VSM 167 FINITE ELEMENT METHOD BASICS

PROJECT 3

GROUP 2 - JIMMIEBOI ANDERSSON & GUSTAV GOODIEBAG

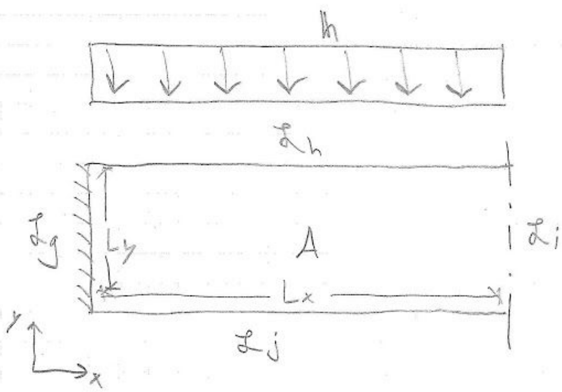
Task 1

a)

Due to symmetry the model has been simplified as the sketch below is shown. The lengths are given in the task.

Strong form

$$-\tilde{\nabla}^T(\mathbf{D}t\tilde{\nabla}\mathbf{u}) = t\mathbf{b}$$



BC's: $\mathbf{u} = 0$ on $\mathcal{L}_g$

$u_x = 0$ on $\mathcal{L}_i$

$\mathbf{u} = \begin{bmatrix} u_x \\ u_y \end{bmatrix}$

$\mathbf{t}$ = prescribed traction $= \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

$\mathbf{t} = \begin{bmatrix} 0 \\ h_y \end{bmatrix}$ on $\mathcal{L}_h$

$t_y = 0$ on $\mathcal{L}_i$

$t_x = t_y = 0$ on $\mathcal{L}_j$

b)

Weak form

$$\int_A (\tilde{\nabla}\mathbf{v})^T \sigma t \, dA = \int_A \mathbf{v}^T \begin{bmatrix} 0 \\ -\rho g \end{bmatrix} t \, dA + \int_{\mathcal{L}_g} \mathbf{v}^T \mathbf{r} t \, d\mathcal{L} + \int_{\mathcal{L}_h} \mathbf{v}^T \begin{bmatrix} 0 \\ h_y \end{bmatrix} t \, d\mathcal{L} + \int_{\mathcal{L}_i} \mathbf{v}^T \begin{bmatrix} t_x \\ 0 \end{bmatrix} t \, d\mathcal{L} + \underbrace{\int_{\mathcal{L}_j} \mathbf{v}^T \begin{bmatrix} 0 \\ 0 \end{bmatrix} t \, d\mathcal{L}}_{=0}$$

$\mathbf{u} = 0$ on $\mathcal{L}_g$

$u_x = 0$ on $\mathcal{L}_i$

We introduce approximations:

$$u_x = \sum_{i=1}^{n} N_i(x,y)\, u_{x,i} \qquad u_y = \sum_{i=1}^{n} N_i(x,y)\, u_{y,i} \qquad N_i(x_j, y_j) = \begin{cases} 1 & i = j \text{ (at node } i) \\ 0 & i \neq j \text{ (at another node)} \end{cases}$$

$$\mathbf{a} = \begin{bmatrix} u_{x,1} \\ u_{y,1} \\ \vdots \\ u_{x,n} \\ u_{y,n} \end{bmatrix} \qquad \mathbf{u} = \mathbf{N}\mathbf{a} \qquad \boldsymbol{\varepsilon} = \tilde{\nabla}\mathbf{u} = \tilde{\nabla}\mathbf{N}\mathbf{a} = (\tilde{\nabla}\mathbf{N})\mathbf{a} = \mathbf{B}\mathbf{a}$$

$$\mathbf{c} = \begin{bmatrix} c_{x,1} \\ c_{y,1} \\ \vdots \\ c_{x,n} \\ c_{y,n} \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} N_i(x,y)\, c_{x,i} \\ \sum_{i=1}^{n} N_i(x,y)\, c_{y,i} \end{bmatrix} = \mathbf{N}\mathbf{c}$$

$$\mathbf{v}^T = \mathbf{c}^T \mathbf{N}^T$$

$$(\tilde{\nabla}\mathbf{N})^T = \mathbf{c}^T \mathbf{B}^T$$

Insert the approximations in weak form together with hooke's law $\sigma = \mathbf{D}\varepsilon$. $\mathbf{c}$ & $\mathbf{a}$ are constant, since eqv. should hold for arbitrary $\mathbf{c}^T$ we put it outside the integral in every term and remove it

FE

$$\int_A \mathbf{B}^T \mathbf{D} \mathbf{B} t \, dA \, \mathbf{a} = \int_A \mathbf{N}^T \mathbf{b} t \, dA + \int_{\mathcal{L}_g} \mathbf{N}^T \mathbf{r} t \, d\mathcal{L} + \int_{\mathcal{L}_h} \mathbf{N}^T \begin{bmatrix} 0 \\ h_y \end{bmatrix} t \, d\mathcal{L} + \int_{\mathcal{L}_i} \mathbf{N}^T \begin{bmatrix} t_x \\ 0 \end{bmatrix} t \, d\mathcal{L}$$

