I set the approximate size of the part to be 50, to reflect the domain size as half the wire, 50 mm. Because of the wire's square cross section, I selected Create Rectangle in the resulting sketch plane. I placed two opposite corners of the rectangle at (-0.5, -0.5) and (0.5, 0.5) to define a square with 1 mm sides. In the resulting extrusion menu, I set the depth to be 50 and added a draft to the extrusion to ensure the wire tapered to a 1 mm thickness at the very end. This draft was determined by a quick geometrical relation of half the wire:

$$\theta = \arctan\left(\frac{(h_1 - h_2)/2}{L}\right) \tag{1}$$

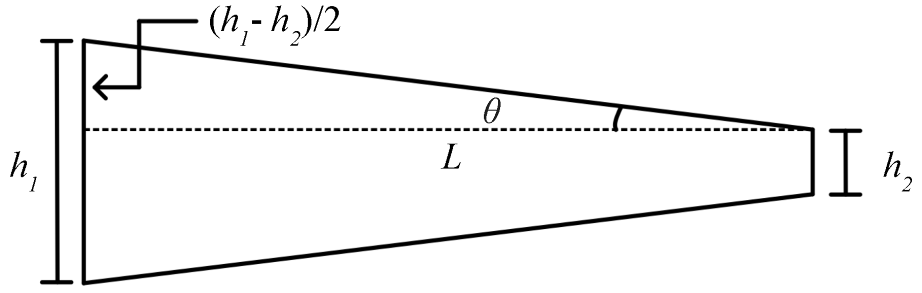


Figure 2: Half Nichrome Wire Taper Relation (not to scale)

As h_1, h_2, and L are all given as 4 mm, 1 mm, and 50 mm, respectively, the resulting taper θ was determined to be 1.71836° using Equation 1. Once half the wire with the taper was extruded, I linked the thermal conductivity of Nichrome to a material by selecting Create Material. I inputted the given value of k = 0.012 $\frac{W}{mm-K}$. Then, I created a section for the wire so that it was defined to be solid and homogeneous with the thermal conductivity that was entered. After that, I assigned the section to the model by clicking Assign Section and selecting the model body. At this point, the model geometry and properties have been established for a steady-state temperature simulation. The only thing left to do was to properly mesh the geometry and solve for the temperature distribution. In the model tree under Part, I clicked the drop down box and selected Mesh, then Seed Mesh. In the Seed Mesh menu, I assigned an approximate global size of 0.5 and kept the rest of the parameters the same. With the global size set, I selected Mesh Part to generate a mesh of 1600 elements. In order to let ABAQUS know that the mesh will be used for heat transfer analysis, I created an Instance of the part under Assembly. Under this Instance, I made the mesh for the part independent and selected Assign Element Type. This menu is where the element type can be set to a standard heat transfer configuration and given a linear order. After the element type was defined, a heat transfer step was added in the model after the initial step that ensured the resulting analysis was for a steady-state response. In order for ABAQUS to properly mimic the heat flux, it was important to place an Analytical Field with a governing expression. The expression input in the Analytical Field was: 0.15/pow((4-3*(50-Z)/50),4) formatted for ABAQUS syntax. Then a body heat flux Load was placed on the model using the Analytical Field that was created. To establish the given temperature boundary condition for the wire, I selected BC in the model tree and selected Create. In this menu, I inputted the 23°C fixed temperature of the rod in Kelvin (296.15) and placed it on the 4 mm x 4 mm face. Then, under Analysis, a Job was created to tell ABAQUS that I intended to run a full analysis on this geometry. After right-clicking the Job and clicking submit, the heat transfer problem was successfully modeled in ABAQUS and was ready for XY output [2].

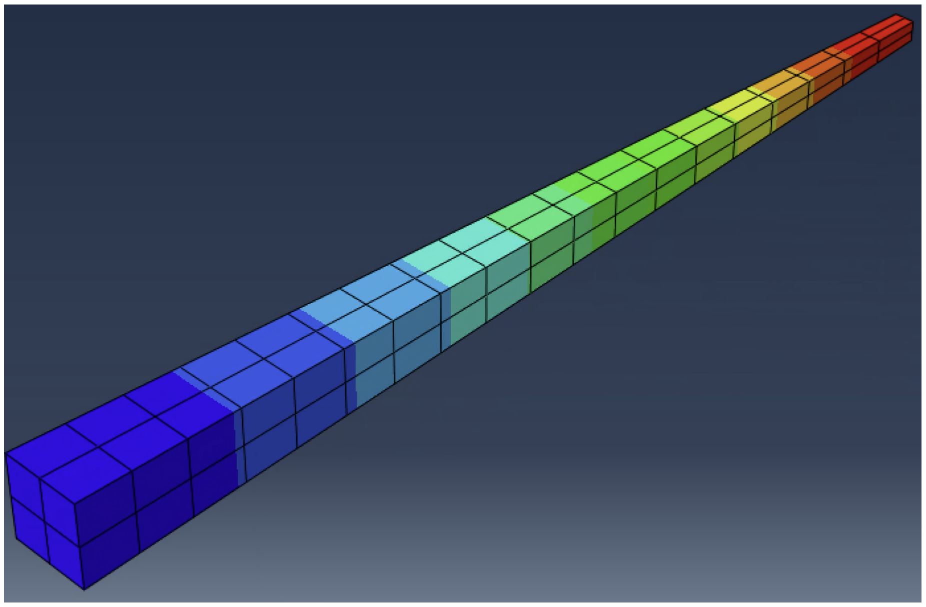


Figure 3: ABAQUS Geometry with Nodal Temperature Contours

B. After creating the geometry and modeling the heat transfer condition in ABAQUS, I constructed the geometry in MATLAB and modeled the heat transfer by deriving an area equation as a function of x,

$$A(x) = [\frac{h_2 + (h_1 - h_2)(L - x)}{L}]^2 \tag{2}$$

and referencing the given heat generation equation,

$$s(x) = \frac{\rho^e I^2}{A(x)} \tag{3}$$

as well as implementing functions that were written in the class in the past for homework assignments and demonstrations. The MATLAB code iterates across the number of elements (76 in this case) using Equations 2 and 3, integrates using Gauss points and their associated weights, and assembles stiffness and heat load matrices to solve for temperature. With the solved temperature (T) and its associated position (x), an interpolation function was used to fit a line to the data points. The XY data from ABAQUS was then overlayed on top to draw a comparison between the two finite element techniques.

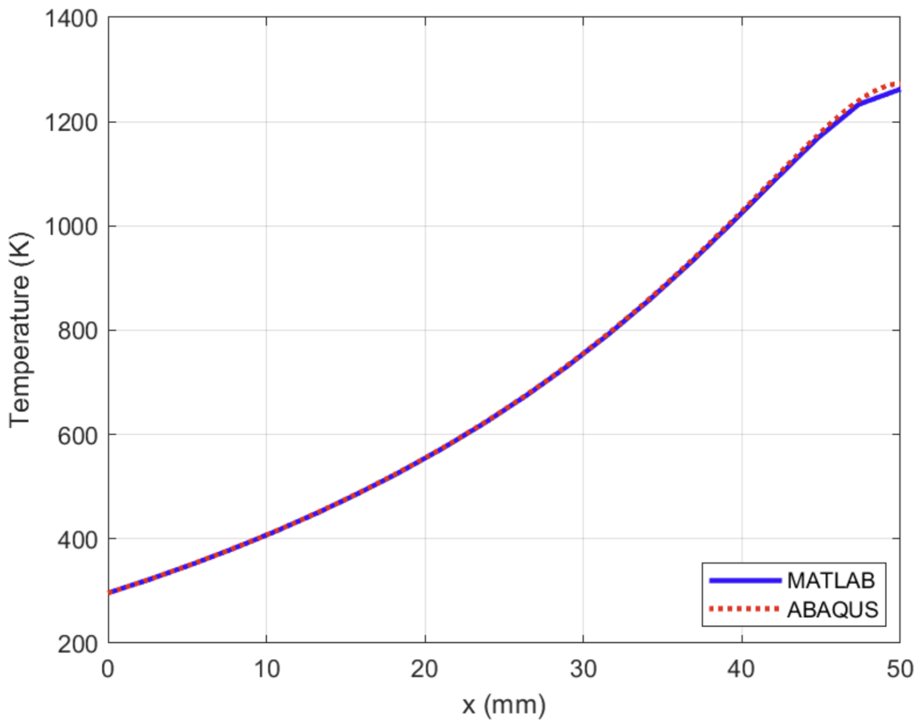
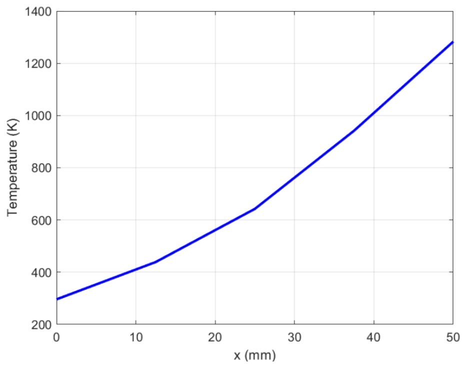


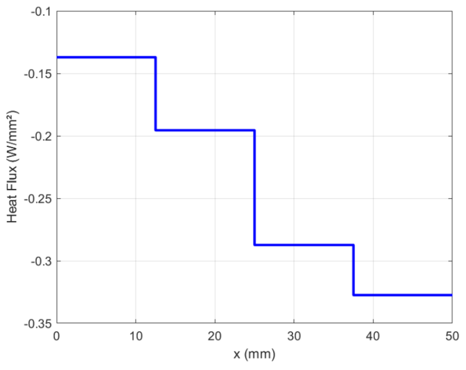
Figure 4: Steady-State Solutions of the Temperature Field Using Finite Elements

From observing the plot, the two solutions very closely mimic each other until around the 45-50 mm mark, in which the MATLAB solution begins to sharply taper off while the ABAQUS solution smoothly tops out. The most likely reason for this is differences in how the model was approximated in ABAQUS vs MATLAB. ABAQUS may use a different interpolation method or higher order integration. Additionally, the method in which the geometry was created could be significant, as in ABAQUS it was made as an extruded draft of a cross section vs a geometrical relation as defined in the MATLAB code, which could potentially affect the element shapes and thus the solved temperature.

