ARIZONA STATE UNIVERSITY

MAE 598: FINITE ELEMENTS IN ENGINEERING

PROJECT 2 – FINAL REPORT

Date – 04/24/2019

Submitted by:

Prof. Jay Oswald

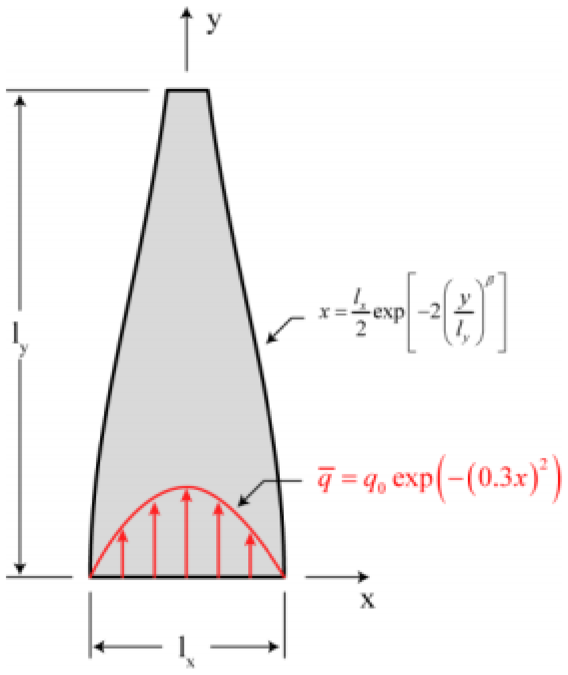
Abhishek Joshi
(1215199687)

Varun Agrawal
(1215318065)

CONTENT

1. Problem Statement

Shown is the basic design of a cooling fin with 2 design parameters l_y (height of the fin) and beta (exponential parameter which controls the curve of the fin). The boundary conditions for the fin are convection on the top, left and right edge, and applied heat flux on bottom edge. The heat flux depends on the distance in x-direction as per the equation shown.
The main objective of the fin was to keep the temperature rise to maximum of 100^{O}C provided the ambient surrounding temperature is 0^{O}C.


y
l_y
$x = \frac{l_x}{2}\exp\left[-2\left(\frac{y}{l_y}\right)^{\beta}\right]$
$\overline{q} = q_0 \exp\left(-(0.3x)^2\right)$
x
l_x

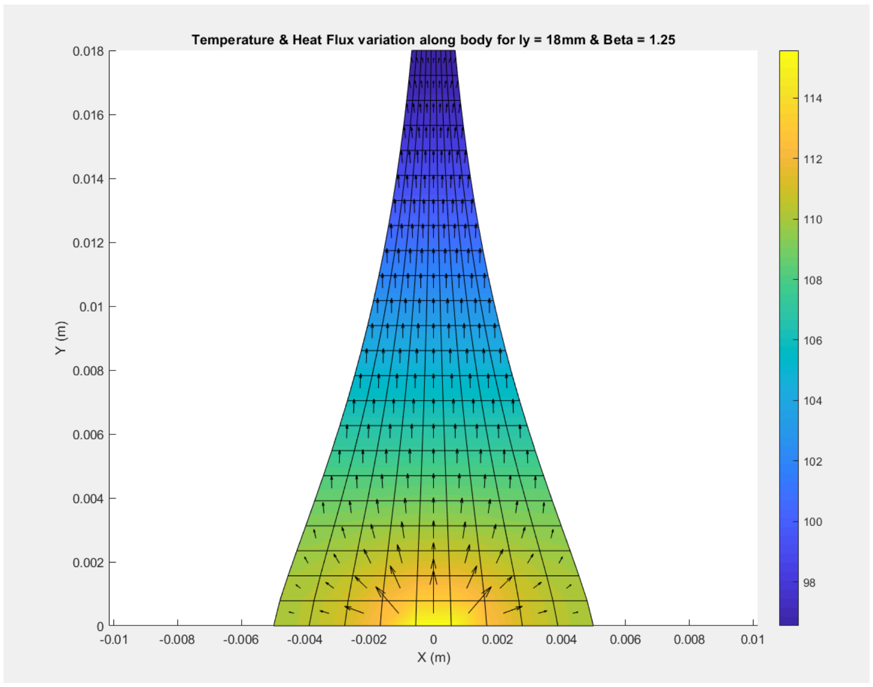
The following steps were followed to solve the problem:

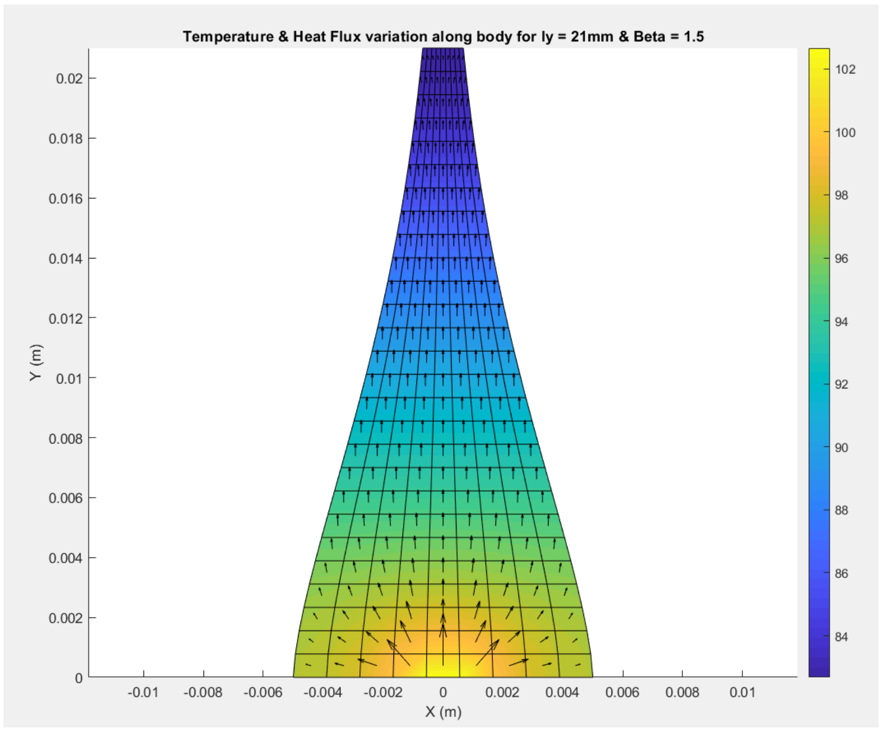
- Generate MATLAB code using FEM techniques to calculate temperature and heat flux distribution.
- Understand convergent test by generating manufactured solution.
- Mesh refinement using Richardson extrapolation.
- Solve the fin problem in commercial software. We chose to solve it in ANSYS Workbench 19.2.
- Compare the solution from MATLAB to ANSYS, and study convergence.
- Determine ly and beta to keep temperature difference below 100^{O}C and minimum area.

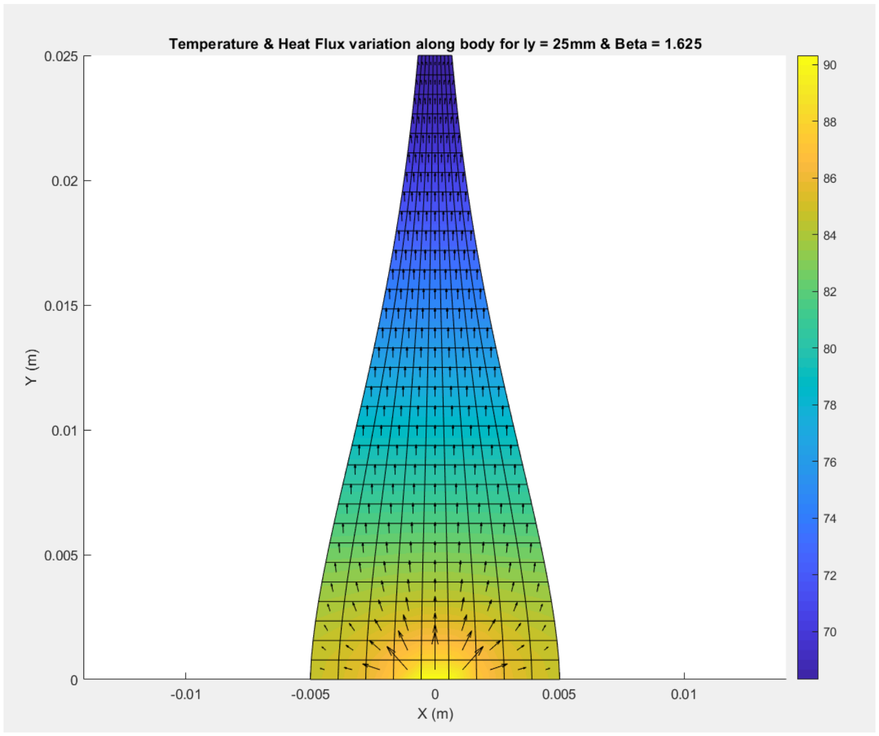
2. Representative Solution

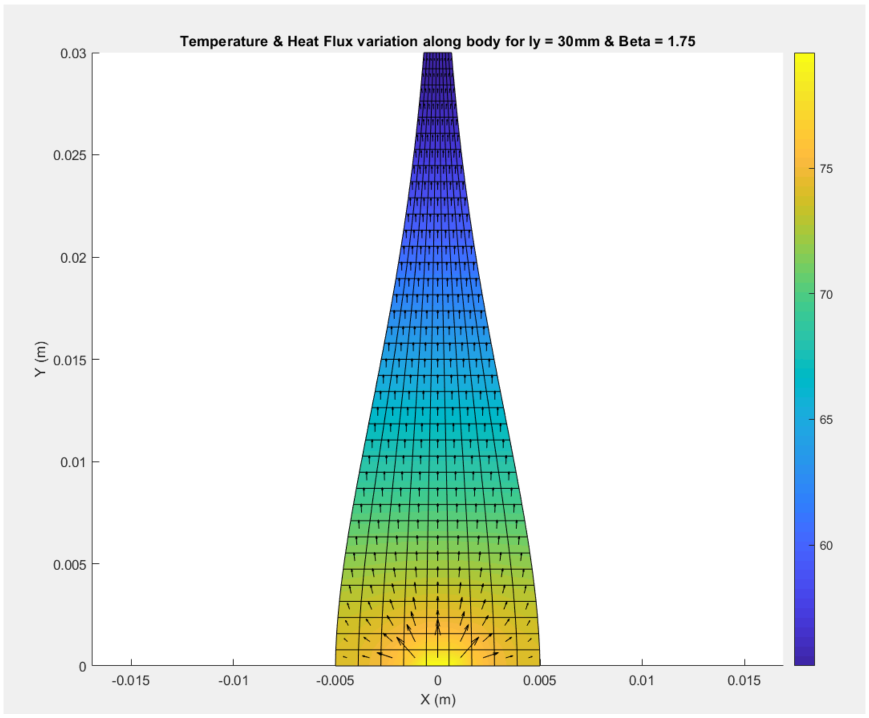
MATLAB code – Appendix 2.1 to plot the temperature and heat flux fields.

