Project Report

FEM: 598 Finite Element Methods

Instructor: Dr. Jay Oswald

Present by:

Purthviraj Vala

Tanmay Dhanote

Contents

Problem Statement

The problem states that there is a plat Impinged by flame at a location and there is heat loss from the other side due to radiation we must determine the steady-state temperature and heat flux by developing a finite element code for the system. After that, also compare the finite element code solution with the exact solution of the problem and simulating it in Abaqus or any commercial finite element code.

Problem Definition

Plate Geometry and Parameters

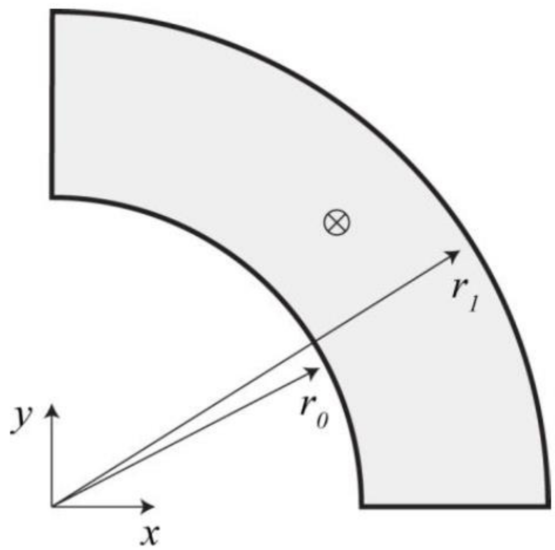


Figure 1: Plate Geometry and Parameters

Geometry Dimensions

$$Plate\ thickness, \Delta_z = 1\ mm$$

$$r1 = 20\ mm$$

$$r0 = 10\ mm$$

Material and It is Properties

$$Thermal\ Conductivity, k = 17\ {}^{W}/_{mK}$$

$$Stefan - Boltzmann\ Constant, \sigma = 5.670e^{-8}\ Wm^{-2}K^{-4}$$

Parameters and Input Variable Equations

$$Ambient\ Temperature, T_\infty = 300\ K$$

$$Radius\ of\ Flame, R = 3\ mm$$

$$Flame\ modeled\ as\ heat\ source, s(x, y) = \exp\left(-\frac{(x - x_c)^2 + (y - y_c)^2}{R^2}\right) W/mm^2$$

$$where, x_c = \frac{1}{2}(r_o + r_1)\cos\left(\frac{\pi}{4}\right) and\ y_c = \frac{1}{2}(r_o + r_1)\sin\left(\frac{\pi}{4}\right)$$

Boundary Condition

$$BC{:}\ 1\ At\ edge\ of\ plate\ flux, q = 0\ \ W/mm^2$$

$$BC{:}\ 2\ Ambient\ Temperature\ T_\infty = 300\ K$$

Governing Equation (Strong Form)

The problem is steady-state heat conduction with heat generation due to flame impingement

$$Governing\ equation, k\Delta_z \nabla^2 T + s = \sigma(T^4 - T_\infty^4) = h_{eff}(T - T_\infty)$$

$$where, h_{eff} = (T^2 + T_\infty^2)(T + T_\infty)$$

Task 1

Weak Form

A weak form is an integral form of differential equations of strong form, which is needed to formulate the finite element method.

Boundary Condition for the derivation of weak form

$$Natural\ BC\ q = \vec{q}.\vec{n}\ on\ \tau_q$$

$$Essenstial\ BC\ T = \bar{T}(x, y)\ on\ \tau_T$$

Step 1: Multiply the strong form with test function (w) and integrate.

$$\int_\Omega w\left(-k\Delta_z \nabla^2 T - s + h_{eff}(T - T_\infty)\right) d\Omega = 0$$

$$\int_\Omega w\left(\vec{\nabla}.\left(-k\Delta_z \vec{\nabla} T\right) - s + h_{eff}(T - T_\infty)\right) d\Omega = 0$$

Step 2: Integration by parts

$$-\int_{\Omega} \vec{\nabla}.\left(wk\Delta_z\vec{\nabla}T\right)d\Omega + \int_{\Omega} \vec{\nabla}w.\,k\Delta_z\vec{\nabla}Td\Omega - \int_{\Omega} ws\,d\Omega + \int_{\Omega} wh_{eff}T\,d\Omega - \int_{\Omega} wh_{eff}T_{\infty}\,d\Omega = 0$$

Step 3: Enforcing w=0 on τ_T

$$\int_{\Omega} \vec{\nabla}w.\,k\Delta_z\vec{\nabla}Td\Omega = -\int_{\tau} wk\Delta_z\vec{\nabla}T.\vec{n}d\tau + \int_{\Omega} ws\,d\Omega - \int_{\Omega} wh_{eff}T\,d\Omega + \int_{\Omega} wh_{eff}T_{\infty}\,d\Omega$$

$$As\ k\Delta_z\nabla T.\vec{n} = -\vec{q} = 0\ on\ \tau_q \rightarrow -\int_{\tau} wk\Delta_z\vec{\nabla}T.\vec{n}d\tau = 0$$

$$\int_{\Omega} \vec{\nabla}w.\,k\Delta_z\vec{\nabla}Td\Omega + \int_{\Omega} wh_{eff}T\,d\Omega = \int_{\Omega} ws\,d\Omega + \int_{\Omega} wh_{eff}T_{\infty}\,d\Omega$$

Step 4: Weak form in matrix notation

$$\int_{\Omega} \nabla w.\,D\Delta_z\nabla Td\Omega + \int_{\Omega} wh_{eff}T\,d\Omega = \int_{\Omega} ws\,d\Omega + \int_{\Omega} wh_{eff}T_{\infty}\,d\Omega$$

$$Where\ D = k\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\ W/mK$$

Discretized Equation

Now we will Discretize the equations

$$w = N^T w^T$$

$$T = Nd$$

$$\nabla T = Bd$$

$$B = \nabla \mathrm{N}$$

$$\left(\int_{\Omega} w^T B^T D B \Delta_z d\Omega + \int_{\Omega} w^T N^T\ h_{eff} N\,d\Omega\right) d = \int_{\Omega} w^T N^T s\,d\Omega + \int_{\Omega} w^T N^T h_{eff} T_{\infty}\,d\Omega$$

The above-presented equation is the final discretized equation.

Task 2

Approach to Deal with Non-Linear Nature of Problem

Step 1: Frist deriving the Discretized equation of our problem from week form.

Note: To make the equation linear, we converted the radiation heat transfer equation from higher-order to linear form as follows:

$$\sigma(T^4 - T_\infty^4) = h_{eff}(T - T_\infty)$$

$$where, h_{eff}(T) = (T^2 + T_\infty^2)(T + T_\infty)$$

Step 2: As can be seen, the heff is surface temperature-dependent. Thus we need the steady-state surface temperature to calculate heff, but as we do not have the temperature function of the surface, we will use the iterative method to solve this problem.

Step 3: We will make an initial guess of the surface temperature, which can be the average temperature of the surface at a steady-state.

Step 4: To determine Tavg we will equate total heat generated to loss of heat due to radiation which will give us the Tavg

$$\int_\Omega s d\Omega = Ac\sigma(T^4 - T_\infty^4)$$

$$Ac\ Area\ of\ plate$$

Step 5: After determining, we can get the initial guess for heff, and now we will loop over the calculation of temperature from the K matrix to T=(K+H)/(Fe+Fh) calculated.

Step 6: Here, we will use the temperature to reach the convergence. The Basic logic is shown below:

Step 7:

Tavg

Heff(Tavg) initial guess

While max(abs(Temperature difference) >=0.01 Criteria of convergence

K H Fe and Fh in the calculation

Then finding

Temperature

Finding the Difference of temperature T with our Tavg for initial

Saving T(n+1) =temperature obtained

Step 8: And now using this new temperature find heff, then recalculate T and loop goes on till the Convergence criteria not achieved.

Approximation of Temperature for the Initial guess of temperature field to Solve the Problem

Finite Element Program in MATLAB

Code

```
    h=1;
    r0=10;          %millimeter
    r1=20;          %millimeter
    etype='t6';     % q4 q8 q9 t3 t6
    R=3;            %millimeter
    sbc=5.670e-14;   %Wmm-2K-4
    nqpts=5;
    T0=300;         % K
    k=eye(2)*0.017;           %W/mm-K
    th=1;

%% input variables %%
xc=0.5*(r1+r0)*cos(pi/4);
yc=0.5*(r1+r0)*sin(pi/4);
Ac=pi*(r1^2-r0^2)/4;
%% Shape function and quarature points
[mesh] = make_project2_mesh(h, r0, r1, etype);
if strcmp(etype, 'q4')
    Shape=@shape_q4;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.conn';
elseif strcmp(etype, 't3')
    Shape=@shape_t3;
    qpts=   [0.1012865073 0.1012865073 0.0629695903;
        0.7974269853 0.1012865073 0.0629695903;
        0.1012865073 0.7974269853 0.0629695903;
        0.4701420641 0.0597158717 0.0661970764;
        0.4701420641 0.4701420641 0.0661970764;
        0.0597158717 0.4701420641 0.0661970764;
        0.3333333333 0.3333333333 0.1125]';
    facep=mesh.conn';
elseif strcmp(etype, 'q8')
    Shape=@shape_q8;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.pconn';
elseif strcmp(etype, 'q9')
    Shape=@shape_q9;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.pconn';
elseif strcmp(etype, 't6')
    Shape=@shape_t6;
```

```
    qpts=   [0.1012865073 0.1012865073 0.0629695903;
        0.7974269853 0.1012865073 0.0629695903;
        0.1012865073 0.7974269853 0.0629695903;
        0.4701420641 0.0597158717 0.0661970764;
        0.4701420641 0.4701420641 0.0661970764;
        0.0597158717 0.4701420641 0.0661970764;
        0.3333333333 0.3333333333 0.1125]';
    facep=mesh.pconn';
end
%% Heat source %%
s=@(x,y) (exp(-(((x-xc)^2+(y-yc)^2)/R^2)));
%% mesh %%
x=mesh.x;
conn=mesh.conn;
%% Average Temperature calculation
fe=0;
for c=conn
    xe=x(:,c);
    for q=qpts
        [N,dNdp]=Shape(q(1:2));
        J=xe*dNdp;
        xp=xe*N;
        S=s(xp(1),xp(2));
        F=S*det(J)*q(end);
        fe=fe+F;
    end
end
Tavg=((fe/(Ac*sbc))+T0^4)^(1/4);
```

Output

Element Type	Tavg (K) FEM
Quad4	1201.8
Quad8	1201.8
Tri3	1201.8
Tri6	1201.8

Method of Manufactured Solution

The method of manufacture solution is for testing the developed code when there is not the availability of exact solution due to the complexity of the problem.

Step 1: The chosen temperature field:

$$T(x,y) = \sin(x) + \cos\,(y)$$

Step 2: Find the Heat flux

$$q = -k\nabla T(x,y) = \begin{matrix} -kcos(x) \\ -kcos(y) \end{matrix}$$

Step 3: Deriving Heat source term for producing the solution

To only our analysis for the method of manufactured solution, we will not consider the heat transfer due to radiation heat transfer, and thus, the governing equation will be:

$$Governing\ equation, k\Delta_z\nabla^2 T + s = 0$$

Solving this to drive the Heat source

$$s(x,y) = k(\sin\,(x) + \sin\,(y))$$

Step 4: Define the Boundary Condition

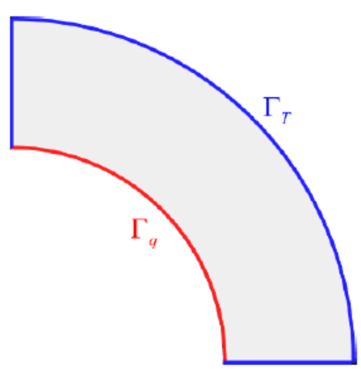


Figure 2: Boundary Condition for Method of Manufactured Solutions

$$Natural\ BC\ \ q = \begin{matrix} -kcos(x) \\ -kcos(y) \end{matrix}\ on\ \tau_q$$

$$Essenstial\ BC\ T = \bar{T}(x,y)\ on\ \tau_T$$

Step 5: Now Considering s(x,y) as our heat source and inputting it in place of our actual heat source function and removing the effect of radiation heat from our Code.
Step 6: Now run the code and obtain Temperature and Heat Flux from our FEM code.
Step 7: Now we will find the error norm to find the convergence of the solution, as we now have exact function of temperature and heat flux.