C. The temperature and heat flux for the wire (still with one side) were calculated and plotted in MATLAB using both linear and quadratic elements and with a varying amount of elements in each to see their effect.

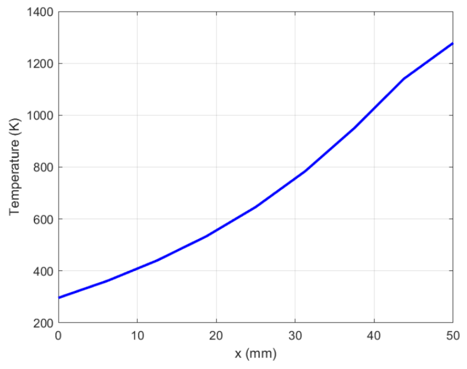


(a) Temperature

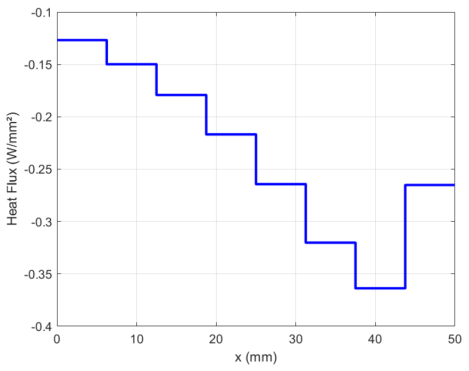


(b) Heat Flux

Figure 5: 4 Linear Elements

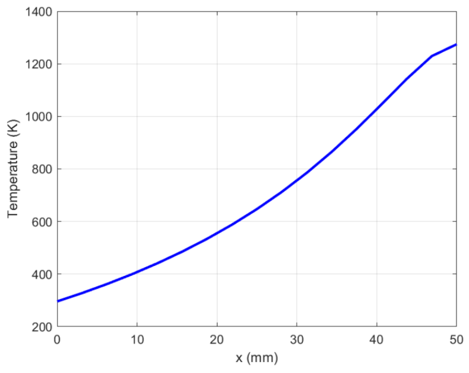


(a) Temperature

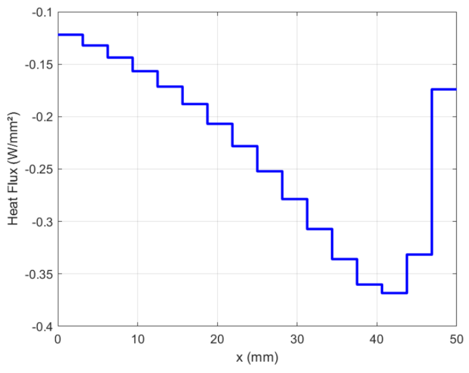


(b) Heat Flux

Figure 6: 8 Linear Elements

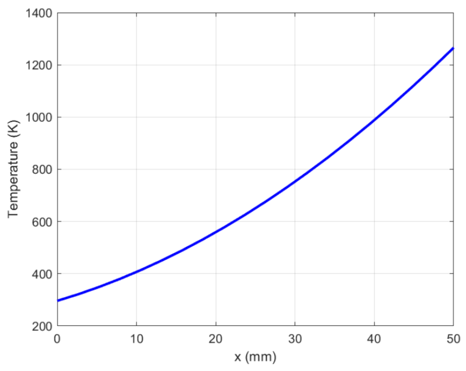


(a) Temperature

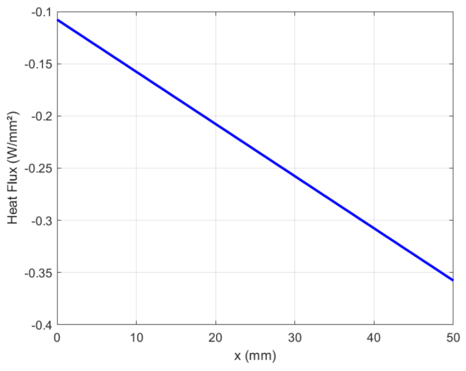


(b) Heat Flux

Figure 7: 16 Linear Elements

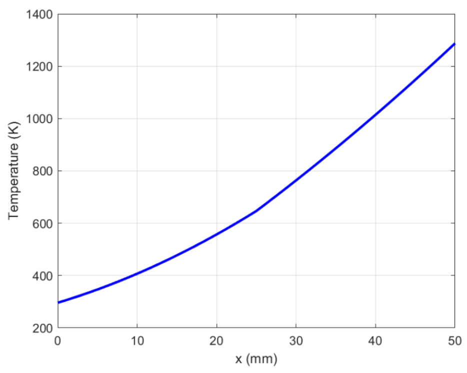


(a) Temperature

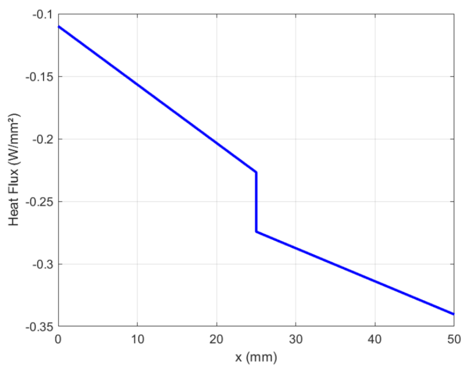


(b) Heat Flux

Figure 8: 1 Quadratic Element

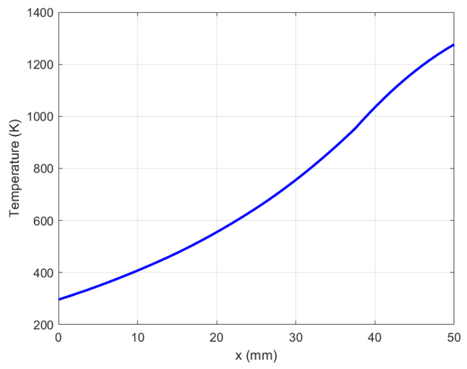


(a) Temperature

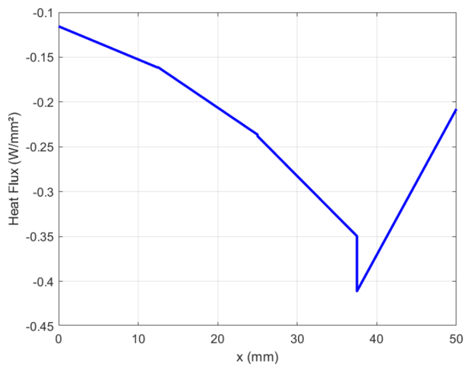


(b) Heat Flux

Figure 9: 2 Quadratic Elements



(a) Temperature



(b) Heat Flux

Figure 10: 4 Quadratic Elements

D. The exact solution to the problem was then calculated utilizing the symbolic math package in MATLAB. Similarly to Part B, this solution was compared to the FEM approximation solved for in ABAQUS. The element size for both of these finite element approximations was 76 elements. As seen below, the curves follow each other closely until the very end.

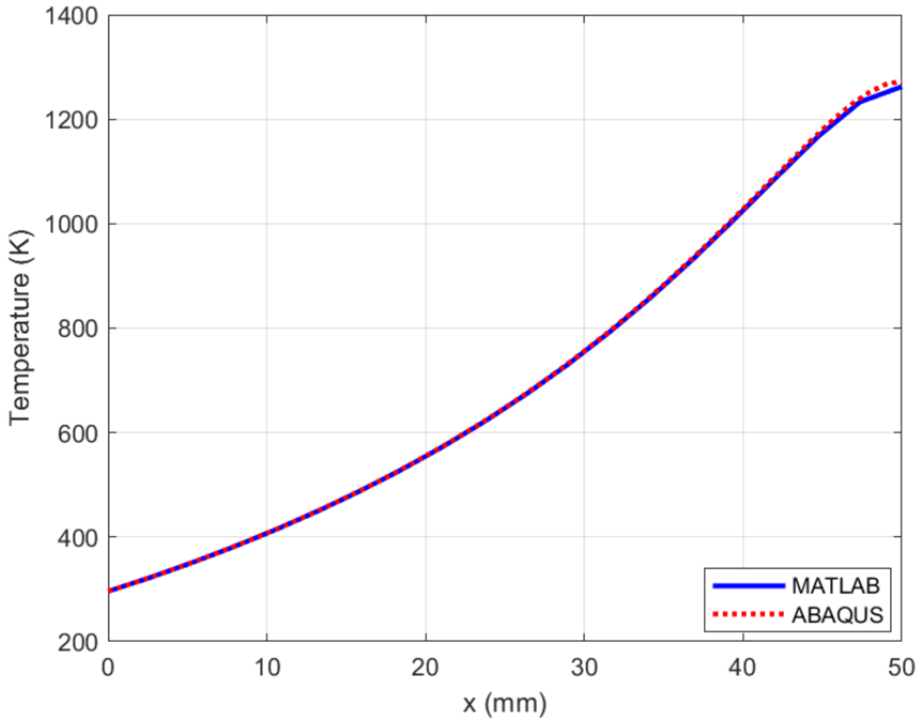


Figure 11: Exact Steady-State Solution of Temperature Field

E. Lastly, the rates of convergence of the temperature field of both linear and quadratic element types were computed by plotting the logarithm of the L2 error norm against the logarithm of the element size for solutions with 4, 8, 16, 32, and 64 elements. The rates of convergence shown below are expected, as the quadratic elements illustrate a higher L2 error norm per increase in element size than the linear elements as a result of the higher order of polynomial utilized in the FEM [1].