QUAD4 element type (element size = 0.0008m)

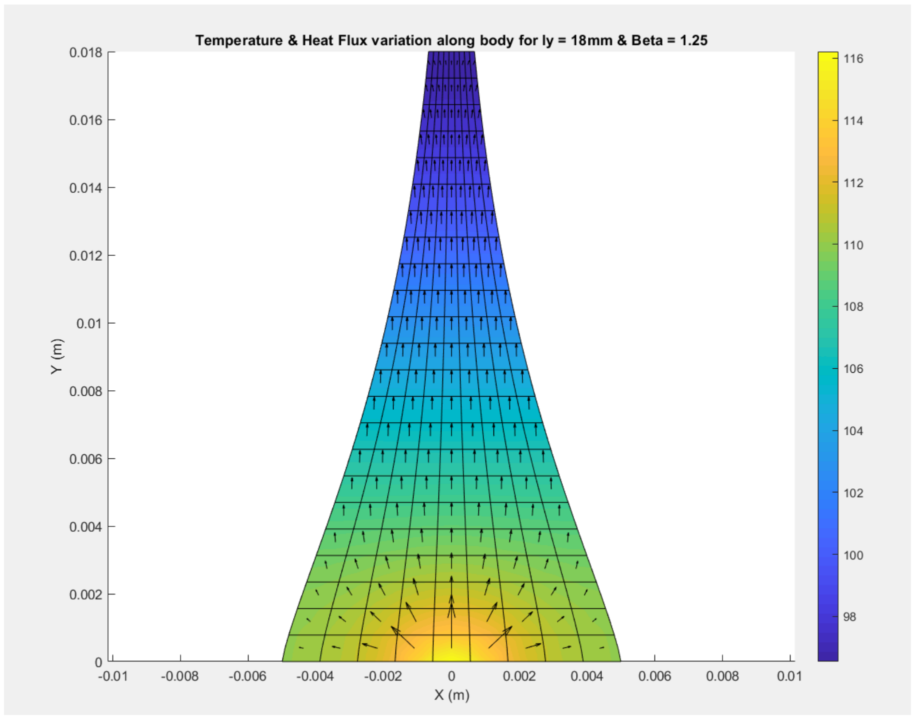


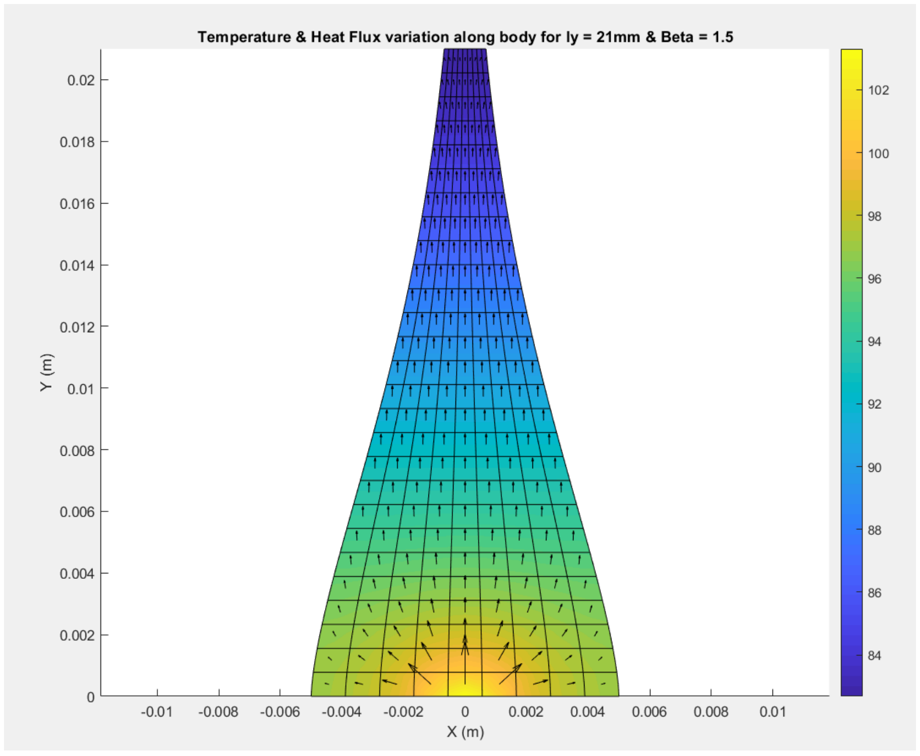


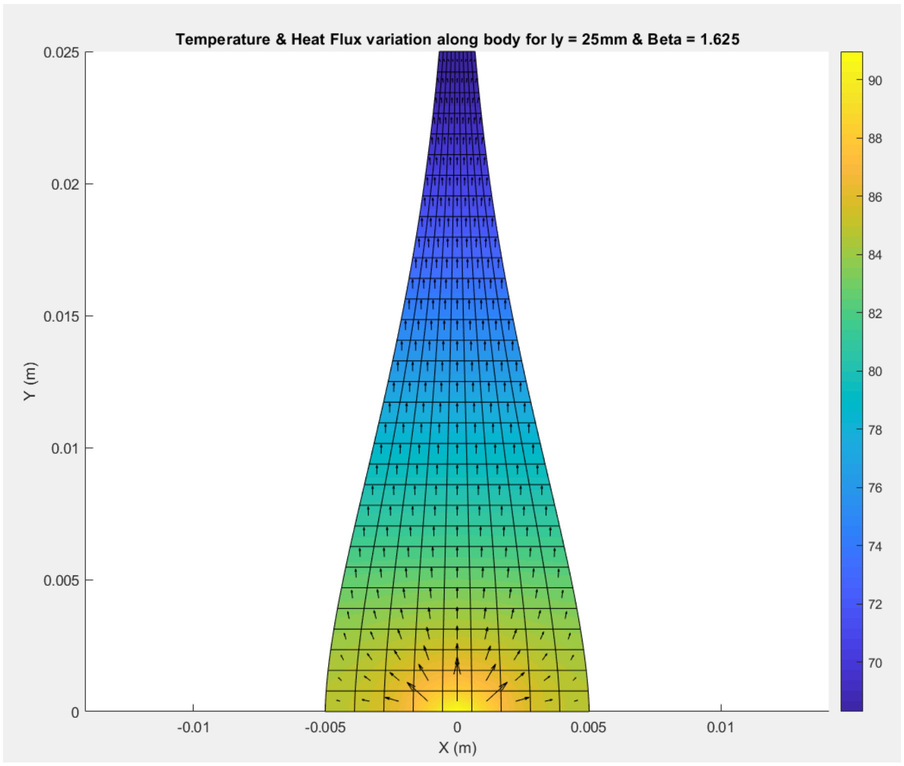

Temperature & Heat Flux variation along body for ly = 25mm & Beta = 1.625
Y (m)
X (m)
0.025
0.02
0.015
0.01
0.005
0
-0.01
-0.005
0
0.005
0.01
90
88
86
84
82
80
78
76
74
72
70

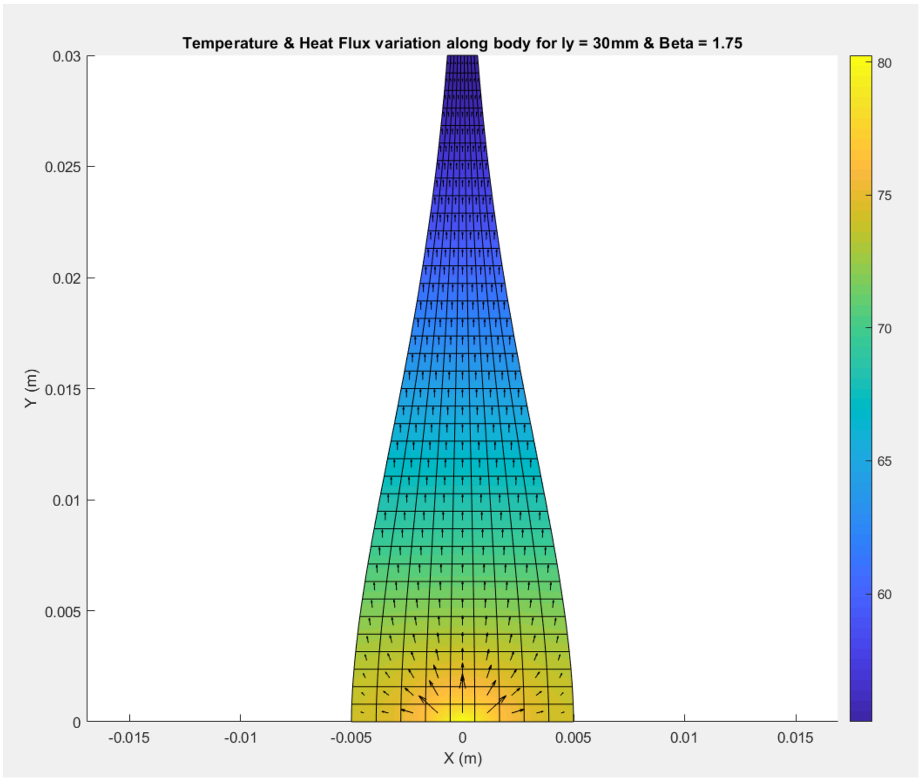

Temperature & Heat Flux variation along body for ly = 30mm & Beta = 1.75
Y (m)
X (m)
0.03
0.025
0.02
0.015
0.01
0.005
0
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
75
70
65
60

QUAD8 element type (element size = 0.0008m)






Temperature & Heat Flux variation along body for ly = 25mm & Beta = 1.625
Y (m)
X (m)
0.025
0.02
0.015
0.01
0.005
0
-0.01
-0.005
0
0.005
0.01
90
88
86
84
82
80
78
76
74
72
70


Temperature & Heat Flux variation along body for ly = 30mm & Beta = 1.75
Y (m)
X (m)
0.03
0.025
0.02
0.015
0.01
0.005
0
-0.015
-0.01
-0.005
0
0.005
0.01
0.015
80
75
70
65
60

3. Convergence Test

From the reference of class notes, considering the actual function of temperature in 2D as

$$T = \sqrt{x^2 + y^2}$$

Using MATLAB, the flux came out as

$$q = -\frac{D*x}{\sqrt{x^2+y^2}} \text{ and } -\frac{D*y}{\sqrt{x^2+y^2}}$$

where D is the thermal conductivity.

Source term is

$$S = -\frac{D}{\sqrt{x^2+y^2}}$$

As, we are not supposed to use convection boundary condition directly in the manufactured solution, following boundary conditions were used –

a. Temperature function, T was given on left and right edge.
b. Heat flux on bottom edge
$$q = -\frac{D*y}{\sqrt{x^2+y^2}}$$
c. Heat flux on top edge
$$q = -\frac{D*x}{\sqrt{x^2+y^2}}$$

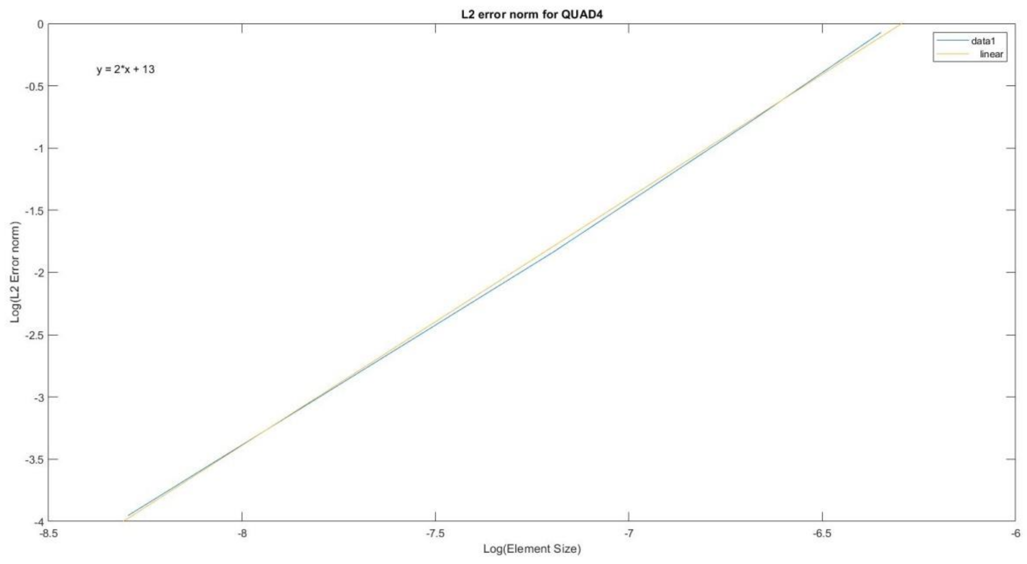
MATLAB code to calculate L2 error norm is attached in Appendix 2.2.

To find the order of convergence, log of error norm v/s log of element size was plotted.