MATLAB Code

```
    h=1;
    r0=10;
    r1=20;
    etype='t6';     % q4 q8 q9 t3 t6
    nqpts=5;
    k=0.017;           %W/mm-K
    th=1;
%% Manufactured Solution Method
H=[2*h h h/2 h/4 h/8];
T_star=@(X,Y) (sin(X) + sin(Y));
Q_heat=@(X,Y) ([-k*cos(X) , -k*cos(Y)]);
%% Heat source %%
S_heat=@(X,Y) (k*(sin(X) + sin(Y)));
e_L2=[];
e_en=[];
leng=[];
for h=H
%% Shape function and quadrature points
[mesh] = make_project2_mesh(h, r0, r1, etype);
[Shape,qpts,facep,shape_edge,nodes_outer,edgeconn_inner]=SQF(etype,nqpts,mesh,r0,r1);
%% mesh %%
x=mesh.x;
conn=mesh.conn;
%% Average Temperature calculation
K=spalloc(length(x),length(x),2*length(x));
Fe=zeros(length(x),1);
    for c=conn
        xe=mesh.x(:,c);
        Ke=zeros(length(c));
        for q=qpts
            [N,dNdp]=Shape(q);
            J=xe*dNdp;
            B=dNdp/J;
            w=q(end);
            Ke=Ke+B*k*eye(2)*th*B'*det(J)*w;
            f=S_heat(xe(1,:)*N,xe(2,:)*N);
            Fe(c)=Fe(c)+N*f*det(J)*w;
        end
        K(c,c)=K(c,c)+Ke;
    end
    for ec=edgeconn_inner
        xe=mesh.x(:,ec);
        for q=quadrature_1D(nqpts)
            [N,dNdp]=shape_edge(q(1));
            J=xe*dNdp;
            n=[0 -1;1 0]*J/norm(J);
            XE=xe*N;
            Q=dot(n,Q_heat(XE(1),XE(2)));
            Fe(ec)=Fe(ec)-N*Q*norm(J)*q(end);
        end
    end
    fixed_nodes_temp=nodes_outer(1:end);
```

```
    K(fixed_nodes_temp, :)= 0;
    K(fixed_nodes_temp,fixed_nodes_temp)=eye(length(fixed_nodes_temp));
    XY=x(:,fixed_nodes_temp);
    Fe(fixed_nodes_temp,1)=(T_star(XY(1,:),XY(2,:)))';
    T=K\Fe;
    %% Convergence
Fun_num=0;
Fun_den=0;
for c=conn
    xe=x(:,c);
    len=norm(xe(:,2)-xe(:,1));
for q=qpts
    [N,dNdp]=Shape(q(1:2));
    xee=xe*N;
    T1=T(c)'*N;
    J=xe*dNdp;
    Temp_Exact=(T_star(xee(1),xee(2)));
    fun_num=(Temp_Exact-T1)^2;
    fun_den=Temp_Exact^2;
    Fun_num=Fun_num+fun_num*det(J)*q(end);
    Fun_den=Fun_den+fun_den*det(J)*q(end);
end
end

Fun_s_num=0;
Fun_s_den=0;
for c=conn
    xe=x(:,c);
    for q=qpts
        [N,dNdp]=Shape(q(1:2));
        J=xe*dNdp;
        B=dNdp/J;
        Tf=T(c);
        Xe=xe*N;
        n=Xe/norm(Xe);
        Q=-dot(n,Tf'*B*eye(2)*k);
        Heat_Exact=dot(n,Q_heat(Xe(1),Xe(2)));
        fun_s_num=(Heat_Exact-Q)^2;
        fun_s_den=Heat_Exact^2;
        Fun_s_num=Fun_s_num+(fun_s_num*det(J)*q(end))/2;
        Fun_s_den=Fun_s_den+(fun_s_den*det(J)*q(end))/2;
    end
end
e_L2(end+1)=sqrt(abs(Fun_num/Fun_den));
e_en(end+1)=sqrt(abs(Fun_s_num/Fun_s_den));
leng(end+1)=len;
end
figure('name',"log(h) vs log(e_L2)");
plot(log(H),log(e_L2),'--o')
title(['log(eL2) vs log(h) for ',etype ,' Element'])
xlabel('log(h)--log of element length')
ylabel('log(eL2)--log of L2 Temperature error norm')
legend([etype ,' Element'])
figure('name',"log(h) vs log(e_en)");
plot(log(H),log(e_en),'--o')
title(['log(een)vs log(LE)for ',etype ,' Element'])
xlabel('log(h)--log of element length')
ylabel('log(een)--log of flux error norm')
legend([etype ,' Element'])



function[N, dNdp] = shape_q4(p)
```

```matlab
N = [(1/4)*(1-p(1))*(1-p(2));
    (1/4)*(1+p(1))*(1-p(2));
    (1/4)*(1+p(1))*(1+p(2));
    (1/4)*(1-p(1))*(1+p(2))];
dNdp = [(1/4)*(p(2)-1),(1/4)*(p(1)-1);
    (1/4)*(1-p(2)),(-1/4)*(p(1)+1);
    (1/4)*(1+p(2)),(1/4)*(1+p(1));
    (-1/4)*(1+p(2)),(1/4)*(1-p(1))];
end
function[N, dNdp] = shape_q8(p)
N = [(-1/4)*(1-p(1))*(1-p(2))*(1+p(1)+p(2));     %1-1
    (-1/4)*(1+p(1))*(1-p(2))*(1-p(1)+p(2));     %3-2
    (-1/4)*(1+p(1))*(1+p(2))*(1-p(1)-p(2));     %5-3
    (-1/4)*(1-p(1))*(1+p(2))*(1+p(1)-p(2));     %7-4
    (1/2)*(1-p(1))*(1+p(1))*(1-p(2));           %2-5
    (1/2)*(1+p(1))*(1+p(2))*(1-p(2));           %4-6
    (1/2)*(1-p(1))*(1+p(1))*(1+p(2));           %6-7
    (1/2)*(1-p(1))*(1+p(2))*(1-p(2))];          %8-8
dNdp = [(-1/4)*(p(2)-1)*(2*p(1)+p(2)),(-1/4)*(p(1)-1)*(2*p(2)+p(1));    %1-1
     (1/4)*(p(2)-1)*(-2*p(1)+p(2)),(1/4)*(p(1)+1)*(2*p(2)-p(1));        %3-2
     (1/4)*(1+p(2))*(2*p(1)+p(2)),(1/4)*(1+p(1))*(p(1)+2*p(2));         %5-3
     (-1/4)*(1+p(2))*(p(2)-2*p(1)),(-1/4)*(p(1)-1)*(2*p(2)-p(1));       %7-4
     p(1)*(p(2)-1),(1/2)*(1+p(1))*(-1+p(1));                            %2-5
     (-1/2)*(p(2)+1)*(p(2)-1),(-1)*(1+p(1))*p(2);                       %4-6
     (-1)*p(1)*(1+p(2)),(-1/2)*(1+p(1))*(p(1)-1);                       %6-7
     (1/2)*(1+p(2))*(p(2)-1),p(2)*(p(1)-1)];                            %8-8
end
function[N, dNdp] = shape_q9(p)
N = [0.5*p(1)*(p(1)-1)*0.5*p(2)*(p(2)-1);
    0.5*p(1)*(p(1)+1)*0.5*p(2)*(p(2)-1);
    0.5*p(1)*(p(1)+1)*0.5*p(2)*(p(2)+1);
    0.5*p(1)*(p(1)-1)*0.5*p(2)*(p(2)+1);
    (1-p(1)^2)*0.5*p(2)*(p(2)-1);
    0.5*p(1)*(p(1)+1)*(1-p(2)^2);
    (1-p(1)^2)*0.5*p(2)*(p(2)+1);
    0.5*p(1)*(p(1)-1)*(1-p(2)^2);
    (1-p(1)^2)*(1-p(2)^2)];
dNdp = [0.5*(2*p(1)-1)*0.5*p(2)*(p(2)-1),0.5*p(1)*(p(1)-1)*0.5*(2*p(2)-1);
    0.5*(2*p(1)+1)*0.5*p(2)*(p(2)-1),0.5*p(1)*(p(1)+1)*0.5*(2*p(2)-1);
    0.5*(2*p(1)+1)*0.5*p(2)*(p(2)+1),0.5*p(1)*(p(1)+1)*0.5*(2*p(2)+1);
    0.5*(2*p(1)-1)*0.5*p(2)*(p(2)+1),0.5*p(1)*(p(1)-1)*0.5*(2*p(2)+1);
    (-2*p(1))*0.5*p(2)*(p(2)-1),(1-p(1)^2)*0.5*(2*p(2)-1);
    0.5*(2*p(1)+1)*(1-p(2)^2),0.5*p(1)*(p(1)+1)*(-2*p(2));
    (-2*p(1))*0.5*p(2)*(p(2)+1),(1-p(1)^2)*0.5*(2*p(2)+1);
    0.5*(2*p(1)-1)*(1-p(2)^2),0.5*p(1)*(p(1)-1)*(-2*p(2));
    (-2*p(1))*(1-p(2)^2),(1-p(1)^2)*(-2*p(2))];
end

function[N, dNdp] = shape_t3(p)
N = [p(1);
    p(2);
    1 - p(1) - p(2)];
dNdp = [1, 0;
    0, 1;
    -1, -1];
end
function[N, dNdp] = shape_t6(p)
N = [p(1)*(2*p(1)-1);
    p(2)*(2*p(2)-1);
    (1-p(2)-p(1))*(2*(1-p(2)-p(1))-1);
    4*p(1)*p(2);
    4*p(2)*(1-p(2)-p(1));
    4*p(1)*(1-p(2)-p(1))];
```

```
dNdp =  [4*p(1) - 1, 0 ;
        0, 4*p(2) - 1;
        -3+4*p(1)+4*p(2),-3+4*p(1)+4*p(2);
        4*p(2), 4*p(1);
        -4*p(2),(4-8*p(2)-4*p(1));
        4-4*p(2)-8*p(1),-4*p(1)];
end
function [N,dNdp] = shape2(p)
N=0.5*[1-p(1);1+p(1)];
dNdp=[-0.5;0.5];
end
function [N, dNdp] = shape3(p) % Define quadratic shape functions and derivatives
N=1/2*[-(1-p)*p ; 2*(1-p)*(1+p); (1+p)*p ];
dNdp=[ p-0.5000 ; -2*p ; p+0.5000];
end

%% Gaussian Quadrature 1D
function [qpts] = quadrature_1D(n)
%   QUADRATURE
%     quadrature(n) returns a quadrature table for a rule with n
%     integration points.  The first row of the table gives the quadrature
%     point location and the second gives the quadrature weights.

    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
[Shape,qpts,facep,shape_edge,nodes_outer,edgeconn_inner]=SQF(etype,nqpts,mesh,r0,r1)
x=mesh.x;
edge_inner=find(abs(x(1,:).^2+x(2,:).^2-r0^2)<1e-10);
edge_outer=find(abs(x(1,:).^2+x(2,:).^2-r1^2)<1e-10);
edge_sidex=find(abs(x(2,:))<1e-10);
edge_sidey=find(abs(x(1,:))<1e-10);
nodes_outer=[edge_sidex edge_outer(2:end-1) edge_sidey];
if strcmp(etype, 'q4')
    Shape=@shape_q4;
    shape_edge=@shape2;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.conn';
    edgeconn_inner=[edge_inner(1:end-1);edge_inner(2:end)];
elseif strcmp(etype, 't3')
    Shape=@shape_t3;
    shape_edge=@shape2;
    qpts=   [0.1012865073 0.1012865073 0.0629695903;
        0.7974269853 0.1012865073 0.0629695903;
        0.1012865073 0.7974269853 0.0629695903;
        0.4701420641 0.0597158717 0.0661970764;
        0.4701420641 0.4701420641 0.0661970764;
        0.0597158717 0.4701420641 0.0661970764;
        0.3333333333 0.3333333333 0.1125]';
    facep=mesh.conn';
    edgeconn_inner=[edge_inner(1:end-1);edge_inner(2:end)];
elseif strcmp(etype, 'q8')
    Shape=@shape_q8;
```

```
     shape_edge=@shape3;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.pconn';
    edgeconn_inner=[edge_inner(1:2:end-2);edge_inner(2:2:end-1);edge_inner(3:2:end)];
elseif strcmp(etype, 'q9')
    Shape=@shape_q9;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.pconn';
elseif strcmp(etype, 't6')
    Shape=@shape_t6;
     shape_edge=@shape3;
    qpts=   [0.1012865073 0.1012865073 0.0629695903;
        0.7974269853 0.1012865073 0.0629695903;
        0.1012865073 0.7974269853 0.0629695903;
        0.4701420641 0.0597158717 0.0661970764;
        0.4701420641 0.4701420641 0.0661970764;
        0.0597158717 0.4701420641 0.0661970764;
        0.3333333333 0.3333333333 0.1125]';
    facep=mesh.pconn';
    edgeconn_inner=[edge_inner(1:2:end-2);edge_inner(2:2:end-1);edge_inner(3:2:end)];
end
end
```

Output Linear Element

Plot of Error Norms for 3 Node Triangular Element

Element size (mm) used = 2.0000 1.0000 0.5000 0.2500 0.1250

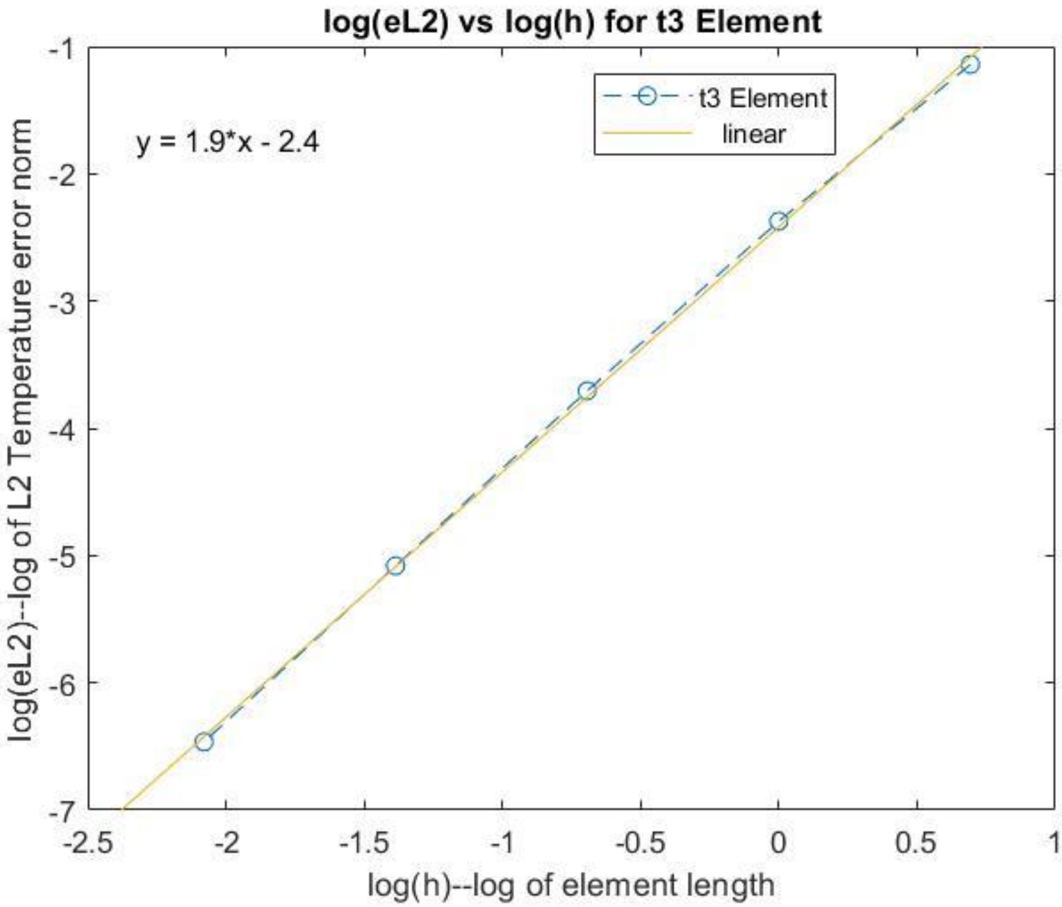


Figure 3: L2 Error Norm for 3 Node Triangular Element

$$L2\ Error\ Norm\ Function, \log(e_{L2}) = C + \alpha \log(h)$$

$$e_{L2} - Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -2.4 + 1.9 \log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 1.9$ *≈2. As per the general Mathematical Literature Theory for Linear element, the rate of convergence is equal to 2 for the L2 error norm, and our result is approximately matching. Taking Further smaller elements will lead to convergence slop to 2.*

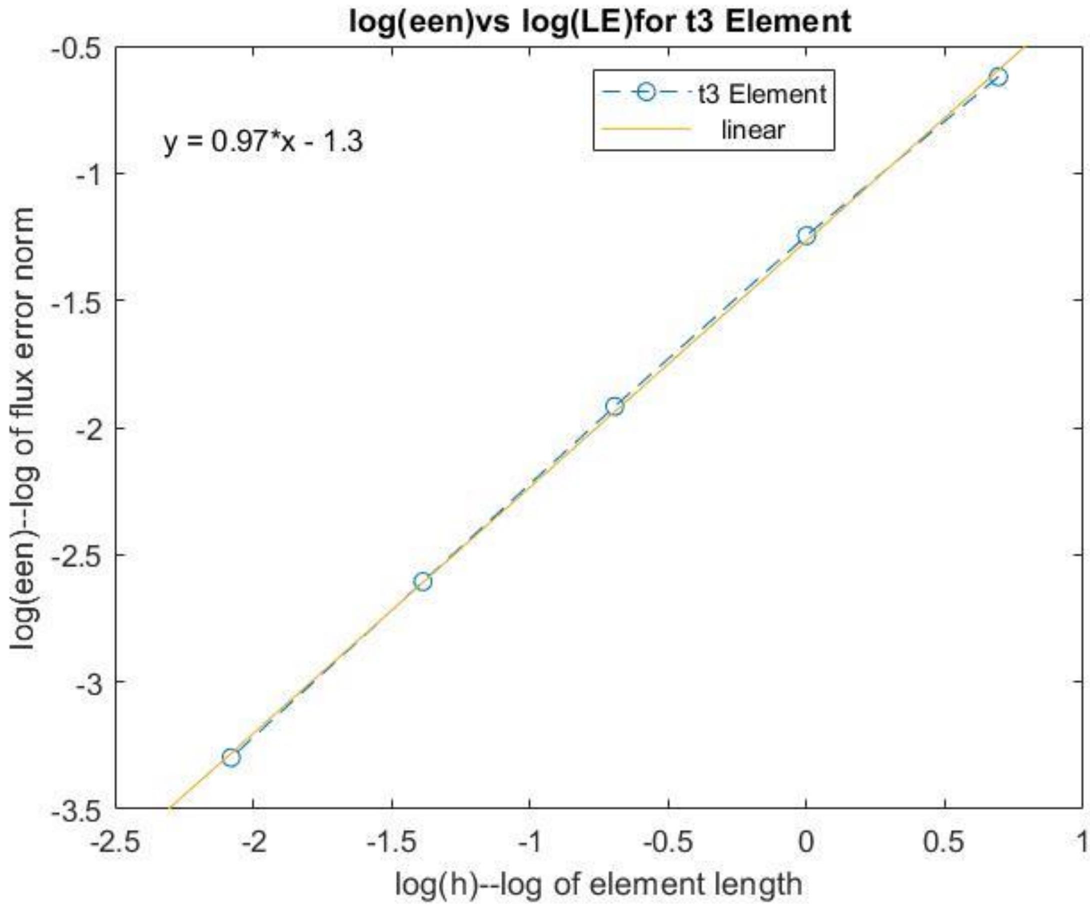


Figure 4: Plot of Energy Norm 3 Node Triangular Element

$$Energy\ Error\ Norm\ Function, \log(e_{en}) = C + \alpha \log(h)$$

$$e_{en} - Energy\ Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -1.3 + 0.97 \log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 0.97$*≈1. As per the general Mathematical Literature Theory for Linear element, the rate of convergence is equal to 1 for energy error norm.*