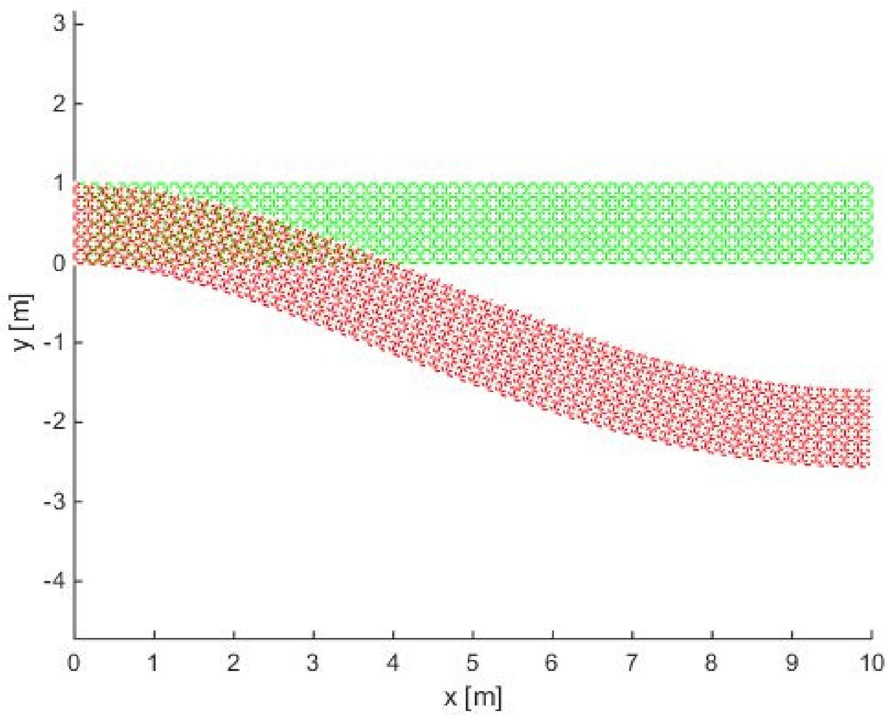
$\mathbf{u} = 0$ on $\mathcal{L}_g$

$u_x = 0$ on $\mathcal{L}_i$

c)

```
clf
clc


Lx = 10; %beam length [m]
Ly = 1; %beam height [m]
Nx = 60; %number of elements in x
Ny = 6; %number of elements in y
elemtype = 1; %1 for tri's
rhoG= 2e4; %rho*g
E = 100e9; % youngs modulus [Pa]
v = 0.3; % poissons ratio [-]
t = 2; %thickness [m]
hy = -1e6; % distributed load [N/m2]

%analysis type 1 = plane stress, 2 = plane strain, 3 = axisymmetry, 4 =
three dim
ptype = 1;


[Edof ,Dof, Coord, Ex, Ey, LeftSide_nodes, TopSide_nodes, RightSide_nodes,
BottomSide_nodes, TopRighty_node, h ] = RectangleMeshGen( Lx, Ly, Nx, Ny,
elemtype );

eldraw2(Ex,Ey,[1,2,0]);

bc1 = [Dof(LeftSide_nodes,1), zeros(size(LeftSide_nodes,1),1)];% Locks
translations in x on left side
bc2 = [Dof(LeftSide_nodes,2), zeros(size(LeftSide_nodes,1),1)]; % Locks
translations in y on left side
bc3 = [Dof(RightSide_nodes,1), zeros(size(LeftSide_nodes,1),1)]; % Locks
translations in x on right side

bc = [bc1;bc2;bc3];

nNodes = size(Dof,1); %Number of nodes
nElements = size(Edof,1); %Number of elements
nDofs = 2*nNodes; %number of dofs

K = zeros(nDofs); % defining the K-matrix
f = zeros(nDofs,1); %defining the f-matrix

eq =[0;-rhoG]; %defining eq (b)
```

```
ep = [ptype t]; %element properties vector
D = hooke(ptype, E,v); %  evaluating constitutive matrix

for i=1:nElements
      % calculating element stiffness matrix
      [Ke, fe] = plante(Ex(i,:), Ey(i,:), ep, D, eq);

   % assembling to global stiffness matrix
      [K,f] = assem(Edof(i,:), K, Ke, f, fe);
end

fl = zeros(nDofs,1);


% distance between top side nodes
topNodeDistance = Lx/Nx;

% adding UDL to load vector
fl(Dof(TopSide_nodes,2),1) = hy*t*topNodeDistance;

% adjusting contribution at corner nodes
for i=1:size(TopSide_nodes,1)
      n = TopSide_nodes(i);
      if(ismember(n, RightSide_nodes) || ismember(n, LeftSide_nodes))
            fl(Dof(n,2),1) = fl(Dof(n,2),1)/2;
end
end

% applying UDL to force vector
f = f+fl;

% solving equations
[a,Q]=solveq(K,f,bc);

% plotting displacements
Ed = extract(Edof,a);
eldisp2(Ex,Ey,Ed, [2,4,0],50)
xlabel('x [m]')
ylabel('y [m]')
```

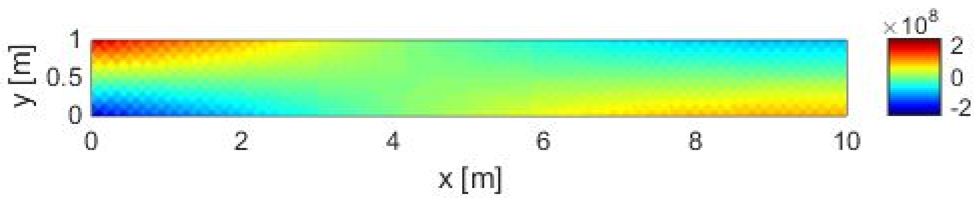


Original beam mesh in green and magnified deflected shape in red

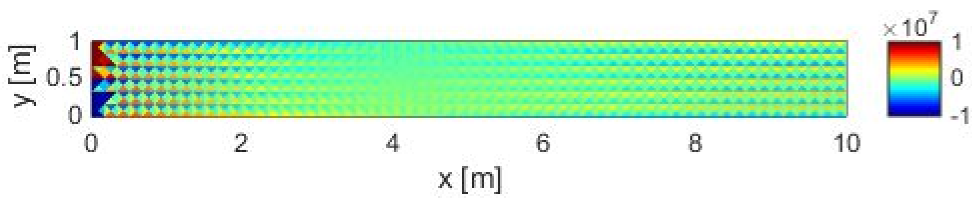
d)

```
%% d)
% calculating stresses
[Es, Et] = plants(Ex,Ey,ep,D,Ed);

%plotting sigma-x
figure(2)
fill(Ex',Ey',[Es(:,1) Es(:,1) Es(:,1)]');
axis equal tight
shading flat
colormap(jet);
colorbar();
xlabel('x [m]')
ylabel('y [m]')
caxis(1e8*[-2.5 2.5])
```

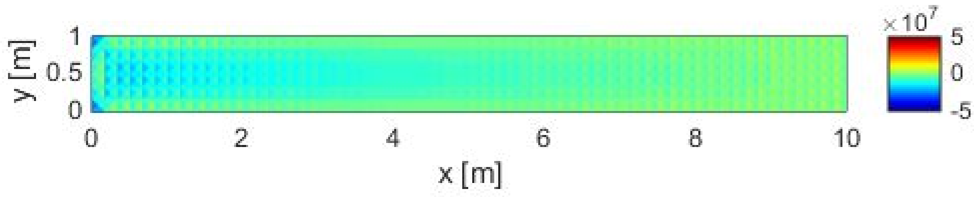
```
%Plotting sigma-y
figure(3)
fill(Ex',Ey',[Es(:,2) Es(:,2) Es(:,2)]');
axis equal tight
shading flat
colormap(jet);
colorbar();
xlabel('x [m]')
ylabel('y [m]')
caxis(1e7*[-1 1])

%plotting sigma-xy
figure(4)
fill(Ex',Ey',[Es(:,3) Es(:,3) Es(:,3)]');
axis equal tight
shading flat
colormap(jet);
colorbar();
xlabel('x [m]')
ylabel('y [m]')
caxis(1e7*[-5 5])
```



Plot showing stress in x-direction (sigma-x) [Pa]



Plot showing stress in y-direction (sigma-y) [Pa]



Plot showing stress in xy-direction (sigma-xy) [Pa]

The results look reasonable in general. In x-direction the results are similar to how the moment diagram would look by using the beam theory. The stresses in y-direction are mostly 0 which makes sense since the beam is free to deform in this direction.

