MAE 404 Project 2

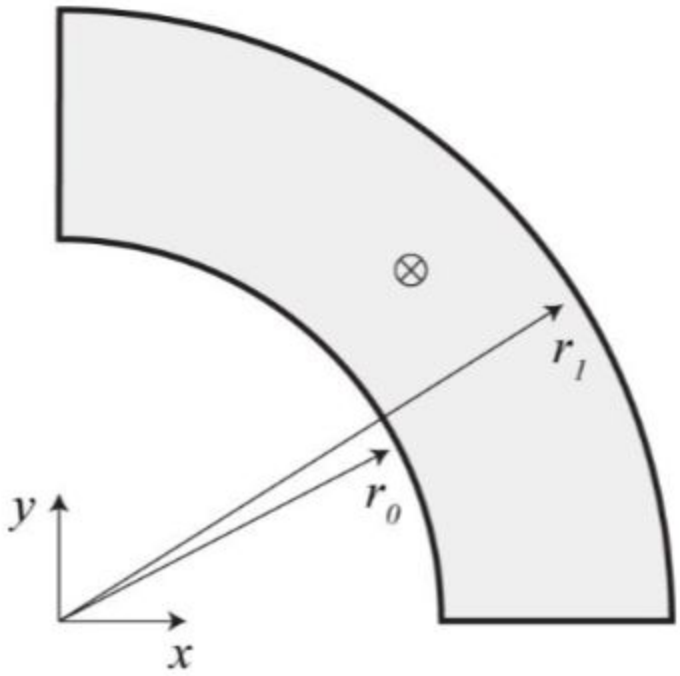
Alek Pensky
Michael Kolodziej

MAE 404/598

5/1/2020

Problem Description:

A plate with small thickness is exposed to a flame heat source, with a given heating distribution across its surface. The plate interacts with its surroundings, at 300 K, through mutual radiation heat transfer. Using finite elements, the team must determine the temperature field and heat flux field for the thin steel plate.



The inner radius r_0 =10mm, the outer radius r_1 =20mm, the thermal conductivity k = 17 W/(m*K), the thickness of the plate Δ_z =1mm, and the ambient temperature T_∞ =300 K.

<u>Linearizing the Governing Equation:</u>

To deal with the nonlinear nature of the problem, a linearized form of the strong form equation was created. The original equation is given below:

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

Using the below equation:

$$\sigma\left(T^4 - T_\infty^4\right) = h_{eff}\left(T - T_\infty\right), \text{ where } h_{eff} = \sigma\left(T^2 + T_\infty^2\right)\left(T + T_\infty\right)$$

The strong form was simplified to:

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

To determine the value of h_eff, the average temperature was calculated and then fed back into the defining equation to find an average h_eff to be used in later calculations:

$$\int s d\Omega = \int_0^{inf} exp(-r^2/9) * 2\pi r dr = A\sigma({T_{bar}}^4 - {T_0}^4)$$

$$9\pi = A\sigma({T_{bar}}^4 - {T_0}^4)$$

$$T_{bar} = 1207.29\ K$$

$$h_{eff} = \sigma({T_{bar}}^2 + {T_0}^2) * (T_{bar} + T_0)$$

$$h_{eff} = 132.259\ W/(m^2 K)$$

Calculating the effective heat transfer coefficient for radiative heat transfer linearizes the radiation term and consequently eliminates the need to solve the finite element problem iteratively.

After obtaining nodal flux magnitudes from all elements, least squares projection was used to consolidate the different heat flux values for the shared nodes of adjacent elements.

Weak Form Development and Discretization

The governing equation for determining the temperature of the plate is given as the following:

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

Where k is the thermal conductivity, Δ_z is the thickness of the plate, T is the temperature, and h_{eff} is the effective heat transfer coefficient.
When the system is successfully taken from its strong form to its weak form, the equation takes on this form:

$$\int w(\nabla k\Delta_z \nabla T + hT) d\Omega = \int ws d\Omega - \int whT_\infty d\Omega$$

The process for deriving this form can be found in the appendix.

The discrete equation for the system was calculated to be:

$$(K + H)d = (f + f_h)$$

The calculations made to reach this conclusion are given in the appendix.

Method of Manufactured Solutions

For the method of manufactured solutions, an arbitrary temperature field (generally, must be smooth and continuous) is chosen. This will yield a flux field and heat source field. The flux field is then used to determine the f vector of the solution. The boundary conditions are defined arbitrarily (i.e. gamma T and gamma q can be on any edge of the domain) based on the analytical fields developed. In this problem, gamma q, where flux is defined, occurs at r=r0. Gamma T, where temperature is defined, contains all other edges of the domain. Computing the "f" and "K" terms will depend on the imposed gamma q and gamma T, respectively. After developing a conductivity matrix, the temperature field is solved for and compared to the analytical temperature field. By changing the element size, the rate of convergence can be determined using L2 error norm metrics.