The plot of L2 Error Norm for 4 Node Quadrilateral Element

Element size (mm) used = 2.0000 1.0000 0.5000 0.2500 0.1250 0.0625 mm

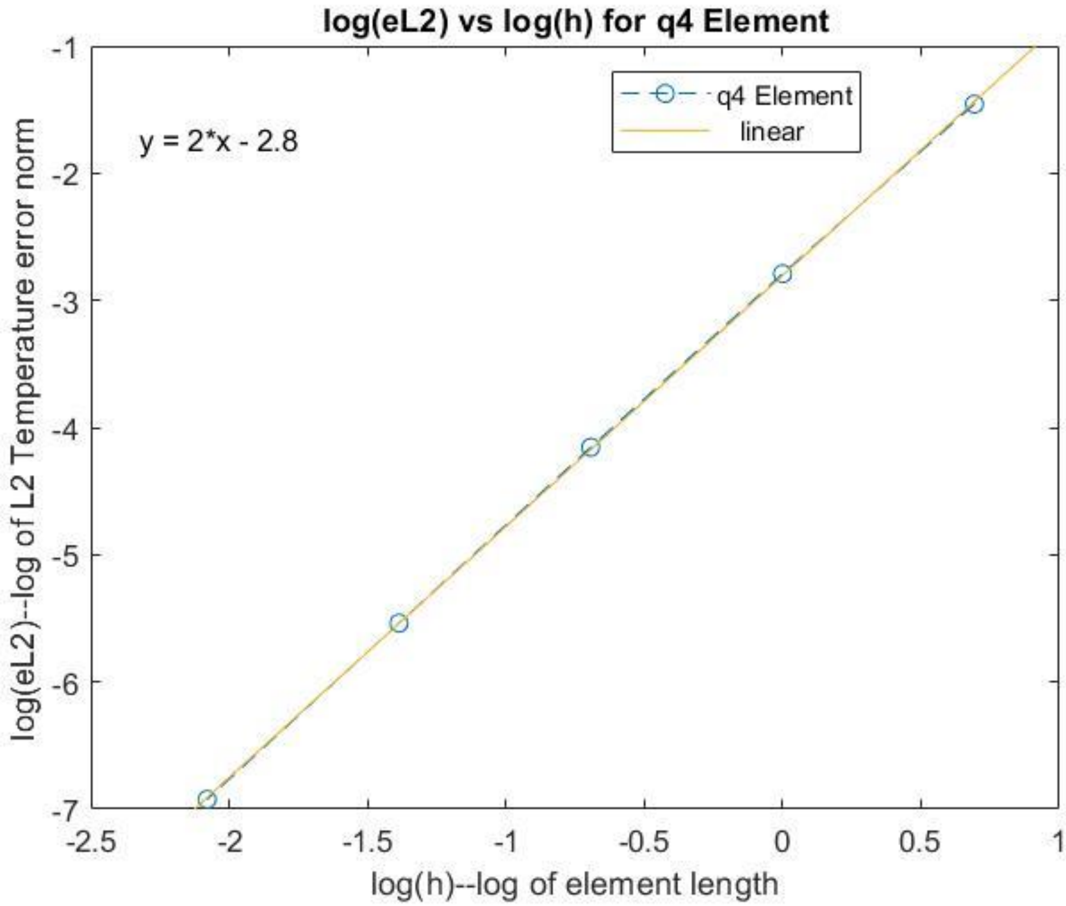


Figure 5: L2 Error Norm for 4 Node Quadrilateral Element

$$L2\ Error\ Norm\ Function, \log(e_{L2}) = C + \alpha \log(h)$$

$$e_{L2} - Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -2.8 + 2\log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 2$. As per the general Mathematical Literature Theory for Linear element, the rate of convergence is equal to 2 for the L2 error norm.

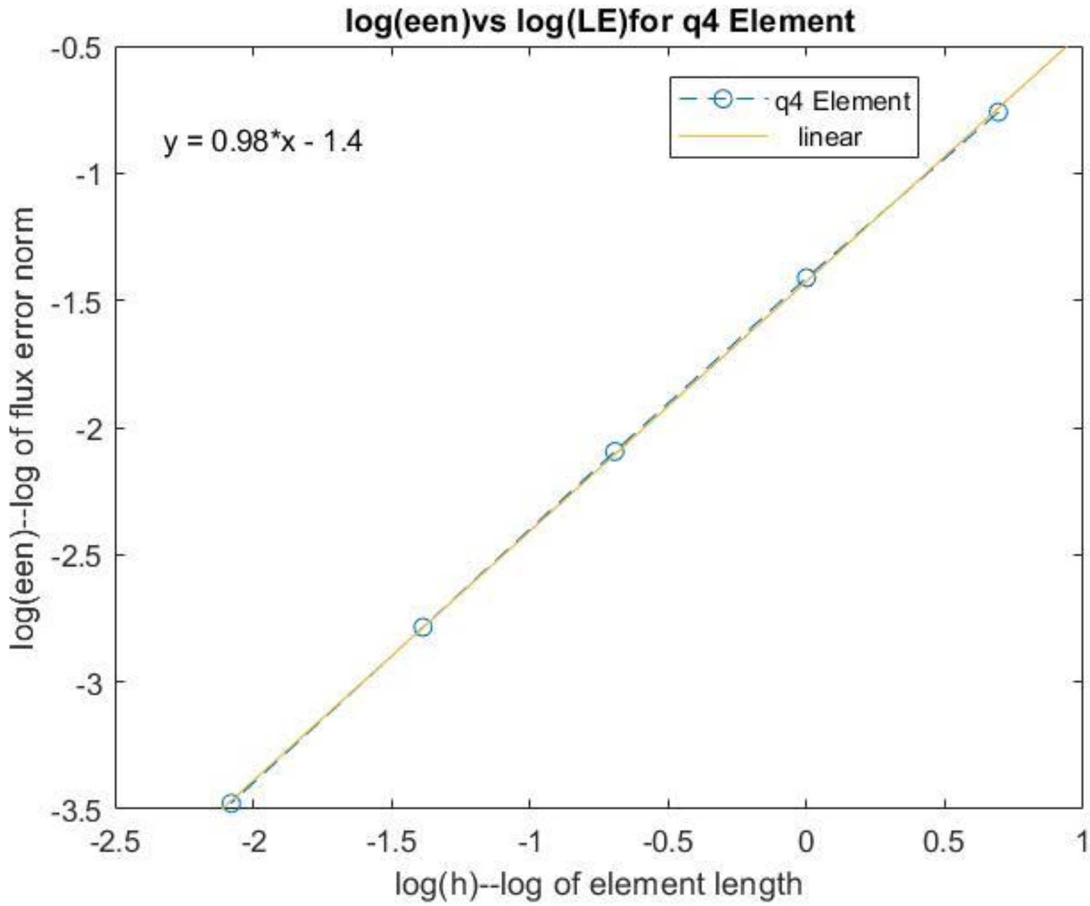


Figure 5: Plot of Energy Norm 4 Node Quadrilateral Element

$$Energy\ Error\ Norm\ Function, \log(e_{en}) = C + \alpha \log(h)$$

$$e_{en} - Energy\ Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -1.4 + 0.98 \log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 0.98$*≈1. As per the general Mathematical Literature Theory for Linear element, the rate of convergence is equal to 1 for energy error norm.*

Output Quadratic Element

The plot of Error Norms for 6 Node Triangular Element

Element size (mm) used = 2.0000   1.0000   0.5000   0.2500   0.1250

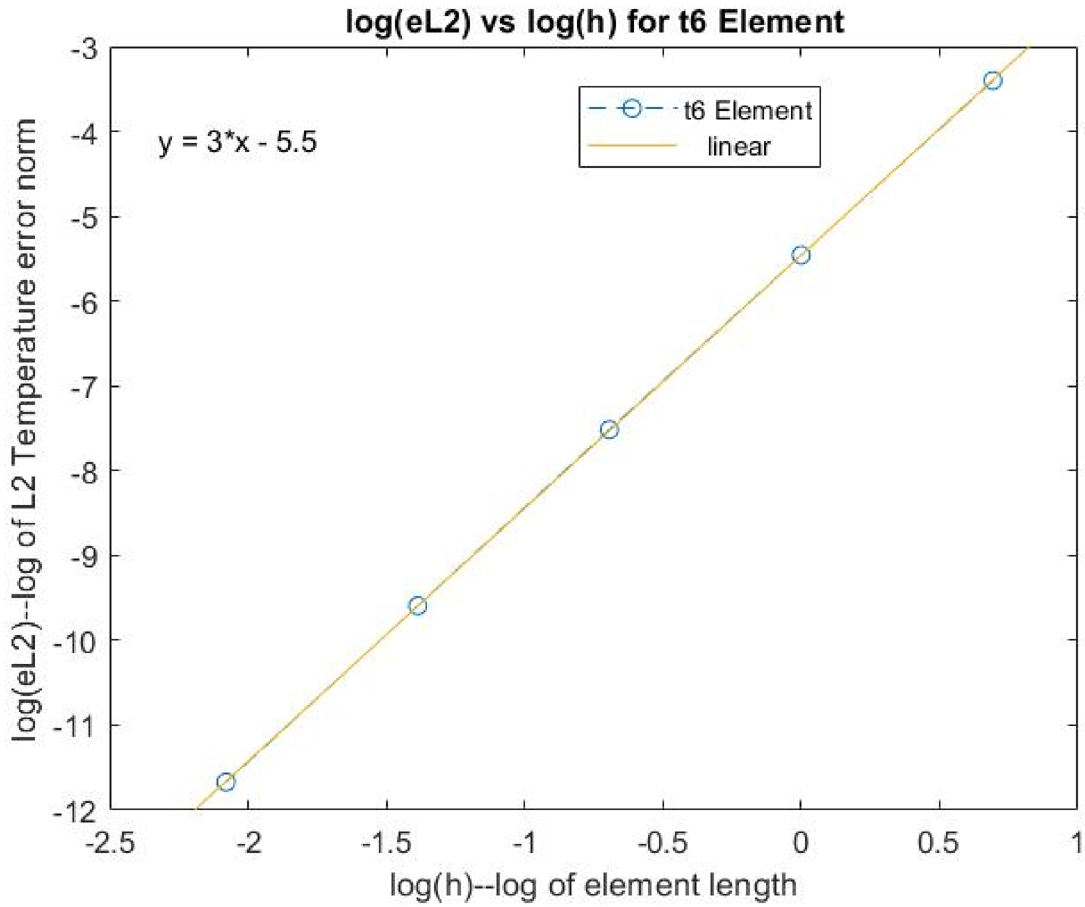


Figure 6: L2 Error Norm for 6 Node Triangular Element

$$L2\ Error\ Norm\ Function, \log(e_{L2}) = C + \alpha \log(h)$$

$$e_{L2} - Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -5.5 + 3\log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 3$. *As per the general Mathematical Literature Theory for the quadratic element, the rate of convergence is equal to 3 for the L2 error norm.*

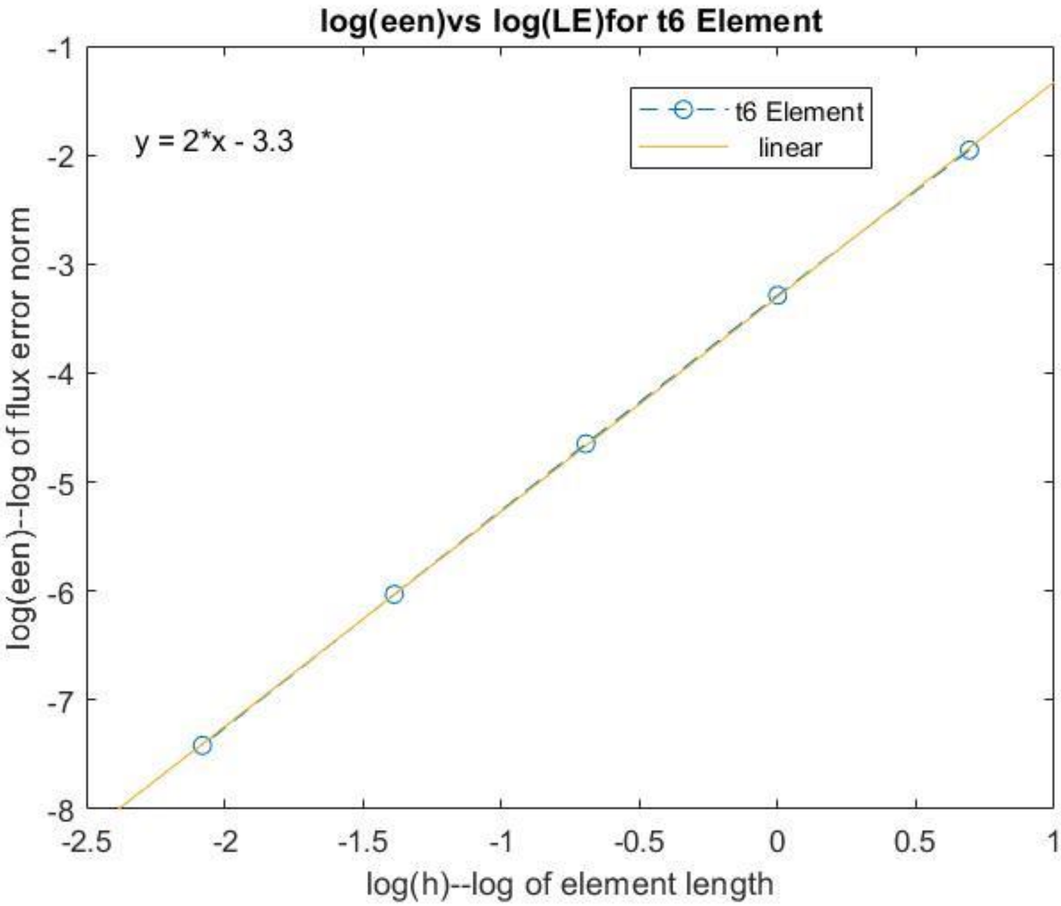


Figure 7: Plot of Energy Norm 6 Node Triangular Element

$$Energy\ Error\ Norm\ Function, \log(e_{en}) = C + \alpha \log(h)$$

$$e_{en} - Energy\ Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -3.3 + 2\log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 2$. As per the general Mathematical Literature Theory for the quadratic element, the rate of convergence is equal to 2 for energy error norm.

The plot of L2 Error Norm for 8 Node Quadrilateral Element

Element size (mm) used = 2.0000 1.0000 0.5000 0.2500 0.1250 0.0625 mm

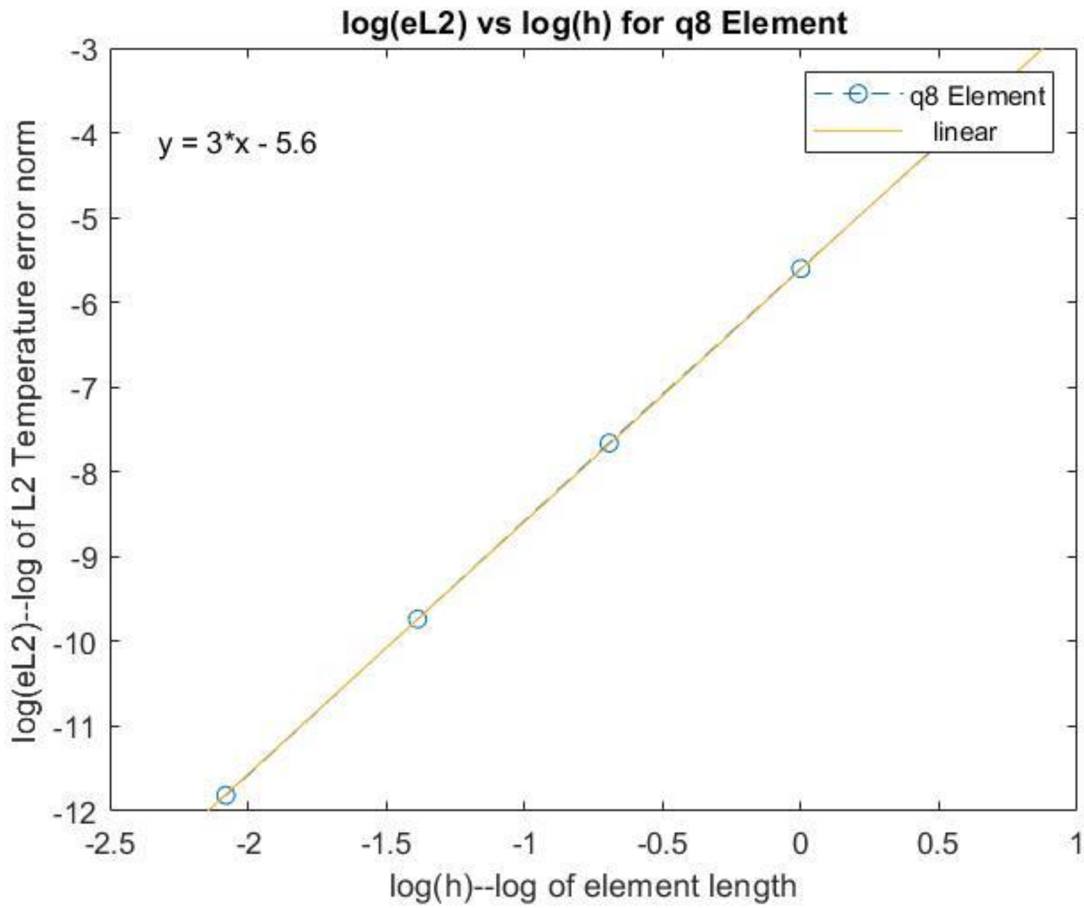


Figure 8: L2 Error Norm for 8 Node Quadrilateral Element

$$L2\ Error\ Norm\ Function, \log(e_{L2}) = C + \alpha \log(h)$$

$$e_{L2} - Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -5.6 + 3\log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 3$. As per the general Mathematical Literature Theory for the quadratic element, the rate of convergence is equal to 3 for the L2 Error norm.