The uneven pattern is a known FE problem for triangular meshes (not a problem for quads) and is due to numerical instability, by using more elements it looks better.

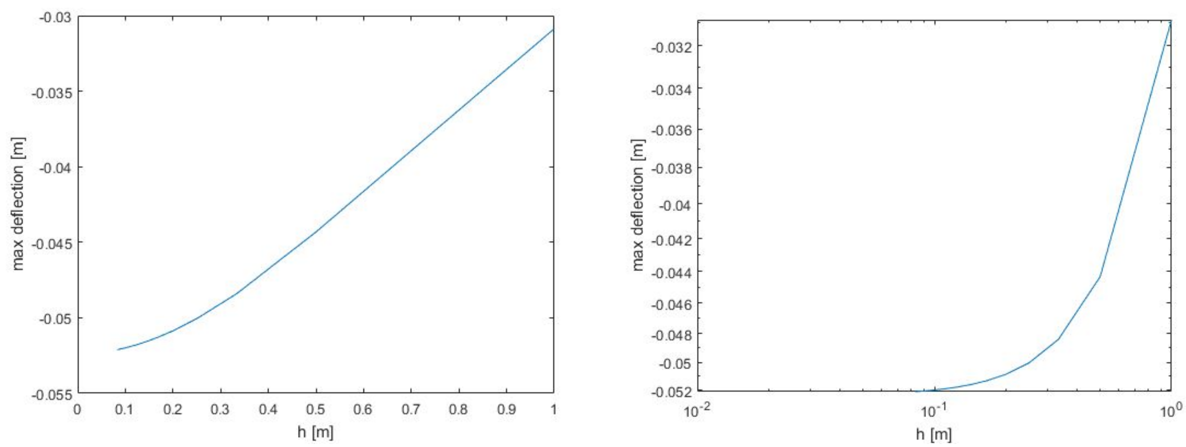
e)

See code below for main entry script:

```
%% e)
Ny = [1 2 3 4 5 6 7 8 9 10 11 12];

y = zeros(1,size(Ny,2));
h = zeros(1,size(Ny,2));

for i = 1:size(h,2)
      [y(i), h(i)] = BeamFunction(Ny(i)*10, Ny(i));
end

% plotting normal graph
figure(1)
plot(h,y)
xlabel('h [m]')
ylabel('max deflection [m]')

% plotting graph with logarithmic x-axis
figure(2)
loglog(h,y)
xlabel('h [m]')
```

```
ylabel('max deflection [m]')

[p, constant ] = ConvergenceRate(y,h)
```

The function **BeamFunction** is essentially task c) in function form. Actual code will be submitted in separate MATLAB-file.

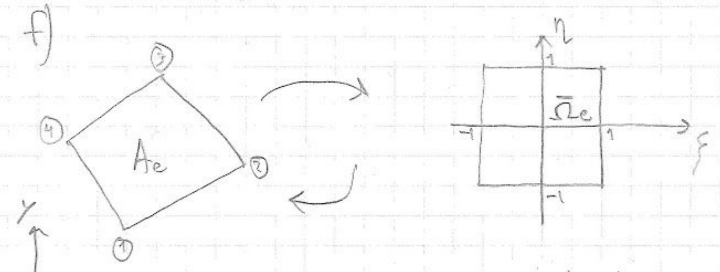


Plotted max deflections for varying h (element size) with linear x-scale (left) and logarithmic (right).

```
p = 2.3641
constant = 0.0385
```

f)

f) (1)



Global domain Parent domain

We describe x & y as functions of η & ξ

$$x = x(\xi, \eta)$$
$$y = y(\xi, \eta)$$

We derivate and get:

$$dx = \frac{dx}{d\xi} \cdot d\xi + \frac{dx}{d\eta} \cdot d\eta$$

$$dy = \frac{dy}{d\xi} \cdot dy + \frac{dy}{d\eta} \cdot d\eta$$

on matrix form:

$$\begin{bmatrix} dx \\ dy \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{dx}{d\xi} & \frac{dx}{d\eta} \\ \frac{dy}{d\xi} & \frac{dy}{d\xi} \end{bmatrix}}_{J - \text{Jacobian matrix}} \begin{bmatrix} d\xi \\ d\eta \end{bmatrix}$$

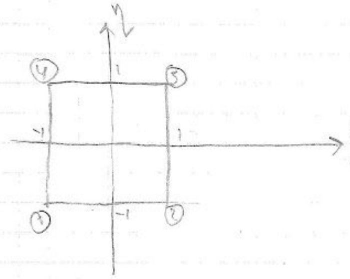
J must be invertible for a unique mapping. $\begin{bmatrix} d\xi \\ d\eta \end{bmatrix} = J^{-1} \begin{bmatrix} dx \\ dy \end{bmatrix}$

$\det(J) \neq 0$. We choose $\det(J) > 0$

It can be shown:

$$\int_{A_e} (\ldots)\, dA = \int_{\bar{\Omega}_e} (\ldots) \det(J)\, d\bar{\Omega} = \int_{-1}^{1}\int_{-1}^{1} (\ldots) \det(J)\, d\xi\, d\eta$$

For a quadrilateral mesh:



$$\bar{N}_1^e = \frac{1}{4}(\xi-1)(\eta-1) \qquad \bar{N}_3^e = \frac{1}{4}(\xi+1)(\eta+1)$$

$$\bar{N}_2^e = -\frac{1}{4}(\xi+1)(\eta-1) \qquad \bar{N}_4^e \; -\frac{1}{4}(\xi-1)(\eta+1)$$

$$x = \bar{\mathbb{N}}\mathbb{x}^e \qquad \bar{\mathbb{N}}^e = \begin{bmatrix} \bar{N}_1^e & 0 & \dots & \bar{N}_4 & 0 \\ 0 & \bar{N}_1^e & \dots & 0 & \bar{N}_4 \end{bmatrix}$$
$$y = \bar{\mathbb{N}}\mathbb{y}^e$$