Triangular elements of linear and quadratic order are used to test convergence behaviour given the temperature field solution:

$$T = 1000sin(x+xy)sin(x+y)$$

This solution was chosen because it was sufficiently smooth and continuous, and its amplitude changes were large enough to not show misleading convergence results.

The heat source equation corresponding to the imposed temperature field was solved symbolically in MATLAB. It is given as:

$$\begin{aligned} s = {} & 34000sin(x+xy)sin(x+y) - 34000xcos(x+xy)cos(x+y) \\ & - 34000cos(x+xy)cos(x+y)(y+1) + 17000x^2sin(x+xy)sin(x+y) \\ & + 17000sin(x+xy)sin(x+y)(y+1)^2 \end{aligned}$$

The source term is the gradient of the flux term. The flux term is derived from Fourier's law of thermal conduction:

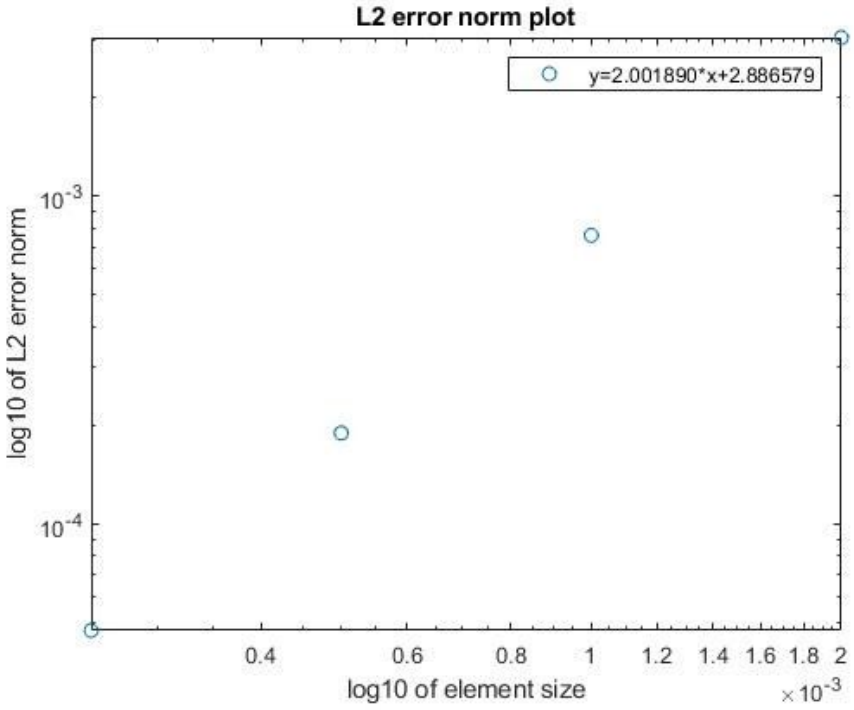
$$q = -k\Delta T$$

Temperature and flux boundary conditions were imposed on their respective boundaries. The first step was to find all of the nodes on different portions of the boundary. The inner radius of the body contains nodes within gamma q. These were found by searching for nodes that have a radial coordinate = inner radius = 10mm. The outer radius, part of gamma T, was found with the same approach but for a radius of 20mm. The other edges' nodes were found by calling out all nodes with an x OR y component equal to zero, which are also part of gamma T. Elements in gamma T will have their corresponding rows of the k matrix replaced with all zeros. Then, their temperature will be assigned by replacing their corresponding rows in the f vector with the temperature solutions at the nodes' respective locations. The f vector corresponds to the heat

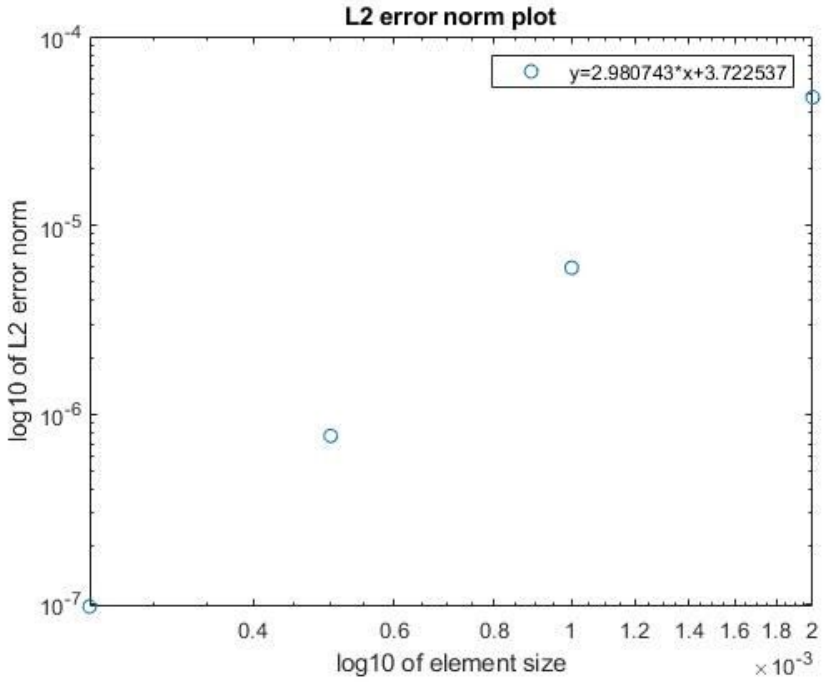
flux entering the domain. This is because in the weak form the gradient of flux over the boundary is equivalent to n*q_bar over the boundary gamma q. Thus, the f vector is integrated over the boundary gamma q for flux that is perpendicular to the direction of the elements. The conductivity matrix is constructed in the standard method. The temperature solution is solved by T=k\f.