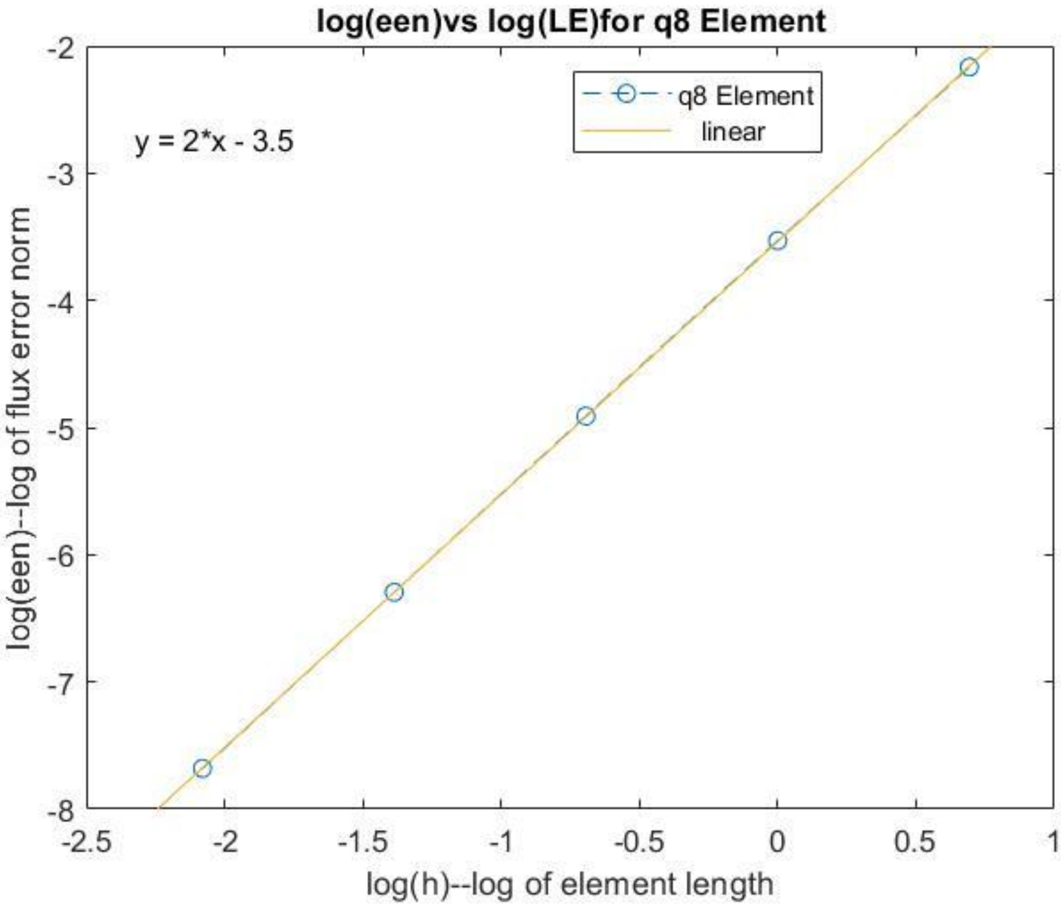


Figure 5: Plot of Energy Norm 8 Node Quadrilateral Element

$$Energy\ Error\ Norm\ Function, \log(e_{en}) = C + \alpha \log(h)$$

$$e_{en} - Energy\ Error\ Norm\ C - Arbitrary\ Constant\ \alpha - Rate\ of\ Convergence\ h - Element\ Size$$

$$L2\ Error\ Norm\ Equatiion\ From\ Plot, \log(e_{L2}) = -3.5 + 2\log(h)$$

Comparing both the Equation, we obtain the convergence Rate $\alpha = 2$. As per the general Mathematical Literature Theory for the quadratic element, the rate of convergence is equal to 2 for energy error norm.

Conclusion Error Norm

1. From the convergence plot L2 error norm of T3 and Q4 element, we observed that the Q4 elements are better than the T3 element as expected.
2. The Convergence in all the L2 and energy error norm is obtained as required.

Finite Element Method MATLAB Solution

MATLAB code

```
function [T,Flux]= project_02(h,r0,r1,etype,R,sbc,nqpts,T0,k)
if nargin == 0
    h=1;
    r0=10;          %milimeter
    r1=20;          %milimeter
    etype='q4';     % q4 q8 q9 t3 t6
    R=3;            %milimeter
    sbc=5.670e-14;   %Wmm-2K-4
    nqpts=5;
    T0=300;         % K
    k=eye(2)*0.017;          %W/mm-K
    th=1;
end
%% input variables %%
xc=0.5*(r1+r0)*cos(pi/4);
yc=0.5*(r1+r0)*sin(pi/4);
Ac=pi*(r1^2-r0^2)/4;
%% Shape function and quarature points
[mesh] = make_project2_mesh(h, r0, r1, etype);
if strcmp(etype, 'q4')
    Shape=@shape_q4;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.conn';
elseif strcmp(etype, 't3')
    Shape=@shape_t3;
    qpts=   [0.1012865073 0.1012865073 0.0629695903;
        0.7974269853 0.1012865073 0.0629695903;
        0.1012865073 0.7974269853 0.0629695903;
        0.4701420641 0.0597158717 0.0661970764;
        0.4701420641 0.4701420641 0.0661970764;
        0.0597158717 0.4701420641 0.0661970764;
        0.3333333333 0.3333333333 0.1125]';
    facep=mesh.conn';
elseif strcmp(etype, 'q8')
    Shape=@shape_q8;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.pconn';
elseif strcmp(etype, 'q9')
    Shape=@shape_q9;
    qpts=Quadrature_2D_Quadilateral_element(nqpts);
    facep=mesh.pconn';
elseif strcmp(etype, 't6')
    Shape=@shape_t6;
    qpts=   [0.1012865073 0.1012865073 0.0629695903;
        0.7974269853 0.1012865073 0.0629695903;
        0.1012865073 0.7974269853 0.0629695903;
        0.4701420641 0.0597158717 0.0661970764;
```

```
        0.4701420641 0.4701420641 0.0661970764;
        0.0597158717 0.4701420641 0.0661970764;
        0.3333333333 0.3333333333 0.1125]';
    facep=mesh.pconn';
end
%% Heat source %%
s=@(x,y) (exp(-(((x-xc)^2+(y-yc)^2)/R^2)));
%% mesh %%
x=mesh.x;
conn=mesh.conn;
%% Average Temperture calculation
fe=0;
for c=conn
    xe=x(:,c);
    for q=qpts
        [N,dNdp]=Shape(q(1:2));
        J=xe*dNdp;
        xp=xe*N;
        S=s(xp(1),xp(2));
        F=S*det(J)*q(end);
        fe=fe+F;
    end
end
Tavg=((fe/(Ac*sbc))+T0^4)^(1/4);
Heff=@(T) (sbc*(T^2+T0^2)*(T+T0));
K=zeros(length(x),length(x));
H=zeros(length(x),length(x));
T=zeros(length(x),1);
Ta=ones(length(x),1)*Tavg;
Fh=zeros(length(x),1);
Fe=zeros(length(x),1);
dT=100;
count=0;
while max(dT)>=0.01
    for c=conn
        xe=mesh.x(:,c);
        Ke=zeros(length(c));
        Hk=zeros(length(c));
        TA=Ta(c);
        for q=qpts
            [N,dNdp]=Shape(q);
            J=xe*dNdp;
            B=dNdp/J;
            w=q(end);
            Ke=Ke+B*k*th*B'*det(J)*w;
            f=s(xe(1,:)*N,xe(2,:)*N);
            TAP=TA'*N;
            hk=N*Heff(TAP)*N'*det(J)*w;
            Hk=Hk+hk;
            fh=N*Heff(TAP)*det(J)*w*T0;
            Fh(c)=Fh(c)+fh;
            Fe(c)=Fe(c)+N*f*det(J)*w;
```

```
        end
        K(c,c)=K(c,c)+Ke;
        H(c,c)=H(c,c)+Hk;
    end
    T=((K+H)\(Fe+Fh));
    dT=abs(T-Ta);
    Ta=T;
    count=count+1;
    plot(count,max(dT),'--ob')
    hold on
end
hold off
figure()
patch('Faces',facep,'Vertices',mesh.x','FaceVertexCData',T,'FaceColor'
,'interp');
colorbar
%%  conn and pconn need to be used in this case.. (works now for both
temperature and flux plots if u want to use)
%% better looking than above code as lines are thinner
% p.vertices =mesh.x';
% p.faces = mesh.conn';
% p.facecolor ='interp';
% p.facevertexcdata=T; %plot temp
% p.edgealpha = 0.2; %transparency
% clf;
% patch(p);
% colorbar
%% flux
%
%% [Add this to the above code once we get correct answer] %%
%
SpA = spalloc(length(mesh.x), length(mesh.x), 9*(length(mesh.x)));
den = zeros(length(mesh.x),2);
for c=conn
    xe=x(:,c);
    SpAe = zeros(length(c));
    for q=qpts
        [N,dNdp]=Shape(q);
        J=xe*dNdp;
        B=dNdp/J;
        w=q(end);
        Tf=T(c);
        Q=-Tf'*B*k;
        SpAe = SpAe+N*N'*det(J)*w;
        den(c,:) = den(c,:) + N*Q*det(J)*w;
    end
    SpA(c,c) = SpA(c,c) + SpAe;
end
flux=SpA\den;
Flux = sqrt(flux(:,1).^2+flux(:,2).^2);
figure()
```

```
patch('Faces',facep,'Vertices',mesh.x','FaceVertexCData',Flux,'FaceCol
or','interp');
colorbar

end

function[N, dNdp] = shape_q4(p)
N = [(1/4)*(1-p(1))*(1-p(2));
     (1/4)*(1+p(1))*(1-p(2));
     (1/4)*(1+p(1))*(1+p(2));
     (1/4)*(1-p(1))*(1+p(2))];
dNdp = [(1/4)*(p(2)-1),(1/4)*(p(1)-1);
     (1/4)*(1-p(2)),(-1/4)*(p(1)+1);
     (1/4)*(1+p(2)),(1/4)*(1+p(1));
     (-1/4)*(1+p(2)),(1/4)*(1-p(1))];
end
function[N, dNdp] = shape_q8(p)
N = [(-1/4)*(1-p(1))*(1-p(2))*(1+p(1)+p(2));     %1-1
     (-1/4)*(1+p(1))*(1-p(2))*(1-p(1)+p(2));     %3-2
     (-1/4)*(1+p(1))*(1+p(2))*(1-p(1)-p(2));     %5-3
     (-1/4)*(1-p(1))*(1+p(2))*(1+p(1)-p(2));     %7-4
     (1/2)*(1-p(1))*(1+p(1))*(1-p(2));           %2-5
     (1/2)*(1+p(1))*(1+p(2))*(1-p(2));           %4-6
     (1/2)*(1-p(1))*(1+p(1))*(1+p(2));           %6-7
     (1/2)*(1-p(1))*(1+p(2))*(1-p(2))];          %8-8
dNdp = [(-1/4)*(p(2)-1)*(2*p(1)+p(2)),(-1/4)*(p(1)-1)*(2*p(2)+p(1));
%1-1
      (1/4)*(p(2)-1)*(-2*p(1)+p(2)),(1/4)*(p(1)+1)*(2*p(2)-p(1));
%3-2
      (1/4)*(1+p(2))*(2*p(1)+p(2)),(1/4)*(1+p(1))*(p(1)+2*p(2));
%5-3
      (-1/4)*(1+p(2))*(p(2)-2*p(1)),(-1/4)*(p(1)-1)*(2*p(2)-p(1));
%7-4
     p(1)*(p(2)-1),(1/2)*(1+p(1))*(-1+p(1));
%2-5
      (-1/2)*(p(2)+1)*(p(2)-1),(-1)*(1+p(1))*p(2);
%4-6
      (-1)*p(1)*(1+p(2)),(-1/2)*(1+p(1))*(p(1)-1);
%6-7
      (1/2)*(1+p(2))*(p(2)-1),p(2)*(p(1)-1)];
%8-8
end
function[N, dNdp] = shape_q9(p)
N = [0.5*p(1)*(p(1)-1)*0.5*p(2)*(p(2)-1);
    0.5*p(1)*(p(1)+1)*0.5*p(2)*(p(2)-1);
    0.5*p(1)*(p(1)+1)*0.5*p(2)*(p(2)+1);
    0.5*p(1)*(p(1)-1)*0.5*p(2)*(p(2)+1);
     (1-p(1)^2)*0.5*p(2)*(p(2)-1);
    0.5*p(1)*(p(1)+1)*(1-p(2)^2);
     (1-p(1)^2)*0.5*p(2)*(p(2)+1);
    0.5*p(1)*(p(1)-1)*(1-p(2)^2);
     (1-p(1)^2)*(1-p(2)^2)];
```

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

function[N, dNdp] = shape_t3(p)
N = [p(1);
    p(2);
    1 - p(1) - p(2)];
dNdp = [1, 0;
    0, 1;
    -1, -1];
end
function[N, dNdp] = shape_t6(p)
N = [p(1)*(2*p(1)-1);
    p(2)*(2*p(2)-1);
    (1-p(2)-p(1))*(2*(1-p(2)-p(1))-1);
    4*p(1)*p(2);
    4*p(2)*(1-p(2)-p(1));
    4*p(1)*(1-p(2)-p(1))];

dNdp =  [4*p(1) - 1, 0 ;
        0, 4*p(2) - 1;
        -3+4*p(1)+4*p(2),-3+4*p(1)+4*p(2);
        4*p(2), 4*p(1);
        -4*p(2),(4-8*p(2)-4*p(1));
        4-4*p(2)-8*p(1),-4*p(1)];