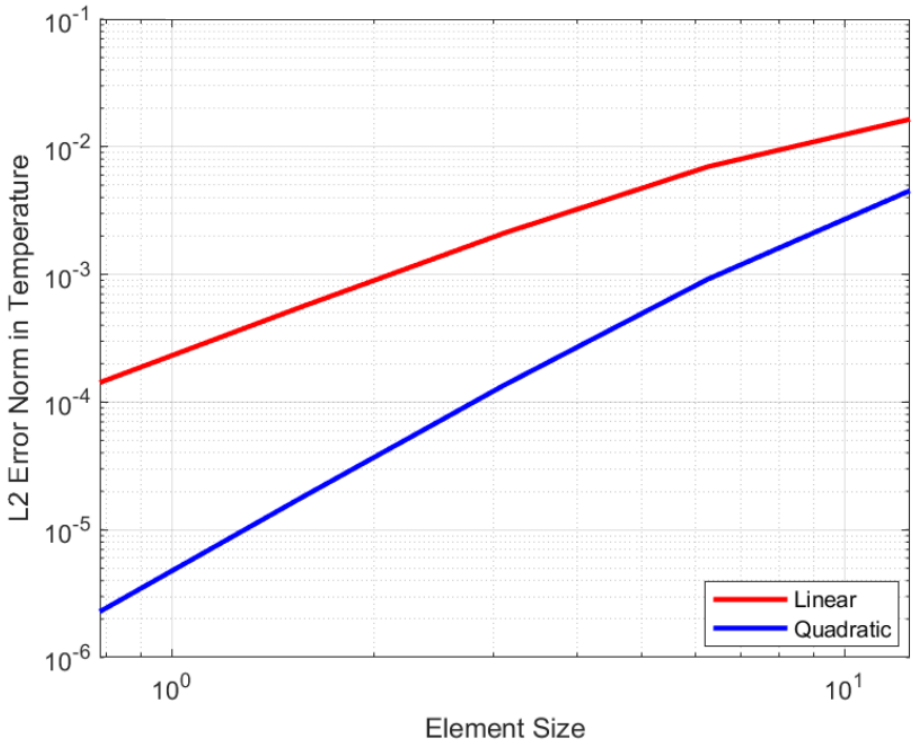


Figure 12: Rates of Convergence of Temperature Field

2 References

References

[1] Fish and Belytschko. *A First Course in Finite Elements*. Wiley, 2007.

[2] Jay Oswald. "Finite Elements in Engineering". MATLAB Codes and ABAQUS Notes.

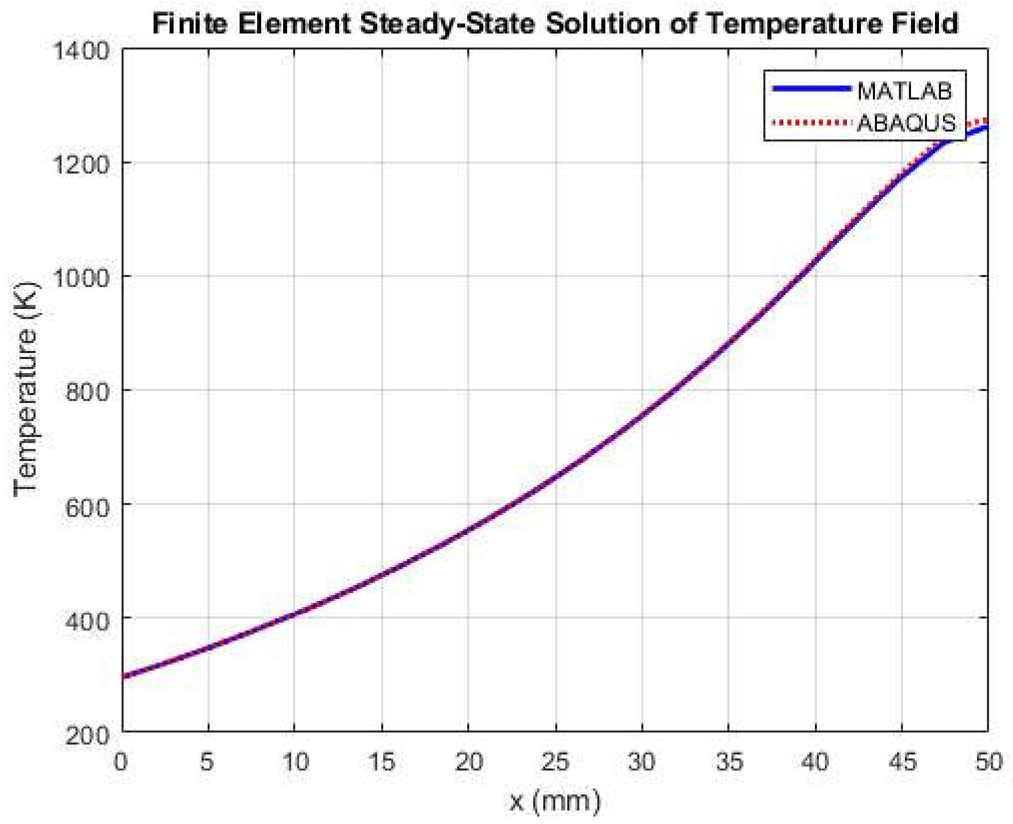
Contents

```
clear all;
clc;
```

Part B

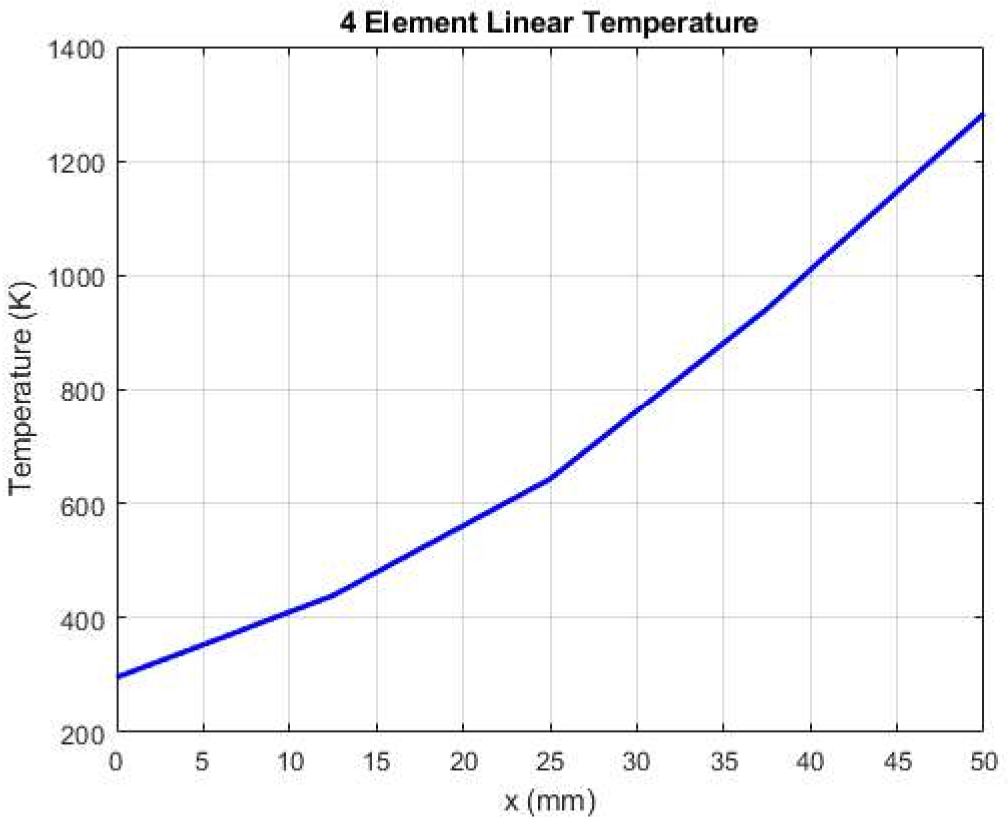
```
L = 50;
h1 = 4;
h2 = 1;
k = 0.012;
rhoe = 0.0015;
I = 10;
T_boundary = 296.15;
taper = (h1 - h2)/L;
area_x = @(x) (h2+(h1-h2)*(L-x)/L).^2;
s_x = @(x) (rhoe*I^2)./area_x(x);
ne = 76;
nn = 2*ne + 1;
mesh.x = linspace(0, L, nn)';
mesh.conn = [1:2:nn-2; 2:2:nn-1; 3:2:nn];
mesh.shape = @shape3;
K = zeros(nn, nn);
load = zeros(nn, 1);
gauss_points = [-sqrt(3/5), 0, sqrt(3/5)];
weights = [5/9, 8/9, 5/9];
for e = 1:ne
    c = mesh.conn(:,e);
    xe = mesh.x(c);
    Ke = zeros(3,3);
    load_e = zeros(3,1);
    for g = 1:length(gauss_points)
        p = gauss_points(g);
        w = weights(g);
        [N, dNdp] = shape3(p);
        J = xe'*dNdp;
        x_value = N'*xe;
        A = area_x(x_value);
        s_value = s_x(x_value);
        dNdx = dNdp/J;
        Ke = Ke+k*A*(dNdx*dNdx')*J*w;
        load_e = load_e+s_value*N*J*w;
    end
    K(c, c) = K(c, c) + Ke;
    load(c) = load(c) + load_e;
end
K(1,:) = 0;
```