For **ly = 25 mm** and **beta = 1.5**, using **QUAD4** element and the grid spacing range as 0.00025:0.0005:0.002 (in meters),

Order of solution is 2.035378e+00.

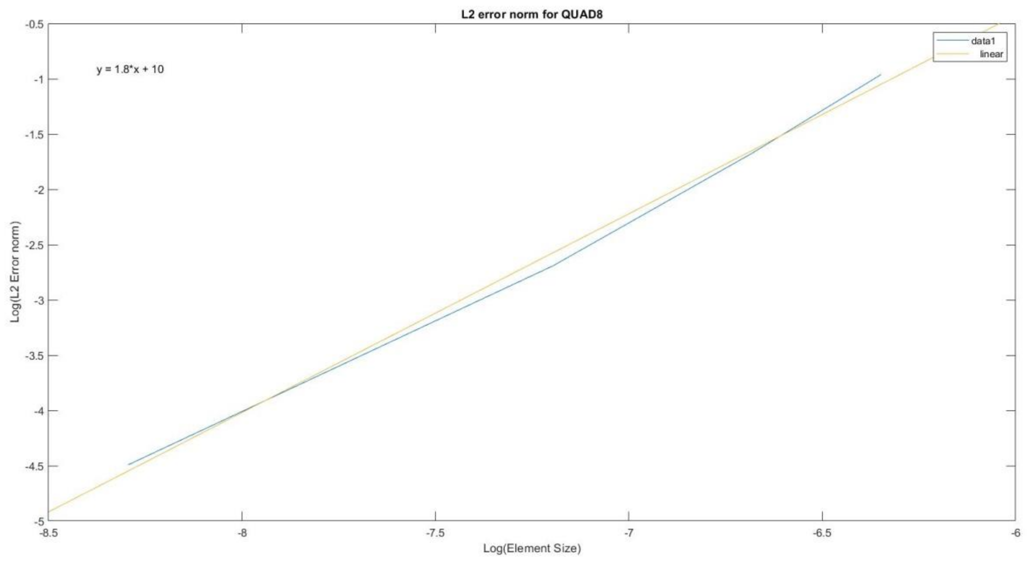
QUAD4 element is only linearly complete and the expected order of convergence is 2. L2 error norm order from MATLAB is approximately equal to 2.



For **ly = 25 mm** and **beta = 1.5**, using **QUAD8** element and the grid spacing range as 0.00025:0.0005:0.002 (in meters),

Order of solution is 1.917202e+00.

QUAD8 element is also linearly complete and the expected order of convergence is 2. L2 error norm order from MATLAB is approximately equal to 2.



4. Mesh Refinement Test

From the reference of –
https://www.grc.nasa.gov/www/wind/valid/tutorial/spatconv.html
https://en.wikipedia.org/wiki/Richardson_extrapolation

Richardson extrapolation is a method for obtaining a higher-order estimate of the continuum value (value at zero grid spacing) from a series of lower-order discrete values.

Three different meshes (f_1, f_2, and f_3) were taken in 'h' that are distinguished by 'r' (grid refinement ratio) and performed three analyses. The minimum value of r should be 1.1 to allow the discretization error to be differentiated from other error sources. We took the value of r as 2.

The order of grid convergence 'p' can be obtained from three solutions using

$$p = \ln\left(\frac{f_3 - f_2}{f_2 - f_1}\right) / \ln(r)$$

To get the exact solution, the Richardson extrapolation can be generalized for a **p-th** order methods and **r**-value of grid ratio (which does not have to be an integer) as:

$$f_{h=0} \cong f_1 + \frac{f_1 - f_2}{r^p - 1}$$

Now, calculate the Grid Convergence Index (GCI).

The **GCI** on the fine grid is defined as:

$$\mathrm{GCI}_{fine} = \frac{F_s\,|\varepsilon|}{(r^p - 1)}$$

where F_s is factor of safety (=1.25 for comparison over three or more grids).

$GCI_{12} = F_s*abs((f_1-f_2)/f_1)/(2^p - 1)$
$GCI_{23} = F_s*abs((f_2-f_3)/f_2)/(2^p - 1)$

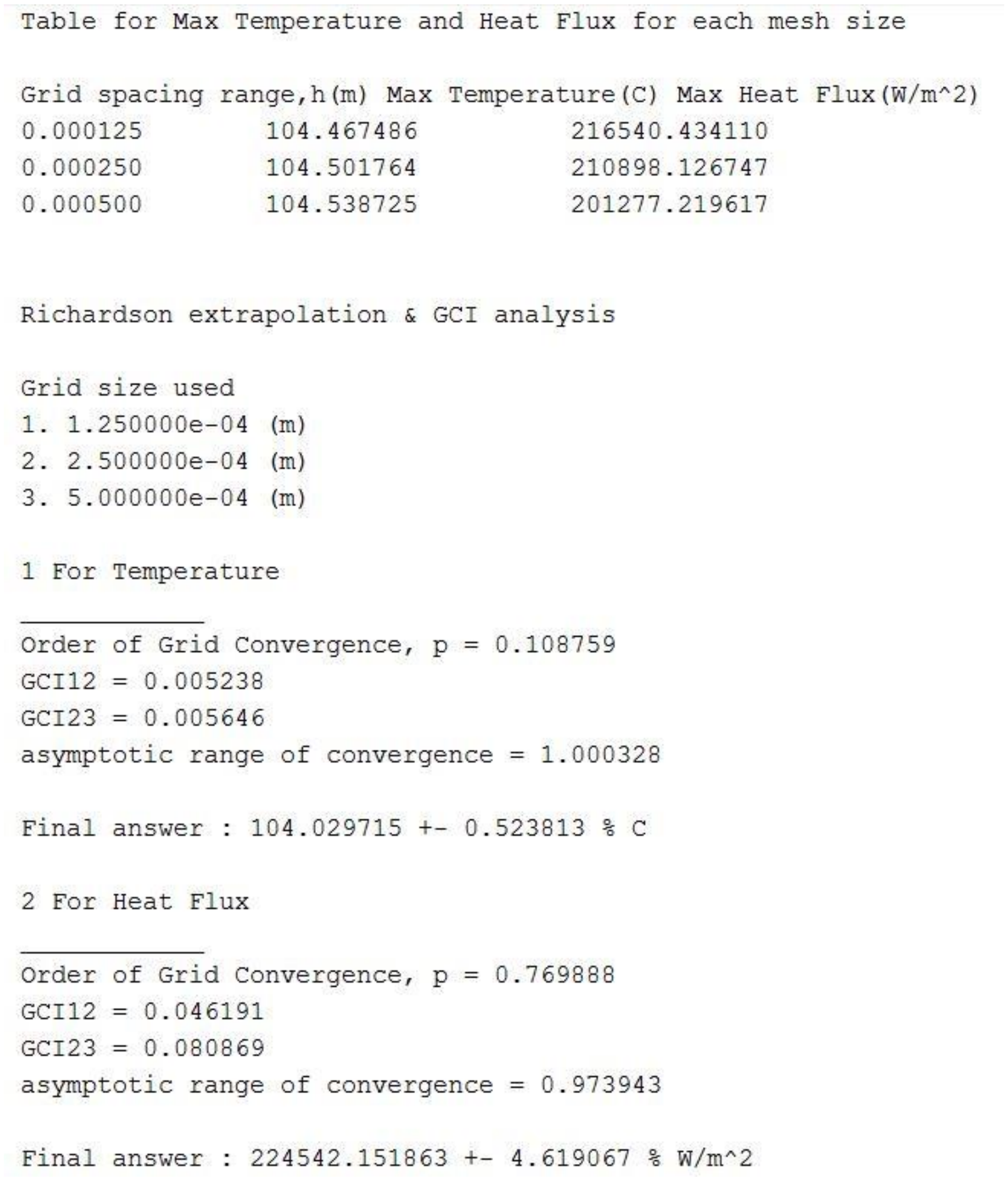
Check the asymptotic range of convergence for the computed solution using

$$\mathbf{GCI_{23} = r^p\, GCI_{12}}$$

Finally, the exact solution is given by $f_{h=0}$ with an accuracy of GCI_{12} percent.

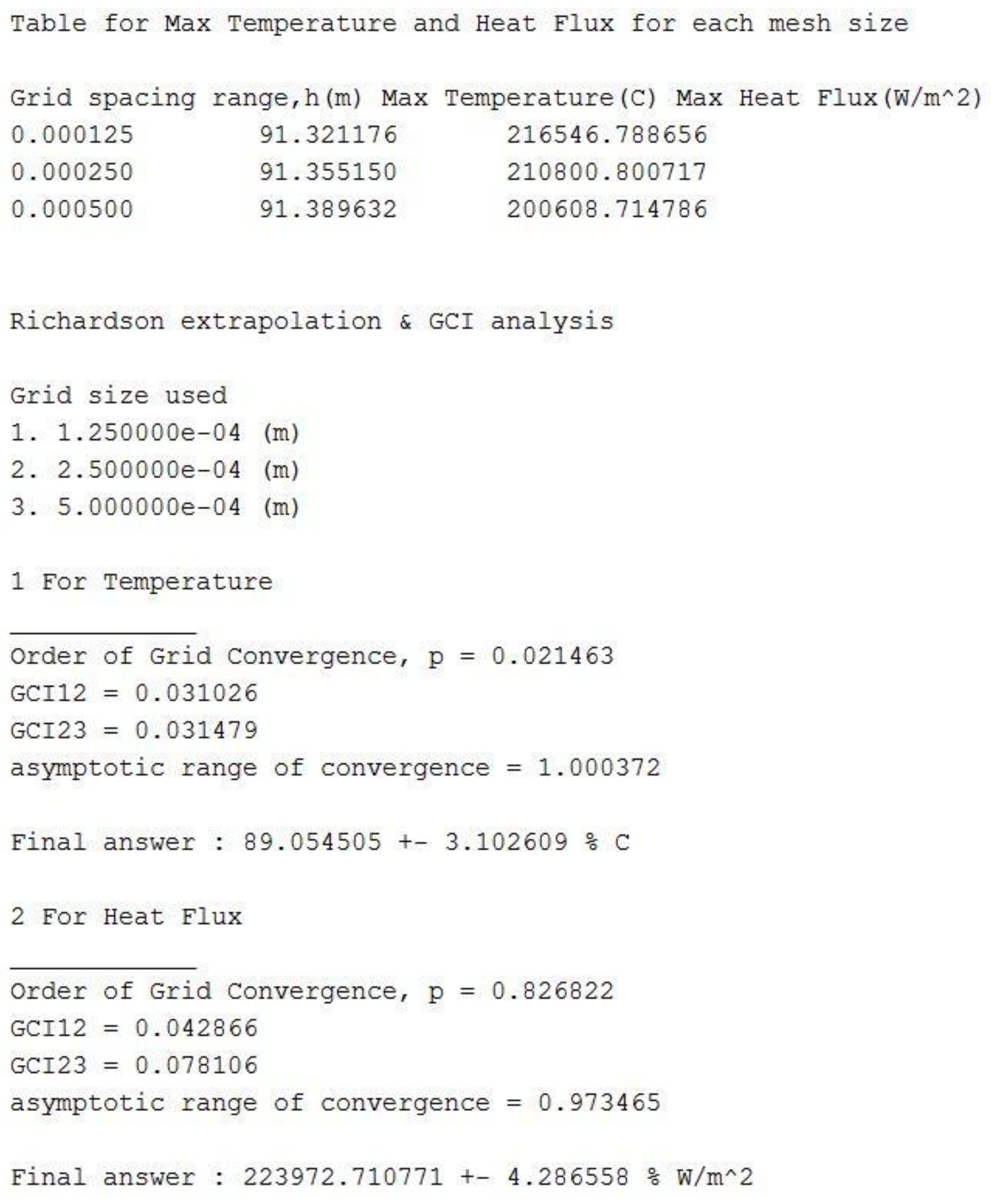
MATLAB code – Appendix 2.3
Output of mesh refinement test for three cases is shown in next pages.