Below are the L2 error norms plots for linear and quadratic elements for primary unknown.

Convergence For Linear Elements:



Convergence For Quadratic Elements:



The above results are expected for linear and quadratic elements. The log-log plot for linear elements should have a slope of 2 (2.001 actual) and the plot for quadratic elements should have a slope of 3 (2.98 actual). This makes sense because quadratic elements should show a faster rate of convergence.

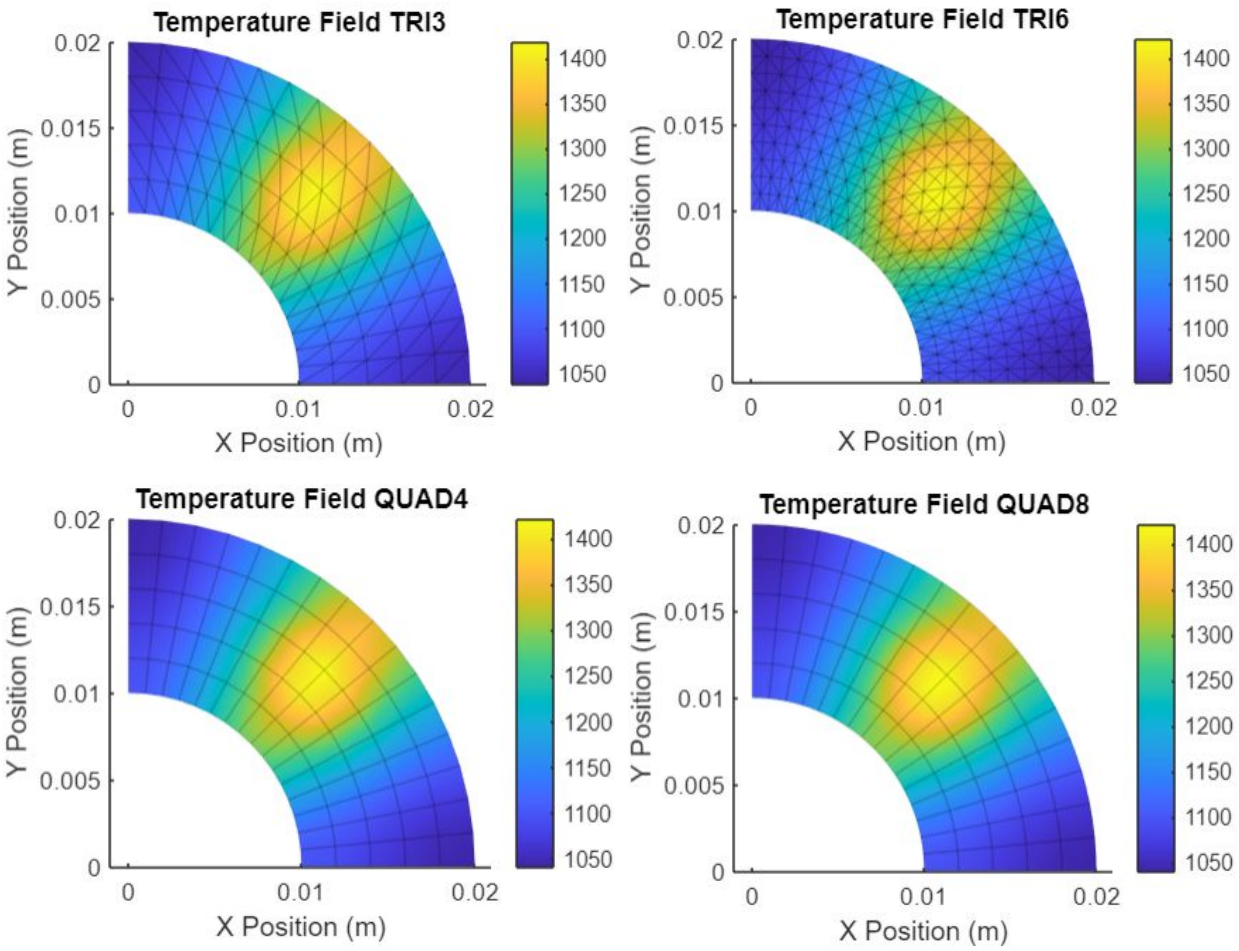
MATLAB Solution

The first step to finding a solution to the problem using MATLAB simulation is to create a baseline equation to solve for our unknown variable. After numerically integrating each term of the matrix equation given in the weak form discretization above, the primary unknown was solved for by the following operation:

$$T = (K + H) \backslash (f + f_h)$$

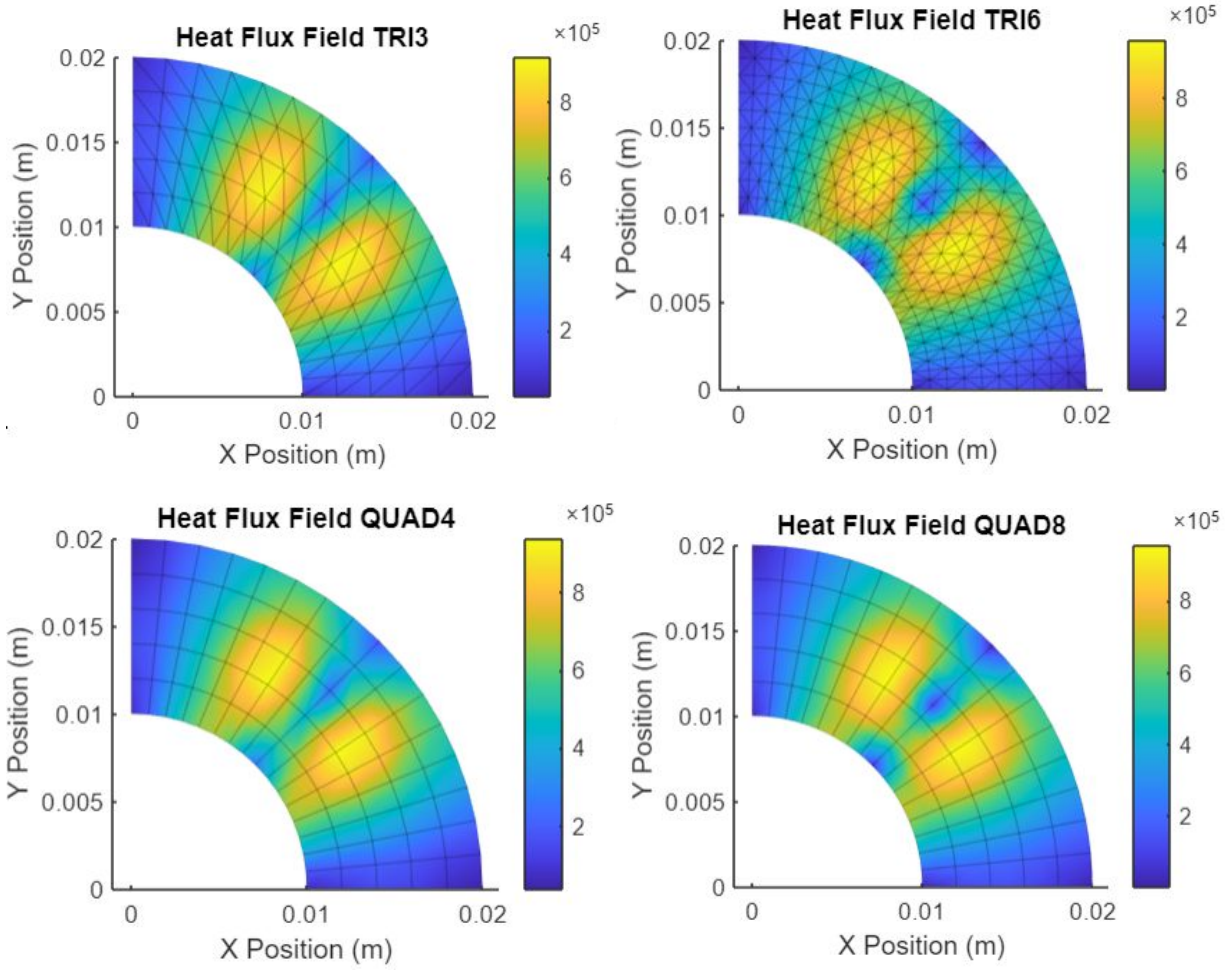
Each term in the above equation was solved for using different 2D element orders and types in order to create a visual representation of the temperature field and heat flux field in this plate. The code for the equations of each of these variables as well as the method of creating these graphs are given in the appendix.

Temperature Fields for Different Element Types



The temperature field graphs above show that the highest temperatures are at the center of the body and dissipate through the body. The accuracy of these temperature fields increase when going from linear to quadratic elements and when going from triangular to quadrilateral elements.