end
```

Outputs

[NOTE: For all the plots presented below the unit of temperature is Kelvin and Unit of heat flux is W/mm²]

Plot for Iterative convergence for Q4 Element

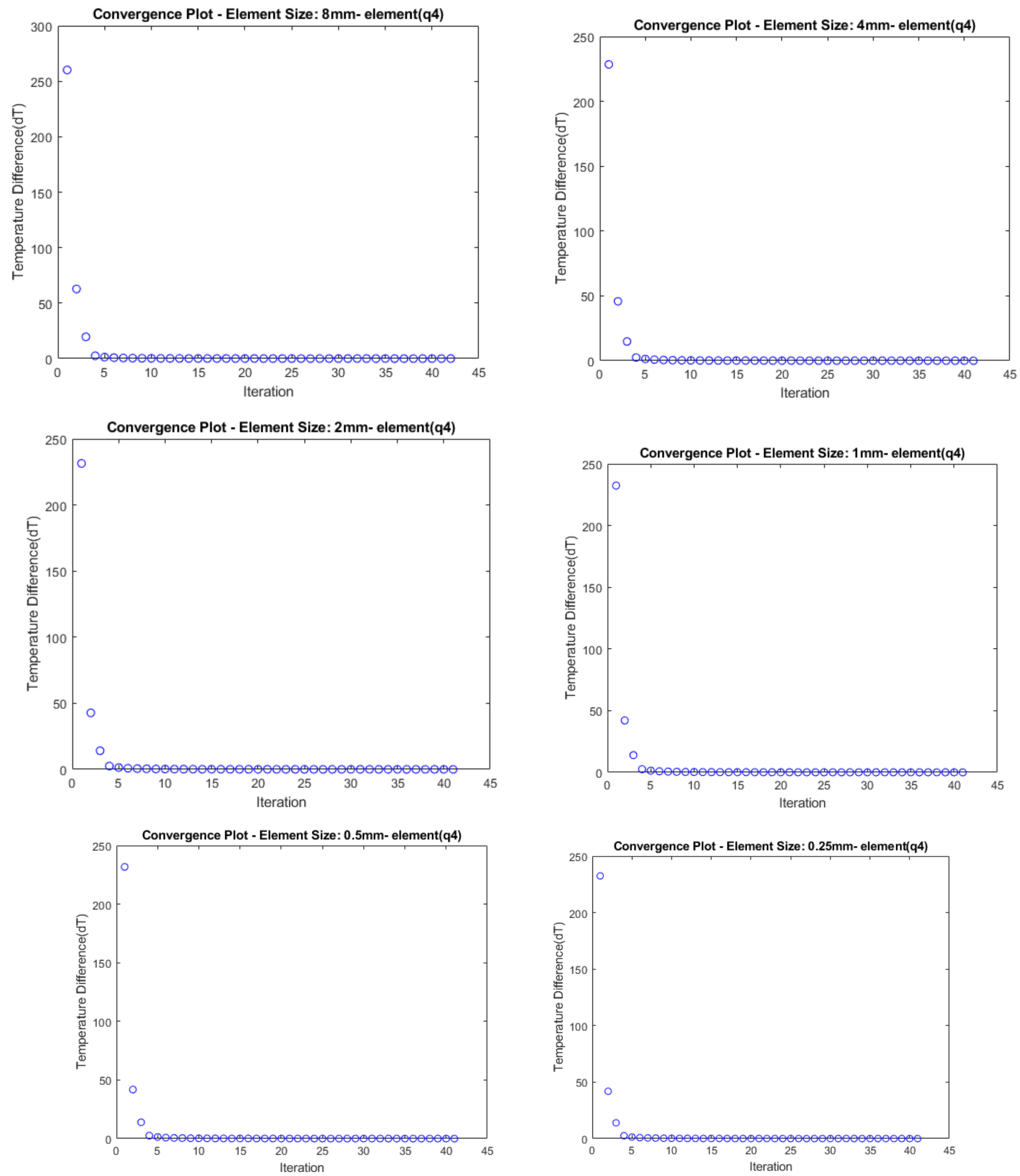


Figure 9: Iteration plot Temperature Convergence for Q4 element

Approximate Number of Iteration observed are 41

The plot of Temperature Profile Q4 Element

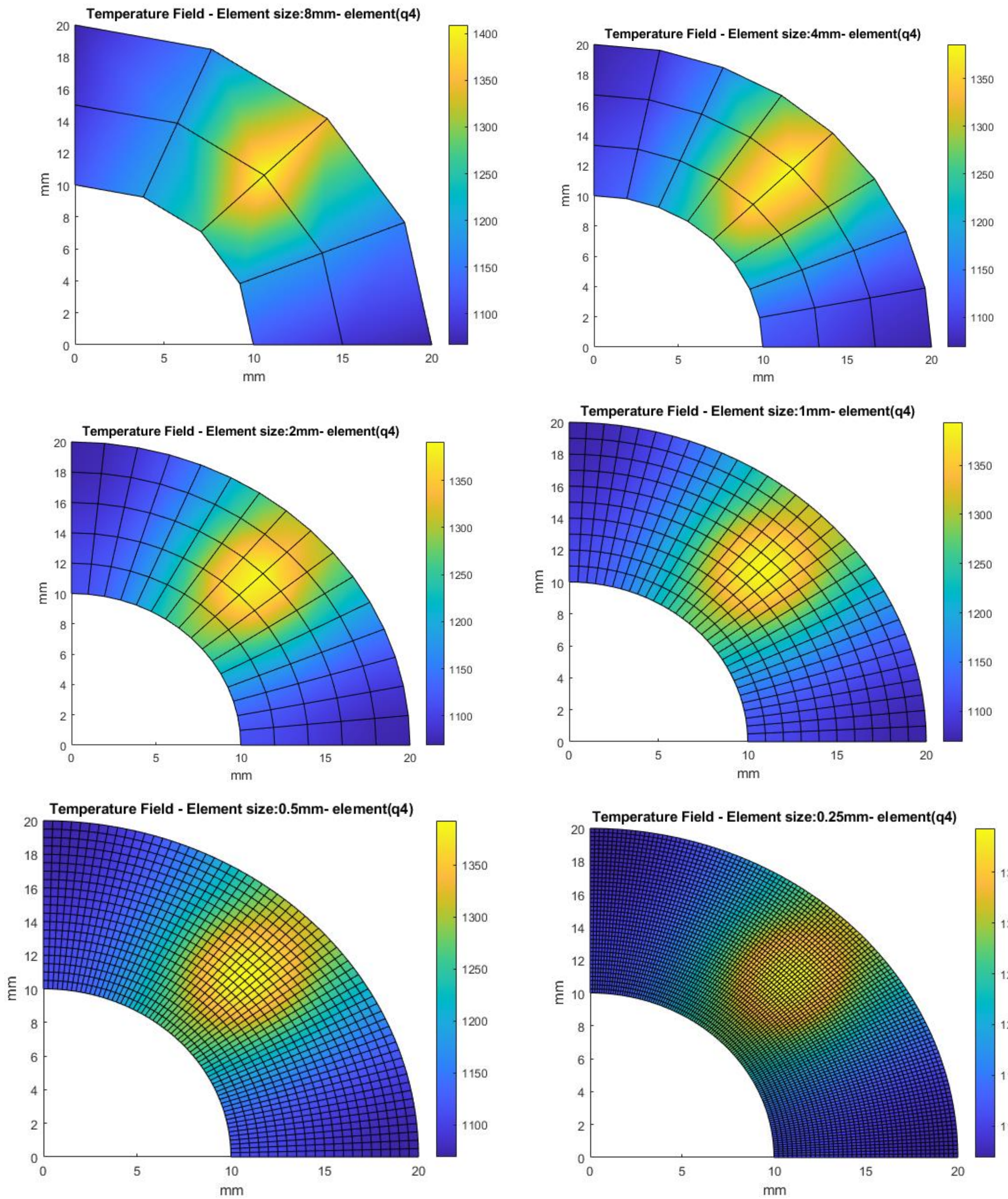


Figure 10: Temperature Field Q4 Element

The plot of Heat Flux Profile Q4 Element

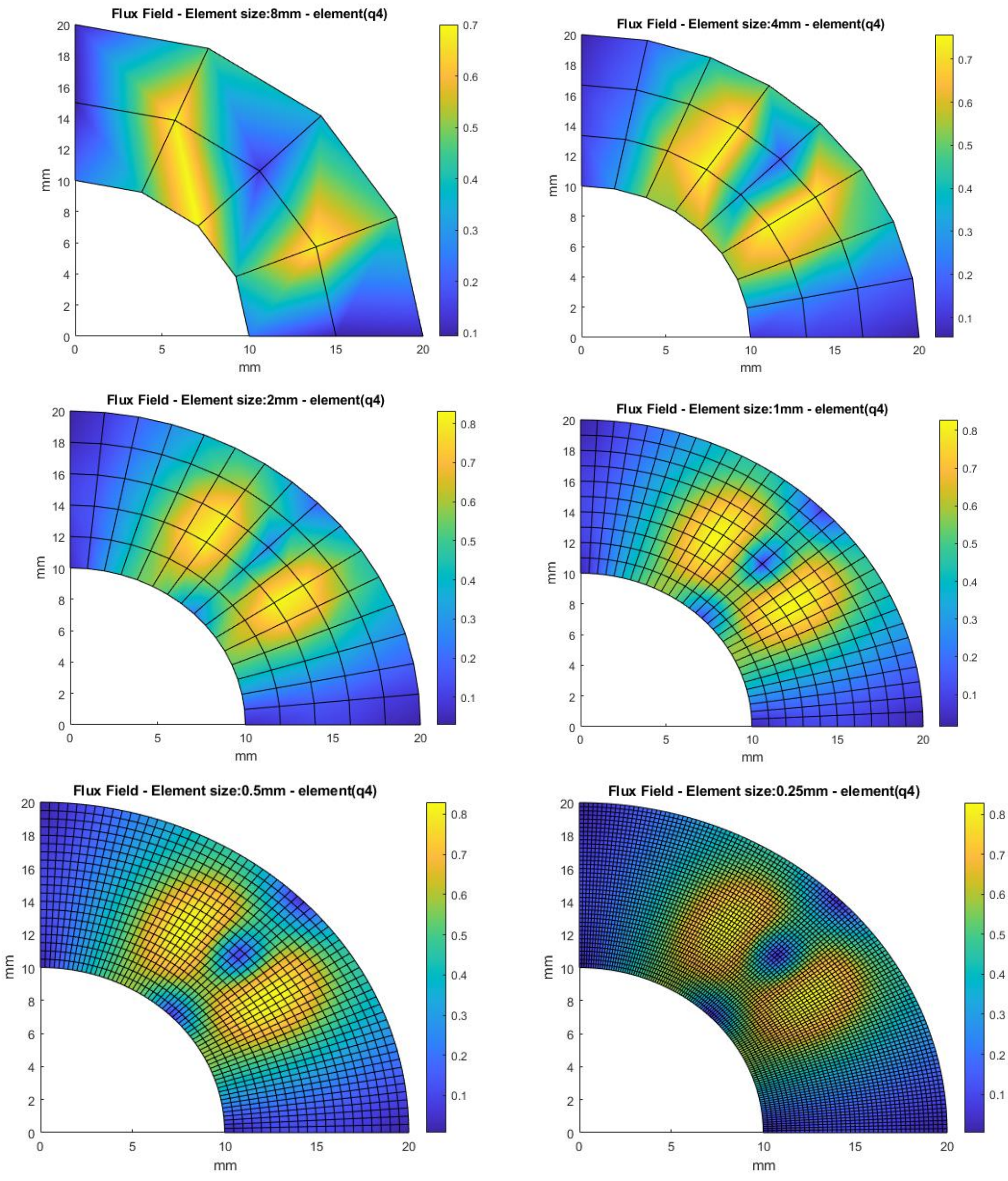


Figure 11: Heat Flux Profile Q4 Element

Plot for Iterative convergence for T3 Element

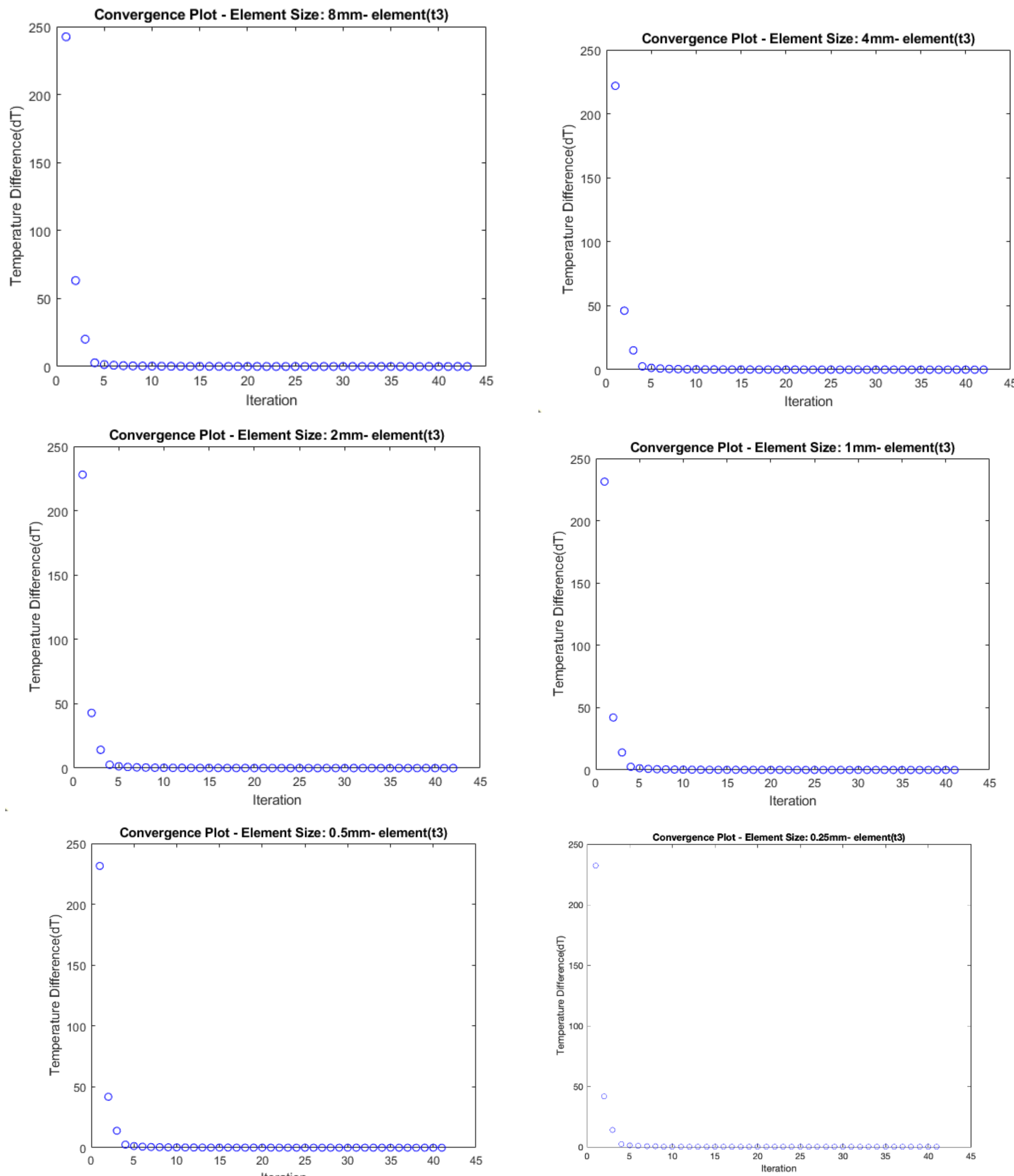


Figure 12: Iteration plot Temperature Convergence for T3 Element

Approximate Number of Iteration observed is 41.

The plot of Temperature Profile T3 Element

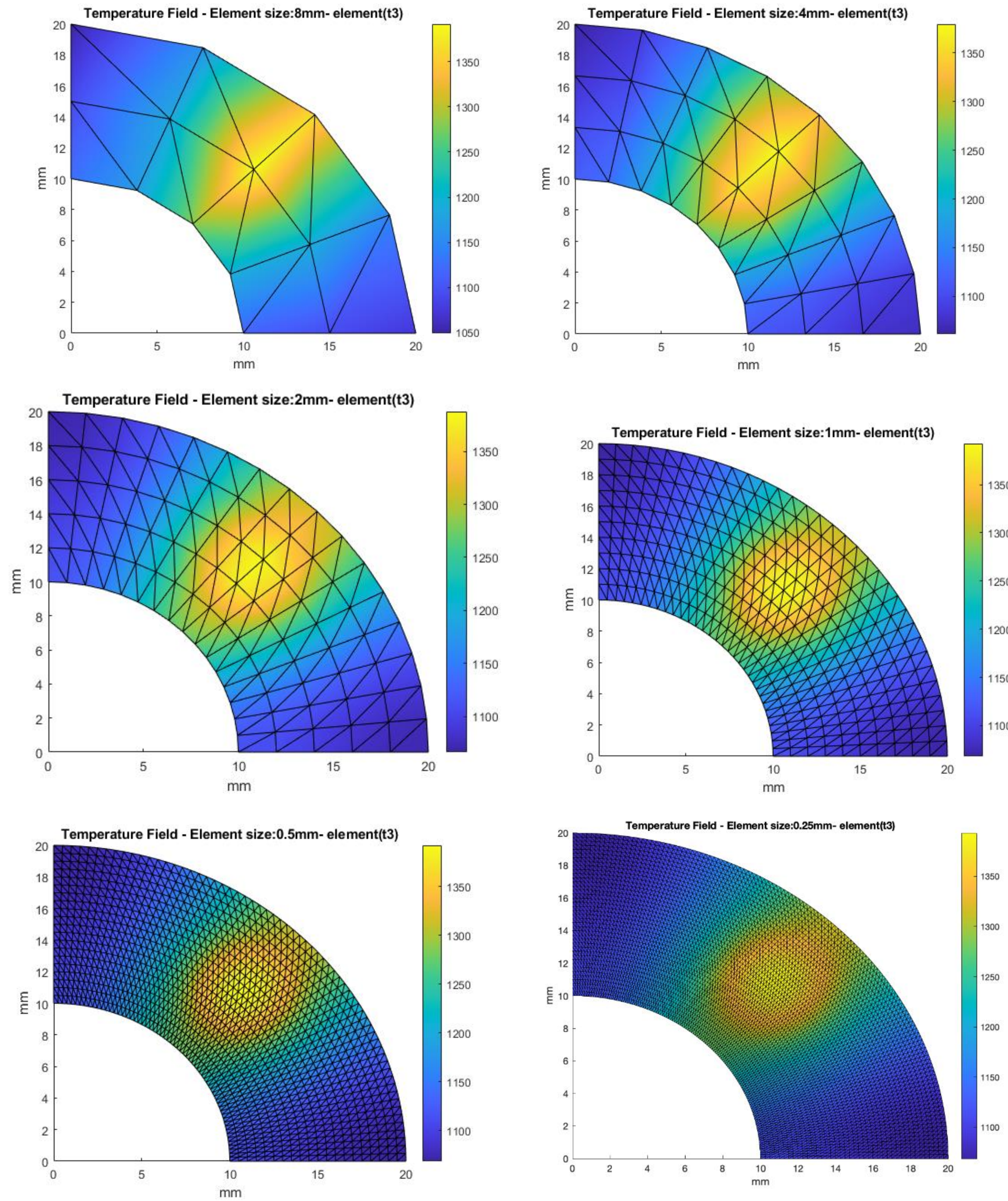


Figure 13: Temperature Profile for T3 Element

Plot for Heat Flux Field for T3 Element

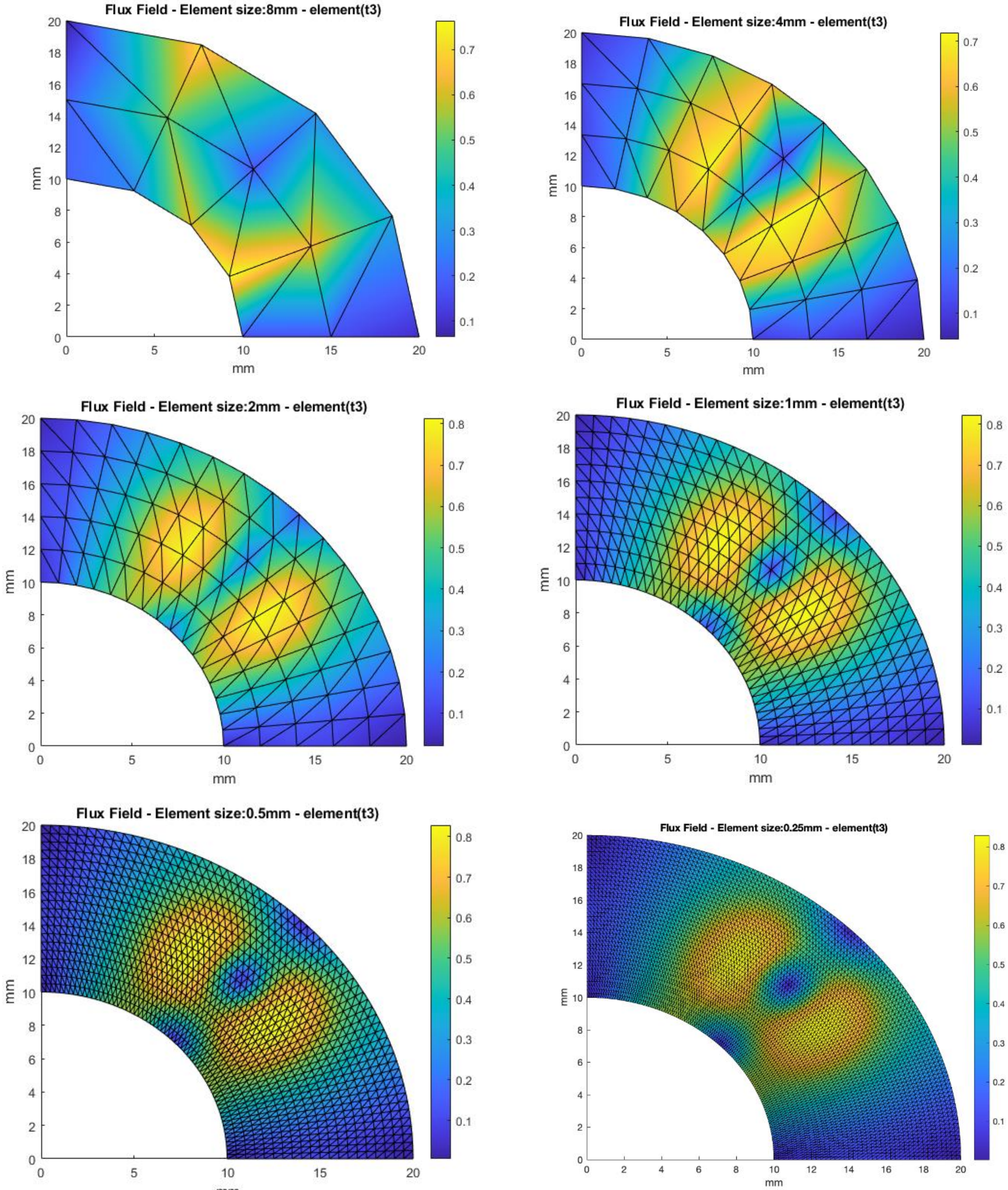


Figure 14: Heat Flux Field for T3 Element