For ly = 20.625 mm and beta = 1.625



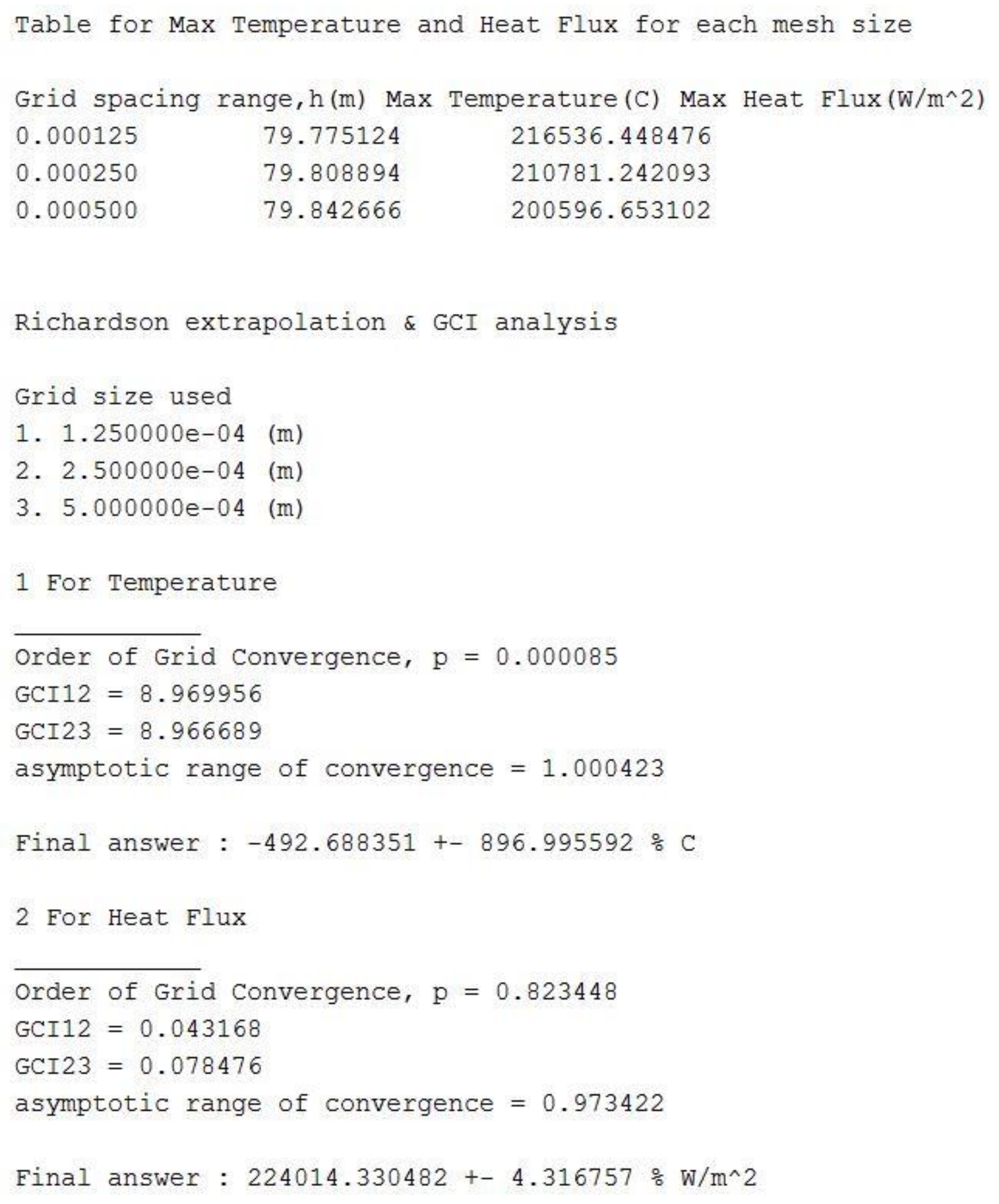
```
Table for Max Temperature and Heat Flux for each mesh size

Grid spacing range,h(m) Max Temperature(C) Max Heat Flux(W/m^2)
0.000125        104.467486          216540.434110
0.000250        104.501764          210898.126747
0.000500        104.538725          201277.219617


Richardson extrapolation & GCI analysis

Grid size used
1. 1.250000e-04 (m)
2. 2.500000e-04 (m)
3. 5.000000e-04 (m)

1 For Temperature
____________
Order of Grid Convergence, p = 0.108759
GCI12 = 0.005238
GCI23 = 0.005646
asymptotic range of convergence = 1.000328

Final answer : 104.029715 +- 0.523813 % C

2 For Heat Flux
____________
Order of Grid Convergence, p = 0.769888
GCI12 = 0.046191
GCI23 = 0.080869
asymptotic range of convergence = 0.973943

Final answer : 224542.151863 +- 4.619067 % W/m^2
```

For ly = 25 mm and beta = 1.5



```
Table for Max Temperature and Heat Flux for each mesh size

Grid spacing range,h(m) Max Temperature(C) Max Heat Flux(W/m^2)
0.000125        91.321176       216546.788656
0.000250        91.355150       210800.800717
0.000500        91.389632       200608.714786


Richardson extrapolation & GCI analysis

Grid size used
1. 1.250000e-04 (m)
2. 2.500000e-04 (m)
3. 5.000000e-04 (m)

1 For Temperature
____________
Order of Grid Convergence, p = 0.021463
GCI12 = 0.031026
GCI23 = 0.031479
asymptotic range of convergence = 1.000372

Final answer : 89.054505 +- 3.102609 % C

2 For Heat Flux
____________
Order of Grid Convergence, p = 0.826822
GCI12 = 0.042866
GCI23 = 0.078106
asymptotic range of convergence = 0.973465

Final answer : 223972.710771 +- 4.286558 % W/m^2
```

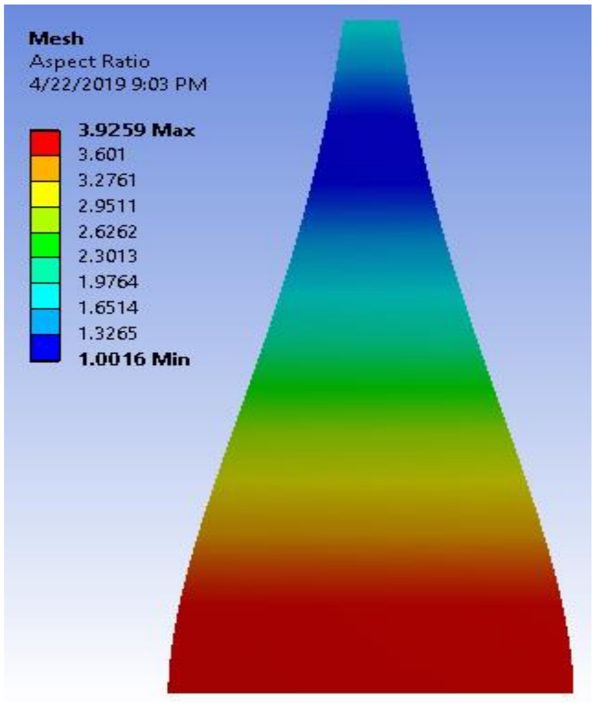
For ly = 25 mm and beta = 1.5

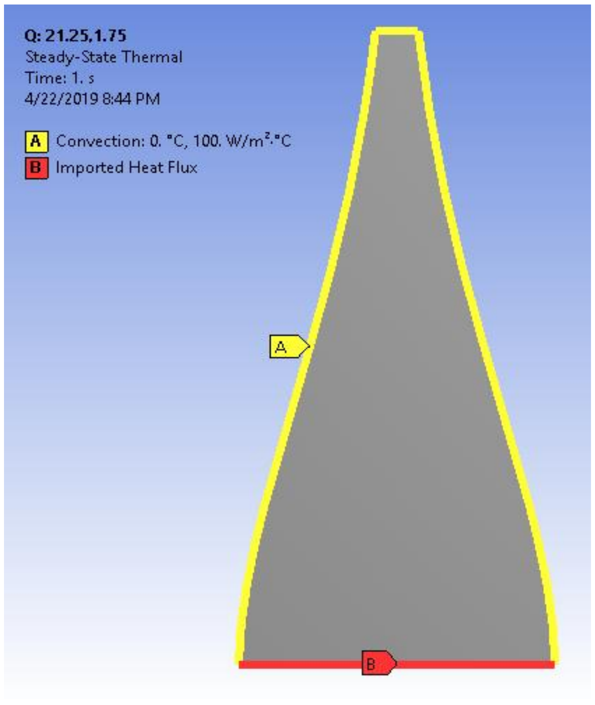


```
Table for Max Temperature and Heat Flux for each mesh size

Grid spacing range,h(m) Max Temperature(C) Max Heat Flux(W/m^2)
0.000125        79.775124       216536.448476
0.000250        79.808894       210781.242093
0.000500        79.842666       200596.653102


Richardson extrapolation & GCI analysis

Grid size used
1. 1.250000e-04 (m)
2. 2.500000e-04 (m)
3. 5.000000e-04 (m)

1 For Temperature
____________
Order of Grid Convergence, p = 0.000085
GCI12 = 8.969956
GCI23 = 8.966689
asymptotic range of convergence = 1.000423

Final answer : -492.688351 +- 896.995592 % C

2 For Heat Flux
____________
Order of Grid Convergence, p = 0.823448
GCI12 = 0.043168
GCI23 = 0.078476
asymptotic range of convergence = 0.973422

Final answer : 224014.330482 +- 4.316757 % W/m^2
```

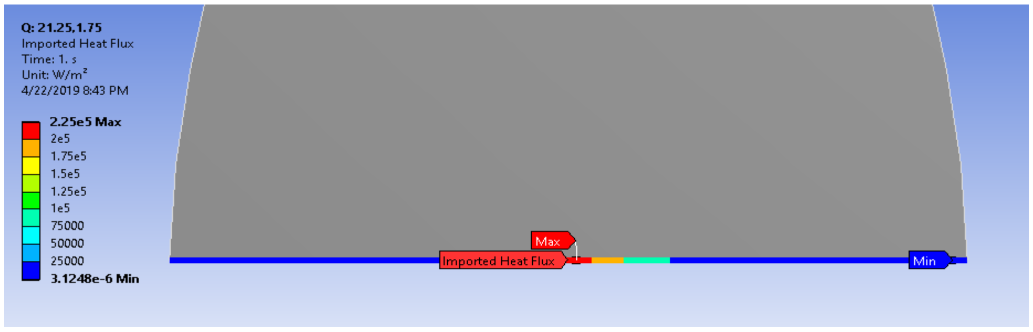
5. Comparison with commercial code solution (ANSYS)

One of the most important objectives while generating mesh in commercial software was to keep the aspect ratio within a certain allowable limit. As found online and experts' opinions, aspect ratio for 2D quadrilateral mesh should be below 5. So, all the analyses done were with an aspect ratio of less than 4 to 4.5. Due to certain licensing issues with use of such commercial software, the mesh size could not have been taken down any further.