Heat Flux Field for Different Element Types



The heat flux field above shows heat fluxes are highest around the area of highest temperature change when compared to the temperature fields. The accuracy of these heat flux fields increase when going from linear to quadratic elements and when going from triangular to quadrilateral elements.

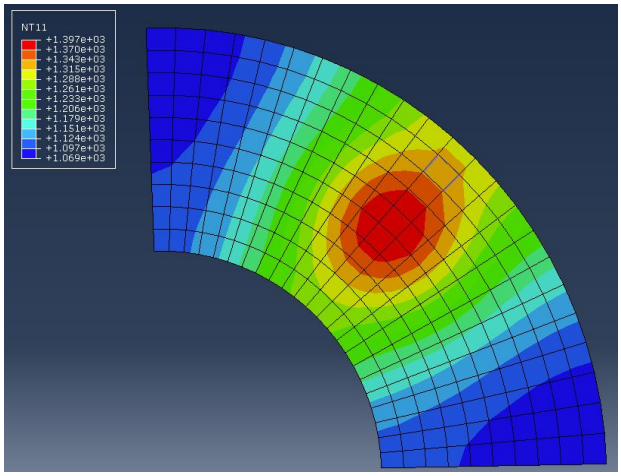
Demonstration of a Refined Mesh

To show how the element mesh for the body is sufficiently refined, the table below shows the change in maximum temperature and heat flux as the element size becomes smaller, as well as the percentage change between the current and previous result value. Linear triangular elements are used.

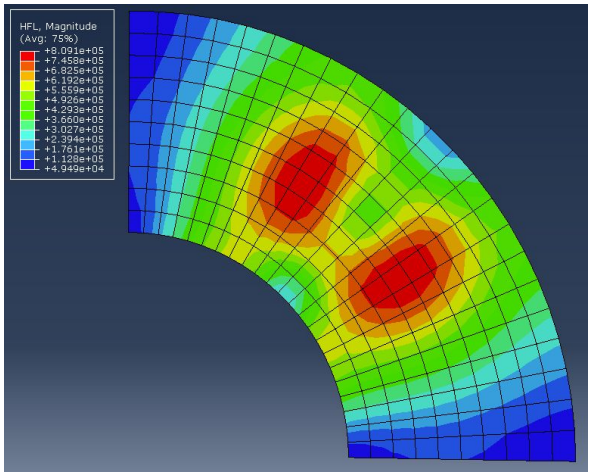
Element Size	Maximum Temperature (K)	Temperature Change	Maximum Internal Heat Flux (W/m^2)	Heat Flux Change
0.02	1531.6	------	7.0771e5	-------
0.01	1310.3	-14.45%	8.3080e5	17.39%
0.005	1402.1	7.01%	9.0189e5	8.56%
0.0025	1411.1	0.64%	8.9347e5	-0.93%

Abaqus Solution

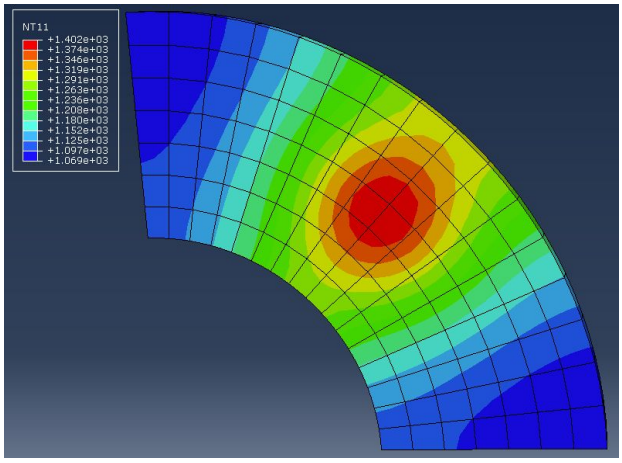
Linear Elements
Temperature Field (K)



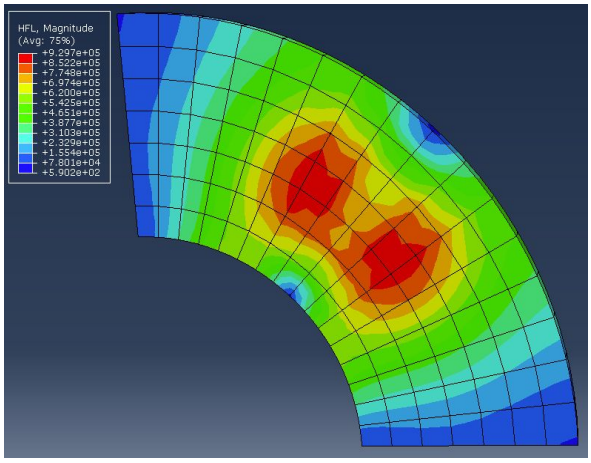
Linear Elements
Internal Flux Field (W/m^2)



Quadratic Elements
Temperature Field (K)



Quadratic Elements
Internal Flux Field (W/m^2)



Summary of Abaqus Results			
Linear elements		Quadratic elements	
Maximum Temperature (K)	Maximum Flux (W/m^2)	Maximum Temperature (K)	Maximum Flux (W/m^2)
1397	8.091e5	1402	9.297e5

The above plots are consistent with each other for temperature and flux fields. The quadratic elements have larger element sizes to accommodate the student software node limitations. Overall, the Abaqus solutions agree well with MATLAB solutions.

