CHALMERS
UNIVERSITY OF TECHNOLOGY

Computer Assignment 1b

TME245 - Finite Element Method: Structures

Academic year 2024/2025, SP3

Group number: 13

Brijesh Niranjan, brijeshn@chalmers.se

David Rasuli, rasulid@chalmers.se

February 13, 2025

Chalmers University of Technology
Department of Industrial and Materials Science
Division of Material and Computational Mechanics
Gothenburg, Sweden

Contents

1

Task 1 - Matlab

In this computer assignment, an analysis of a bimetal strip consisting of steel and aluminium was conducted. The problem was divided into four subtasks, namely task 1a to 1d, to prepare for the ultimate goal of performing an FE analysis of the bimetal strip, which deforms elastically as well as plastically due to a drastic change of the ambient temperature. Task 1 was completely implemented in MATLAB.
Each subtask will be further explained in its respective subsection. For task 1a until 1c, a 2D square consisting of two triangular elements is regarded. This serves the purpose to teach the student a non-linear finite element analysis step-by-step. Task 1d is the final task, in which the analysis of the bimetal strip is conducted, using the knowledge and the functions from the previous exercises.

1.1 Task 1a - Implementing the non-linear programme

In this task, the element routine from CE1 was modified to contain the Newton algorithm in order to solve a linear elastic problem using constant strain triangles (CST). Furthermore, a simplified 2D square problem, discretised with two triangle elements, is regarded as seen on Fig. 1.1.
The mesh consists of a total of four nodes, one in every corner of the square. Each of the nodes have two degrees of freedom, which result in a total of eight degrees of freedom for this FE problem.

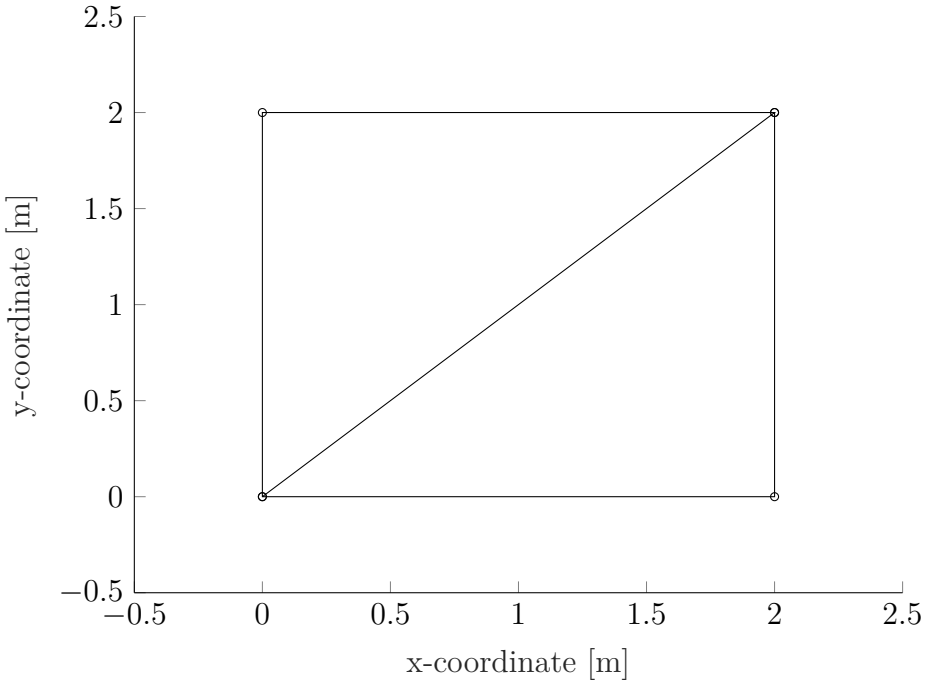

Figure 1.1: FE mesh and geometry used in task 1a.

The material and geometric parameters are defined in Table 1.1.

Property	**Unit**	**Value**
Length (x)	m	2
Height (y)	m	2
Depth (z)	m	1
Young's modulus	GPa	210
Poisson's ratio	-	0.3

Table 1.1: Material and geometric properties for task 1a.

The square was fixed on the left side which imposes Dirichlet boundary conditions in x- and y-direction for both nodes on the left side. Additionally, a stepwise displacement control was introduced to the top right node in negative y-direction, starting with a displacement of $a = 0mm$ and increasing linearly to the maximum displacement of $a = 1mm$. The resulting internal force at the same DoF is shown against the stepwise increase of the displacement in Fig. 1.2.