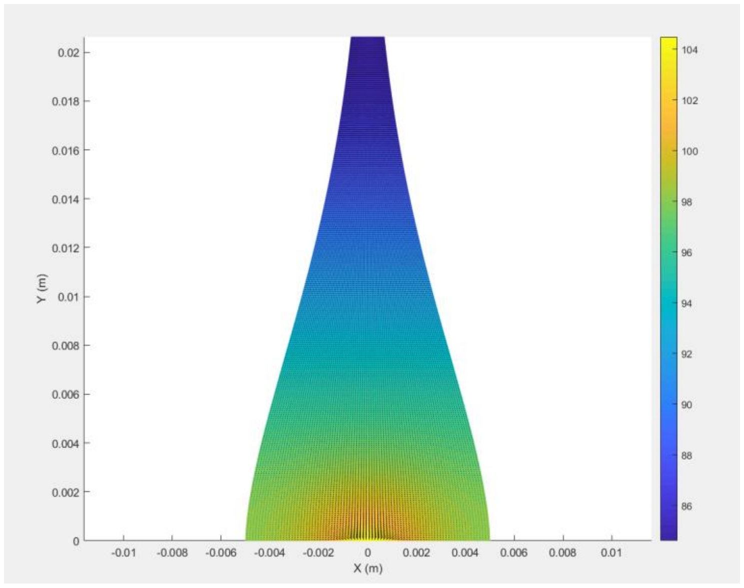


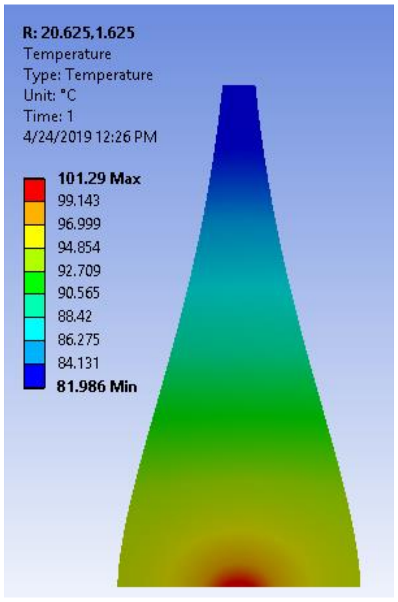
As for the Boundary condition, the 3 boundary edges were subjected to convection with co-efficient of 100 W/m^2 ^{O}C and ambient temperature 0 ^{O}C. Bottom edge is subjected to variable heat flux as provided in the problem statement. This data was provided in ANSYS using external data option. Drawback to doing so is the fact that it cannot be provided heat flux as function of distance. Instead, the heat flux is specified at various distances. To reduce these errors, the variations in distances were kept as 0.5 mm.



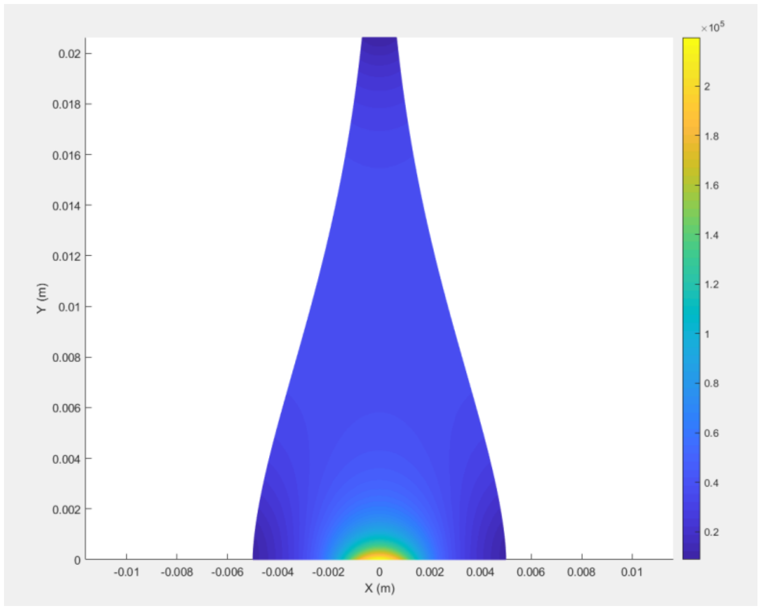
For ly = 20.625, beta =1.625

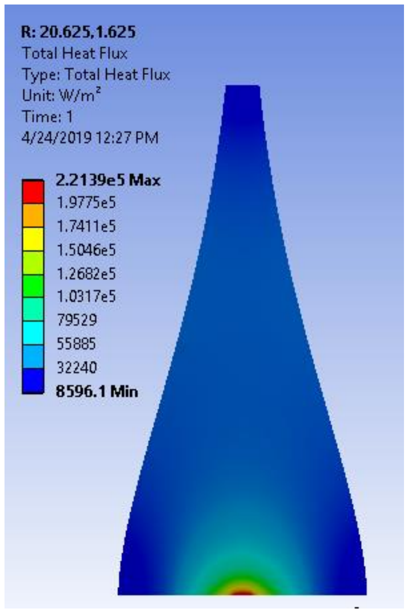
Temperature (℃)





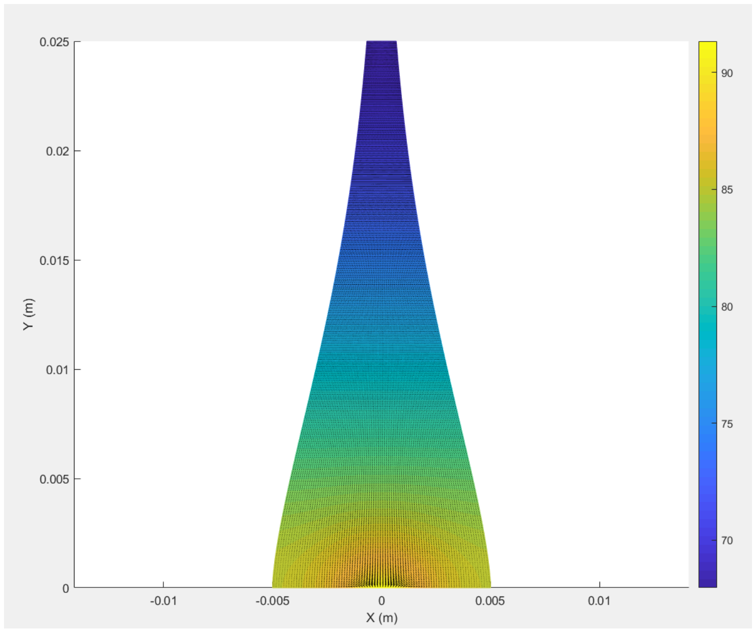
Heat flux ($\frac{W}{m^2}$)

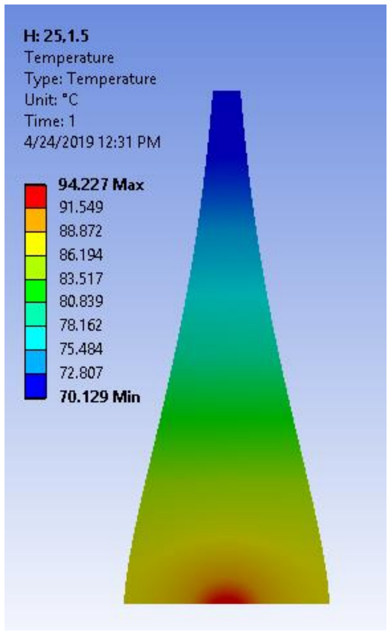




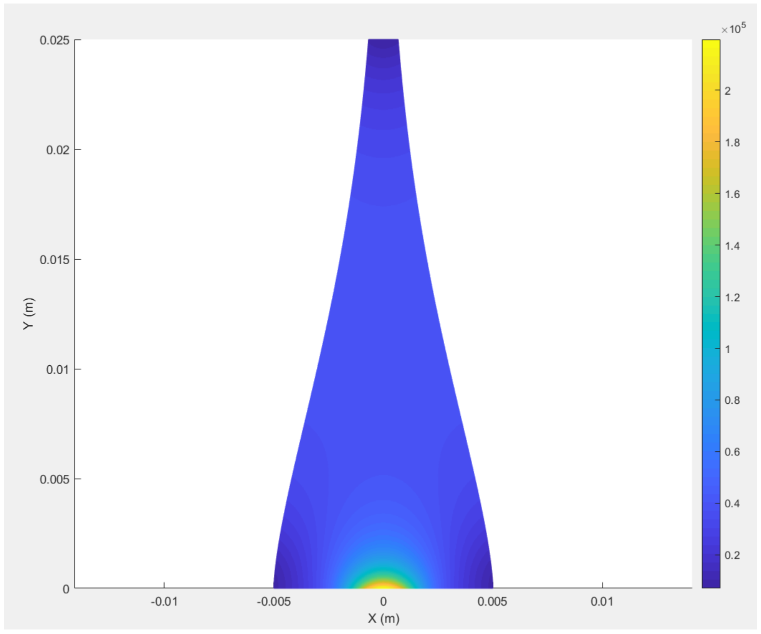
For ly = 25, beta =1.5

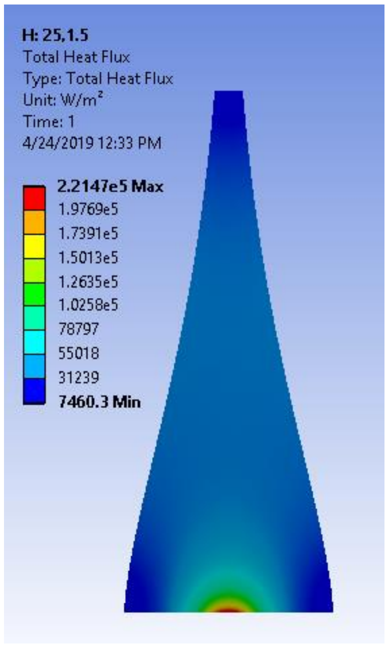
Temperature (℃)





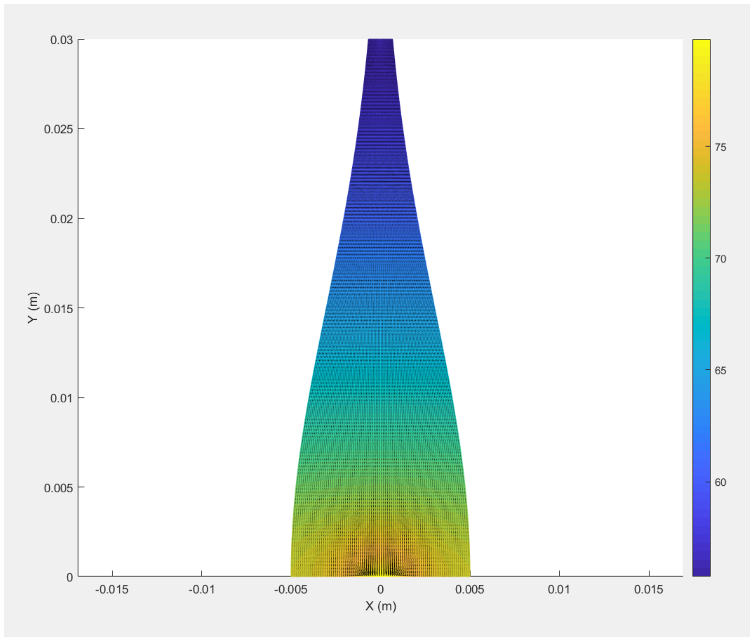
Heat flux ($\frac{W}{m^2}$)

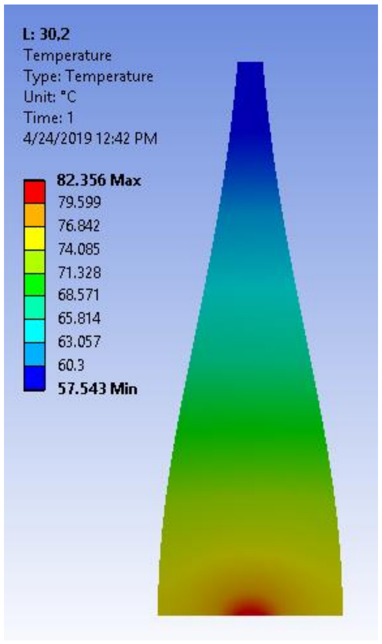




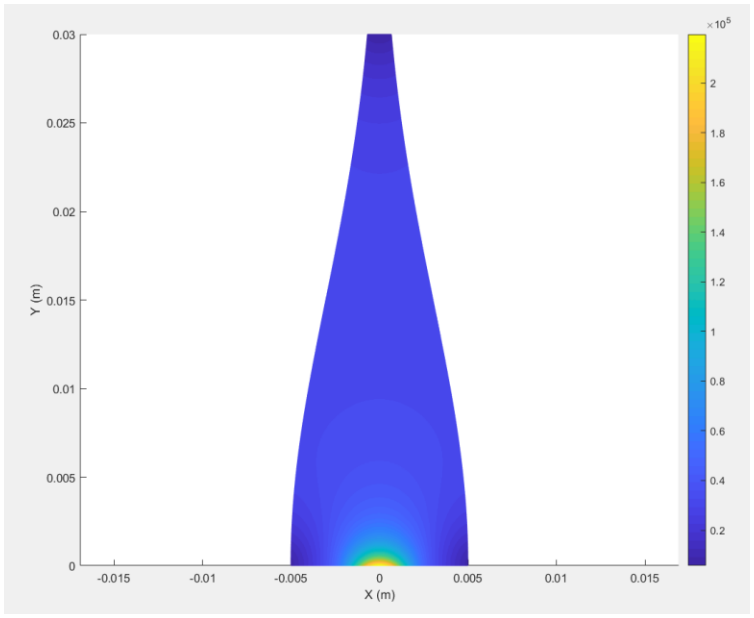
For ly = 30, beta =2

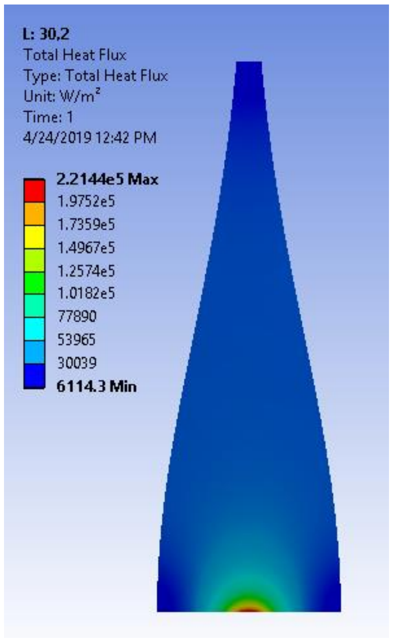
Temperature (℃)