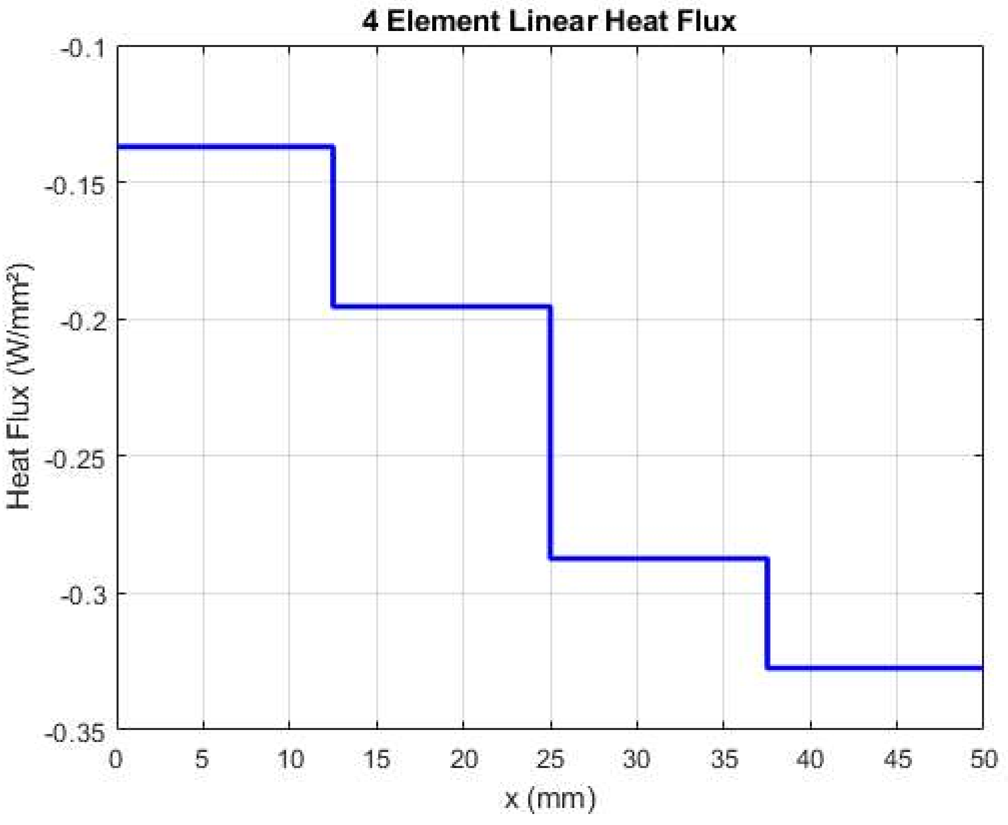
```
K(1,1) = 1;
load(1) = T_boundary;
T = K\load;
[x, T, dTdx] = interpolate(mesh, T);
abaqus_data = readtable("abaqus_data.xlsx",'VariableNamingRule', 'preserve');
x_abaqus = abaqus_data{:,2};
T_abaqus = abaqus_data{:,3};
figure;
plot(x_abaqus, T_abaqus, 'b-', LineWidth=2)
hold on;
plot(x, T, 'r:', LineWidth=2);
xlabel('x (mm)');
ylabel('Temperature (K)');
title('Finite Element Steady-State Solution of Temperature Field');
legend('MATLAB','ABAQUS');
grid on;
```