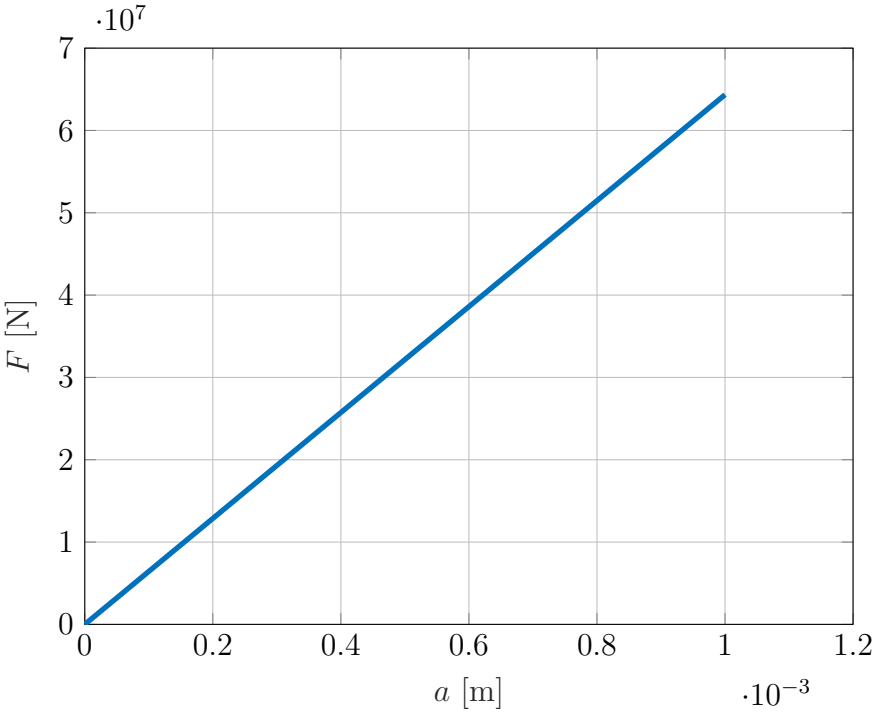

Figure 1.2: Internal force at top right node in y-direction with respect to the current displacement.

The maximum internal force is reached at the maximum displacement of $a = 1mm$, being $F_{int} = 64.34 \cdot 10^6 N$. In addition to that, the Newton iteration converges in a maximum of two steps as can be seen in Fig. 1.3.

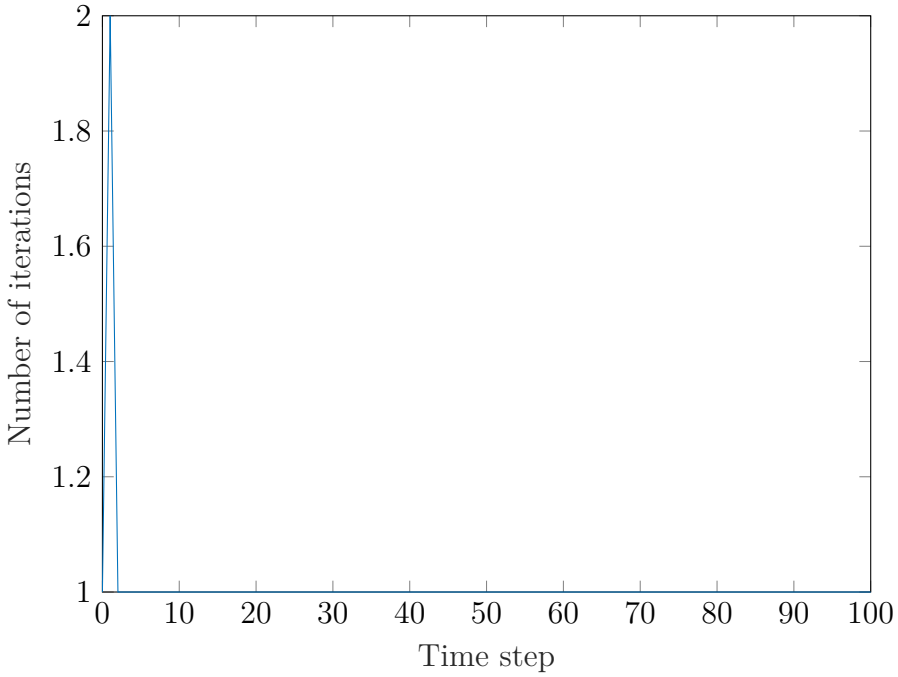

Figure 1.3: Number of iterations needed at each time step.

The solution was verified using the analytical solution for a cantilever beam. The maximum displacement of a cantilever beam is calculated as follows [GF07]

$$a_{max} = \frac{Pl^3}{3EI_y} \tag{1.1}$$

where P is the force and the area moment of inertia I_y is calculated as

$$I_y = \frac{1}{12}bh^3. \tag{1.2}$$

Using these equations, it is possible to determine the force for a displacement of $a = 1mm$ analytically

$$P = 52.5 \cdot 10^6 N. \tag{1.3}$$

The results are reasonable, due to an error of about 20%. This discrepancy is likely to result from the stiff nature of CST elements and the coarse meshing of the geometry. It is expected to become more accurate with a finer mesh or by using triangular elements with quadratic shape functions.

1.2 Task 1b - Quadratic isoparametric triangular elements

In this task, the previous code had to be adjusted, to not use CST elements anymore, but instead work with quadratic triangular elements, which possess six nodes instead of three. In addition to that, it is expected, that the new element routine includes the isoparametric mapping of the new element type and the numerical integration of said elements.
The goal is to create a function, which returns the element internal force vector $\mathbf{f_{e,int}}$ and the tangent stiffness matrix $\mathbf{K_e}$ of the element.
In the final step, the element routine has to be tested using a given test case and and implemented for the previous problem in task 1a.

Since the shape functions of the new elements now result in a non-constant $\mathbf{B}$-matrix, it has to be integrated with regards to the area of the element. This is done numerically using gaussian quadrature. For this element type, it is sufficient to take three gauss points and multiply them with the respective weights. The gauss points are

$$\underline{IP} = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{1}{6} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{6} \end{bmatrix} \tag{1.4}$$

with their respective weights

$$\underline{w} = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix}. \tag{1.5}$$

In the code, a routine was implemented which pre-calculates the $\mathbf{B}$ matrix and the $\mathbf{F}_{isop}$ matrix with respect to the three gauss points. This approach returns three pre-defined matrices for $\mathbf{B}$ and $\mathbf{F}_{isop}$, which are saved as three dimensional matrices using an external function file. This approach avoids the usage of symbolic operations and saves the matrices externally. For further information, refer to the code snippet A.2.1 in the appendix.