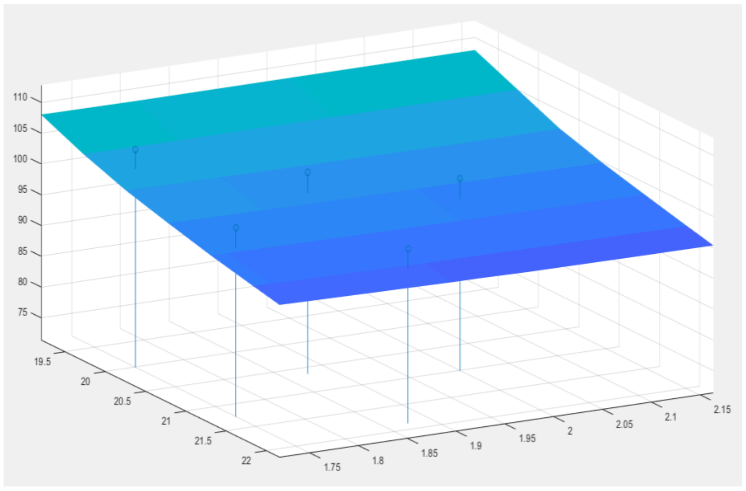
Heat flux $(\frac{W}{m^2})$

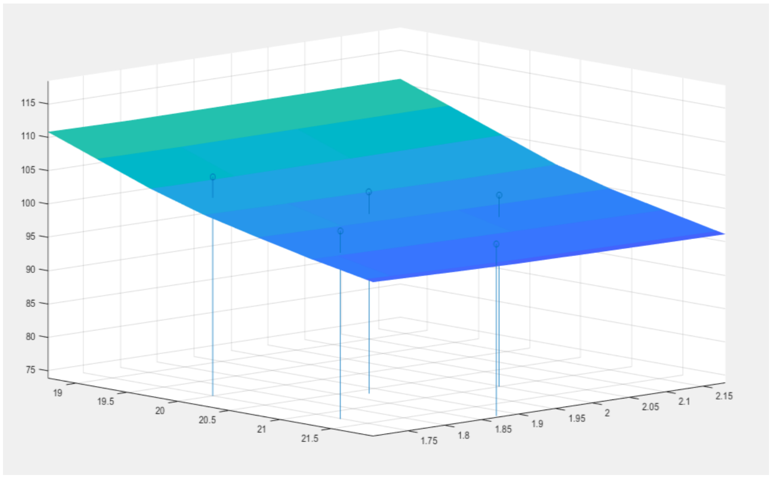




For the comparison purpose above, mesh Size in MATLAB and ANSYS is kept at 0.08mm. Apart from the ANSYS and MATLAB results shown above, all the remaining iterations done in both the software is tabulated below. This data helps understand the variations in commercial software and MATLAB.

The variation in both the results can be easily observed. Below is the representation of temperature surface as variation in ly and beta for QUAD 4 and QUAD 8 elements. Surface values represent the ANSYS solution and the points just above the curve are answers from MATLAB. This helps us conclude that the ANSYS results are going to converge.



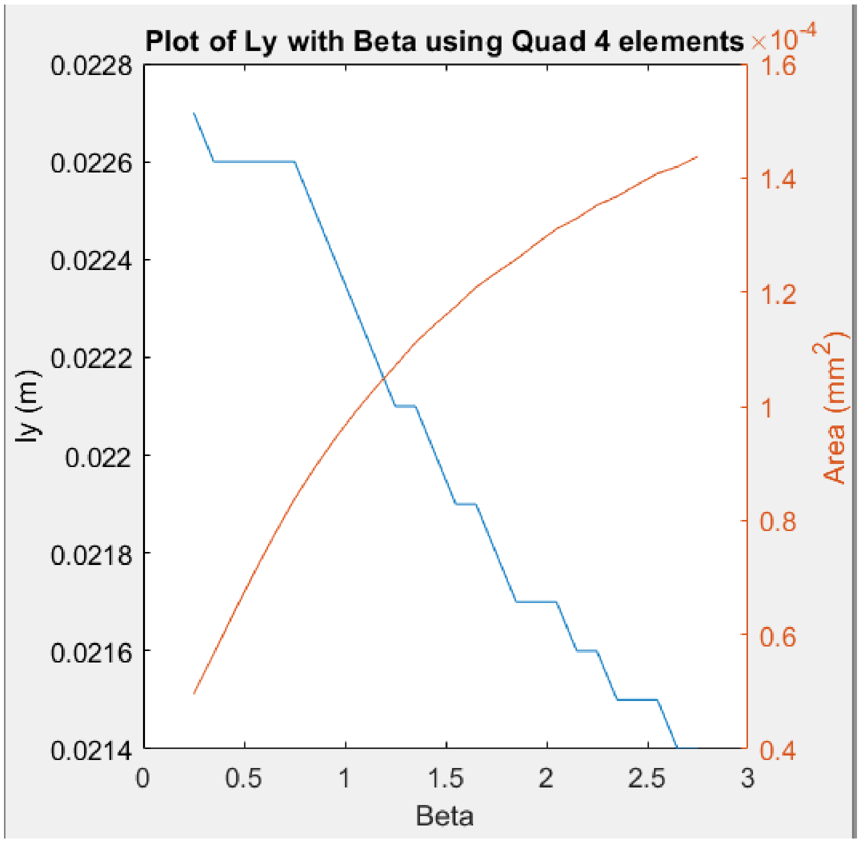


Height of Fin	beta	Temp. ANSYS (QUAD4)	Temp. ANSYS (QUAD8)	Temp. MATLAB (QUAD4)	Temp. MATLAB (QUAD8)
15	1.5	132.55	132.52	133.0261	133.0241
15	2	131.89	131.86	132.362	132.3601
25	1.5	94.27	94.22	91.3417	91.34
25	2	87.556	87.52	90.3146	90.3129
20	1.5	103.84	103.81	107.1258	107.124
20	1.75	103.44	103.41	106.6737	106.672
20.625	1.625	101.29	101.25	104.488	104.4863
20.625	1.875	100.87	100.84	104.06	104.058
20.625	2.5	100.09	100.06	103.2517	103.2501
21.875	1.875	96.59	96.555	99.6384	99.6367
21.25	1.75	98.872	98.837	101.9942	101.9925
22.5	2	94.433	94.398	97.4131	97.4114
21	2	99.351	99.316	102.4891	102.4874
21	3	98.288	98.254	101.3925	101.3908
21	1.5	100.2	100.16	103.3642	103.3625
21	1.6	100	99.97	103.1622	103.1605

6. Optimal Design Parameters

For QUAD4 element-
MATLAB code – Appendix 2.4 to find optimal ly and beta for the fin design. We have plotted ly with beta to satisfy the temperature condition of 100 ℃ difference from the ambient temperature.

Also, area of the fin is also plotted with ly and beta.



For the very lower value of beta, the fin does not form a curve shape. It should be high enough (say 0.25) for the fin to form a curve shape.