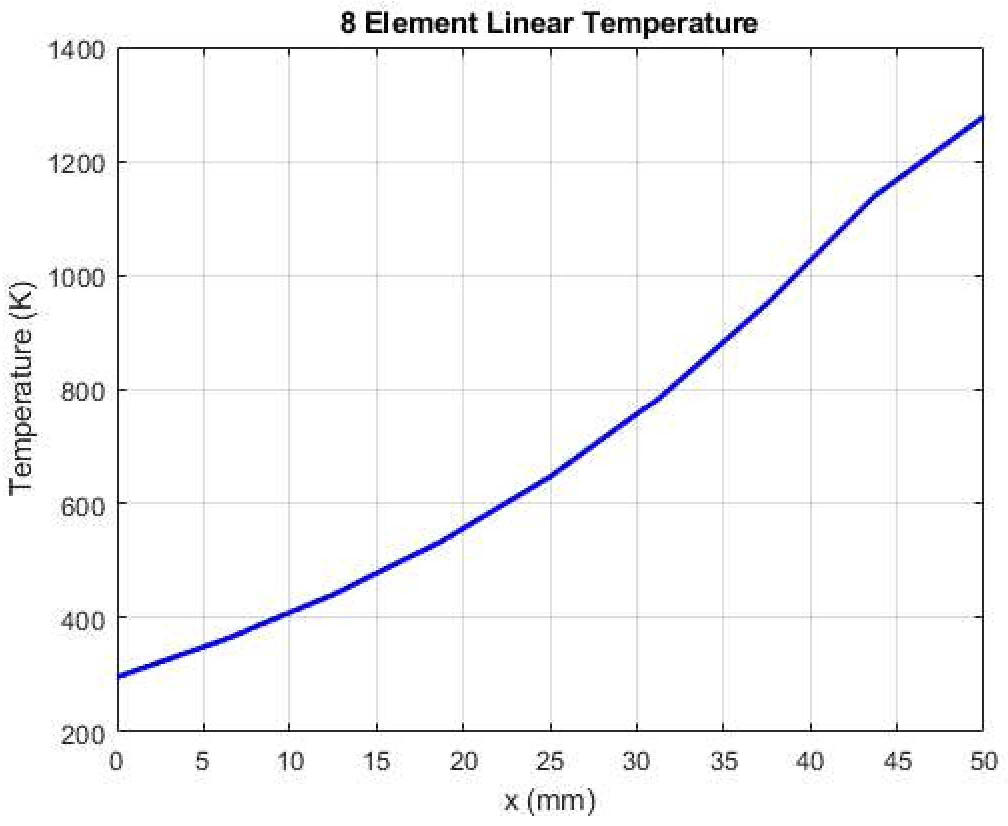


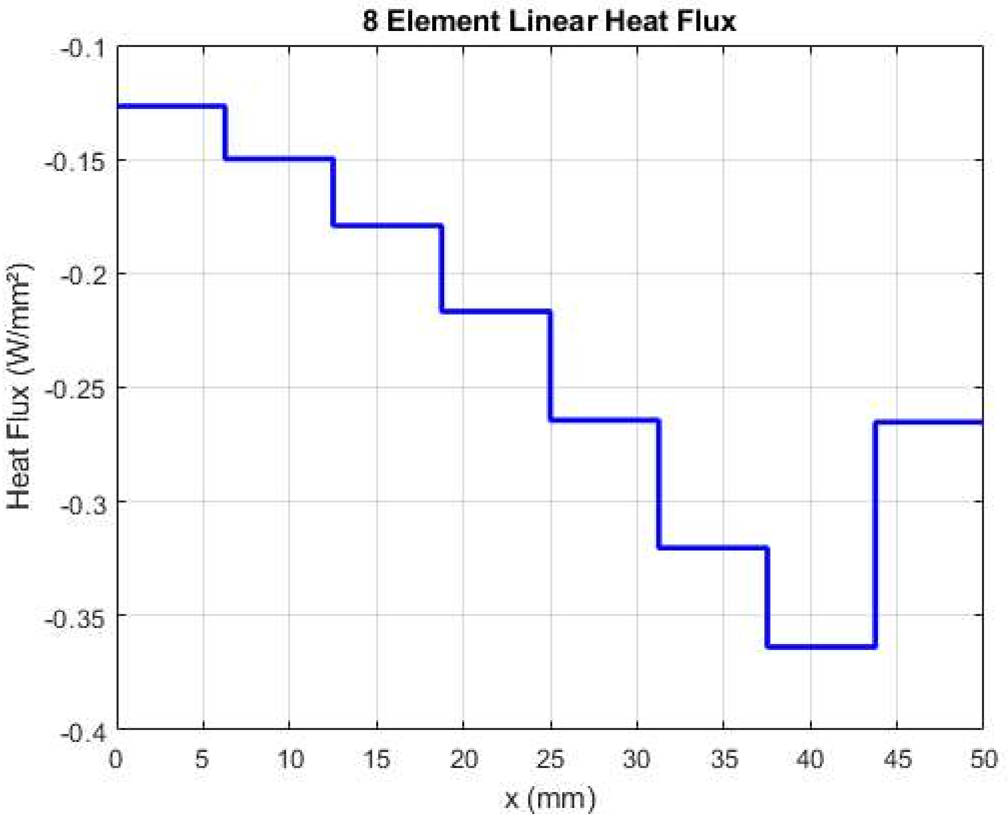
Part C

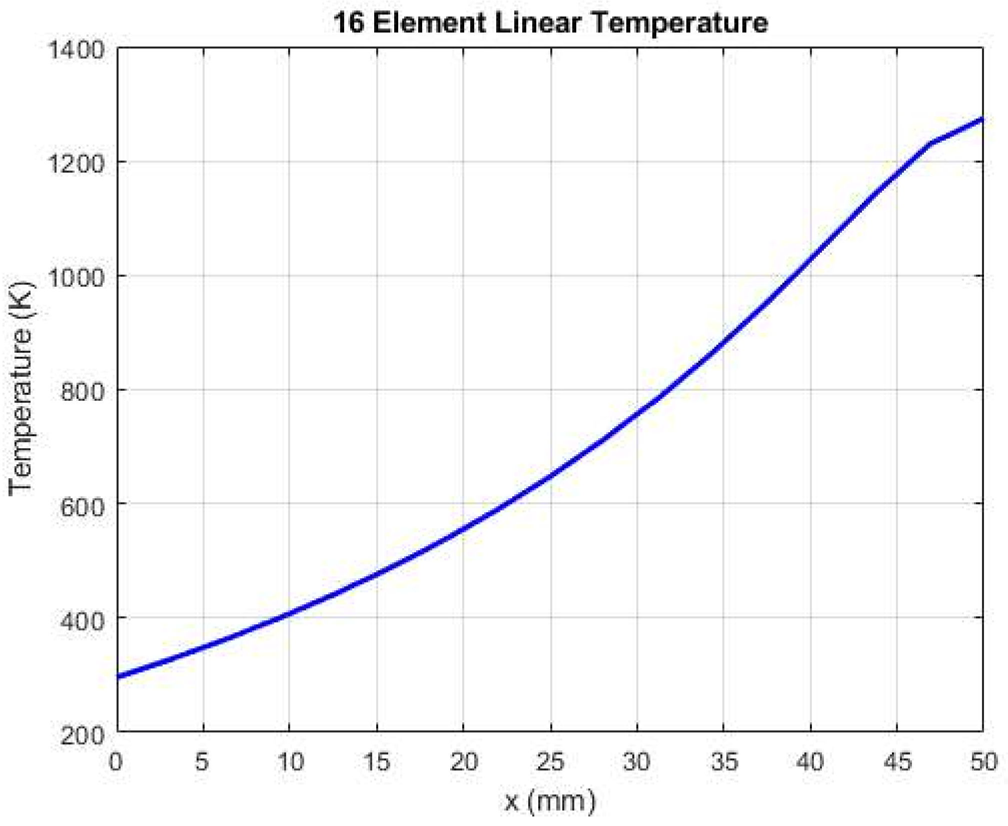
```
linear = [4, 8, 16];
quadratic = [1, 2, 4];
for i = 1:length(linear)
    ne = linear(i);
    nn = ne+1;
    mesh.x = linspace(0, L, nn)';
    mesh.conn = [1:nn-1;2:nn];
    mesh.shape = @shape2;
    K = zeros(nn, nn);
    load = zeros(nn,1);
    for e = 1:size(mesh.conn,2)
        c = mesh.conn(:,e);
        xe = mesh.x(c);
        Ke = zeros(size(c,1), size(c,1));
        load_e = zeros(size(c,1), 1);
        for g = 1:length(gauss_points)
            p = gauss_points(g);
            w = weights(g);
            [N, dNdp] = mesh.shape(p);
            J = xe'*dNdp;
            x_value = N'*xe;
            A = area_x(x_value);
            s_value = s_x(x_value);
            dNdx = dNdp/J;
            Ke = Ke+k*A*(dNdx*dNdx')*J*w;
            load_e = load_e+s_value*N*J*w;
        end
        K(c, c) = K(c, c)+Ke;
        load(c) = load(c)+load_e;
    end
    K(1,:) = 0;
    K(1,1) = 1;
    load(1) = T_boundary;
    T = K\load;
    [x, T, dTdx] = interpolate(mesh, T);
    figure;
    plot(x, T, 'b-', LineWidth=2);
    xlabel('x (mm)');
    ylabel('Temperature (K)');
    title(sprintf('%d Element Linear Temperature', ne));
    grid on;
    figure;
    plot(x, -k*dTdx, 'b-', LineWidth=2);
    xlabel('x (mm)');
    ylabel('Heat Flux (W/mm²)');
    title(sprintf('%d Element Linear Heat Flux', ne));
    grid on;
end

for i = 1:length(quadratic)
    ne = quadratic(i);
    nn = 2*ne+1;
    mesh.x = linspace(0, L, nn)';
    mesh.conn = [1:2:nn-2;2:2:nn-1;3:2:nn];
    mesh.shape = @shape3;
    K = zeros(nn, nn);
    load = zeros(nn,1);
    for e = 1:size(mesh.conn,2)
        c = mesh.conn(:,e);
        xe = mesh.x(c);
        Ke = zeros(size(c,1), size(c,1));
        load_e = zeros(size(c,1), 1);
        for g = 1:length(gauss_points)
            p = gauss_points(g);
            w = weights(g);
            [N, dNdp] = mesh.shape(p);
            J = xe'*dNdp;
            x_value = N'*xe;
            A = area_x(x_value);
            s_value = s_x(x_value);
            dNdx = dNdp/J;
            Ke = Ke+k*A*(dNdx * dNdx')*J*w;
            load_e = load_e+s_value*N*J*w;
        end
        K(c, c) = K(c, c)+Ke;
        load(c) = load(c)+load_e;
    end
    K(1,:) = 0;
    K(1,1) = 1;
    load(1) = T_boundary;
    T = K\load;
    [x, T, dTdx] = interpolate(mesh, T);
    figure;
    plot(x, T, 'b-', LineWidth=2);
    xlabel('x (mm)');
    ylabel('Temperature (K)');
    title(sprintf('%d Element Quadratic Temperature', ne));
    grid on;
    figure;
    plot(x, -k*dTdx, 'b-', LineWidth=2);
    xlabel('x (mm)');
    ylabel('Heat Flux (W/mm²)');
    title(sprintf('%d Element Quadratic Heat Flux', ne));
    grid on;
end
```

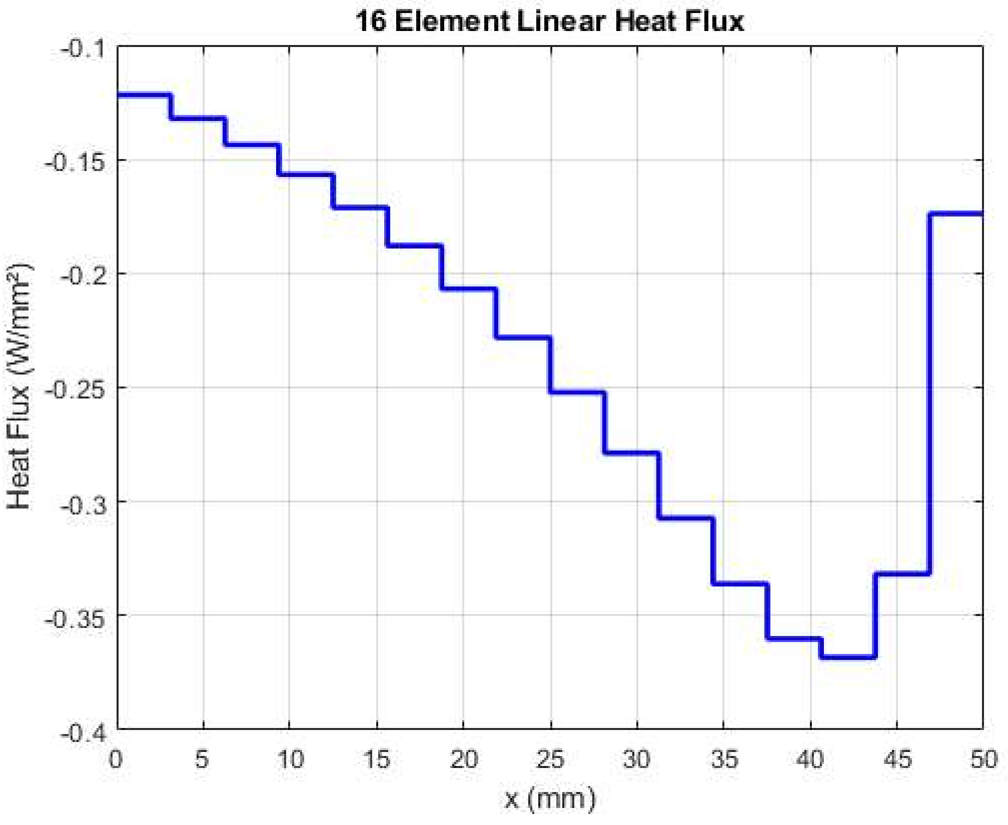

4 Element Linear Temperature
Temperature (K)
1400
1200
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50
x (mm)

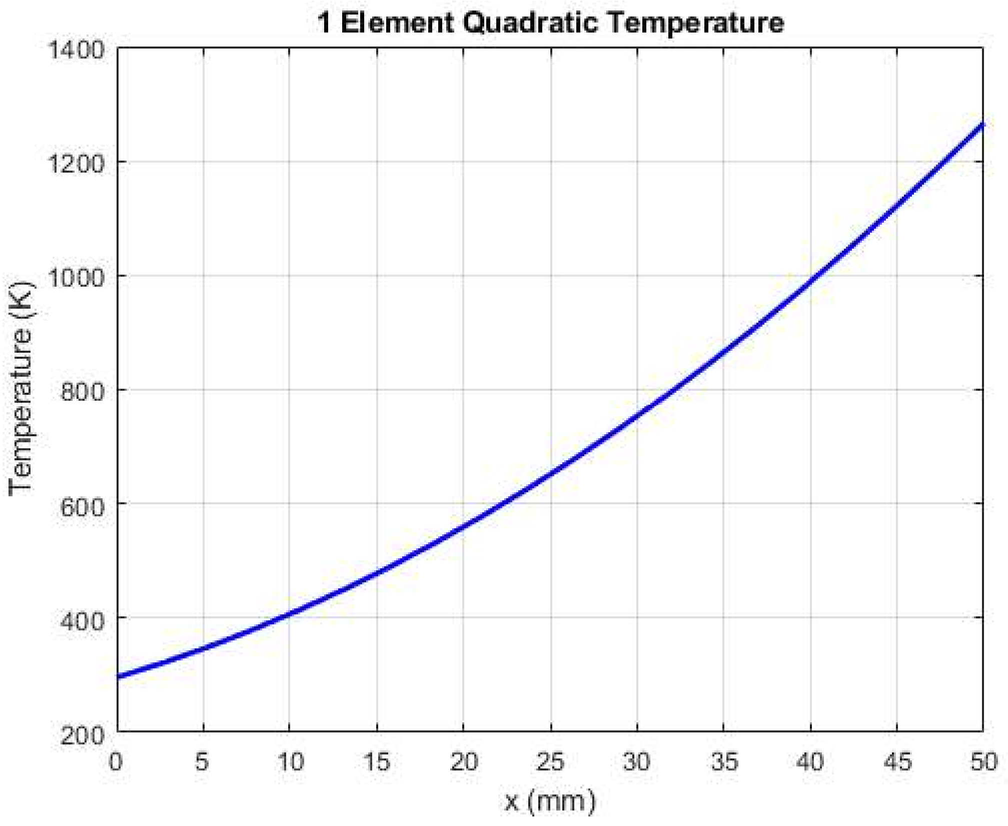

4 Element Linear Heat Flux
Heat Flux (W/mm²)
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
0
5
10
15
20
25
30
35
40
45
50
x (mm)