To check the code, the test case was implemented and successfully calculated. The expected result provided by the exercise sheet is the internal force vector with the following entries

$$\mathbf{f}_{e,int} = \begin{bmatrix} 1.5860 \cdot 10^{-13} \\ -77.856 \\ -6.8669 \cdot 10^{-13} \\ 77.856 \\ -2.9447 \cdot 10^{-13} \\ -7.9494 \cdot 10^{-16} \\ -1.6032 \cdot 10^{-14} \\ -8.8818 \cdot 10^{-14} \\ -8.9864 \cdot 10^{-13} \\ 311.42 \\ 1.7372 \cdot 10^{-12} \\ -311.42 \end{bmatrix} N. \tag{1.6}$$

The new element routine calculated an internal force vector with the values

$$\mathbf{f}_{e,int} = \begin{bmatrix} 1.4211 \cdot 10^{-14} \\ -77.8556 \\ -1.0034 \cdot 10^{-14} \\ 77.8556 \\ 3.5527 \cdot 10^{-15} \\ -1.9409 \cdot 10^{-14} \\ 1.4211 \cdot 10^{-14} \\ 2.1316 \cdot 10^{-13} \\ 2.8422 \cdot 10^{-14} \\ 311.4222 \\ -3.5527 \cdot 10^{-14} \\ -311.4222 \end{bmatrix} N \tag{1.7}$$

where the differences can be easily explained by the numerical inaccuracies of the algorithm. This proves, that the element routine is indeed functional and determines the internal forces correctly.

Back to the previous example at hand, all geometric and material properties stay the same as in task 1a and Table 1.1. Since the problem is now defined using quadratic triangular elements, five additional nodes and therefore 10 additional degrees of freedom are introduced, increasing the number of DoFs to 18. The new mesh is depicted in Fig. 1.4.

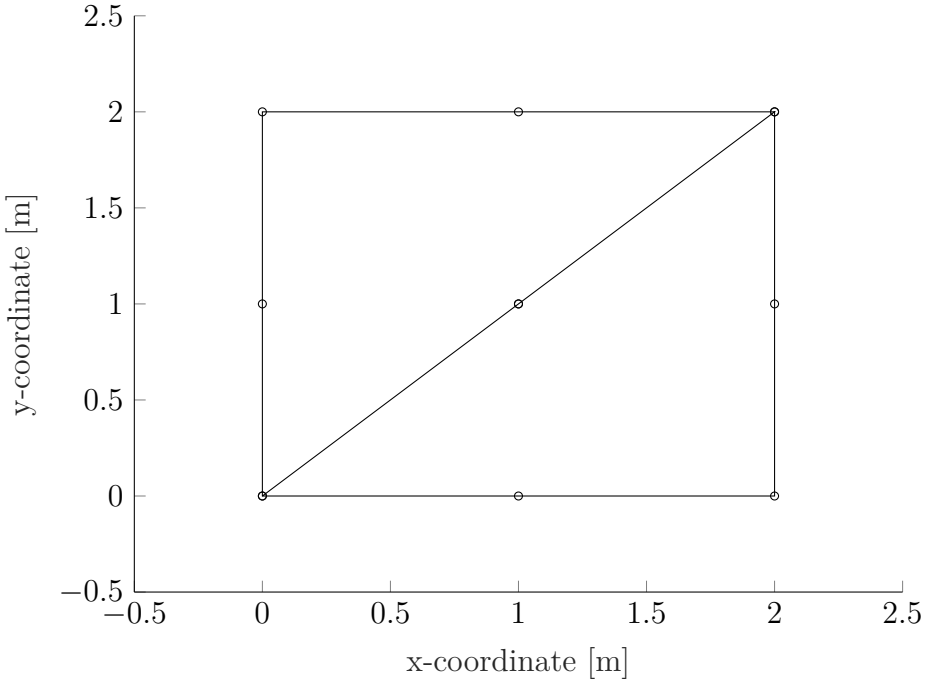

Figure 1.4: FE mesh and geometry used in task 1b.

The square is yet again fixed at the left side, constraining the three nodes on the left in x- and y-direction and a stepwise linear displacement control from $a = 0mm$ until $a = 1mm$ is introduced yet again.

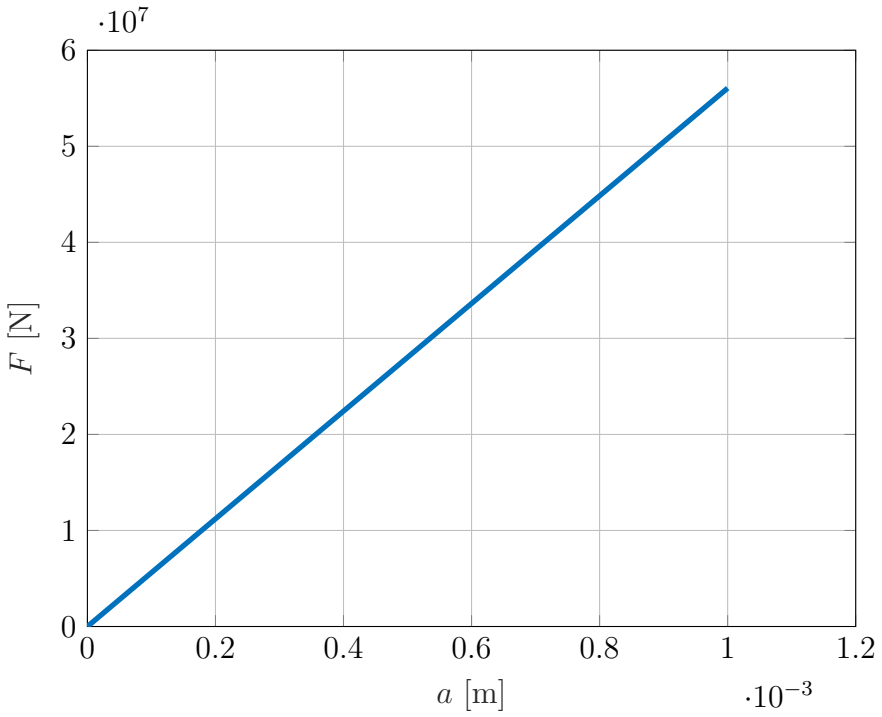

Figure 1.5: Internal force at top right node in y-direction with respect to the current displacement

The maximum force can be seen in Fig. 1.5 being $F_{int} = 56.0562 \cdot 10^6 N$ at a displacement of $a = 1mm$. This solution is much closer to the analytical solution of $P = 52.5 \cdot 10^6 N$ from Eq. 1.3.