Plot for Iterative convergence for Q8 Element

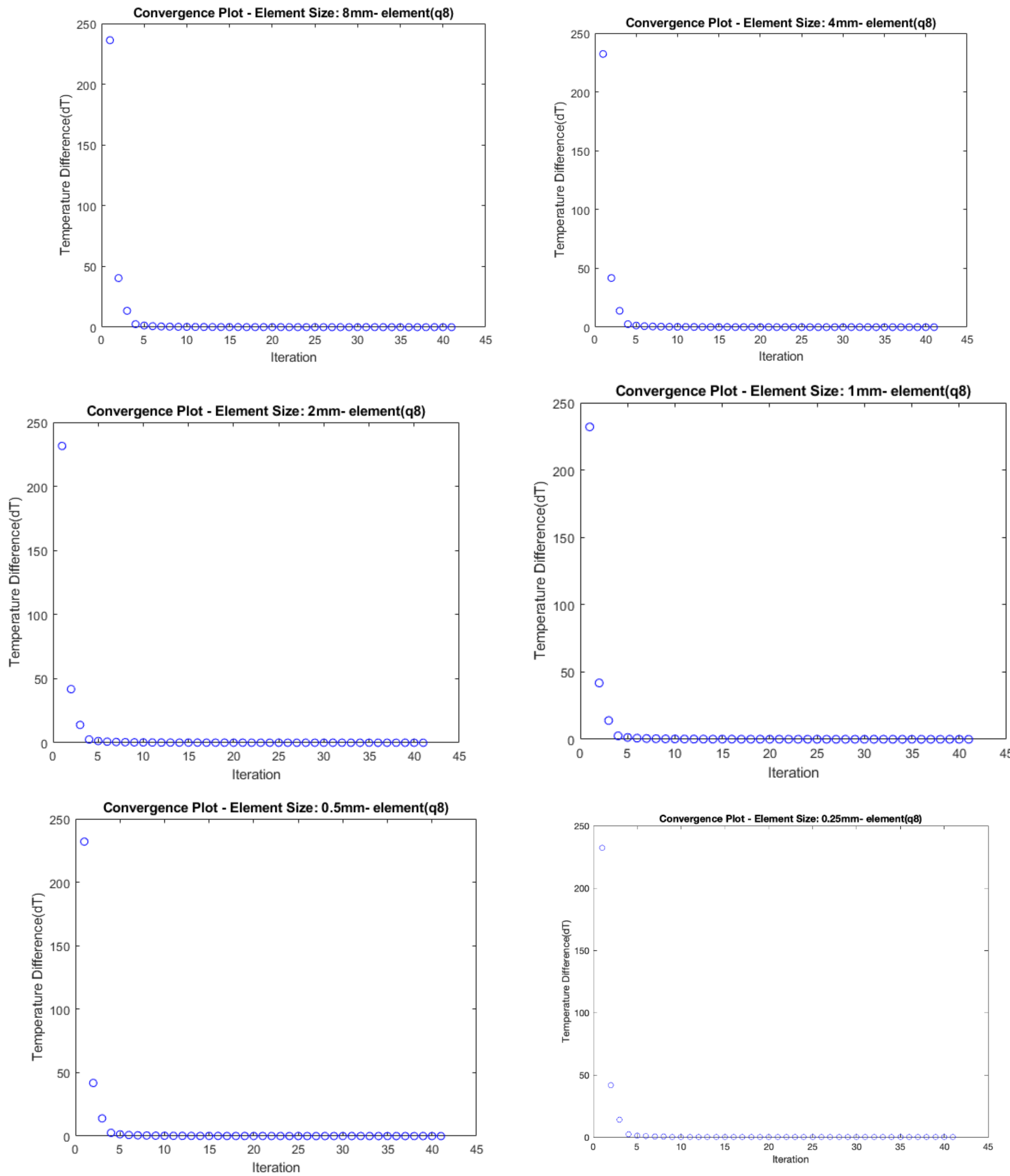


Figure 15: Iteration plot Temperature Convergence for Q8 element

Approximate Number of Iteration observed are 41

The plot of Temperature Profile Q8 Element

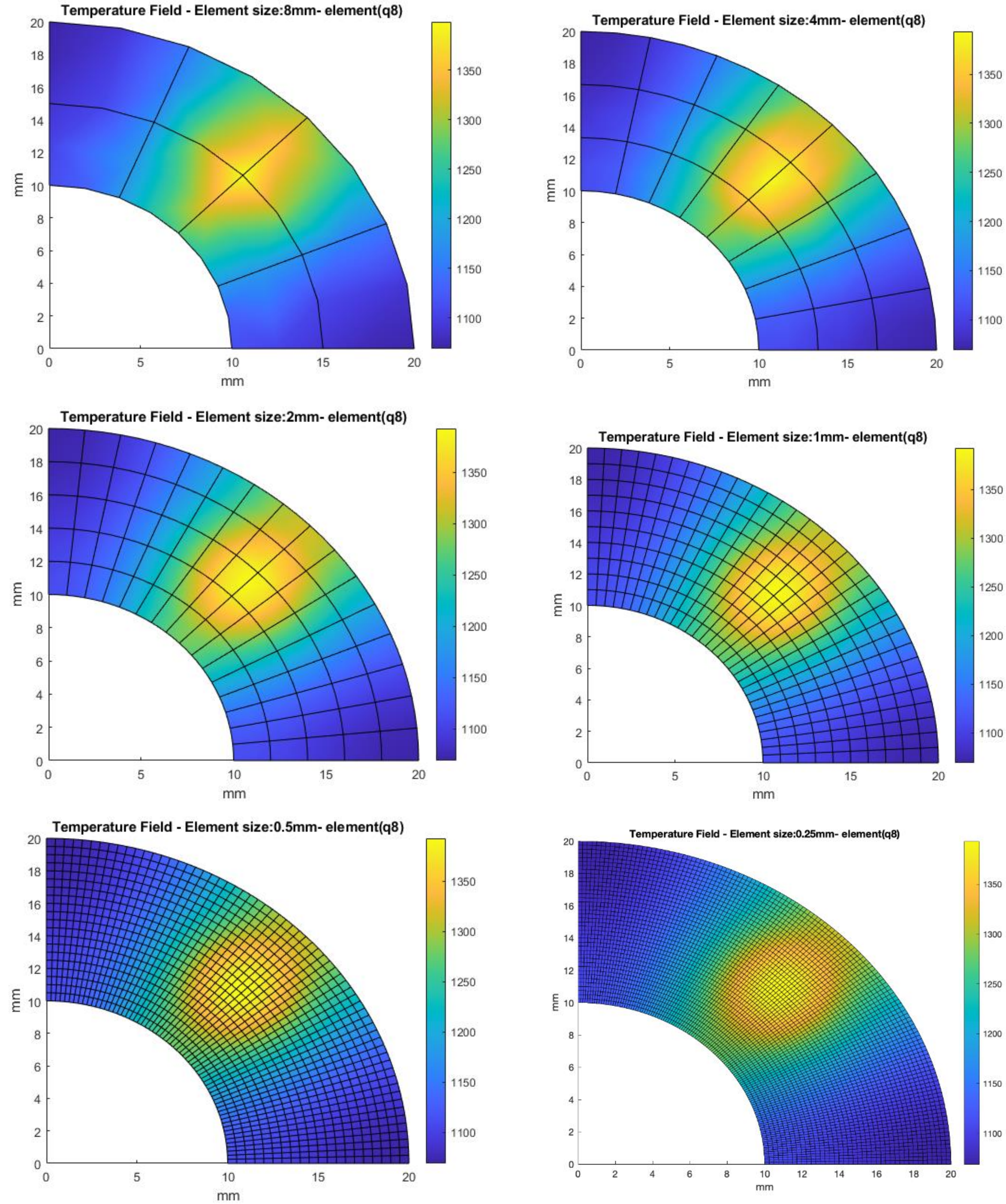


Figure 16: Temperature Field Q8 Element

Plot for Heat Flux field for Q8 Element

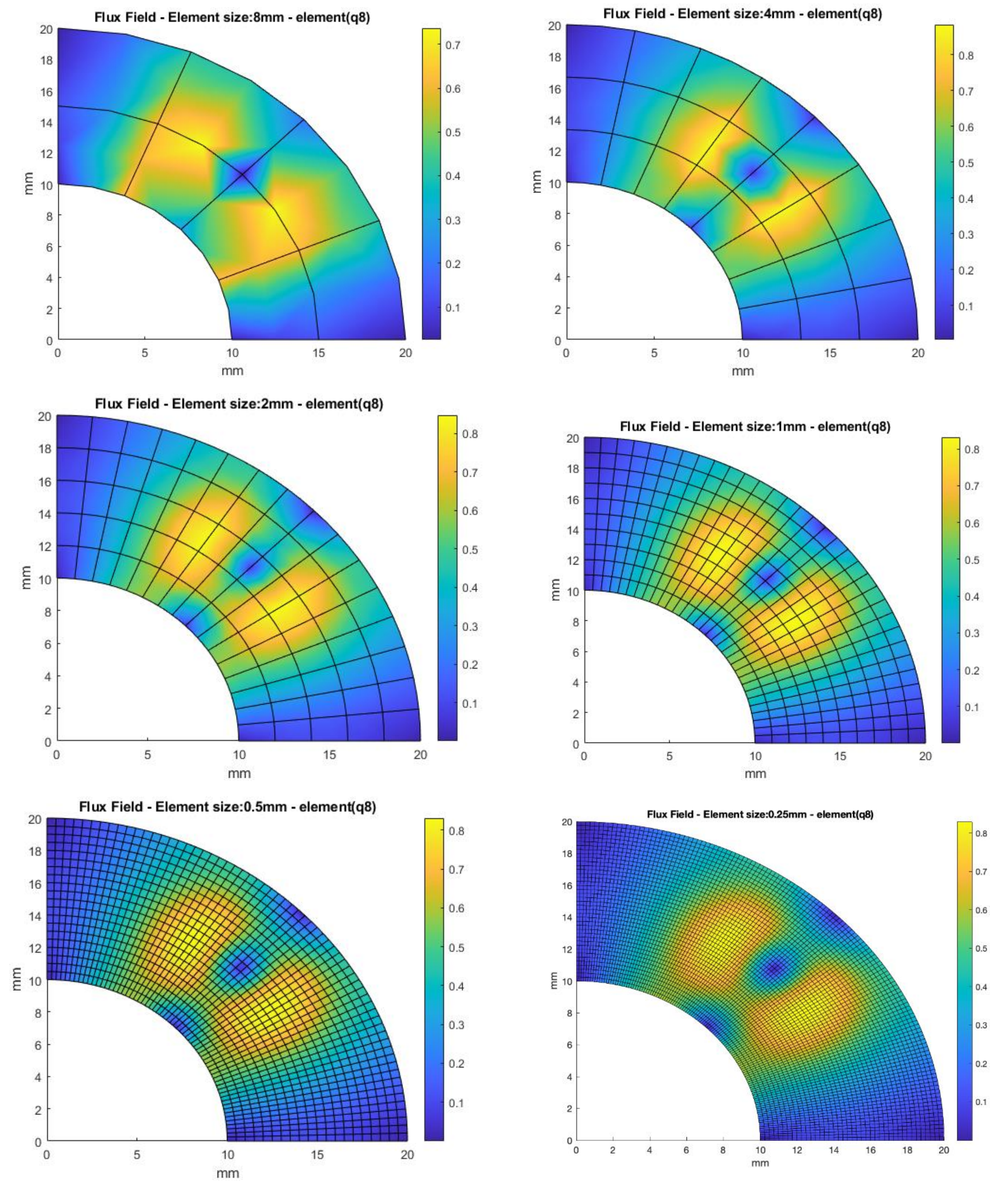


Figure 17: Heat Flux Field Q8 Element

Plot for Iterative convergence for T6 Element

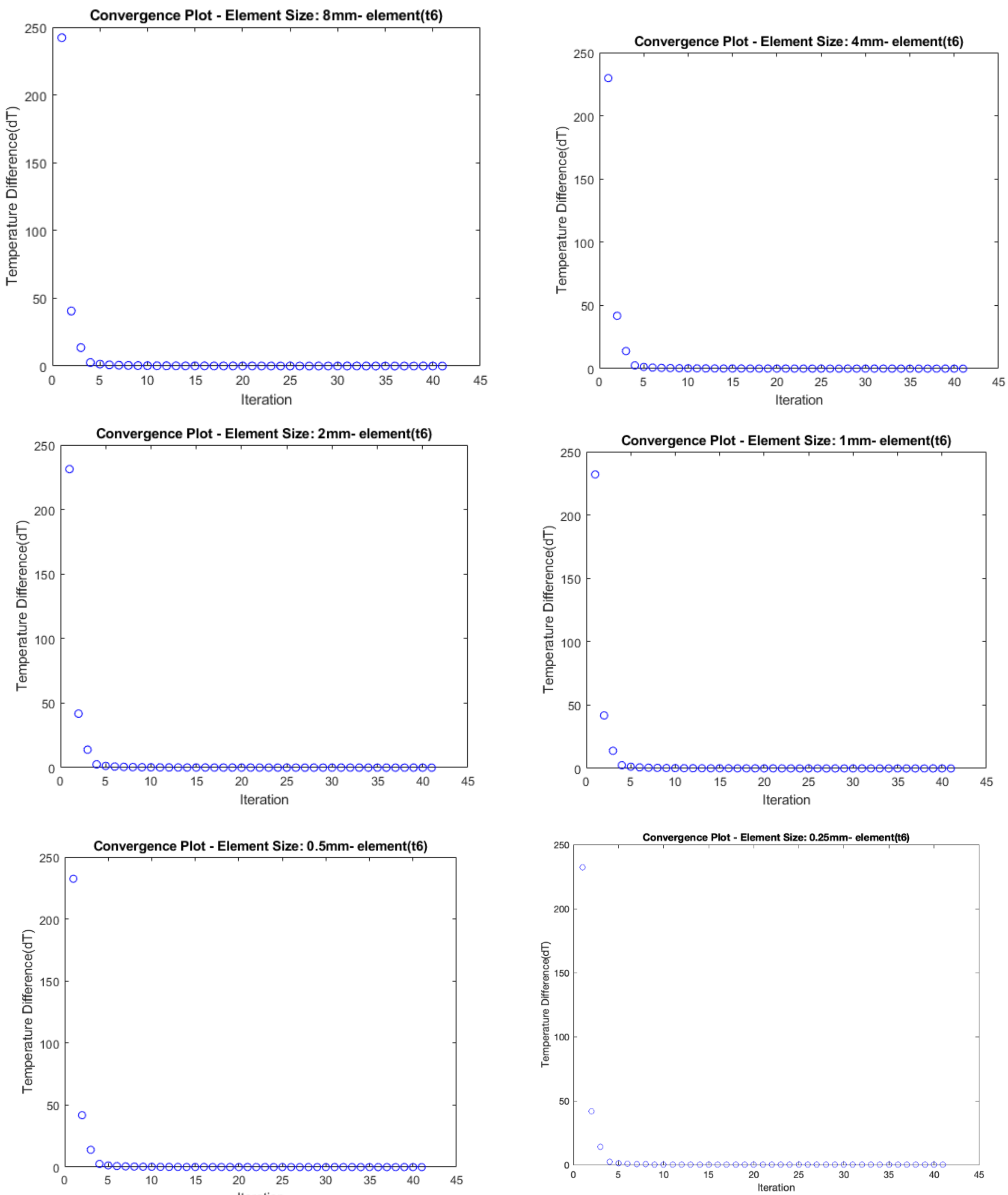


Figure 18: Iteration Plot Temperature Convergence for T6 Element

The plot of Temperature Profile Q8 Element

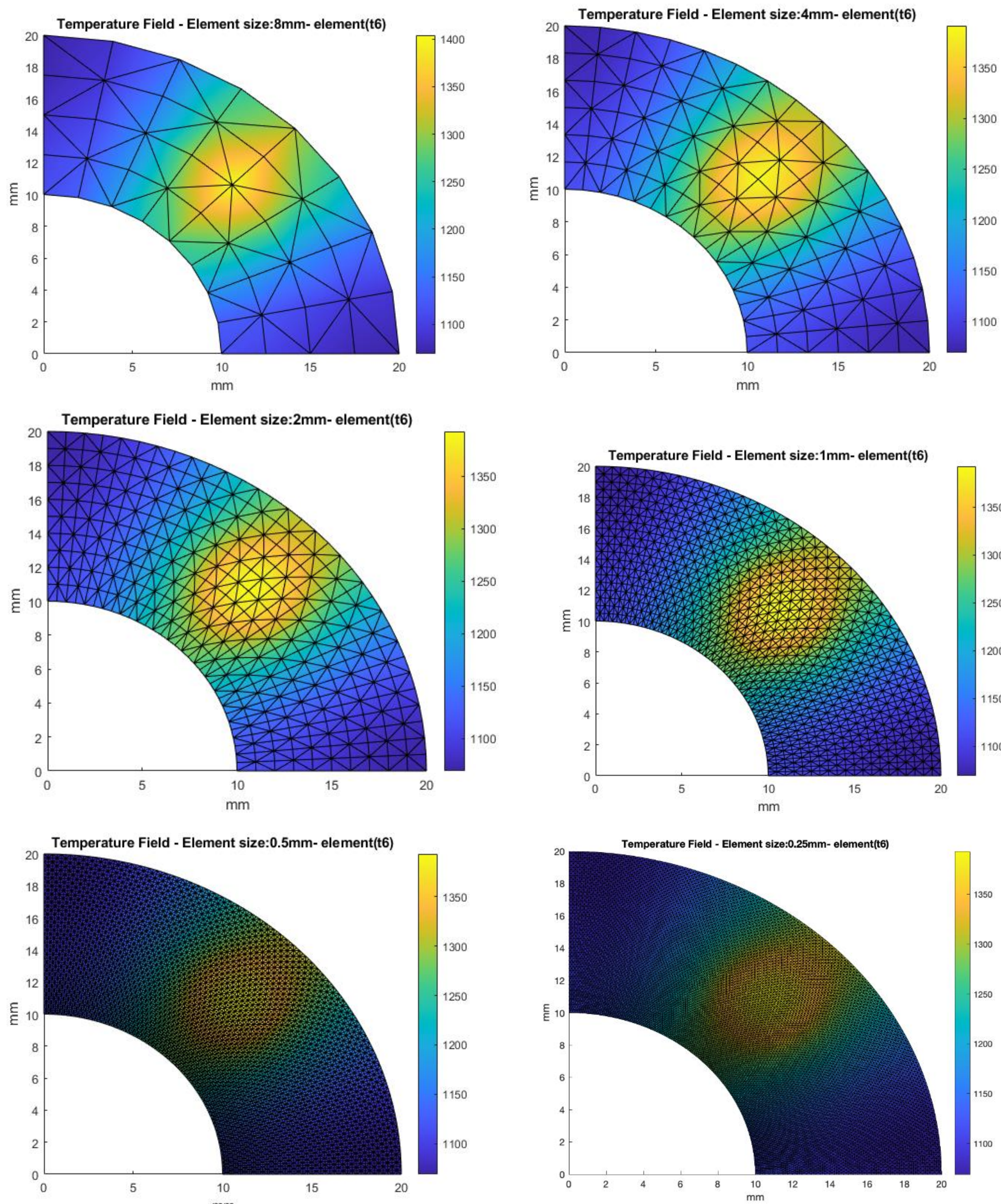


Figure 19: Temperature Profile T6 Element

Plot for Heat Flux field for T6 Element

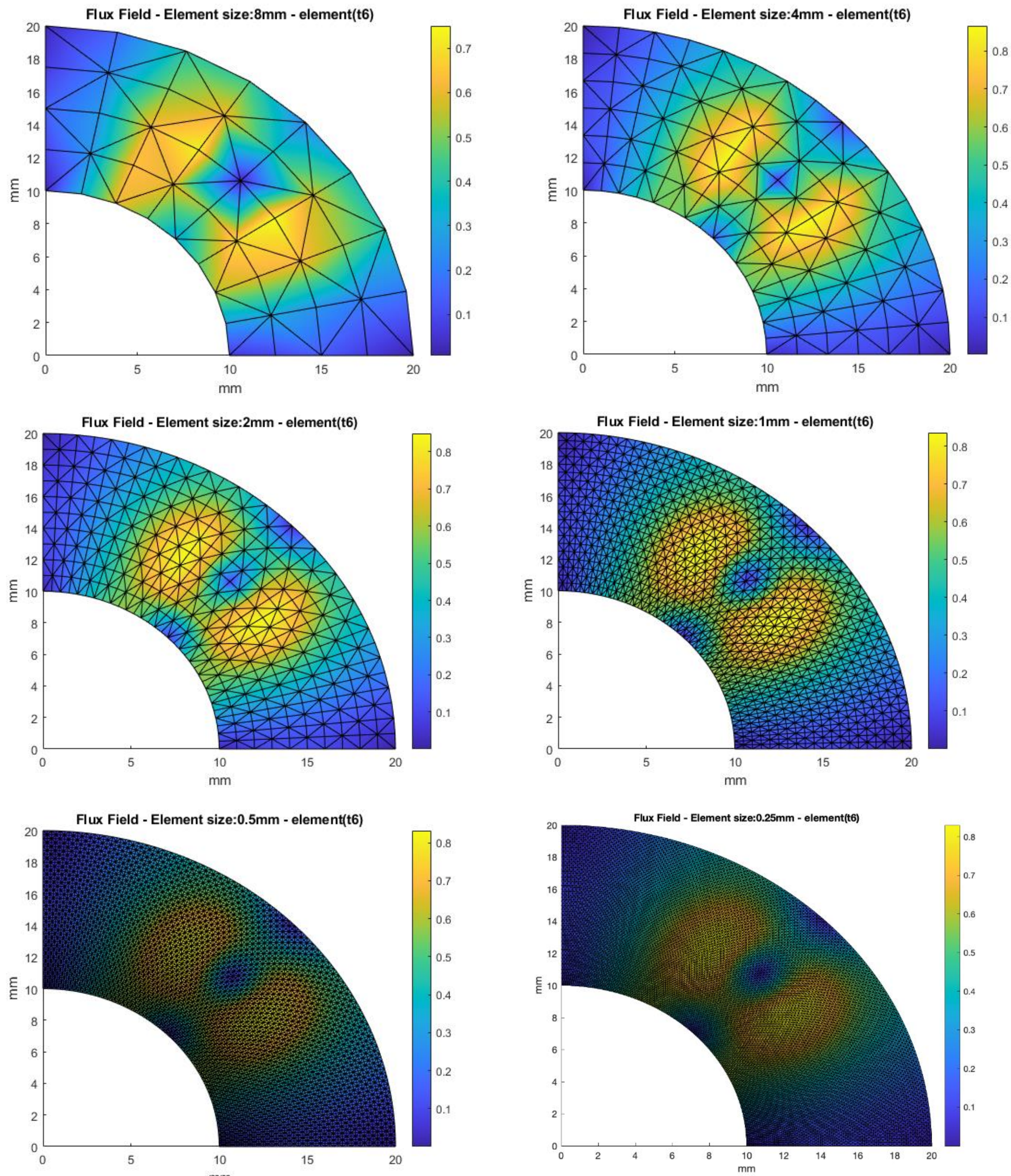


Figure 20: Heat Flux Field T6 Element

Conclusion

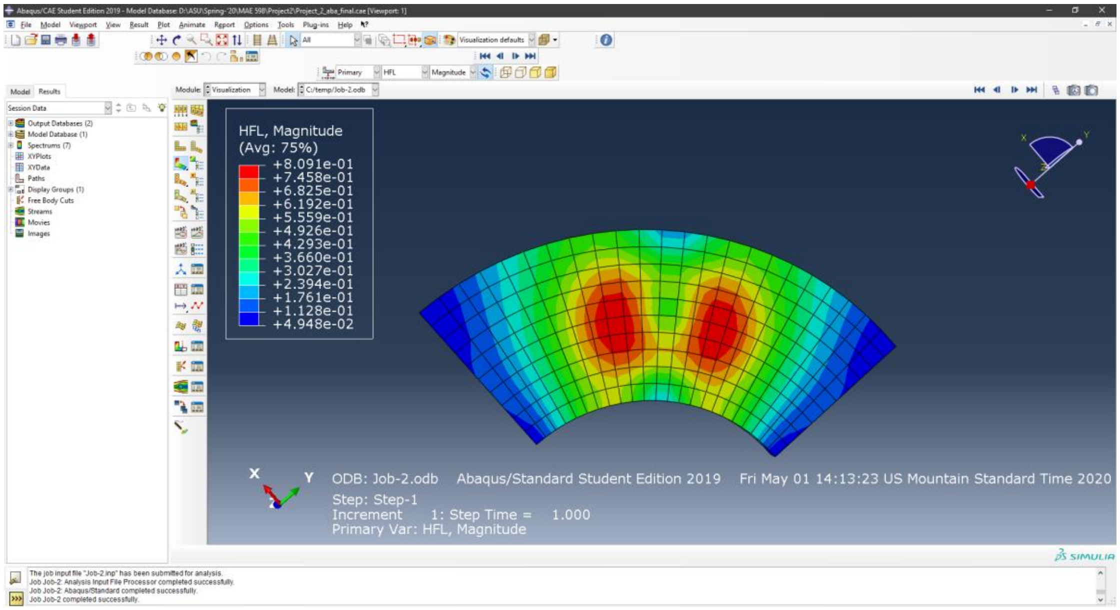
Table: Data of Each Element Type with Different Element Size

Element Type	Element Size (mm)	Maximum Temperature (K)	Maximum Heat Flux (W/mm^2)
Q4	8	1409.196	0.700332