We evaluate $\mathbb{J}$:

$$\begin{cases} \dfrac{dx}{d\xi} = \dfrac{d\bar{\mathbb{N}}_e}{d\xi}\cdot \mathbb{x}^e & \dfrac{dx}{d\eta} = \dfrac{d\bar{\mathbb{N}}_e}{d\eta}\cdot\mathbb{x}^e \\[2ex] \dfrac{dy}{d\xi} = \dfrac{d\bar{\mathbb{N}}_e}{d\xi}\cdot \mathbb{y}^e & \dfrac{dy}{d\eta} = \dfrac{d\bar{\mathbb{N}}_e}{d\eta}\cdot\mathbb{y}^e \end{cases}$$

We derivate $\bar{\mathbb{N}}$ with respect to ξ & η

$$\frac{d\bar{\mathbb{N}}_e}{d\xi} = \frac{d\bar{\mathbb{N}}_e}{dx}\cdot\frac{dx}{d\xi} + \frac{d\bar{\mathbb{N}}_e}{dy}\cdot\frac{dy}{d\xi}$$

$$\frac{d\bar{\mathbb{N}}_e}{d\eta} = \frac{d\bar{\mathbb{N}}_e}{dx}\cdot\frac{dx}{d\eta} + \frac{d\bar{\mathbb{N}}_e}{dy}\cdot\frac{dy}{d\eta}$$

In matrix form it can be written:

$$\begin{bmatrix} \dfrac{d\bar{\mathbb{N}}_e}{d\xi} \\[2ex] \dfrac{d\bar{\mathbb{N}}_e}{d\eta} \end{bmatrix} = \underbrace{\begin{bmatrix} \dfrac{dx}{d\xi} & \dfrac{dy}{d\xi} \\[2ex] \dfrac{dx}{d\eta} & \dfrac{dy}{d\eta} \end{bmatrix}}_{\mathbb{J}^T} \begin{bmatrix} \dfrac{d\bar{\mathbb{N}}_e}{dx} \\[2ex] \dfrac{d\bar{\mathbb{N}}_e}{dy} \end{bmatrix} \Rightarrow \begin{bmatrix} \dfrac{d\bar{\mathbb{N}}_e}{dx} \\[2ex] \dfrac{d\bar{\mathbb{N}}_e}{dy} \end{bmatrix} = \left(\mathbb{J}^T\right)^{-1} \begin{bmatrix} \dfrac{d\bar{\mathbb{N}}_e}{d\xi} \\[2ex] \dfrac{d\bar{\mathbb{N}}_e}{d\eta} \end{bmatrix}$$

$$\mathbb{B}^e = \begin{bmatrix} \frac{\partial \mathbb{N}^e}{\partial x} \\ \frac{\partial \mathbb{N}^e}{\partial y} \end{bmatrix}$$

$$\bar{\mathbb{B}}_e = \begin{bmatrix} \frac{\partial N_1^e}{\partial x} & 0 & \frac{\partial N_2^e}{\partial x} & 0 & \frac{\partial N_3^e}{\partial x} & 0 & \frac{\partial N_4^e}{\partial x} & 0 \\ 0 & \frac{\partial N_1^e}{\partial y} & 0 & \frac{\partial N_2^e}{\partial x} & 0 & \frac{\partial N_3^e}{\partial y} & 0 & \frac{\partial N_4^e}{\partial y} \\ \frac{\partial N_1^e}{\partial y} & \frac{\partial N_1^e}{\partial x} & \frac{\partial N_2^e}{\partial y} & \frac{\partial N_2^e}{\partial x} & \frac{\partial N_3^e}{\partial y} & \frac{\partial N_3^e}{\partial x} & \frac{\partial N_4^e}{\partial y} & \frac{\partial N_4^e}{\partial x} \end{bmatrix}$$

$$\mathbb{K}^e = \int_{A^e} \mathbb{B}^{eT} \mathbb{D} \mathbb{B}^e t \, dA \qquad \mathbb{f}_\Omega^e = \int_{A^e} \mathbb{N}^{eT} \mathbb{Q} t \, dA$$

We use numerical integration to compute $\int_{-1}^{1} f(\xi) d\xi$ by evaluating the function $f(\xi)$ in certain points, ξ_i, and multiply by weights H_i

$$\int_{-1}^{1}\int_{-1}^{1} f(\xi,\eta) d\xi \, d\eta \approx \sum_{i=1}^{n}\sum_{j=1}^{m} f(\xi_i, \eta_j) H_i H_j \qquad \text{we choose } m=n$$

$$\mathbb{K}^e = \int_{-1}^{1}\int_{-1}^{1} \bar{\mathbb{B}}^{eT} \bar{\mathbb{D}} \bar{\mathbb{B}}^e t \det(\mathbb{J}) \, d\xi \, d\eta = \sum_{i=1}^{n}\sum_{j=1}^{m} \bar{\mathbb{B}}^{eT} \bar{\mathbb{D}} \bar{\mathbb{B}}^e t \det(\mathbb{J}) H_i H_j$$

$$\mathbb{f}_\Omega^e = \int_{-1}^{1}\int_{-1}^{1} \bar{\mathbb{N}}^{eT} \mathbb{Q} t \, d\xi \, d\eta = \sum_{i=1}^{n}\sum_{j=1}^{m} \bar{\mathbb{N}}^{eT} \mathbb{Q} t H_i H_j \qquad \text{where } \mathbb{Q} = \begin{bmatrix} 0 \\ -\rho g \end{bmatrix}$$

H_i & H_j are chosen from a table based on m & n.