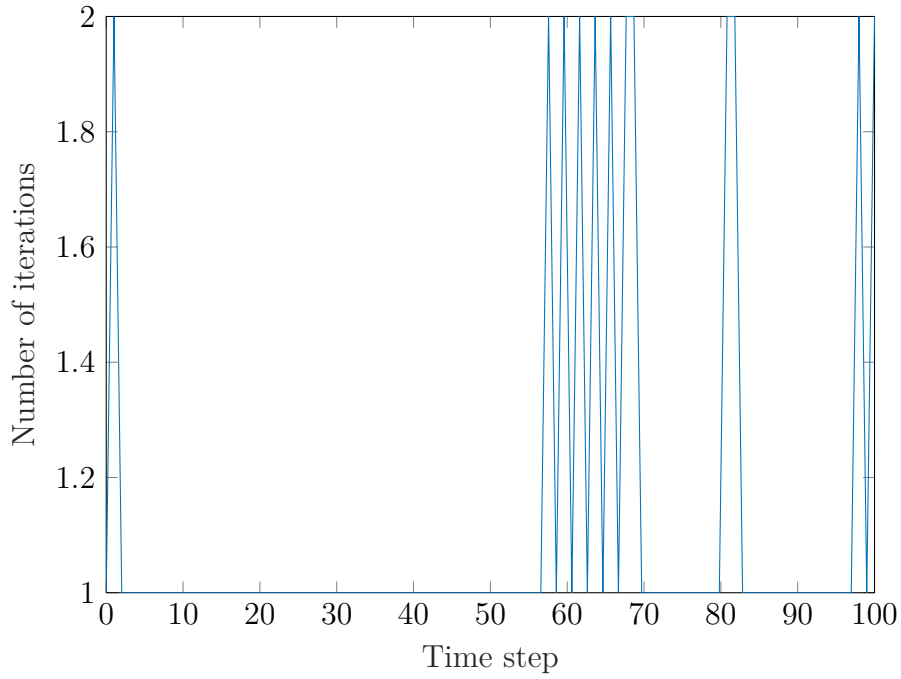

Figure 1.6: Number of iterations needed at each time step.

Fig. 1.6 depicts the number of iterations needed at each time step and just like in the previous case, the solution is always achieved with a maximum number of two iterations.

The difference between the CST element and the quadratic triangular element can be explained by the stiffer nature of the CST elements, which in turn result in higher forces to reach the same deformation.

1.3 Task 1c - Implementing a thermo-elastoplastic stress analysis

In this task the element routine from task 1b was modified such that the material accounted to a thermo-elastoplastic behaviour.
Here, the goal is to let the same example from task 1b undergo a linear temperature increase and determine at each step, whether elastic or plastic deformation is occurring. Depending on the result, the function shall decide, whether linear elasticity is suitable or whether Mises plasticity describes the behaviour better.
This will be decided through the CALFEM environment, more specifically the `mises` and the `dmises` functions.

Just as in task 1b, the FE mesh looks exactly the same, being depicted in Fig. 1.7.

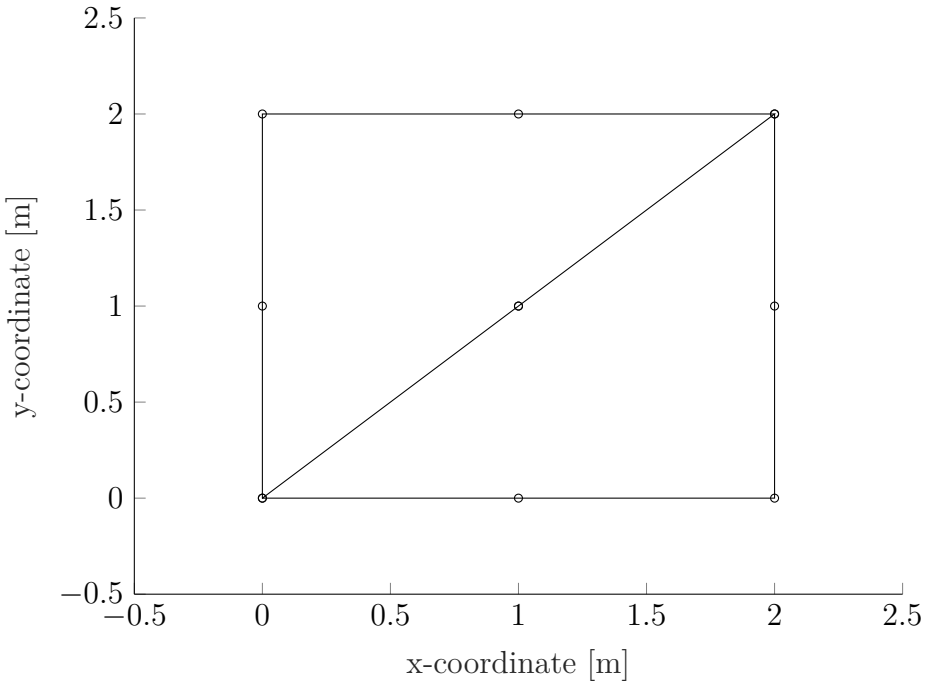

Figure 1.7: FE mesh and geometry used in task 1c.