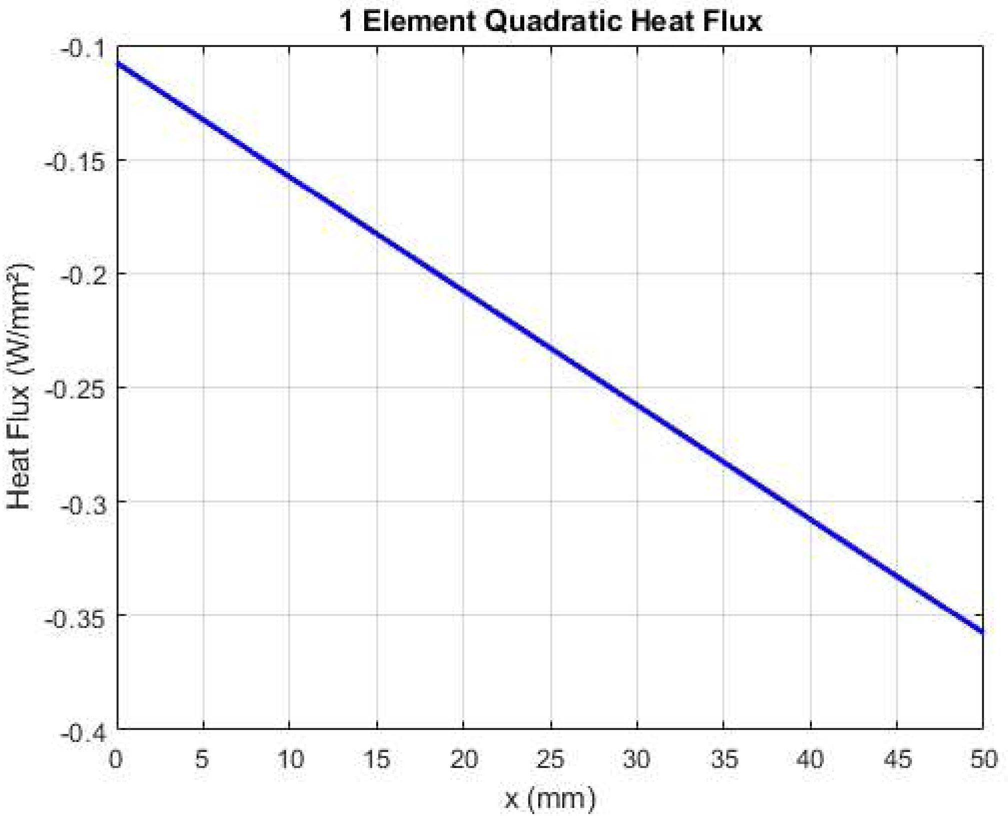

8 Element Linear Temperature
Temperature (K)
x (mm)
1400
1200
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50

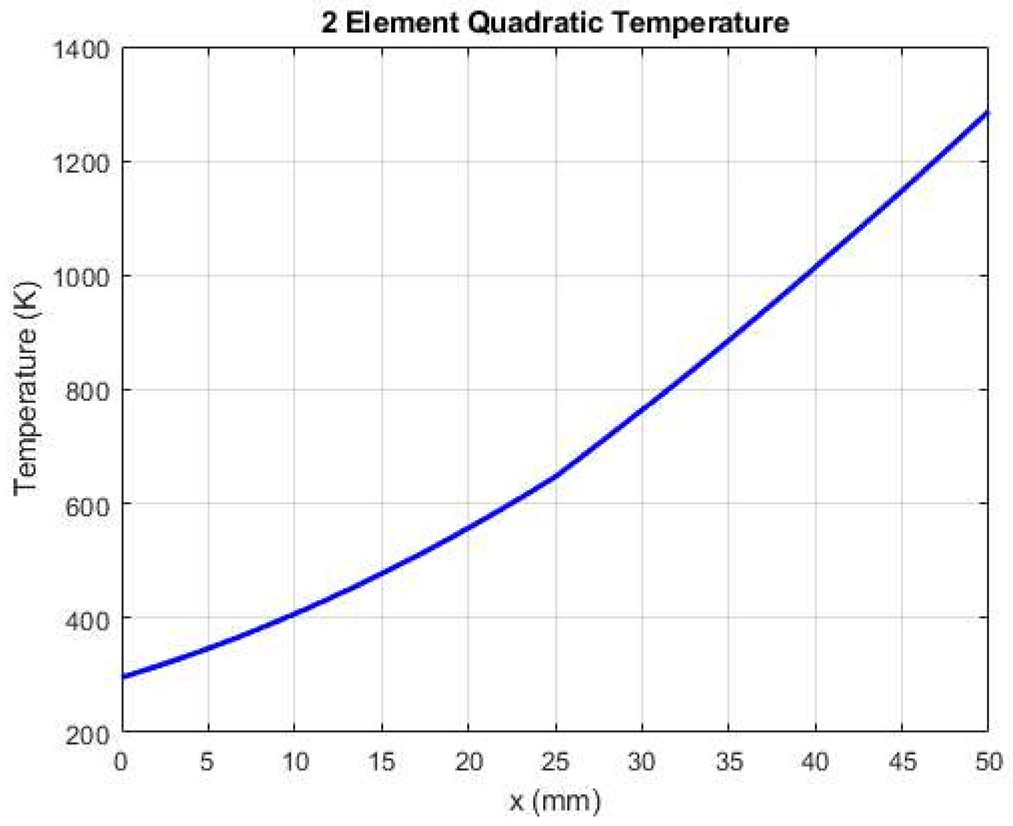

8 Element Linear Heat Flux
Heat Flux (W/mm²)
x (mm)
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
0
5
10
15
20
25
30
35
40
45
50


16 Element Linear Temperature
Temperature (K)
1400
1200
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50
x (mm)

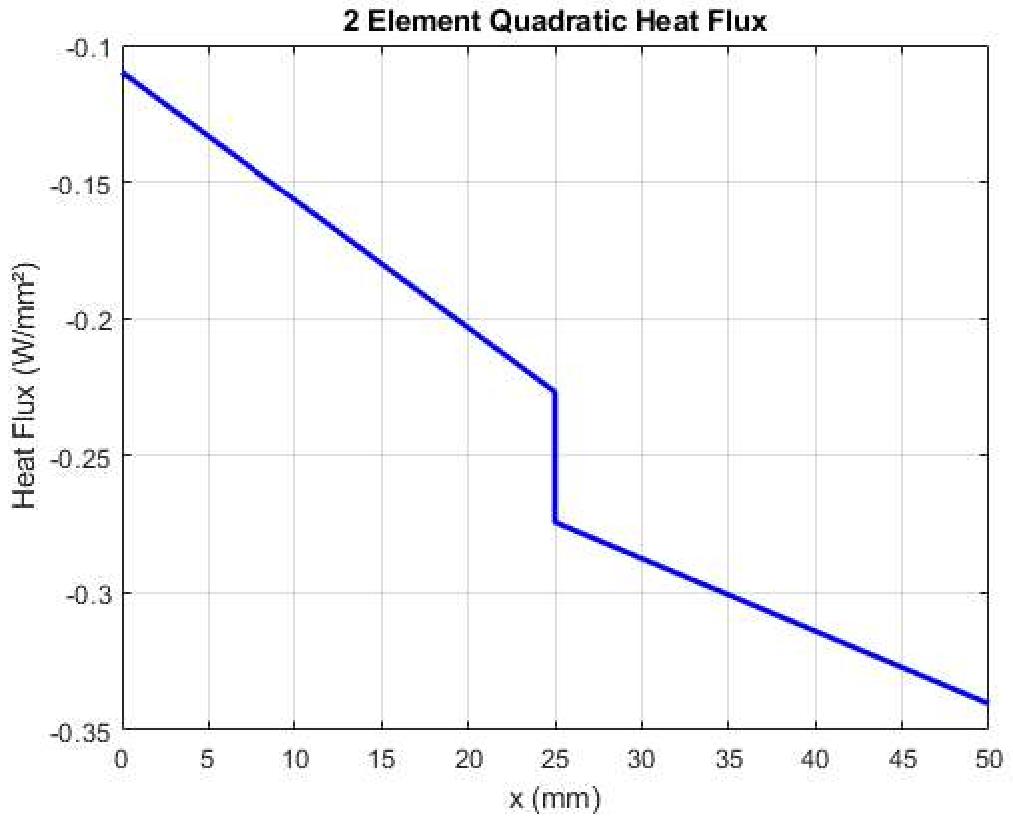

16 Element Linear Heat Flux
Heat Flux (W/mm²)
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
0
5
10
15
20
25
30
35
40
45
50
x (mm)