<u>For plane stress</u>

$$\mathbb{D} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}$$

<u>For plane strain</u>

$$\mathbb{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}$$

See below for plan4bilin.m code:

```
function [ Ke, fe ] = plan4bilin( ex, ey, ep, eq )

% Pick out parameters from ep to facilitate readability of the code
ptype = ep(1);
t     = ep(2);
ngp   = ep(3); % NoGaussPoits per direction
Emod  = ep(4);
ny    = ep(5);

%  Initialize Ke and fe with zeros for all of their elements
Ke = zeros(2*size(ex,2));
fe = zeros(2*size(ex,2),1);


%  Tolerance for the Jacobian determinant
minDetJ = 1.e-16;


% Determine constitutive matrix D for plane strain or plane stress
if ptype == 1
 D=(Emod/(1-ny^2))*[1 ny 0; ny 1 0; 0 0 (1-ny)/2];
else
 D=(Emod/((1+ny)*(1-2*ny)))*[1-ny ny 0;ny 1-ny 0; 0 0 (1-2*ny)/2];
end

%  Set the appropriate Gauss integration scheme for 1, 2 or 3 Gauss points
%  in each direction (ksi, eta). (or else 1, 4 or 9 Gauss points in total)
if ngp == 1
 intWeight   = 2;
 GaussPoints = 0;

elseif ngp == 2
 pos = 1/sqrt(3);
 intWeight = [1 1];
 GaussPoints = [-pos pos];

elseif ngp == 3
 pos = sqrt(3/5);
 w1 =8/9;
 w2 =  5/9;
 intWeight = [w2 w1 w2];
 GaussPoints = [-pos 0 pos];

else
 error('Only 1,2 or 3 Gauss Points in each direction apply)'
end
```

```
%  Loop over all integration points to compute Ke and fe
for i = 1:ngp
      xsi = GaussPoints(i);
      weightXsi = intWeight(i);

      for j = 1:ngp
            eta   = GaussPoints(j);
            weightEta = intWeight(j);

            % Compute the element shape functions Ne (use xsi and eta from
              above)
            Ne1 = (1/4)*(xsi-1)*(eta-1);
            Ne2 = (-1/4)*(xsi+1)*(eta-1);
            Ne3 = (1/4)*(xsi+1)*(eta+1);
            Ne4 = (-1/4)*(xsi-1)*(eta+1);

            % Compute derivatives (with respect to xsi and eta) of the
            % shape functions at coordinate (xsi,eta). Since the element is
            % isoparametic, these are also the derivatives of the basic
              functions
            dXsi = (1/4)*[eta-1 -(eta-1) eta+1 -(eta+1)];
            dEta = (1/4)*[xsi-1 -(xsi+1) xsi+1 -(xsi-1)];

            %  Use shape function derivatives and element vertex coordinates
            %  to establish the Jacobian matrix.
            J = [dXsi*ex' dEta*ex'; dXsi*ey' dEta*ey'];

            %  Compute the determinant of the Jacobian and check that it is
               OK
            detJ = det(J);

            if ( detJ < minDetJ )
                  fprintf( 1, 'Bad element geometry in function plan4bilin:
                              detJ = %0.5g\n', detJ );
      return;
            end

            % Determinant seems OK - invert the transpose of the Jacobian
            Jinv = inv(J');

            %   Compute derivatives with respect to x and y, of all basis
            functions,
            dNxy = Jinv*[dXsi;dEta];

            % Use the derivatives of the shape functions to compute the element
            % B-matrix, Be
            Be = [dNxy(1,1) 0 dNxy(1,2) 0 dNxy(1,3) 0 dNxy(1,4) 0
```

```
             0 dNxy(2,1) 0 dNxy(2,2) 0 dNxy(2,3) 0 dNxy(2,4)
       dNxy(2,1) dNxy(1,1) dNxy(2,2) dNxy(1,2) dNxy(2,3) ...
                   dNxy(1,3) dNxy(2,4) dNxy(1,4)];

       Ne = [Ne1 0 Ne2 0 Ne3 0 Ne4 0; 0 Ne1 0 Ne2 0 Ne3 0 Ne4];

       % Compute the contribution to element stiffness and load matrix
       % from current Gauss points
       % (check for plane strain or plane stress again!)

       Ke = Ke + Be'*D*Be*t*detJ*weightXsi*weightEta;
       fe = fe + Ne'*eq*t*detJ*weightXsi*weightEta;  <

   end
end
%--------------------------end--------------------------------
```

See code below for plan4bilinStress.m:

```
function [ stress, strain ] = plan4bilinStress( ex, ey, ep, u_e )
% Pick out parameters from ep to facilitate readability of the code


ptype = ep(1);
Emod  = ep(2);
ny    = ep(3);
minDetJ = 1.e-16;

% Determine constitutive matrix D for plane strain or plane stress
if ptype==1
      D=(Emod/(1-ny^2))*[1 ny 0; ny 1 0; 0 0 (1-ny)/2];
else
      D=(Emod/((1+ny)*(1-2*ny)))*[1-ny ny 0;ny 1-ny 0; 0 0 (1-2*ny)/2];
end

% For 1 gauss point the coordinates (xsi, eta) in the parent domain are:
xsi = 0;
eta = 0;

% Compute the derivatives (with respect to xsi and eta) of the
```

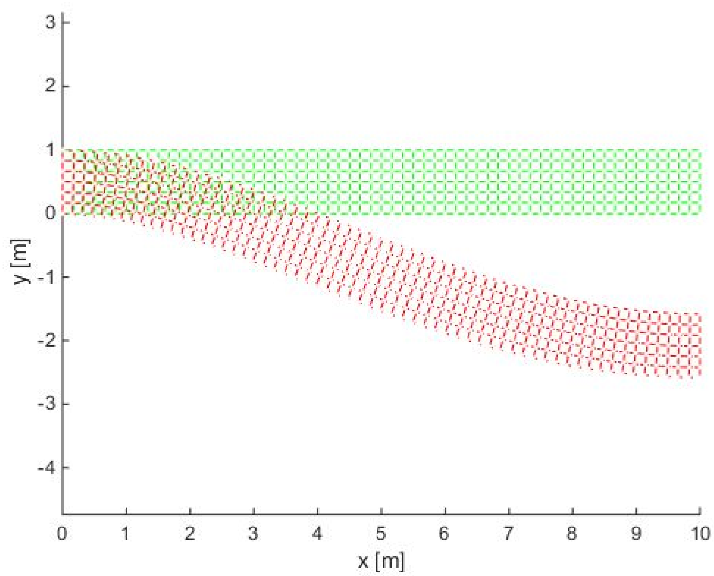
```
% shape functions at coordinate (xsi,eta).
Ne1 = (1/4)*(xsi-1)*(eta-1);
Ne2 = (-1/4)*(xsi+1)*(eta-1);
Ne3 = (1/4)*(xsi+1)*(eta+1);
Ne4 = (-1/4)*(xsi-1)*(eta+1);

dXsi = (1/4)*[eta-1 -(eta-1) eta+1 -(eta+1)];
dEta = (1/4)*[xsi-1 -(xsi+1) xsi+1 -(xsi-1)];

% Compute Jacobian matrix and invert the transpose of the Jacobian
J = [dXsi*ex' dEta*ex'; dXsi*ey' dEta*ey'];

%  Compute the determinant of the Jacobian and check that it is OK
detJ = det(J);

if ( detJ < minDetJ )
      fprintf( 1, 'Bad element geometry in function plan4bilin: detJ =
%0.5g\n', detJ );
      return;
end

% Determinant seems OK - invert the transpose of the Jacobian
Jinv = inv(J');

% Compute derivatives with respect to x and y, of all basis functions
dNxy = Jinv*[dXsi;dEta];

% Use the derivatives of the shape functions to compute the element
% B-matrix, Be
Be = [dNxy(1,1) 0 dNxy(1,2) 0 dNxy(1,3) 0 dNxy(1,4) 0
      0 dNxy(2,1) 0 dNxy(2,2) 0 dNxy(2,3) 0 dNxy(2,4)
      dNxy(2,1) dNxy(1,1) dNxy(2,2) dNxy(1,2) dNxy(2,3) dNxy(1,3) ...
            dNxy(2,4) dNxy(1,4)];

% Compute the strain and store it in strain
strain = Be*u_e;

% Compute the stress and store it in stress
stress = D*strain;
end
%--------------------------end--------------------------------
```

g)