For the first of two test cases, all outside boundaries have to be constrained, which leaves only the centre node free. Then a temperature control variable is added, which increases the temperature linearly until $\Delta T = 10K$ is reached.
The trial stress is given as

$$\begin{bmatrix} {}^{(k)}\sigma_{xx}^{\text{trial}} \\ {}^{(k)}\sigma_{yy}^{\text{trial}} \\ {}^{(k)}\sigma_{zz}^{\text{trial}} \\ {}^{(k)}\sigma_{xy}^{\text{trial}} \end{bmatrix} = \begin{bmatrix} {}^{n}\sigma_{xx} \\ {}^{n}\sigma_{yy} \\ {}^{n}\sigma_{zz} \\ {}^{n}\sigma_{xy} \end{bmatrix} + \underline{D} \left(\begin{bmatrix} {}^{(k)}\Delta\varepsilon_{xx} \\ {}^{(k)}\Delta\varepsilon_{yy} \\ {}^{(k)}\Delta\varepsilon_{zz} \\ {}^{(k)}\Delta\gamma_{xy} \end{bmatrix} - \alpha\Delta T \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right) \tag{1.8}$$

where $\underline{D}$ is the constitutive matrix for elasticity, α is the thermal expansion coefficient and ΔT is the temperature difference. With this trial stress, the `mises` function is able to determine whether the deformation is elastic or plastic.

For the problem at hand, the internal force in x-direction with respect to the temperature change is depicted in Fig. 1.8. It is obvious, that the internal force increases linearly. This result is expected.

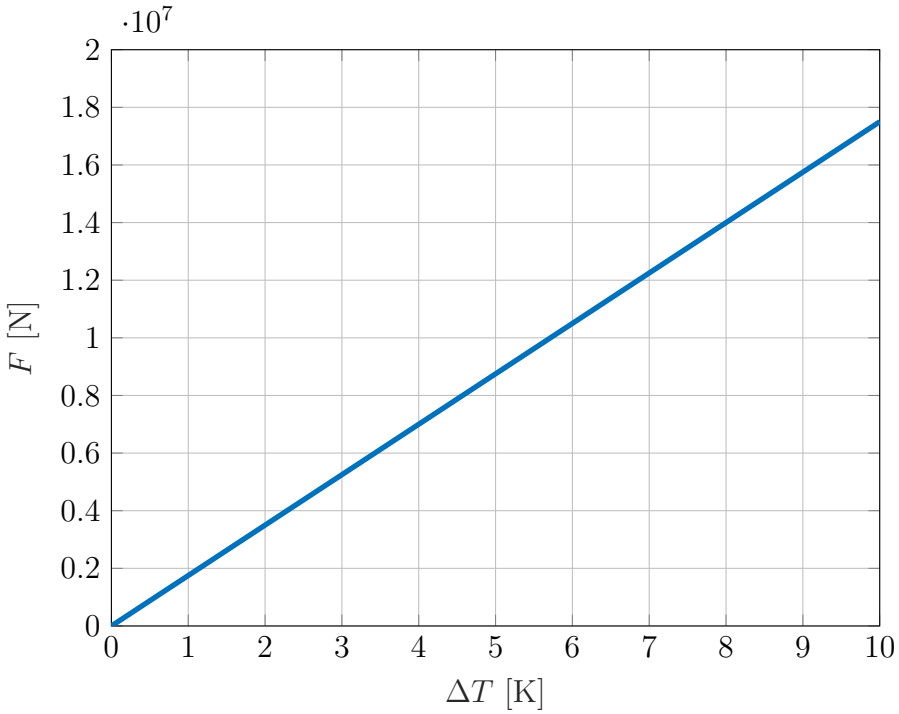

Figure 1.8: Internal force at top right node in x-direction with respect to temperature change.

Furthermore, Fig. 1.9 depicts the number of iterations needed to reach an acceptable solution. Every time step needed exactly one iteration to calculate to the defined precision.

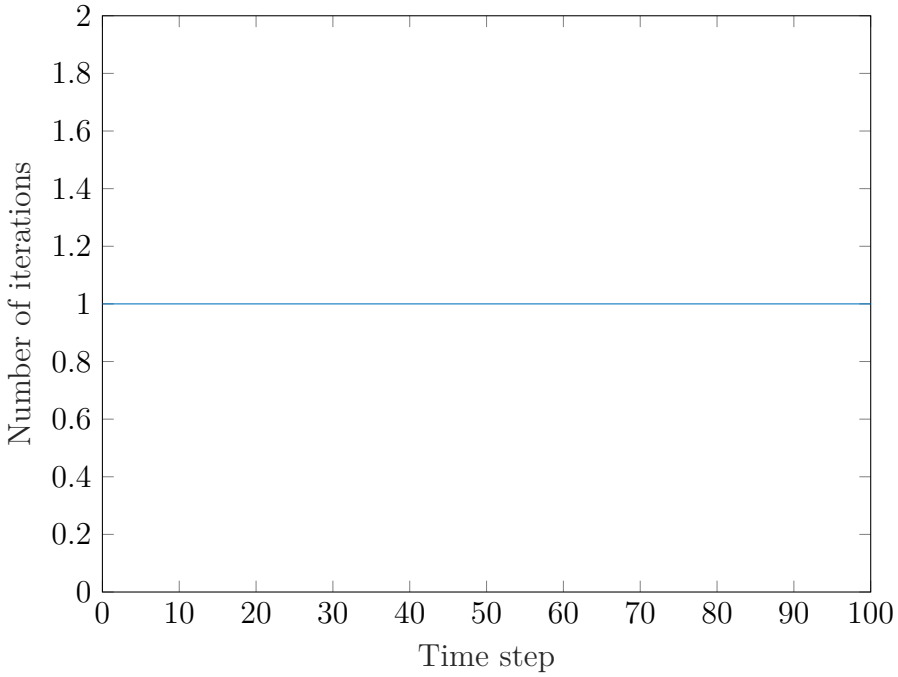

Figure 1.9: Number of iterations needed at each time step.