APPENDIX

APPENDIX 2.1
REPRESENTATIVE SOLUTION

```
function [d,qf,Area] = tqanalysis(ly, h, beta, etype)
% ly=0.030; %height of the fin, in m
% h=0.00008; %approximate element size,m
% beta=2; %exponential factor controlling the rate of fin taper

% etype='QUAD4';
D = 45; %thermal conductivity, in W/m-K
lx = 0.010; %base width, in m
Hc = 100; %convection coefficient, in W/m^2-K
mesh = fin_mesh(lx, ly, h, beta, etype);

if strcmp(etype, 'QUAD4')
    nne = 4;
    shape=@ shape2;
    shapeq=@shapeq4;
    %Connectivity matric for left, top, right, and bottom edge
for QUAD4
    conn.left =[mesh.left_nodes(1:end-
1);mesh.left_nodes(2:end)];
    conn.top =[mesh.top_nodes(1:end-1); mesh.top_nodes(2:end)];
    conn.right =[mesh.right_nodes(1:end-
1);mesh.right_nodes(2:end)];
    conn.bottom =[mesh.bottom_nodes(1:end-1);
mesh.bottom_nodes(2:end)];

elseif strcmp(etype, 'QUAD8')
    nne = 8;
    shape=@ shape3;
    shapeq=@shapeq8;
    %Connectivity matric for left, top, right, and bottom edge
for QUAD8
    conn.left = [mesh.left_nodes(1:2:end-2);
mesh.left_nodes(2:2:end-1); mesh.left_nodes(3:2:end)];
    conn.top = [mesh.top_nodes(1:2:end-2);
mesh.top_nodes(2:2:end-1); mesh.top_nodes(3:2:end)];
    conn.right = [mesh.right_nodes(1:2:end-2);
mesh.right_nodes(2:2:end-1); mesh.right_nodes(3:2:end)];
    conn.bottom = [mesh.bottom_nodes(1:2:end-2);
mesh.bottom_nodes(2:2:end-1); mesh.bottom_nodes(3:2:end)];
end
```

```
qpts = quadrature(4); %quadrature points
%Stiffness Matrix
K = spalloc(length(mesh.x),length(mesh.x),9*length(mesh.x));
%flux matrix
f = zeros(length(K),1);
Area = 0;

%for the stiffness matrix
for c = mesh.conn
    xe = mesh.x(:,c);
    ke = zeros(length(c));
    for qx = qpts
        for qy = qpts
            q = [qx(1) qy(1)];
            [~,dNdp] = shapeq(q);
            J = xe*dNdp;
            B = dNdp/J;
            ke = ke + B*D*B'*qx(2)*qy(2)*det(J);
            Area = Area + det(J)*qx(2)*qy(2); %Area of the fin
using jacobian and quadrature points
        end
    end
    K(c,c) = K(c,c)+ke;
end

%Applying convection bondary conditions on left, top, and right
edges
for c = [conn.left conn.right conn.top]
    xe = mesh.x(:,c);
    he = zeros(length(c));
    for q = qpts
      [N,dNdp] = shape(q(1));
      J = xe*dNdp;
      he = he + N*Hc*N'*q(end)*norm(J);
    end
K(c,c) = K(c,c)+he;
end
 %Applying heat flux on bottom edge of domain
for c = conn.bottom
    xe = mesh.x(:,c);
    Fe = zeros(length(c),1);
    for q = qpts
       [N,dNdp] = shape(q(1));
       x = xe*N;
       J = xe*dNdp;
       qbar = 225*1000*exp(-(1000*x(1))^2); %flux in W/m^2
       Fe = Fe + N*norm(J)*qbar*q(end);
    end
f(c) = f(c)+Fe;
end

d = K\f; %temperature calculation
max(d)

%Calculating heat flux
A = spalloc(length(mesh.x),length(mesh.x),9*length(mesh.x));
y = zeros(length(A),2);

for c = mesh.conn
    xe = mesh.x(:,c);
    de = d(c)';
    Ae = zeros(length(c));
    for qx = qpts
        for qy = qpts
            q = [qx(1) qy(1)];
            [N,dNdp] = shapeq(q);
            J = xe*dNdp;
            B = dNdp/J;
            q = -D*de*B;
            Ae = Ae + N*N'*qx(2)*qy(2)*det(J);
            y(c,:) = y(c,:) + N*q*det(J)*qx(2)*qy(2);
        end
    end
    A(c,c) = A(c,c)+Ae;
end

qf = A\y;

for i=1:length(qf)
   qfresult(i)=sqrt(qf(i,1)^2+qf(i,2)^2);
end

%plotting the results
%plot for temperature
titlely=num2str(ly*1000);
titleb=num2str(beta);
title1=strcat("Temperature & Heat Flux variation along body for
ly = ",titlely,"mm & Beta = ", titleb);
figure(1)
clf;
p.vertices = mesh.x';
p.faces = mesh.conn';

if nne == 8
    p.faces = mesh.conn([1,5,2,6,3,7,4,8],:)';
```

```
end

p.facecolor = 'interp';
p.facevertexcdata = d;
p.edgecolor='interp'
patch(p)
colorbar
axis equal;
xlabel("X (m)")
ylabel("Y (m)")
title(title1)
hold on

for heat flux
    qq = [];
    xx = [];
for c = mesh.conn
    xe = mesh.x(:,c);
    qe = qf(c,:)';
    [N,~] = shapeq([0;0]);
    xx(end+1, :) = xe * N;
    qq(end+1, :) = qe * N;
end

quiver(xx(:,1), xx(:,2), qq(:,1), qq(:,2),1, 'color', 'k');

end
 %plot for heat flux
figure(2)
clf;
p.vertices = mesh.x';
p.faces = mesh.conn';

if nne == 8
    p.faces = mesh.conn([1,5,2,6,3,7,4,8],:)';
end

p.facecolor = 'interp';
p.facevertexcdata = qfresult';
p.edgecolor='interp';
patch(p)
colorbar
axis equal;
xlabel("X (m)")
ylabel("Y (m)")
% title(title1)
%Linear shape function and its derivative
function [N, dNdp] = shape2(p)
```

```
N= [(1-p)/2; (1+p)/2];
dNdp = [-1/2; 1/2];
end

%Quadratic shape functions and its derivative
function [N, dNdp] = shape3(p)
N = [p*(p-1)/2; (p+1)*(1-p); p*(p+1)/2];
dNdp = [(2*p-1)/2; -2*p; (2*p+1)/2];
end

function [N, dNdp] = shapeq4(p)
N = [(1-p(1))*(1-p(2))/4; (1+p(1))*(1-p(2))/4;
(1+p(1))*(1+p(2))/4; (1-p(1))*(1+p(2))/4];
dNdp = (1/4)*[-(1-p(2)) -(1-p(1)); (1-p(2)) -(1+p(1)); (1+p(2))
(1+p(1)); -(1+p(2)) (1-p(1))];
end

function [N, dNdp] = shapeq8(p)
N  = [-(1+p(1)+p(2))*(1-p(1))*(1-p(2))/4; (p(1)-p(2)-
1)*(1+p(1))*(1-p(2))/4; (p(1)+p(2)-1)*(1+p(1))*(1+p(2))/4; -
(1+p(1)-p(2))*(1-p(1))*(1+p(2))/4;(1-p(1)*p(1))*(1-p(2))/2;(1-
p(2)*p(2))*(1+p(1))/2;(1-p(1)*p(1))*(1+p(2))/2; (1-
p(2)*p(2))*(1-p(1))/2];
dNdp = 0.25*[-((1-p(1))*(1-p(2))+(1+p(1)+p(2))*-1*(1-p(2))),-
((1-p(1))*(1-p(2))+(1+p(1)+p(2))*(1-p(1))*-1); (1+p(1))*(1-
p(2))+(p(1)-p(2)-1)*(1-p(2)),-1*(1+p(1))*(1-p(2))+(p(1)-p(2)-
1)*(1+p(1))*-1; (1+p(1))*(1+p(2))+(p(1)+p(2)-
1)*(1+p(2)),1*(1+p(1))*(1+p(2))+(p(1)+p(2)-1)*(1+p(1)); -((1-
p(1))*(1+p(2))+(p(1)-p(2)+1)*-1*(1+p(2))),-(-1*(1-
p(1))*(1+p(2))+(p(1)-p(2)+1)*(1-p(1))); 2*((-2*p(1))*(1-
p(2))),2*((1-p(1)^2)*-1); 2*(1*(1-p(2)^2)),2*((1+p(1))*(-
2*p(2))); 2*((1+p(2))*(-2*p(1))),2*(1*(1-p(1)^2)); 2*(-1*(1-
p(2)^2)),2*((1-p(1))*(-2*p(2)))];
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
```

APPENDIX 2.2
CONVERGENCE TEST

```
ly=0.025; %height of the fin, in m
rangeh=0.00025:0.0005:0.002; %grid spacing range (m)
beta=1.5; %exponential factor controlling the rate of fin taper

etype='QUAD8';
D = 45; %thermal conductivity, in W/m-K
lx = 0.010; %base width, in m
Hc = 100; %convection coefficient, in W/m^2-K

for hcount=1:length(rangeh)
    h=rangeh(hcount);
    mesh = fin_mesh(lx, ly, h, beta, etype);

    if strcmp(etype, 'QUAD4')
        nne = 4;
        shape=@ shape2;
        shapeq=@shapeq4;
        %Connectivity matric for left, top, right, and bottom
edge for QUAD4
        conn.left =[mesh.left_nodes(1:end-
1);mesh.left_nodes(2:end)];
        conn.top =[mesh.top_nodes(1:end-1);
mesh.top_nodes(2:end)];
        conn.right =[mesh.right_nodes(1:end-
1);mesh.right_nodes(2:end)];
        conn.bottom =[mesh.bottom_nodes(1:end-1);
mesh.bottom_nodes(2:end)];

    elseif strcmp(etype, 'QUAD8')
        nne = 8;
        shape=@ shape3;
        shapeq=@shapeq8;
        %Connectivity matric for left, top, right, and bottom
edge for QUAD8
        conn.left = [mesh.left_nodes(1:2:end-2);
mesh.left_nodes(2:2:end-1); mesh.left_nodes(3:2:end)];
        conn.top = [mesh.top_nodes(1:2:end-2);
mesh.top_nodes(2:2:end-1); mesh.top_nodes(3:2:end)];
        conn.right = [mesh.right_nodes(1:2:end-2);
mesh.right_nodes(2:2:end-1); mesh.right_nodes(3:2:end)];
        conn.bottom = [mesh.bottom_nodes(1:2:end-2);
mesh.bottom_nodes(2:2:end-1); mesh.bottom_nodes(3:2:end)];
    end
```

```
qpts = quadrature(4); %quadrature points
%Stiffness Matrix
K = spalloc(length(mesh.x),length(mesh.x),9*length(mesh.x));
%flux matrix
f = zeros(length(K),1);
%Heat source
Sf=zeros(length(K),1);

%for the stiffness matrix
for c = mesh.conn
    xe = mesh.x(:,c);
    ke = zeros(length(c));
    for qx = qpts
        for qy = qpts
            q = [qx(1) qy(1)];
            [N,dNdp] = shapeq(q);
            J = xe*dNdp;
            B = dNdp/J;
            ke = ke + B*D*B'*qx(2)*qy(2)*det(J);
            x=xe(1,:)*N;
            y=xe(2,:)*N;
            S=-D/(x^2+y^2)^(1/2);
            Sf(c)=Sf(c)+N*S*det(J)*qx(2)*qy(2);
        end
    end
    K(c,c) = K(c,c)+ke;
end

%Temperature bondary condition on left and right edges
for c = [conn.left conn.right]
    xe = mesh.x(:,c);
    K(c,:) = 0.0;
    K(c,c) = eye(length(c));
    x=xe(1,1);
    y=xe(2,1);
    Tbar=sqrt(x^2+y^2);
    f(c) = Tbar;
end

%Heat flux on bottom edge
for c = conn.bottom
    xe = mesh.x(:,c);
    Fe = zeros(length(c),1);
    for q = qpts
        [N,dNdp] = shape(q(1));
        x=xe(1,:)*N;
        y=xe(2,:)*N;
```

```
            J = xe*dNdp;
            qbar = -(D*y)/(x^2 + y^2)^(1/2);
            Fe = Fe + N*norm(J)*qbar*q(end);
        end
        f(c) = f(c)+Fe;
    end

    %Heat flux on top edge
    for c = conn.top
        xe = mesh.x(:,c);
        Fe = zeros(length(c),1);
        for q = qpts
            [N,dNdp] = shape(q(1));
            x=xe(1,:)*N;
            y=xe(2,:)*N;
            J = xe*dNdp;
            qbar = -(D*x)/(x^2 + y^2)^(1/2);
            Fe = Fe + N*norm(J)*qbar*q(end);
        end
        f(c) = f(c)+Fe;
    end

    %flux
    F=f+Sf;
    %temperature
    d = K\F;

    %L2 error norm calculation
    enum=0;
    eden=0;
    for c=mesh.conn
        xe=mesh.x(:,c);
        de=d(c)';
        for qx=qpts
            for qy=qpts
                q = [qx(1) qy(1)];
                [N,dNdp] = shapeq(q);
                J = xe*dNdp;
                x=xe(1,:)*N;
                y=xe(2,:)*N;
                Tx = sqrt(x^2+y^2);
                enum = enum + (Tx - de*N)^2 *
det(J)*qx(end)*qy(end);
                eden = eden + Tx^2 * det(J)*qx(end)*qy(end);
            end
        end
    end
    e1(hcount) = sqrt(enum/eden);
end

%order calculation
for i=1:length(rangeh)-1
    order(i)=[log(e1(i+1))-log(e1(i))]/[log(rangeh(i+1))-
log(rangeh(i))];
end

fprintf("Order of solution is %d ",mean(order));
figure(2)
plot(log(rangeh),log(e1))
xlabel("Log(Element Size)"); ylabel("Log(L2 Error norm)")
title("L2 error norm for QUAD8")


%Linear shape function and its derivative
function [N, dNdp] = shape2(p)
N= [(1-p)/2; (1+p)/2];
dNdp = [-1/2; 1/2];
end

%Quadratic shape functions and its derivative
function [N, dNdp] = shape3(p)
N = [p*(p-1)/2; (p+1)*(1-p); p*(p+1)/2];
dNdp = [(2*p-1)/2; -2*p; (2*p+1)/2];
end

function [N, dNdp] = shapeq4(p)
N = [(1-p(1))*(1-p(2))/4; (1+p(1))*(1-p(2))/4;
(1+p(1))*(1+p(2))/4; (1-p(1))*(1+p(2))/4];
dNdp = (1/4)*[-(1-p(2)) -(1-p(1)); (1-p(2)) -(1+p(1)); (1+p(2))
(1+p(1)); -(1+p(2)) (1-p(1))];
end

function [N, dNdp] = shapeq8(p)
N  = [-(1+p(1)+p(2))*(1-p(1))*(1-p(2))/4; (p(1)-p(2)-
1)*(1+p(1))*(1-p(2))/4; (p(1)+p(2)-1)*(1+p(1))*(1+p(2))/4; -
(1+p(1)-p(2))*(1-p(1))*(1+p(2))/4;(1-p(1)*p(1))*(1-p(2))/2;(1-
p(2)*p(2))*(1+p(1))/2;(1-p(1)*p(1))*(1+p(2))/2; (1-
p(2)*p(2))*(1-p(1))/2];
dNdp = 0.25*[-((1-p(1))*(1-p(2))+(1+p(1)+p(2))*-1*(1-p(2))),-
((1-p(1))*(1-p(2))+(1+p(1)+p(2))*(1-p(1))*-1); (1+p(1))*(1-
p(2))+(p(1)-p(2)-1)*(1-p(2)),-1*(1+p(1))*(1-p(2))+(p(1)-p(2)-
1)*(1+p(1))*-1; (1+p(1))*(1+p(2))+(p(1)+p(2)-
1)*(1+p(2)),1*(1+p(1))*(1+p(2))+(p(1)+p(2)-1)*(1+p(1)); -((1-
p(1))*(1+p(2))+(p(1)-p(2)+1)*-1*(1+p(2))),-(-1*(1-
p(1))*(1+p(2))+(p(1)-p(2)+1)*(1-p(1))); 2*((-2*p(1))*(1-
p(2))),2*((1-p(1)^2)*-1); 2*(1*(1-p(2)^2)),2*((1+p(1))*(-
2*p(2))); 2*((1+p(2))*(-2*p(1))),2*(1*(1-p(1)^2)); 2*(-1*(1-
p(2)^2)),2*((1-p(1))*(-2*p(2)))];
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
```