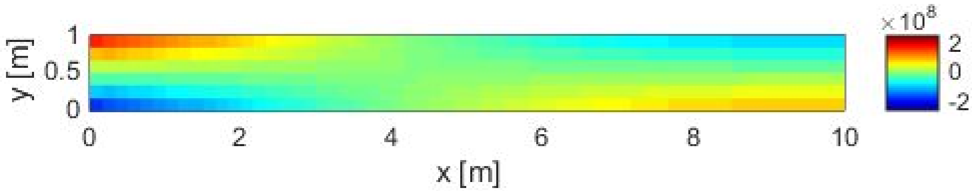
```
%% f)
% setting mesh resolution
Nx = 60;
Ny = 6;

elemtype= 2; %2 for quads
ngp = 2; % number of gauss points

[Edof ,Dof, Coord, Ex, Ey, LeftSide_nodes, TopSide_nodes, RightSide_nodes,
BottomSide_nodes, TopRighty_node, h ] = RectangleMeshGen( Lx, Ly, Nx, Ny,
elemtype );

% plotting mesh
eldraw2(Ex,Ey,[1,2,0]);

bc1 = [Dof(LeftSide_nodes,1), zeros(size(LeftSide_nodes,1),1)];% Locks
translations in x on left side
bc2 = [Dof(LeftSide_nodes,2), zeros(size(LeftSide_nodes,1),1)]; % Locks
translations in y on left side
bc3 = [Dof(RightSide_nodes,1), zeros(size(LeftSide_nodes,1),1)]; % Locks
translations in y on right side

bc = [bc1;bc2;bc3];

nNodes = size(Dof,1); %Number of nodes
nElements = size(Edof,1); %Number of elements
nDofs = 2*nNodes; %number of dofs

K = zeros(nDofs); % defining the K-matrix
f = zeros(nDofs,1); %defining the f-matrix

%%updating element properties to fit our function
ep = [ptype t ngp E v];

for i=1:nElements
      % looping through elements and assembling K & f contributions.

   [Ke, fe] = plan4bilin(Ex(i,:), Ey(i,:), ep, eq);
   [K,f] = assem(Edof(i,:), K, Ke, f, fe);
end

fl = zeros(nDofs,1);

topNodeDistance = Lx/Nx;
```

```
% adding UDL to load vector
fl(Dof(TopSide_nodes,2),1) = hy*t*topNodeDistance;

%finds corner nodes and halves their load vector
% (if node is both a topside node and a rightside/leftside node => corner
% node
for i=1:size(TopSide_nodes,1)

      n = TopSide_nodes(i);

      if(ismember(n, RightSide_nodes) || ismember(n, LeftSide_nodes))
            fl(Dof(n,2),1) = fl(Dof(n,2),1)/2;
      end
end

f = f+fl;

%solving equations
[a,Q]=solveq(K,f,bc);

%extracting and plotting displacements
Ed = extract(Edof,a);
eldisp2(Ex,Ey,Ed, [2,4,0],50)
xlabel('x [m]')
ylabel('y [m]')

%defining stress and strain vectors
Es = zeros(nElements,3);
Et = zeros(nElements,3);

ep = [ptype E v];

%looping through elements and extracting stress and strain
for i = 1:nElements
      [stress, strain] = plan4bilinStress(Ex(i,:), Ey(i,:),ep, Ed(i,:)');

      Es(i,:) = stress;
      Et(i,:) = strain;
end

%plotting sigma-x
figure(2)
fill(Ex',Ey',[Es(:,1) Es(:,1) Es(:,1) Es(:,1)]');
axis equal tight
shading flat
colormap(jet);
colorbar();
xlabel('x [m]')
```

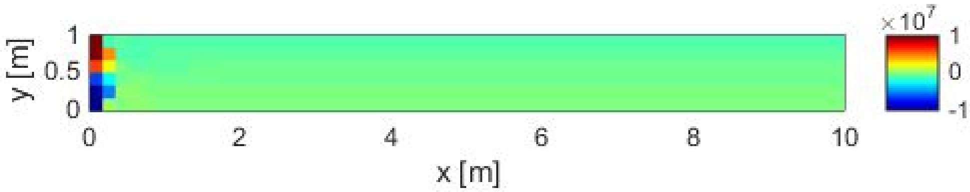
```
ylabel('y [m]')
caxis(1e8*[-2.5 2.5])


%plotting sigma-y
figure(3)
fill(Ex',Ey',[Es(:,2) Es(:,2) Es(:,2) Es(:,2)]');
axis equal tight
shading flat
colormap(jet);
colorbar();
xlabel('x [m]')
ylabel('y [m]')
caxis(1e7*[-1 1])

%plotting sigma-xy
figure(4)
fill(Ex',Ey',[Es(:,3) Es(:,3) Es(:,3) Es(:,3)]');
axis equal tight
shading flat
colormap(jet);
colorbar();
xlabel('x [m]')
ylabel('y [m]')
caxis(1e7*[-5 5])
```



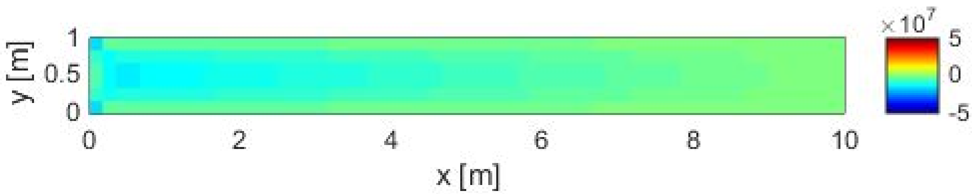
Original beam mesh in green and magnified deflected shape in red



Plot showing stress in x-direction (sigma-x) [Pa]



Plot showing stress in y-direction (sigma-y) [Pa]