Apart from that, the average stress is calculated using arithmetic mean of the three calculated principal stresses as follows

$$\sigma_{avg} = \frac{\sigma_{11} + \sigma_{22} + \sigma_{33}}{3} \tag{1.9}$$

which results in the following average stress for the FE solution

$$\sigma_{avg} = -5.25 \cdot 10^7 MPa. \tag{1.10}$$

The three principal stresses can be analytically determined by calculating

$$\sigma_{avg} = \frac{\left\| \underline{D} \cdot \alpha \cdot \Delta T \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\|_2}{3} \tag{1.11}$$

which also results in an average stress of

$$\sigma_{avg} = 5.25 \cdot 10^7 MPa. \tag{1.12}$$

For the second test case the same example has to be constrained in y-direction for all nodes, such that deformation is only allowed in x-direction. Furthermore, the temperature is now linearly increased until $\Delta T = 300K$. The resulting reaction force from the upper bound with respect to the temperature change is depicted in Fig. 1.10.

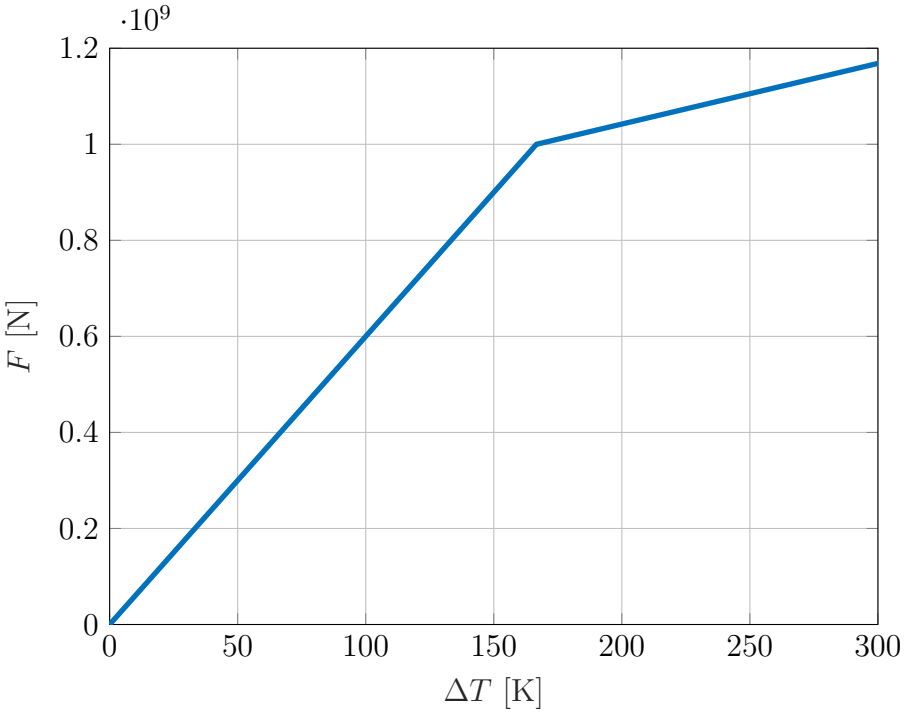

Figure 1.10: Sum of all internal forces in y-direction at the upper boundary against the temperature change.

The point where the elastic deformation changes to plastic deformation is clearly visible at time step $t = 56$ or at $\Delta T = 166.67K$. The reaction force at that temperature across all three nodes in y-direction is $F_{int} = 1000kN$.

Fig. 1.11 shows the number of iterations needed to solve the problem with sufficient accuracy. Every time step needed at most two iterations to reach a sufficient accuracy.

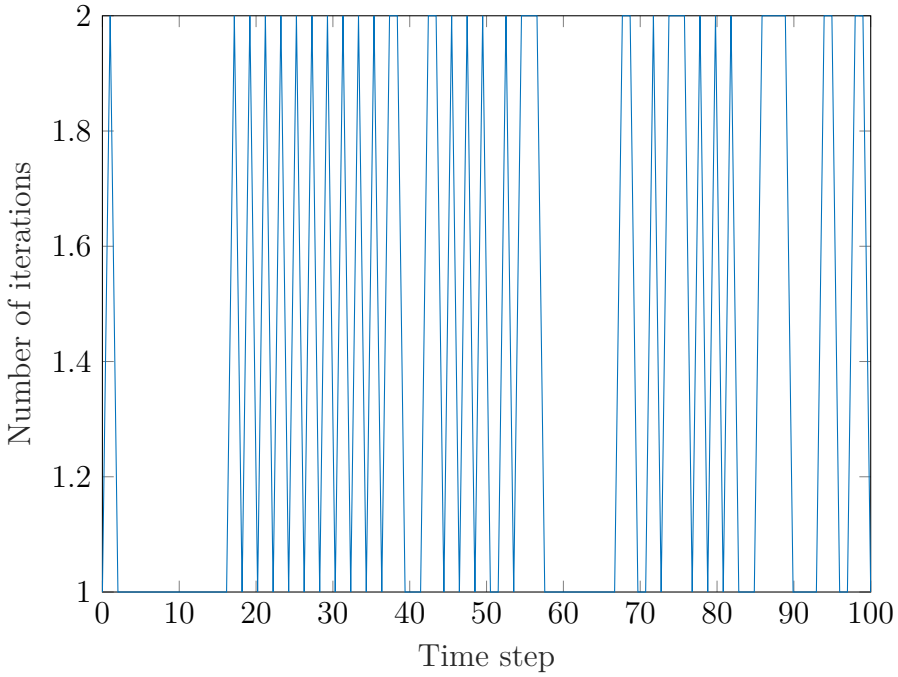

Figure 1.11: Number of iterations needed at each time step.