1 Element Quadratic Temperature
Temperature (K)
x (mm)
1400
1200
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50

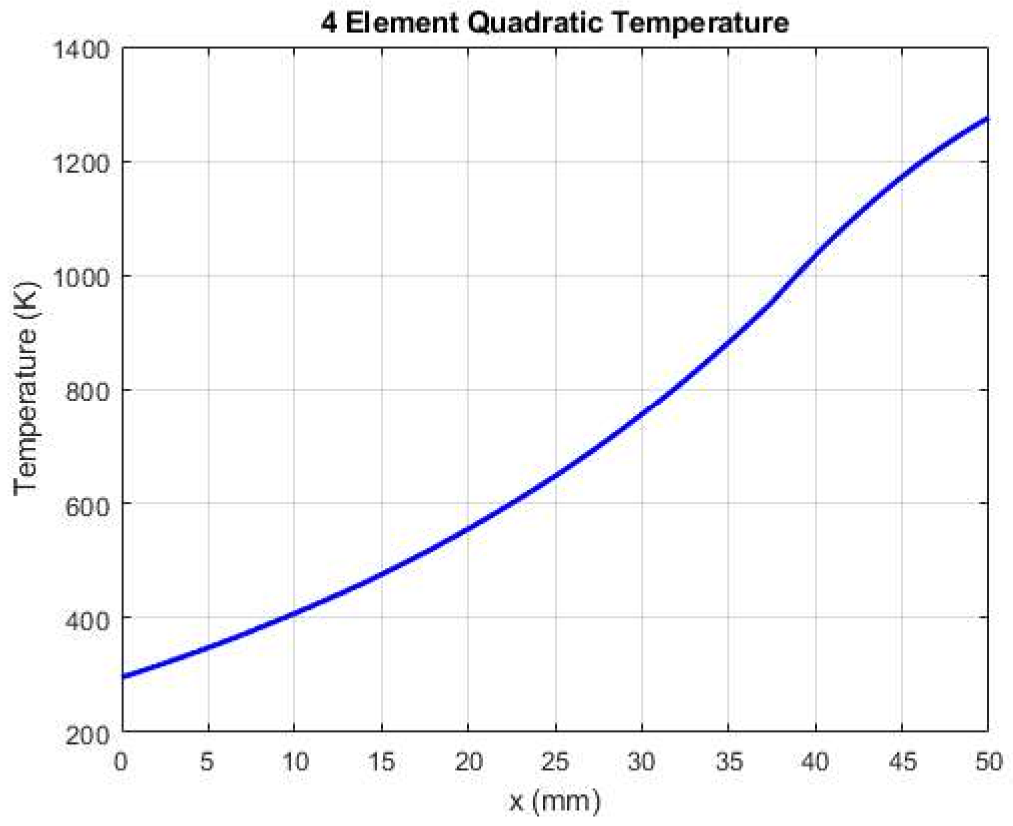

1 Element Quadratic Heat Flux
Heat Flux (W/mm²)
x (mm)
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
-0.4
0
5
10
15
20
25
30
35
40
45
50

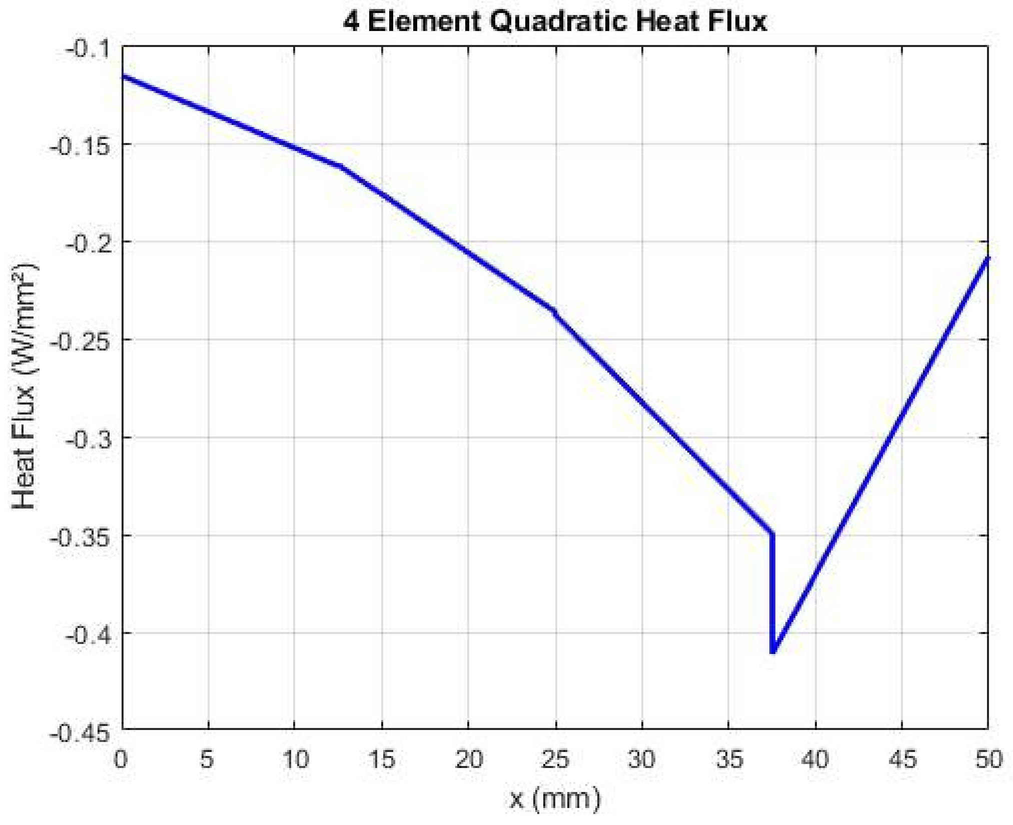


2 Element Quadratic Temperature
Temperature (K)
x (mm)
1400
1200
1000
800
600
400
200
0
5
10
15
20
25
30
35
40
45
50



2 Element Quadratic Heat Flux
Heat Flux (W/mm²)
x (mm)
-0.1
-0.15
-0.2
-0.25
-0.3
-0.35
0
5
10
15
20
25
30
35
40
45
50

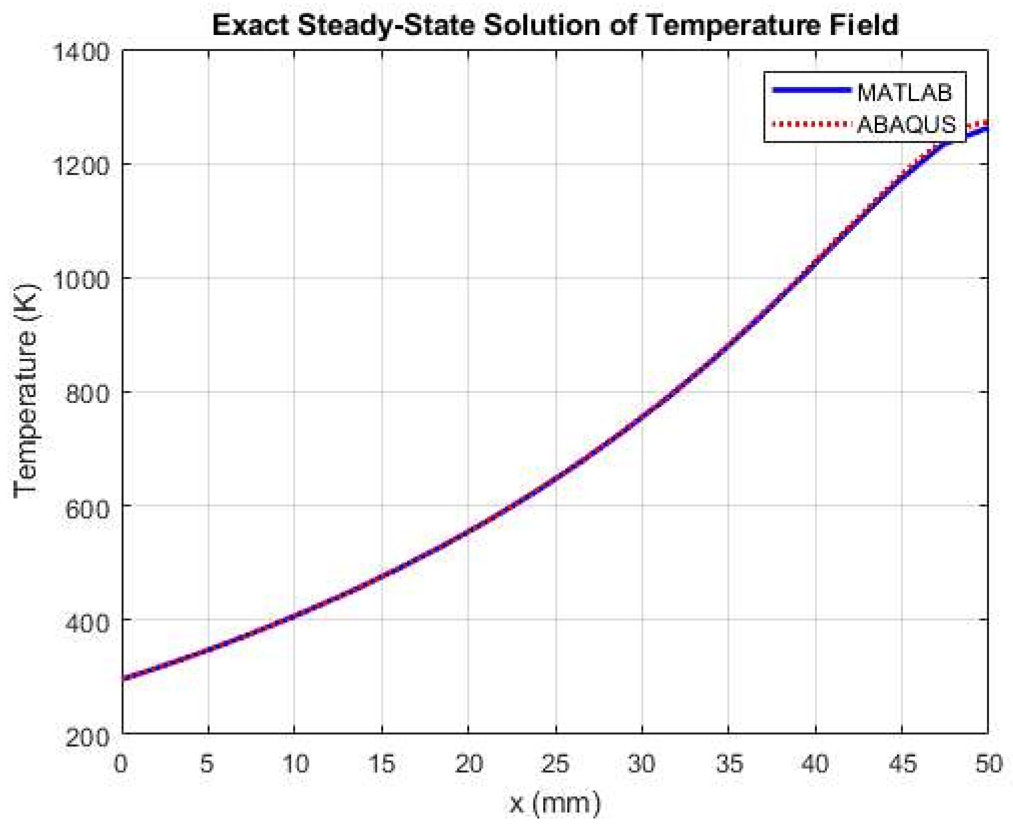




Part D

```
syms x_sym c1 c2
A_sym = (h1 - taper*x_sym)^2;
s_sym = (rhoe*I^2)/A_sym;
```