	4	1385.599	0.756773
	2	1390.876	0.832064
	1	1393.604	0.828168
	0.5	1392.483	0.829734
	0.25	1393.247	0.829626
Q8	8	1398.906	0.736952
	4	1393.639	0.883866
	2	1392.8	0.847098
	1	1392.717	0.831534
	0.5	1392.784	0.830922
	0.25	1392.789	0.829775
T3	8	1391.727	0.763293
	4	1379.363	0.718728
	2	1387.722	0.813719
	1	1392.67	0.823244
	0.5	1392.255	0.828709
	0.25	1392.3	0.8294
T6	8	1404.069	0.750546
	4	1391.285	0.86598
	2	1392.544	0.849187
	1	1392.747	0.835563
	0.5	1393.189	0.831821
	0.25	1393.2	0.8302

Abaqus Solution

Plot temperature and heat flux fields for linear elements

Q4 Element



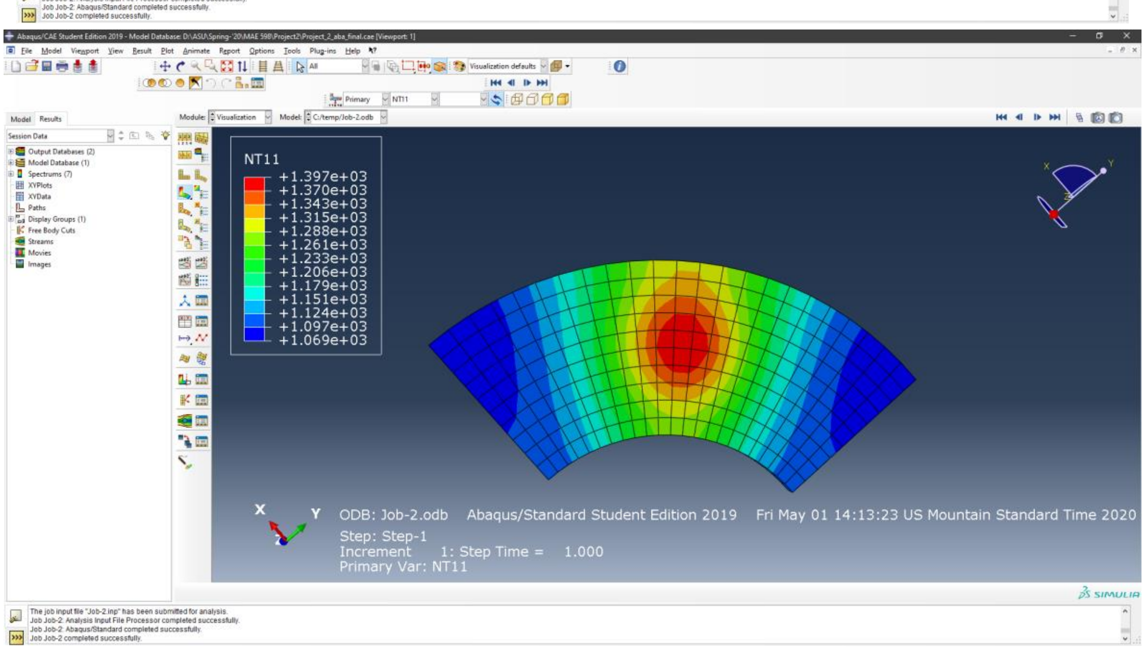
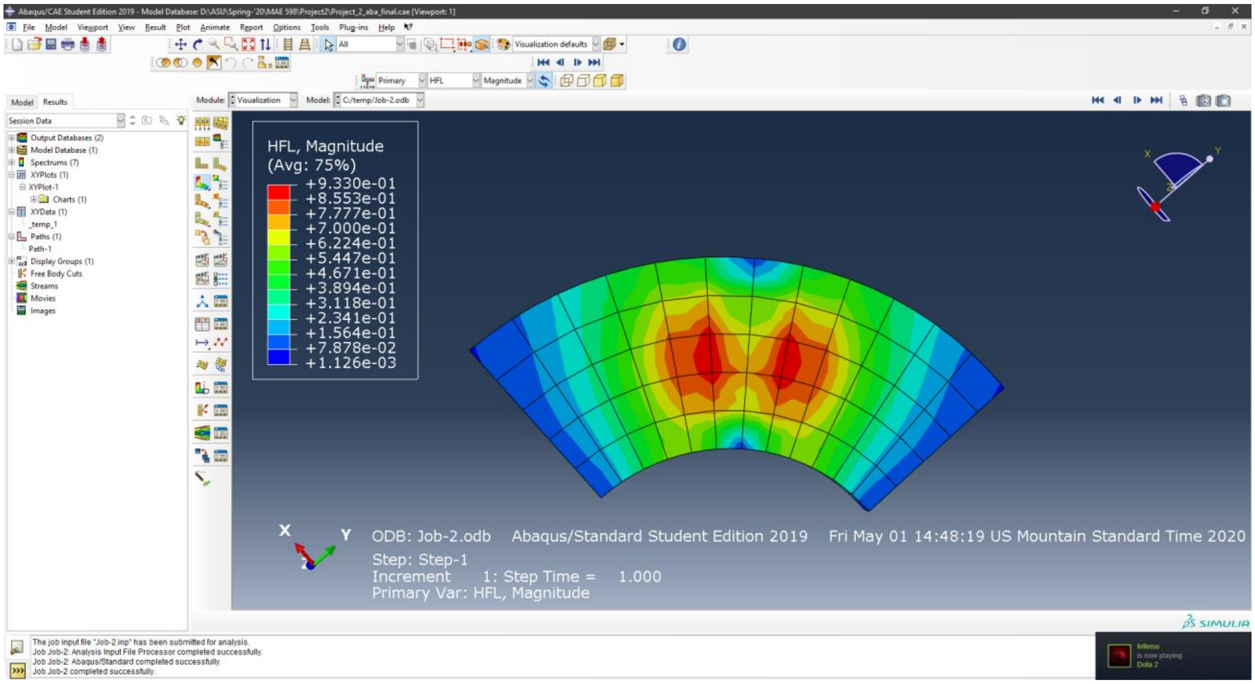


Figure 21: Heat Flux and Temperature Field Abaqus Solution Q4 Element

	Element Size (h)	1 mm	Absolute Error (%)
Temperature	Abaqus	1397	0.24
	FEM	1393.604	
Heat Flux	Abaqus	0.8091	2.3
	FEM	0.828168	

Q8 Element



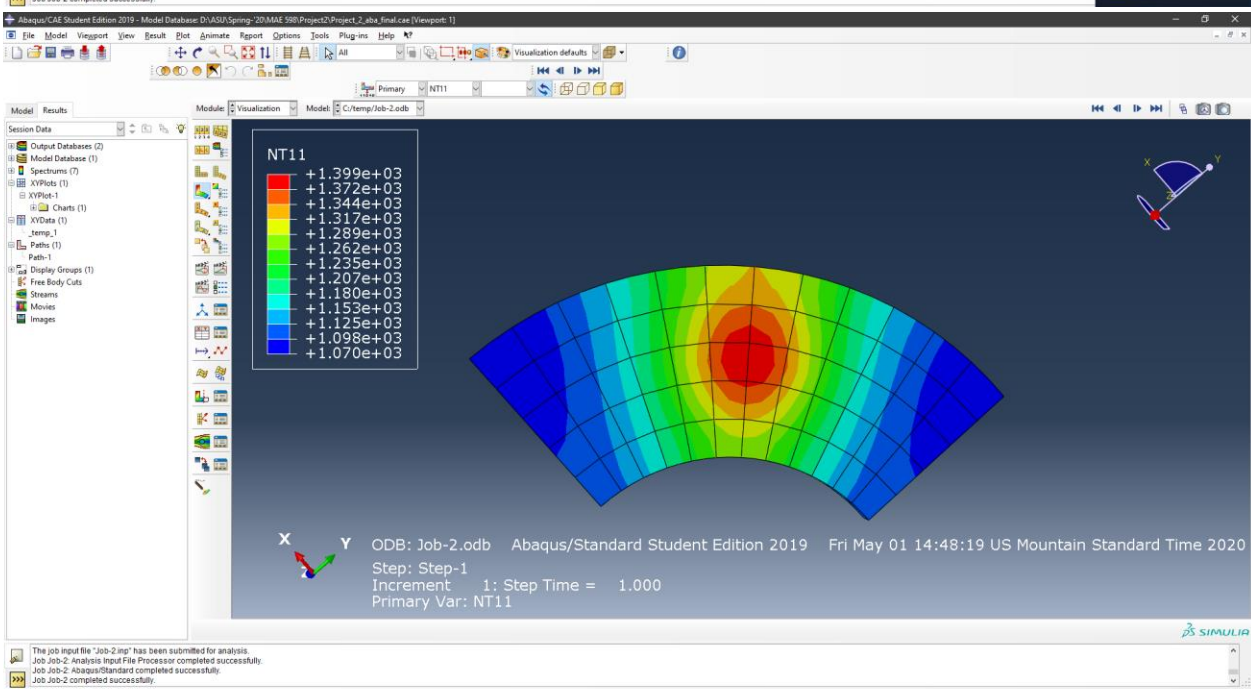
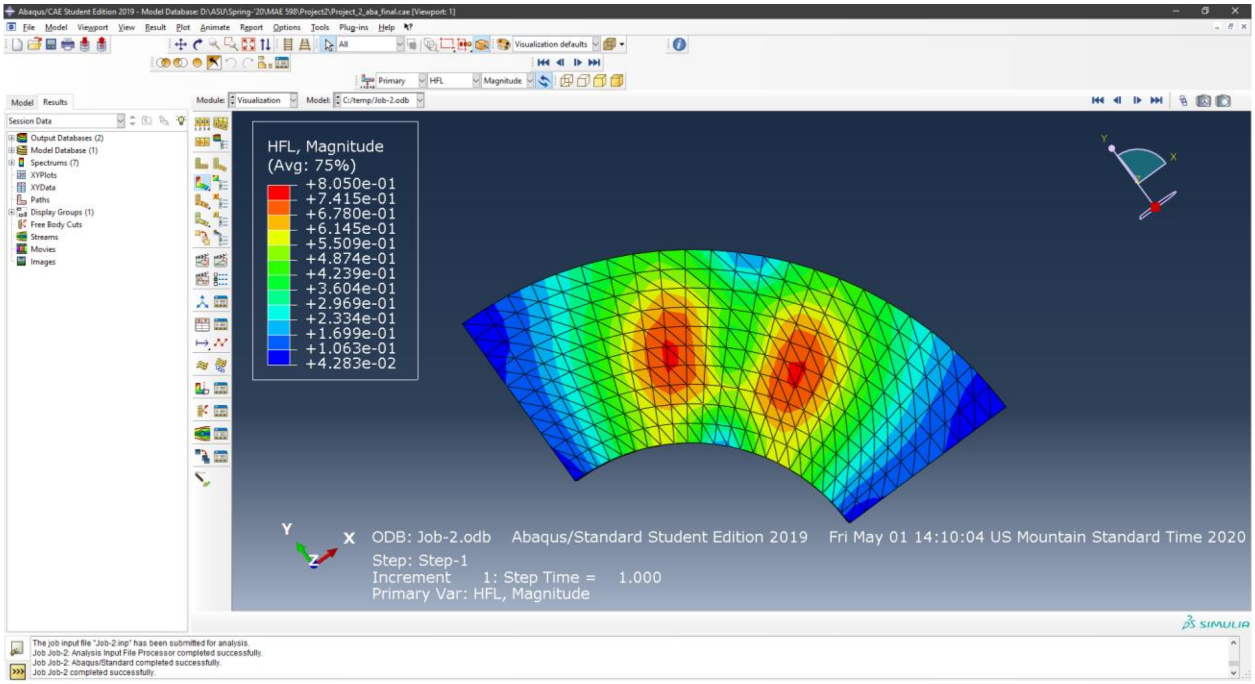


Figure 22: Heat Flux and Temperature Field Abaqus Solution Q8 Element

	Element Size (h)	2 mm	Absolute Error (%)