APPENDIX 2.3
MESH REFINEMENT TEST

```
ly=0.030; %height of the fin, in m
rangeh=[0.000125 0.00025 0.0005]; %grid spacing range (m)
beta=2; %exponential factor controlling the rate of fin taper

etype='QUAD4';
D = 45; %thermal conductivity, in W/m-K
lx = 0.010; %base width, in m
Hc = 100; %convection coefficient, in W/m^2-K

for hcount=1:length(rangeh)
    h=rangeh(hcount);
    mesh = fin_mesh(lx, ly, h, beta, etype);

    if strcmp(etype, 'QUAD4')
        nne = 4;
        shape=@ shape2;
        shapeq=@shapeq4;
        %Connectivity matric for left, top, right, and bottom
edge for QUAD4
        conn.left =[mesh.left_nodes(1:end-
1);mesh.left_nodes(2:end)];
        conn.top =[mesh.top_nodes(1:end-1);
mesh.top_nodes(2:end)];
        conn.right =[mesh.right_nodes(1:end-
1);mesh.right_nodes(2:end)];
        conn.bottom =[mesh.bottom_nodes(1:end-1);
mesh.bottom_nodes(2:end)];

    elseif strcmp(etype, 'QUAD8')
        nne = 8;
        shape=@ shape3;
        shapeq=@shapeq8;
        %Connectivity matric for left, top, right, and bottom
edge for QUAD8
        conn.left = [mesh.left_nodes(1:2:end-2);
mesh.left_nodes(2:2:end-1); mesh.left_nodes(3:2:end)];
        conn.top = [mesh.top_nodes(1:2:end-2);
mesh.top_nodes(2:2:end-1); mesh.top_nodes(3:2:end)];
        conn.right = [mesh.right_nodes(1:2:end-2);
mesh.right_nodes(2:2:end-1); mesh.right_nodes(3:2:end)];
        conn.bottom = [mesh.bottom_nodes(1:2:end-2);
mesh.bottom_nodes(2:2:end-1); mesh.bottom_nodes(3:2:end)];
    end
    qpts = quadrature(3); %quadrature points
    %Stiffness Matrix
    K = spalloc(length(mesh.x),length(mesh.x),9*length(mesh.x));
    %flux matrix
    f = zeros(length(K),1);

    %for the stiffness matrix
    for c = mesh.conn
        xe = mesh.x(:,c);
        ke = zeros(length(c));
        for qx = qpts
            for qy = qpts
                q = [qx(1) qy(1)];
                [N,dNdp] = shapeq(q);
                J = xe*dNdp;
                B = dNdp/J;
                ke = ke + B*D*B'*qx(2)*qy(2)*det(J);
            end
        end
        K(c,c) = K(c,c)+ke;
    end

    %Applying convection bondary conditions on left, top, and
right edges
    for c = [conn.left conn.right conn.top]
        xe = mesh.x(:,c);
        he = zeros(length(c));
        for q = qpts
            [N,dNdp] = shape(q(1));
            J = xe*dNdp;
            he = he + N*Hc*N'*q(end)*norm(J);
        end
        K(c,c) = K(c,c)+he;
    end

    %Applying heat flux on bottom edge of domain
    for c = conn.bottom
        xe = mesh.x(:,c);
        Fe = zeros(length(c),1);
        for q = qpts
            [N,dNdp] = shape(q(1));
            x = xe*N;
            J = xe*dNdp;
            qbar = 225*1000*exp(-(1000*x(1))^2); %flux in W/m^2
            Fe = Fe + N*norm(J)*qbar*q(end);
        end
        f(c) = f(c)+Fe;
    end

    d = K\f; %temperature calculation

    %Calculating heat flux
    A = spalloc(length(mesh.x),length(mesh.x),9*length(mesh.x));
    y = zeros(length(A),2);
    for c = mesh.conn
        xe = mesh.x(:,c);
        de = d(c)';
        Ae = zeros(length(c));
        for qx = qpts
            for qy = qpts
                q = [qx(1) qy(1)];
                [N,dNdp] = shapeq(q);
                J = xe*dNdp;
                B = dNdp/J;
                q = -D*de*B;
                Ae = Ae + N*N'*qx(2)*qy(2)*det(J);
                y(c,:) = y(c,:) + N*q*det(J)*qx(2)*qy(2);
            end
        end
        A(c,c) = A(c,c)+Ae;
    end
    qf = A\y;

    %Resultant flux
    for i=1:length(qf)
        qf_t(i)=sqrt((qf(i,1))^2+(qf(i,2))^2);
    end

    %Richardson Extrapolation and Grid Convergence Index (GCI)
for mesh
    %refinement test

    %finding and saving maximum heat flux and temperature
    % add=find(mesh.x(2,:) == 0);
    % add=length(add);
    % if rem(add,2)~=0
    % H1(hcount,1)=d((add+1)/2);
    % H1(hcount,2)=qf_temp((add+1)/2);
    % else
    % H1(hcount,1)=(d(add/2)+d(add/2+1))/2;
    % H1(hcount,2)=(qf_temp(add/2)+qf_temp(add/2+1))/2;
    % end

    H1(hcount,1)=max(d);
    H1(hcount,2)=max(qf_t);
end

fprintf("\n Table for Max Temperature and Heat Flux for each
mesh size \n")
fprintf("\n Grid spacing range,h(m) Max Temperature(C) Max Heat
Flux(W/m^2)\n")
fprintf(" %f \t \t %f \t \t %f \n",rangeh(1),H1(1,:));
fprintf(" %f \t \t %f \t \t %f \n",rangeh(2),H1(2,:));
fprintf(" %f \t \t %f \t \t %f \n\n",rangeh(3),H1(3,:));
fprintf("\n Richardson extrapolation & GCI analysis \n")
fprintf("\n Grid size used \n 1. %d (m) \n 2. %d (m) \n 3. %d
(m)",rangeh);

Fs=1.25; %factor of safety for comparisons over three grids

for i=1:2
    if i==1
        fprintf("\n \n %d For Temperature ",i)
    else
        fprintf("\n \n %d For Heat Flux ",i)
    end

    fprintf("\n ____________")
    p(i)=log(abs(H1(3,i)-H1(2,i))/abs(H1(2,i)-H1(1,i)))/log(2);

    %order of accuracy
    fprintf("\n Order of Grid Convergence, p = %f",p(i));
    fh0(i)=H1(1,i)+(H1(1,i)-H1(2,i))/(2^p(i)-1);

    %calculating the GCIs
    GC1(i,1)=Fs*abs((H1(1,i)-H1(2,i))/H1(1,i))/(2^p(i)-1);
    fprintf("\n GCI12 = %f",GC1(i,1));
    GC1(i,2)=Fs*abs((H1(2,i)-H1(3,i))/H1(2,i))/(2^p(i)-1);
    fprintf("\n GCI23 = %f",GC1(i,2));

    %asymptotic range of convergence
    convergence(i)=GC1(i,1)*2^(p(i))/GC1(i,2);

    fprintf("\n asymptotic range of convergence = %f
",convergence(i));
    if i==1
        fprintf("\n \n Final answer : %f +- %f %%
C",fh0(i),GC1(i,1)*100);
    else
        fprintf("\n \n Final answer : %f +- %f %%
W/m^2",fh0(i),GC1(i,1)*100);
```

```
    end
end

fprintf("\n \n");

%Linear shape function and its derivative
function [N, dNdp] = shape2(p)
N= [(1-p)/2; (1+p)/2];
dNdp = [-1/2; 1/2];
end

%Quadratic shape functions and its derivative
function [N, dNdp] = shape3(p)
N = [p*(p-1)/2; (p+1)*(1-p); p*(p+1)/2];
dNdp = [(2*p-1)/2; -2*p; (2*p+1)/2];
end

function [N, dNdp] = shapeq4(p)
N = [(1-p(1))*(1-p(2))/4; (1+p(1))*(1-p(2))/4;
(1+p(1))*(1+p(2))/4; (1-p(1))*(1+p(2))/4];
dNdp = (1/4)*[-(1-p(2)) -(1-p(1)); (1-p(2)) -(1+p(1)); (1+p(2))
(1+p(1)); -(1+p(2)) (1-p(1))];
end

function [N, dNdp] = shapeq8(p)
N  = [-(1+p(1)+p(2))*(1-p(1))*(1-p(2))/4; (p(1)-p(2)-
1)*(1+p(1))*(1-p(2))/4; (p(1)+p(2)-1)*(1+p(1))*(1+p(2))/4; -
(1+p(1)-p(2))*(1-p(1))*(1+p(2))/4;(1-p(1)*p(1))*(1-p(2))/2;(1-
p(2)*p(2))*(1+p(1))/2;(1-p(1)*p(1))*(1+p(2))/2; (1-
p(2)*p(2))*(1-p(1))/2];
dNdp = 0.25*[-((1-p(1))*(1-p(2))+(1+p(1)+p(2))*-1*(1-p(2))),-
((1-p(1))*(1-p(2))+(1+p(1)+p(2))*(1-p(1))*-1); (1+p(1))*(1-
p(2))+(p(1)-p(2)-1)*(1-p(2)),-1*(1+p(1))*(1-p(2))+(p(1)-p(2)-
1)*(1+p(1))*-1; (1+p(1))*(1+p(2))+(p(1)+p(2)-
1)*(1+p(2)),1*(1+p(1))*(1+p(2))+(p(1)+p(2)-1)*(1+p(1)); -((1-
p(1))*(1+p(2))+(p(1)-p(2)+1)*-1*(1+p(2))),-(-1*(1-
p(1))*(1+p(2))+(p(1)-p(2)+1)*(1-p(1))); 2*((-2*p(1))*(1-
p(2))),2*((1-p(1)^2)*-1); 2*(1*(1-p(2)^2)),2*((1+p(1))*(-
2*p(2))); 2*((1+p(2))*(-2*p(1))),2*(1*(1-p(1)^2)); 2*(-1*(1-
p(2)^2)),2*((1-p(1))*(-2*p(2)))];
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
```

APPENDIX 2.4
OPTIMAL DESIGN PARAMETERS

```
etype = 'QUAD4';
rangely = 25:-0.1:15;
Betarange = 2.75:-0.1:0.25;

for j=1:length(Betarange)
    for i=1:length(rangely)
        h=0.0005;
        ly = rangely(i)/1000;
        beta = Betarange(j);
        [d, ~,Area] = tqanalysis(ly, h, beta, etype);
        fprintf("ly= %f, beta= %f , max T= %f \n
\n",ly,beta,max(d));
        if max(d)>100
            break;
        end
    end
    LY(j)=ly;
    BT(j)=beta;
    AR(j)=Area;
end

%Plot of ly, beta, and area
plot(BT,LY)
xlabel("Beta")
ylabel("ly (m)")
yyaxis right
plot(BT,AR)
ylabel("Area (mm^2)")
title("Plot of Ly with Beta using Quad 4 elements")
```