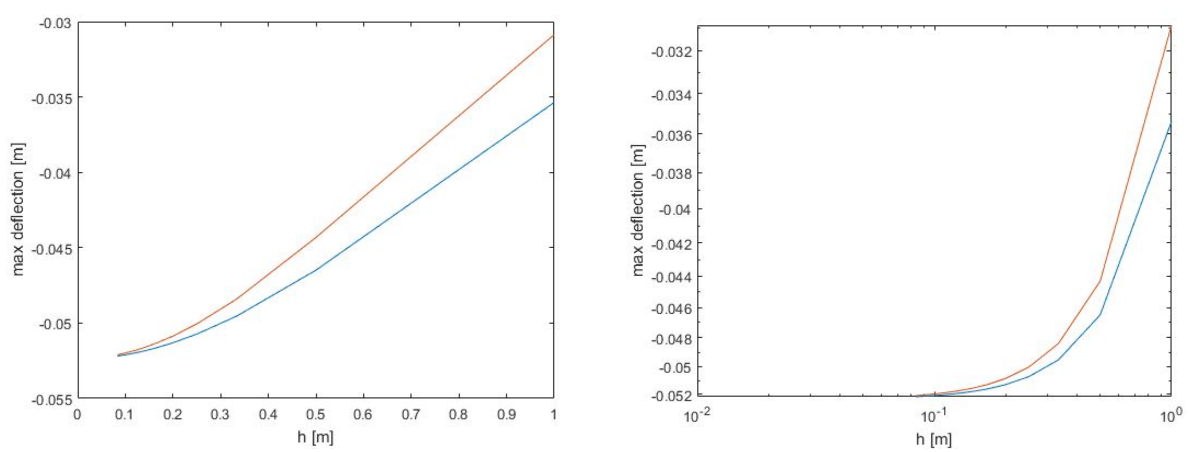
Plot showing stress in xy-direction (sigma-xy) [Pa]

See code below for convergence rate:

```
%% convergence rate

Ny = [1 2 3 4 5 6 7 8 9 10 11 12];

%Empty matrices for quads
yQ = zeros(1,size(Ny,2));
hQ = zeros(1,size(Ny,2));

%Empty matrices for tris
yT = zeros(1,size(Ny,2));
hT = zeros(1,size(Ny,2));

%loop through Ny values and add results in matrices
for i = 1:size(h,2)
[yQ(i), hQ(i)] = BeamFunction2(Ny(i)*10, Ny(i));
[yT(i), hT(i)] = BeamFunction(Ny(i)*10, Ny(i));
end
```

```
% plotting normal graph
figure(1)
plot(hQ,yQ)
hold on
plot(hT,yT)
xlabel('h [m]')
ylabel('max deflection [m]')


% plotting graph with logarithmic x-axis
figure(2)
loglog(hQ,yQ)
hold on
loglog(hT,yT)
xlabel('h [m]')
ylabel('max deflection [m]')


[p, constant ] = ConvergenceRate(y,h)
```

The functions **BeamFunction** and **BeamFunction2** are essentially task c) and g) in function form. Actual code for these will be submitted in separate MATLAB-files.



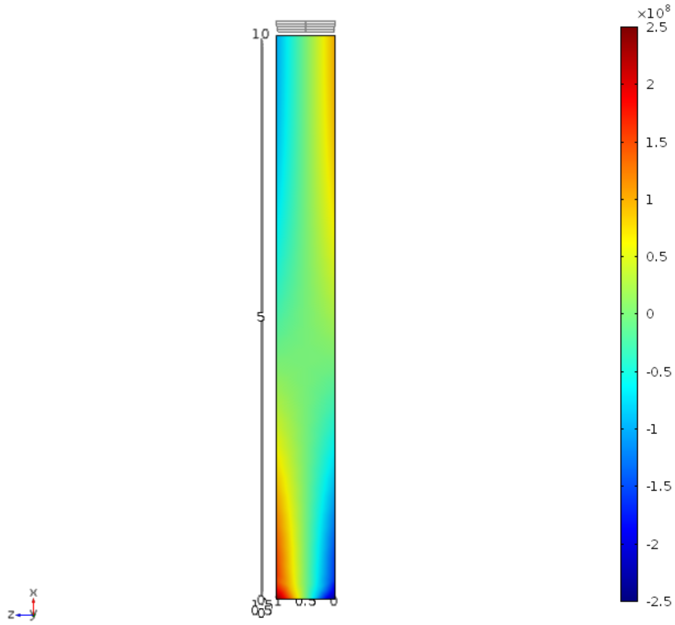
Plotted max deflections for varying h (element size) with linear x-scale (left) and logarithmic (right). Red curves represent values using triangular elements, while blue represent values using bilinear quad elements.

```
p = 2.4028
constant = 0.0294
```

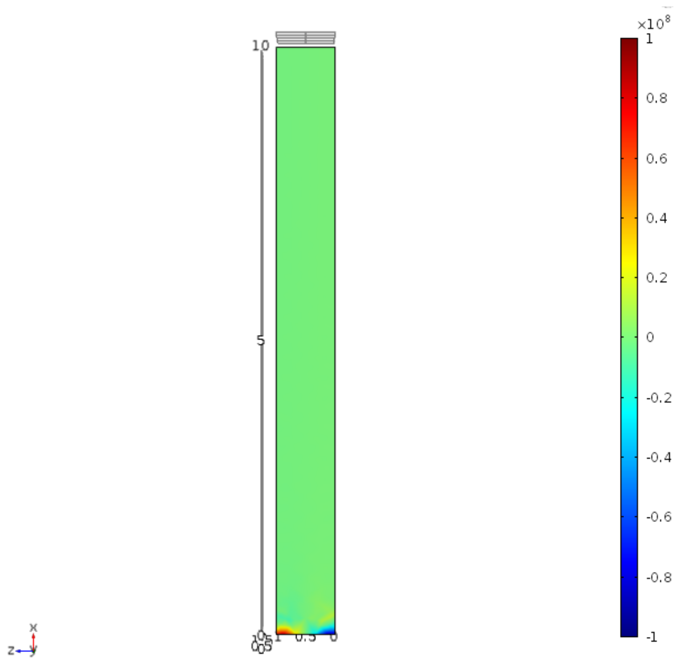
From the graph, it is evident that quads (in blue) converge slightly earlier than triangular elements (in red). This is also obvious from the convergence rate values, where a large p and smaller constant will result in smaller errors.