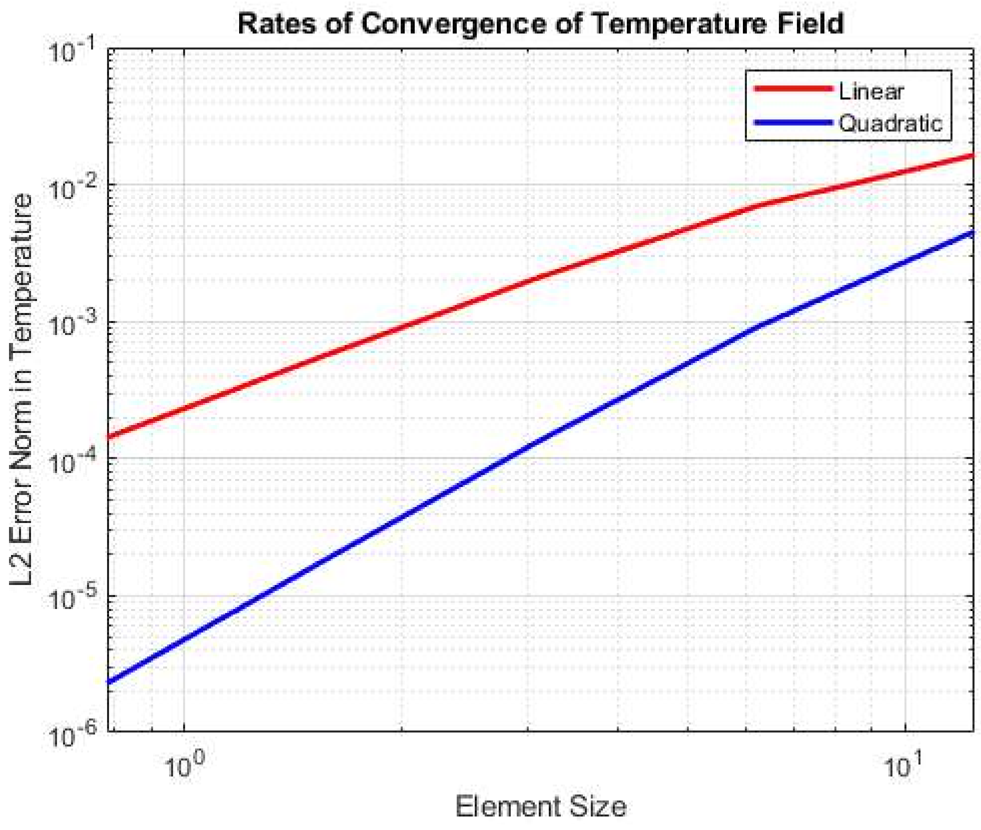
```
T_grad = int(-s_sym, x_sym);
dTdx_sym = (T_grad + c1)/(k*A_sym);
T_sym = int(dTdx_sym, x_sym)+c2;
bc1 = subs(T_sym, x_sym, 0) == T_boundary;
bc2 = subs(dTdx_sym, x_sym, L) == 0;
soln = solve([bc1, bc2], [c1, c2]);
T_exact_sym = simplify(subs(T_sym, soln));
T_exact_function = matlabFunction(T_exact_sym, 'Vars', x_sym);
x_exact = linspace(0, L, 50);
T_exact = T_exact_function(x_exact);
figure;
plot(x_abaqus, T_abaqus, 'b-', LineWidth=2)
hold on;
plot(x_exact, T_exact, 'r:', LineWidth=2);
xlabel('x (mm)');
ylabel('Temperature (K)');
title('Exact Steady-State Solution of Temperature Field');
legend('MATLAB','ABAQUS');
grid on;
```



Part E

```
elements = [4, 8, 16, 32, 64];
dT_exact_sym = diff(T_exact_sym, x_sym);
dT_exact_function = matlabFunction(dT_exact_sym, 'Vars', x_sym);
linear_error = zeros(size(elements));
quadratic_error = zeros(size(elements));
linear_size = zeros(size(elements));
quadratic_size = zeros(size(elements));

for i = 1:length(elements)
```

```
    ne = elements(i);
    nn = ne + 1;
    linear_size(i) = L/ne;
    mesh.x = linspace(0, L, nn)';
    mesh.conn = [1:nn-1;2:nn];
    mesh.shape = @shape2;
    K = zeros(nn, nn);
    load = zeros(nn,1);
    for e = 1:ne
        c = mesh.conn(:,e);
        xe = mesh.x(c);
        Ke = zeros(size(c,1), size(c,1));
        load_e = zeros(size(c,1), 1);
        for g = 1:length(gauss_points)
            p = gauss_points(g);
            w = weights(g);
            [N, dNdp] = mesh.shape(p);
            J = xe'*dNdp;
            x_value = N'*xe;
            A = area_x(x_value);
            s_value = s_x(x_value);
            dNdx = dNdp/J;
            Ke = Ke + k*A*(dNdx*dNdx')*J*w;
            load_e = load_e + s_value*N*J*w;
        end
        K(c, c) = K(c, c) + Ke;
        load(c) = load(c) + load_e;
    end
    K(1,:) = 0;
    K(1,1) = 1;
    load(1) = T_boundary;
    T = K\load;
    [error] = error_norm_compute(mesh, T, T_exact_function, dT_exact_function);
    linear_error(i) = error;
end

for i = 1:length(elements)
    ne = elements(i);
    nn = 2*ne + 1;
    quadratic_size(i) = L/ne;
    mesh.x = linspace(0, L, nn)';
    mesh.conn = [1:2:nn-2;2:2:nn-1;3:2:nn];
    mesh.shape = @shape3;
    K = zeros(nn, nn);
    load = zeros(nn,1);
    for e = 1:ne
        c = mesh.conn(:,e);
        xe = mesh.x(c);
        Ke = zeros(size(c,1), size(c,1));
        load_e = zeros(size(c,1), 1);
        for g = 1:length(gauss_points)
            p = gauss_points(g);
            w = weights(g);
            [N, dNdp] = mesh.shape(p);
            J = xe'*dNdp;
            x_value = N'*xe;
            A = area_x(x_value);
            s_value = s_x(x_value);
            dNdx = dNdp/J;
            Ke = Ke + k*A*(dNdx*dNdx')*J*w;
            load_e = load_e + s_value*N*J*w;
        end
        K(c, c) = K(c, c) + Ke;
        load(c) = load(c) + load_e;
    end
    K(1,:) = 0;
    K(1,1) = 1;
    load(1) = T_boundary;
    T = K\load;
    [error] = error_norm_compute(mesh, T, T_exact_function, dT_exact_function);
    quadratic_error(i) = error;
end

figure;
loglog(linear_size, linear_error, 'r-', LineWidth=2);
hold on;
loglog(quadratic_size, quadratic_error, 'b-', LineWidth=2);
xlabel('Element Size');
ylabel('L2 Error Norm in Temperature');
legend('Linear','Quadratic');
title('Rates of Convergence of Temperature Field');
grid on;
```



Functions Used

```
function [x, T, dTdx] = interpolate(mesh, d)
    x = [];
    T = [];
    dTdx = [];
    for e = 1:size(mesh.conn,2)
        c = mesh.conn(:,e);
        xe = mesh.x(c);
        de = d(c)';
        for p = linspace(-1,1,20)
            [N, dNdp] = mesh.shape(p);
            J = xe'*dNdp;
            dNdx = dNdp/J;
            x(end+1) = xe'*N;
            T(end+1) = de*N;
            dTdx(end+1) = de*dNdx;
        end
    end
end

function [error] = error_norm_compute(mesh, T, T_exact_function, dT_exact_function)
    qpts = quadrature(3);
    num_e1 = 0;
    den_e1 = 0;
    num_e2 = 0;
    den_e2 = 0;
    ne = size(mesh.conn,2);
    for e = 1:ne
        c = mesh.conn(:,e);
        xe = mesh.x(c);
        de = T(c);
        for i = 1:size(qpts,2)
            p = qpts(1,i);
            w = qpts(2,i);
            [N, dNdp] = mesh.shape(p);
            xCoord = N'*xe;
            dxdp = dNdp'*xe;
            J = dxdp;
            dNdx = dNdp/J;
            u_h  = dot(N,de);
            du_h = dot(dNdx,de);
            u_exact  = T_exact_function(xCoord);
            du_exact = dT_exact_function(xCoord);
            num_e1 = num_e1+(u_exact-u_h)^2*w*J;
            den_e1 = den_e1+(u_exact)^2*w*J;
            num_e2 = num_e2+(du_exact-du_h)^2*w*J;
            den_e2 = den_e2+(du_exact)^2*w*J;
        end
    end
    error = sqrt(num_e1 / den_e1);
end

function [qpts] = quadrature(n)
    u = 1:n-1;
    u = u ./ sqrt(4*u.^2 - 1);
    A = zeros(n);
    A(2:n+1 : n*(n-1)) = u;
    A(n+1 : n+1 : n^2-1) = u;
    [v, x] = eig(A);
    [x, k] = sort(diag(x));
    qpts = [x'; 2*v(1,k).^2];
end
```

```
function [N, dNdp] = shape2(p)
    N = 0.5*[1-p; 1+p];
    dNdp = [-0.5; 0.5];
end

function [N, dNdp] = shape3(p)
    N = [-0.5*(1-p)*p; (1-p^2); 0.5*(1+p)*p];
    dNdp = [-0.5+p; -2*p; 0.5+p];
end

function [N,dNdp] = shape4(p)
    N = [-9/16*(p+1/3)*(p-1/3)*(p-1);
         27/16*(p+1)*(p-1/3)*(p-1);
        -27/16*(p+1)*(p+1/3)*(p-1);
         9/16*(p+1)*(p+1/3)*(p-1/3)];
    dNdp = [-9/16*((p-1/3)*(p-1)+(p+1/3)*(p-1)+(p+1/3)*(p-1/3));
             27/16*((p-1/3)*(p-1)+(p+1)*(p-1)+(p+1)*(p-1/3));
            -27/16*((p+1/3)*(p-1)+(p+1)*(p-1)+(p+1)*(p+1/3));
             9/16*((p+1/3)*(p-1/3)+(p+1)*(p-1/3)+(p+1)*(p+1/3))];
end
```