1.4 Task 1d - Analysis of deformation for a bimetal strip

In this task, a bimetal strip made of aluminium and steel is regarded. The goal is to perfom an analysis about the deformation of the structure, due to the rise of the temperature by $\Delta T = 300K$. The strip is fixed on the left side and is free on the right side with no constraints.
In addition to that, the bimetal strip is made of aluminium on the lower half and steel on the upper half. The following geometric and material properties were used.

Property	Unit	Value
L	mm	150
H	mm	10
b	mm	40
Es	GPa	210
ν_s	-	0.3
$\sigma_{\text{yield},s}$	MPa	500
H_s	MPa	40,000
α_s	K^{-1}	$10 \cdot 10^{-6}$
Ea	GPa	80
ν_a	-	0.2
$\sigma_{\text{yield},a}$	MPa	200
H_a	MPa	1,000
α_a	K^{-1}	$20 \cdot 10^{-6}$
# FE Elements	-	240
# DoF	-	1098

Table 1.2: Material mechanical Properties

As previously mentioned, the bimetal strip is heated up linearly until $\Delta T = 300K$ and afterwards cooled down linearly to the original temperature. The temperature progression is depicted in Fig. 1.12.

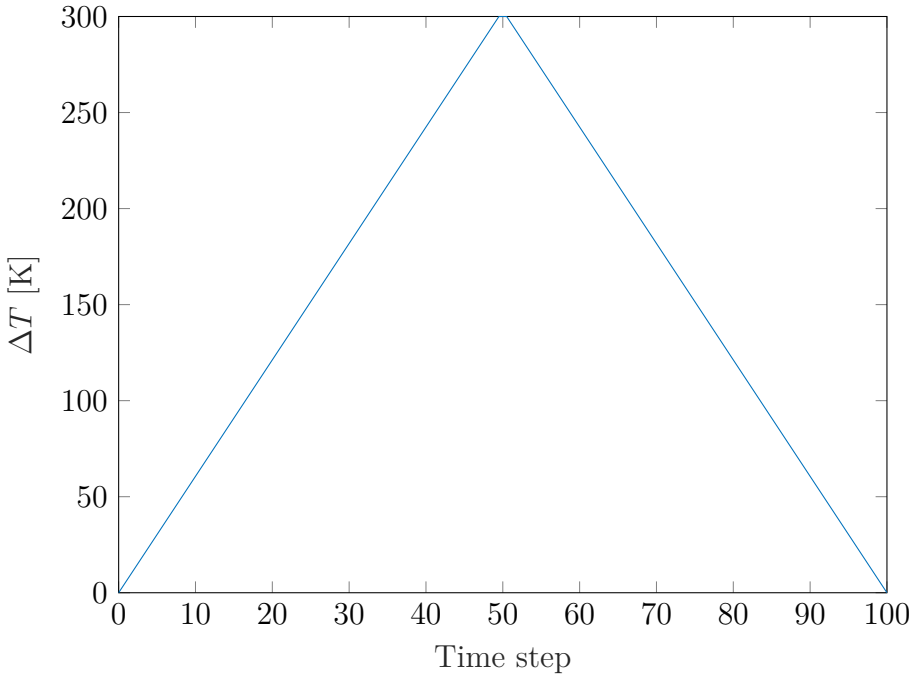

Figure 1.12: Linear temperature progression with respect to the time step.

With the temperature progression from Fig. 1.12, the simulation for the bimetal strip was run and the resulting maximum deformation was found to be node 5 with the 10th DoF of the system. Node 5 is the top most node at the free end of the strip. Fig. 1.13 shows the deformation of node 5 in y-direction.

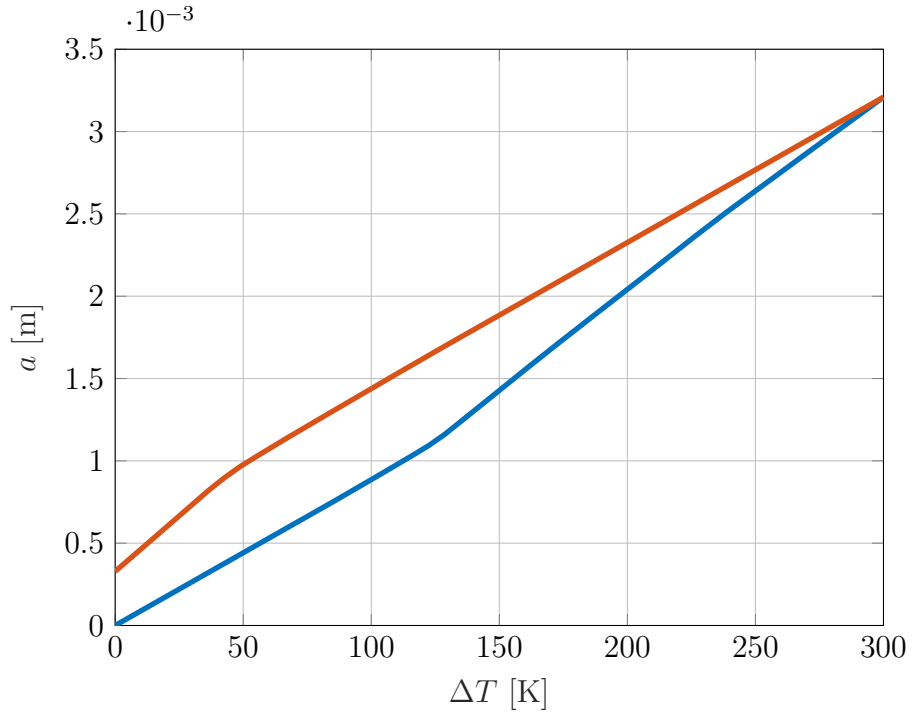

Figure 1.13: Deformation of node 5 in y-direction with respect to temperature, where blue denotes a rising temperature and red denotes a sinking temperature.