Task 2

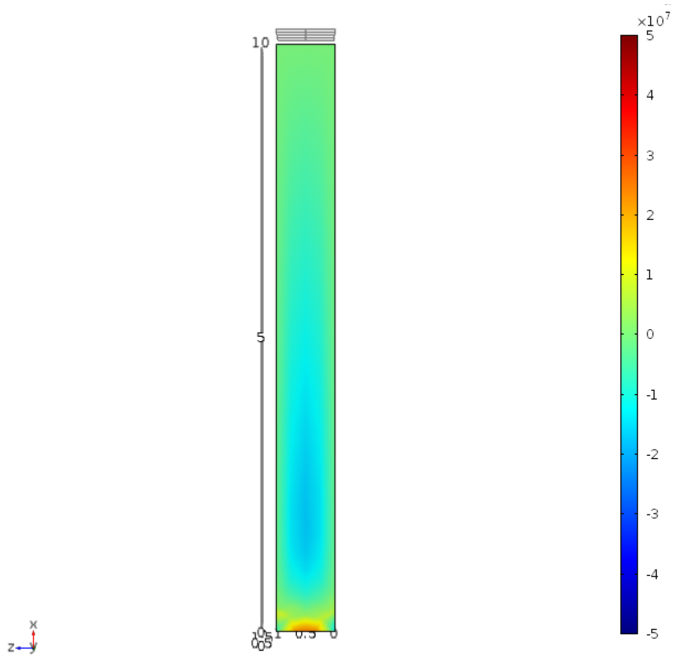
Outer surface: Stress tensor, Gauss-point evaluation x-component(Pa):



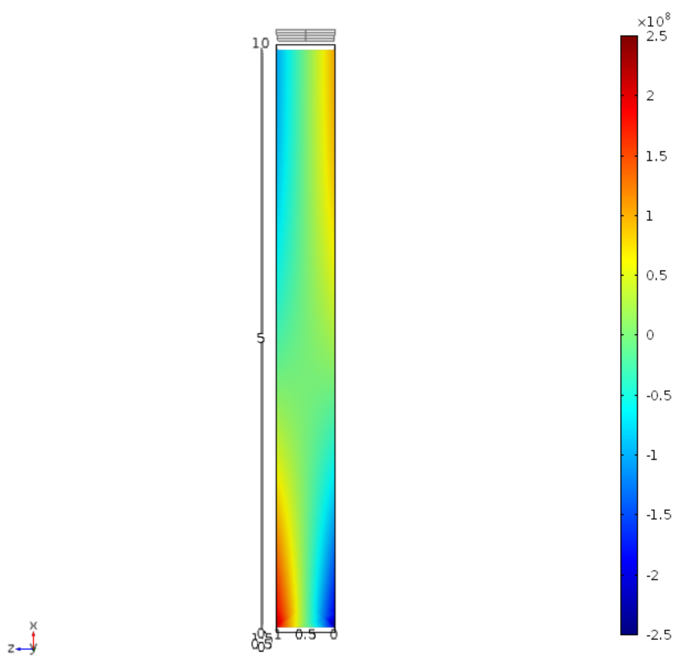
Outer surface: Stress tensor, Gauss-point evaluation y-component (z-direction in figure)(Pa):



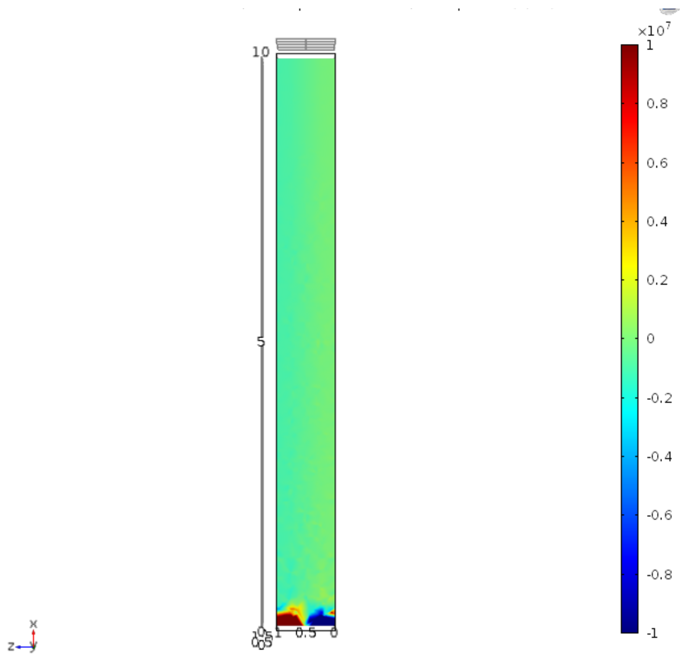
Outer surface: Stress tensor, Gauss-point evaluation xy component(xz-direction in figure) (Pa):



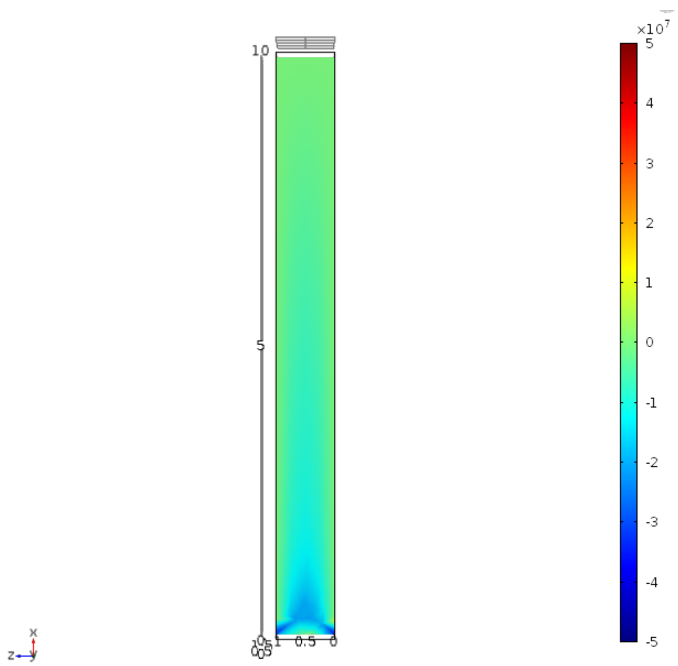
Mid slice: Stress tensor, Gauss-point evaluation x component(Pa):



Mid slice: Stress tensor, Gauss-point evaluation y (z-direction in figure) component(Pa):



Mid slice: Stress tensor, Gauss-point evaluation xy (xz-direction in figure) component(Pa):



Maximum y-displacement at the outer surface with COMSOL : -0.051845 m
Maximum y-displacement with triangular mesh in matlab: -0.0514 m
Maximum y-displacement with triangular mesh in matlab: -0.0517 m.

In regards to displacement in y-direction the COMSOL results are very similar to our MATLAB implementation. Our 2D assumptions are reasonable compared with the outer surface of the beam, if comparing y-direction displacement.

In regards to stress, the sliced results show similar patterns and values compared to the MATLAB implementation. However, values on the surface show a significant increase. Considering the rather large thickness of the beam (t=2m), perhaps using plate elements is not suitable for an accurate representation.