I think that the Abaqus solution is consistent with the MATLAB solution because temperature and flux fields are similar based on visual inspection. Additionally, the maximum temperature in the domain for abaqus is 1397 K for linear triangular elements, while for MATLAB the max temperature is 1411.1 K. These results agree very well with each other. Internal flux results show similar values for MATLAB and abaqus, which are 8.9347e5 W/m^2 and 8.091e5 W/m^2, respectively.

Appendix:

Work to turn the strong form equation to the weak form:

Strong Form: (Linearized)

$$K\Delta_z \nabla^2 T + S - h(T - T_\infty) = 0$$

Multiply by w, integrate over $\mathcal{N}$.
Integrate 1st term by parts by separating ∇.

$$\int \vec{\nabla}(wK\Delta_z \vec{\nabla} T)\, d\mathcal{N} - \int_{\mathcal{N}} \nabla w K \Delta_z \nabla T\, d\mathcal{N}$$

$$+ \int_{\mathcal{N}} ws\, d\mathcal{N} - \int wh(T - T_\infty)\, d\mathcal{N} = 0$$

Rearranging terms: $-K\vec{\nabla} T = \vec{q}$

$$-\int_{\mathcal{N}} \nabla w K \Delta_z \nabla T \cdot d\mathcal{N} = -\int \vec{\nabla} \cdot (w\Delta_z \vec{q})\, d\mathcal{N}$$

$$+ \int_{\mathcal{N}} ws\, d\mathcal{N} - \int wh(T - T_\infty)\, d\mathcal{N} = 0$$

Recognize that $\int \vec{\nabla} w \cdot \Delta_z \vec{q} = \underbrace{\int_{\Gamma_q} w\Delta_z \bar{q} \vec{n}\, d\Gamma}_{=0}$

Weak Form:

$$\int_{\mathcal{N}} w \nabla K \Delta_z \nabla T\, d\mathcal{N} = \int_{\mathcal{N}} ws\, d\mathcal{N} - \int_{\mathcal{N}} wh(T - T_\infty)\, d\mathcal{N}$$

$$\int_{\mathcal{N}} w(\nabla K \Delta_z \nabla T + hT)\, d\mathcal{N} = \int_{\mathcal{N}} ws\, d\mathcal{N} + \int_{\mathcal{N}} whT_\infty\, d\mathcal{N}$$

Rearranging Terms

$$\int_{\mathcal{N}} w(\nabla K \Delta_z \nabla + h) T\, d\mathcal{N} = \int_{\mathcal{N}} ws\, d\mathcal{N} + \int_{\mathcal{N}} whT_\infty\, d\mathcal{N}$$

Work to create the discrete (matrix) equation:

$$w = \underset{\sim}{w}^T \underset{\sim}{N}^T$$

$$T = \underset{\sim}{N}\underset{\sim}{d}$$

$$\underset{\sim}{H}^e = \int_{ve} (N^e)^T h_{eff} (N^e) \, dv$$

$$\underset{\sim}{H} = \sum_e (L^e)^T H \, L^e$$

$$\underset{\sim}{K}^e = \int_{ve} (B^e)^T k \Delta_z (B^e) \, dv$$

$$\underset{\sim}{K} = \sum_e (L^e)^T \underset{\sim}{K}^e L^e$$

$$\underset{\sim}{f}^e = \int_{ve} (N^e)^T s \, dv$$

$$\underset{\sim}{f} = \sum_e (L^e)^T \underset{\sim}{f}^e$$

$$\underset{\sim}{fh}^e = \int_{ve} (N^e)^T h T_\infty \, dv$$

$$\underset{\sim}{fh} = \sum (L^e)^T f_h^e$$

$$(H+K)T = f_s + f_h$$

MATLAB code:

```
clc

r0=.01;
r1=.02;
R = .003;
T_inf = 300;
heff=132.259;
delta_z=.001;
k_cond = 17;
d_max=[];
magn_flux_max=[];
factor_q=[];
factor_t=[];
change_q=[];
change_t=[];
counter=1;
mesh_initial=0.02;
% 0.02 is good!
for h=[mesh_initial mesh_initial/2 mesh_initial/4 mesh_initial/8]

clearvars -
except r0 r1 R T_inf heff delta_z k_cond h magn_flux_max d_max factor change_t cha

    etype = 't3';
    mesh=make_project2_mesh(h,r0,r1,etype);
    %assume 'etype' is input variable for element type
    %'t3' is TRI3, 't6' is TRI6, 'q4' is QUAD4, and 'q8' is QUAD8
    %assume q is quadrature

    if strcmp(etype,'t3')
        shape = @shapet3;
        qpts = [1/6  2/3 1/6; 1/6 1/6  2/3 ; 1/6 1/6 1/6 ];

    elseif strcmp(etype,'t6')
        shape = @shapet6;
        qpts = [0.1012865073 0.7974269853 0.1012865073 0.4701420641
 0.4701420641 0.0597158717 0.3333333333;
                0.1012865073 0.1012865073 0.7974269853 0.0597158717
 0.4701420641 0.4701420641 0.3333333333;
                0.0629695903 0.0629695903 0.0629695903 0.0661970764
 0.0661970764 0.0661970764 0.1125];

    elseif strcmp(etype,'q4')
        shape = @shapeq4;
        qpts = qpt_gen_quad(4);
    else
        shape = @shapeq8;
        qpts = qpt_gen_quad(8);
    end


k=zeros(length(mesh.x));
```

```
% compute conductivity matrix
for c=mesh.conn
    xe=mesh.x(:,c); %x and y of the 8 coords in each element
    ke=zeros(length(c));
    for q=qpts
        [N, dNdp] = shape(q);
        J = xe*dNdp; %(2x8)(8x2)
        B=dNdp/J;
    ke = ke + B*k_cond*delta_z*B'*q(end)*det(J); %how to find D?
    end
    k(c,c)=k(c,c) + ke;
end

%integrate heat source (flame)
f=zeros(length(mesh.x),1);
%iterate thru all qpts
%iso parametric approach

for c=mesh.conn % 3 by num elements
        xe=mesh.x(:,c); %2 by 3
        fe=zeros(length(c),1);
       for q = qpts
            [N,dNdp]=shape(q);
            x=xe*N; % 2x1 centroid of the element (isoparametric
 mapping)
            J=xe*dNdp; % 2x2
            fe = fe + N*heatsource(x)*det(J)*q(end);
       end
    f(c)=f(c)+fe;
end

%H integral
H=zeros(length(mesh.x));
for c=mesh.conn
     xe=mesh.x(:,c); %x and y coords in each element
     he=zeros(length(c));
    for q=qpts
         [N, dNdp] = shape(q);
         J = xe*dNdp;
     he = he + N*heff*N'*q(end)*det(J);
     end
%     Scattering into h matrix
     H(c,c)=H(c,c) + he;
end

fh=zeros(length(mesh.x),1);

for c=mesh.conn
     xe=mesh.x(:,c); %x and y coords in each element
     fhe=zeros(length(c),1);
    for q=qpts
         [N, dNdp] = shape(q);
         J = xe*dNdp;
         fhe = fhe+ N*heff*T_inf*q(end)*det(J);
```

```
     end
    %Scattering into h matrix
     fh(c)=fh(c)+ fhe;
end

d=(k+H)\(f+fh);

d_max(end+1)=max(d);


%  if strcmp(etype,'t3') || strcmp(etype,'q4')
% % linear only
% figure()
% p.faces = mesh.conn';
% p.vertices = mesh.x';
% p.facecolor = 'interp';
% p.facevertexcdata = d;
% p.edgealpha = 0.1;
% patch(p);
% colorbar
% axis equal
% title('Temperature Field QUAD4')
% xlabel('X Position (m)')
% ylabel('Y Position (m)')
%  else
%
% figure()
% p.faces = mesh.pconn';
% p.vertices = mesh.x';
% p.facecolor = 'interp';
% p.facevertexcdata = d;
% p.edgealpha = 0.1;
% patch(p);
% colorbar