Temperature	Abaqus	1399	0.45
	FEM	1392.8	
Heat Flux	Abaqus	0.9330	10.1
	FEM	0.847098	

T3 Element



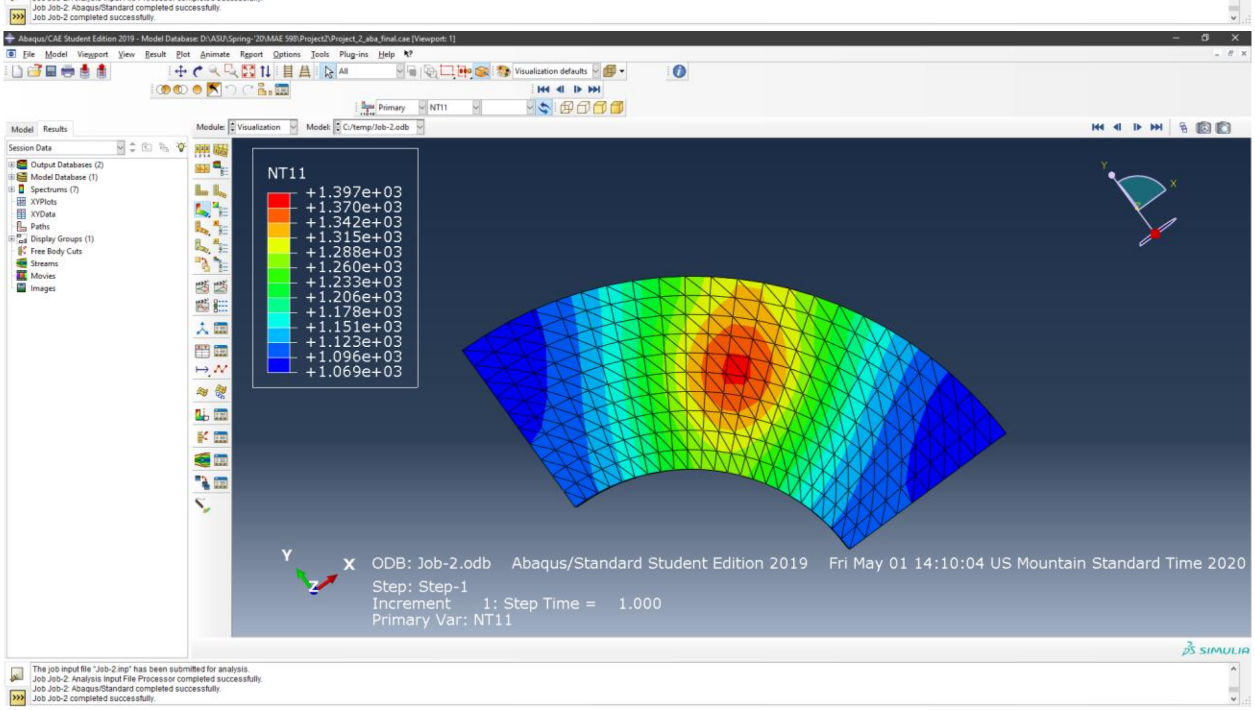


Figure 23: Heat Flux and Temperature Field Abaqus Solution T3 Element

	Element Size (h)	1 mm	Absolute Error
Temperature	Abaqus	1397	0.31
	FEM	1392.67	
Heat Flux	Abaqus	0.8050	2.2
	FEM	0.823244	

T6 Element

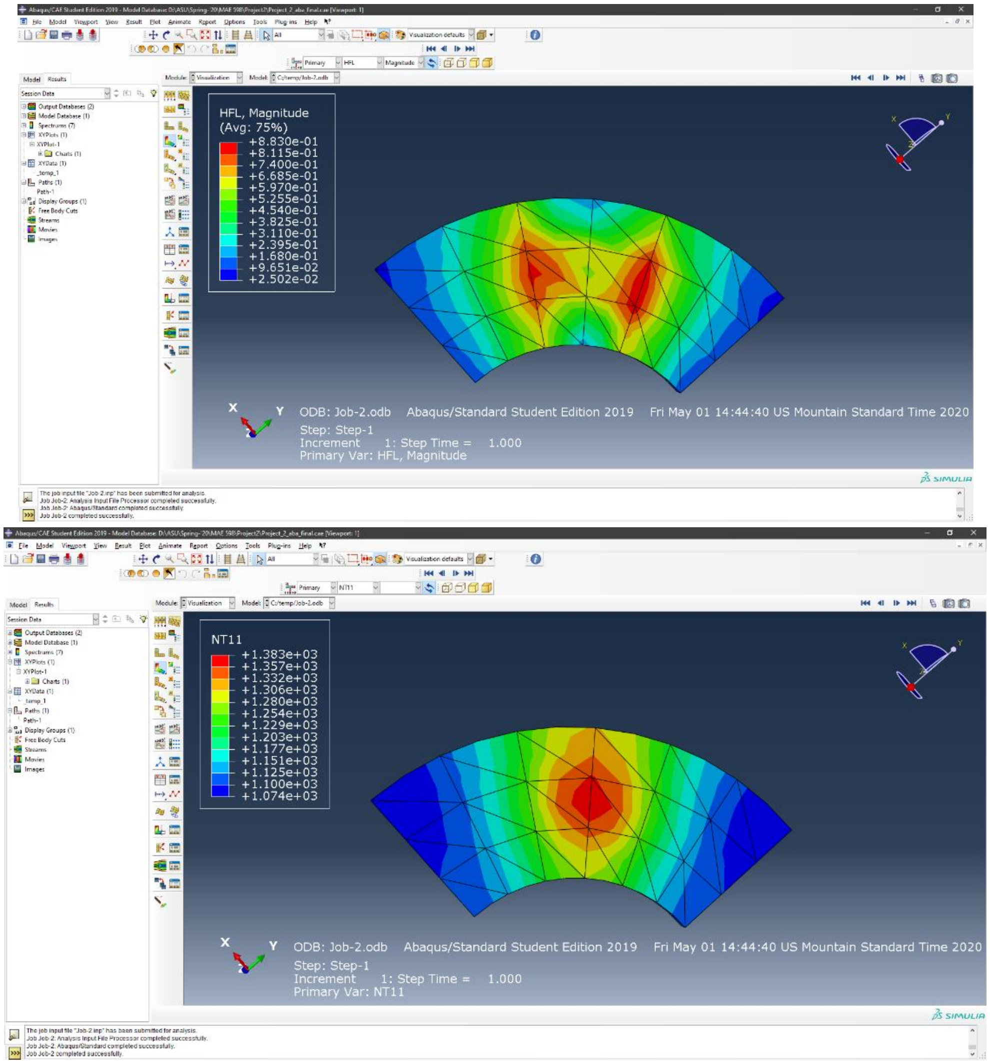


Figure 24: Heat Flux and Temperature Field Abaqus Solution T6 Element

	Element Size (h)	2 mm	Absolute Error (%)
Temperature	Abaqus	1383	0.68
	FEM	1392.544	
Heat Flux	Abaqus	0.8830	3.9
	FEM	0.849187	

Conclusion

The overall error observed between the FEM and Abaqus solution for temperature is below 1%, and for heat flux, it is below 10%. Based on the error, we can say the solution is consistent. However, the error with Flux seems to be significant, but in practicality, this much variation in calculation of heat flux is advisable. Thus, we can say even tough with node limitation, the solution of ABAQUS and FEM MATLAB function is consistent and acceptable.