% axis equal
% title('Temperature Field QUAD8')
% xlabel('X Position (m)')
% ylabel('Y Position (m)')
%  end

A=spalloc(length(mesh.x),length(mesh.x),9*length(mesh.x));
y=zeros(length(mesh.x),2);
for c=mesh.conn
    xe=mesh.x(:,c);
    de=d(c)';
    Ae=zeros(length(c));
    for q=qpts
        [N,dNdp]=shape(q);
        J=xe*dNdp;
        dNdx=dNdp/J;
        q_flux=-k_cond*de*dNdx;
        Ae=Ae + N*N'*det(J)*q(end);
        y(c,:)=y(c,:) + N*q_flux*det(J)*q(end);
    end
```

```
    A(c,c)=A(c,c)+Ae;
end
flux=A\y;

magn_flux=zeros(length(mesh.x),1);



for count=1:length(mesh.x)

    magn_flux(count)=norm(flux(count,:));

end

magn_flux_max(end+1)=max(magn_flux)

if counter>1
    change_q(end+1)=[(magn_flux_max(end)-magn_flux_max(end-1))/
magn_flux_max(end-1)];
    change_t(end+1)=[(d_max(end)-d_max(end-1))/d_max(end-1)];
%     factor_q=[];
%     factor_t=[];

end

%  if strcmp(etype,'t3') || strcmp(etype,'q4')
% %linear only
% figure()
% p.faces = mesh.conn';
% p.vertices = mesh.x';
% p.facecolor = 'interp';
% p.facevertexcdata = magn_flux;
% p.edgealpha = 0.1;
% patch(p);
% colorbar
% axis equal
% title('Heat Flux Field QUAD4')
% xlabel('X Position (m)')
% ylabel('Y Position (m)')
%
%  else
%
% % quadratic only
% figure()
% p.faces = mesh.pconn';
% p.vertices = mesh.x';
% p.facecolor = 'interp';
% p.facevertexcdata = magn_flux;
% p.edgealpha = 0.1;
% patch(p);
% colorbar
% axis equal
% title('Heat Flux Field QUAD8')
% xlabel('X Position (m)')
```

```
% ylabel('Y Position (m)')
%  end

 counter=counter+1;
 change_q
 change_t
end

    function[N,dNdp] = shapet3(q)
        N=[q(1);q(2);1-q(1)-q(2)];
        dNdp=[1 0; 0 1; -1 -1];
    end

    function[N,dNdp]=shapet6(q)
    a=q(1);
    b=q(2);

    N=[a*(2*a-1); b*(2*b-1); (1-a-b)*(2*(1-a-b)-1); 4*a*b; 4*b*(1-a-
b); 4*a*(1-a-b)];


    dNdp=[-1+4*a 0;...
        0 -1+4*b;...
        1-4*(1-a-b) 1-4*(1-a-b);...
        4*b 4*a;...
        -4*b 4*(1-a-b)-4*b;...
        -4*a+4*(1-a-b) -4*a];

    end

    function[N,dNdp] = shapeq4(q)
        N = [0.25*(1-q(1))*(1-q(2)); 0.25*(1+q(1))*(1-q(2));
 0.25*(1+q(1))*(1+q(2)); 0.25*(1-q(1))*(1+q(2))];

        dNdp = [0.25*(-1+q(2)) 0.25*(1-q(2)) 0.25*(1+q(2)) 0.25*(-1-
q(2));
                0.25*(-1+q(1)) 0.25*(-1-q(1)) 0.25*(1+q(1)) 0.25*(1-
q(1))];
            dNdp=dNdp';
    end

function [N, dNdp] = shapeq8(p)
    N = 0.25*[-(1-p(1))*(1-p(2))*(1+p(1)+p(2));
              -(1+p(1))*(1-p(2))*(1-p(1)+p(2));
              -(1+p(1))*(1+p(2))*(1-p(1)-p(2));
              -(1-p(1))*(1+p(2))*(1+p(1)-p(2));
              2*(1-p(1)^2)*(1-p(2));
              2*(1+p(1))  *(1-p(2)^2);
              2*(1-p(1)^2)*(1+p(2));
              2*(1-p(1))  *(1-p(2)^2)];
    dNdp = [-0.25*(p(2)-1)*(p(2)+2*p(1)),
 -0.25*(p(1)-1)*(2*p(2)+p(1));
            0.25*(p(2)-1)*(p(2)-2*p(1)),   0.25*(p(1)+1)*(2*p(2)-
p(1));
```

```
            0.25*(p(2)+1)*(p(2)+2*p(1)),
 0.25*(p(1)+1)*(2*p(2)+p(1));
           -0.25*(p(2)+1)*(p(2)-2*p(1)),  -0.25*(p(1)-1)*(2*p(2)-
p(1));
            p(1)*(p(2)-1),                 0.5*(p(1)^2-1);
            0.5*(1-p(2)^2),               -p(2)*(p(1)+1);
           -p(1)*(p(2)+1),                 0.5*(1-p(1)^2);
            0.5*(p(2)^2-1),                p(2)*(p(1)-1)];
end

function [s] = heatsource(x_s)
r0=.01;
r1=.02;
R = .003;

        x=x_s(1);
        y=x_s(2);

        xc=.5*(r1+r0)*cos(pi/4);
        yc=.5*(r1+r0)*sin(pi/4);

        s=exp(-((x-xc)^2+(y-yc)^2)/R^2)*1e6;

end

function [qpts] = qpt_gen_quad(n)
% Construct the quadrature table, qpts here.

one_D_quad = quadrature(n);

qpts = zeros(3,n^2);
col=1;
for qpt1 = one_D_quad %2x1 of qpt and weight

    for qpt2 = one_D_quad %2x1 of qpt and weight
        qpts(1,col)=qpt1(1); %kesi
        qpts(2,col)=qpt2(1);% zeta
        qpts(3,col)=qpt1(2)*qpt2(2);

        col=col+1;
    end
end
end
```