In Fig. 1.13 it is possible to see the maximum deformation at $\Delta T = 300K$ with

$a_{max} = 3.2mm$. Furthermore, it is interesting to note, that plastic deformation occurs at $\Delta T = 122.45K$ with a deformation of $a = 1.1mm$.
After cooling down, the plastic deformation remains and leaves a displacement of $a_p = 0.33mm$.

In Fig. 1.14 an upscaled version of the deformation at $\Delta T = 300K$ in blue and the plastically deformed shape after the temperature progression in red are compared to the initial shape of the bimetal strip before the temperature progression.

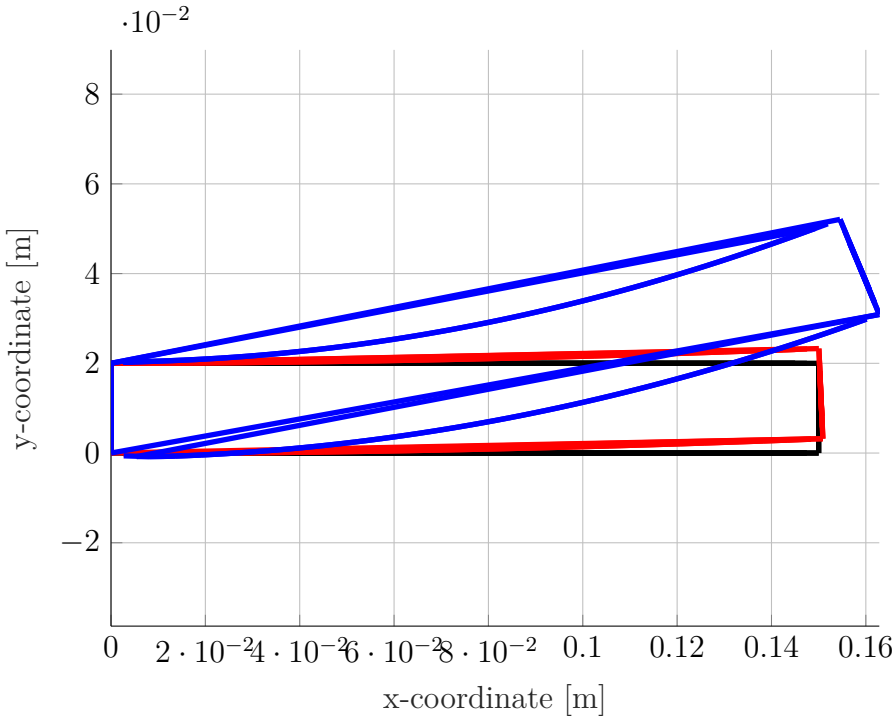

Figure 1.14: Upscaled visualisation of the deformation at $\Delta T = 300K$ in blue, plastically deformed shape in red, next to the original shape in black.

In Fig. 1.15 the number of iterations are plotted which were needed at each time step. Unlike before, where every time step was sufficiently computed within a maximum of two iterations, it takes the algorithm up to nine iterations to compute a sufficiently accurate solution to the problem.

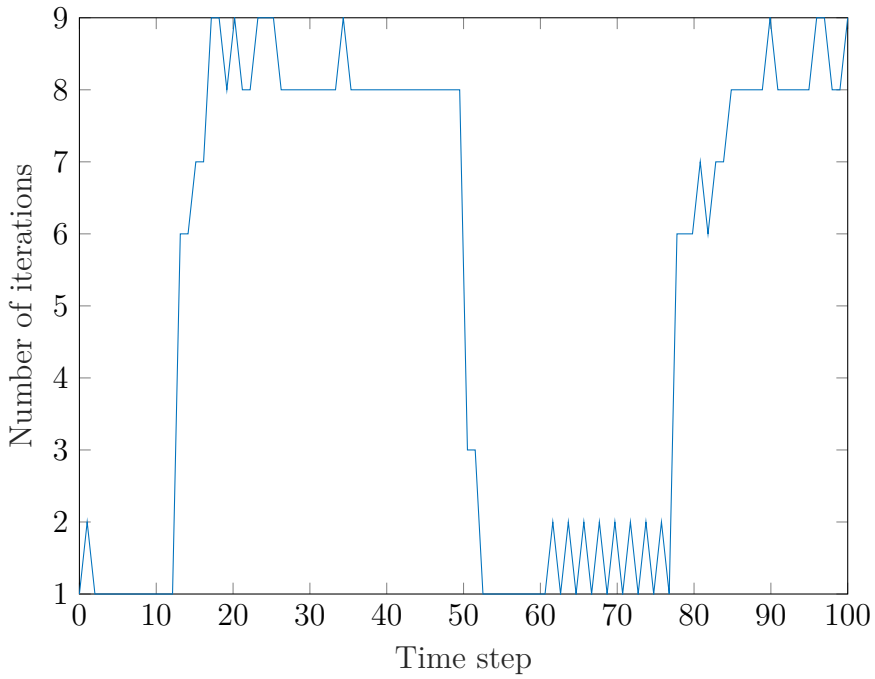

Figure 1.15: Number of iterations needed at each time step.

2

Task 2 - Abaqus

In this task, a predefined 3D geometry file for ABAQUS was provided containing all the dimensions in mm. An analysis for a thermo-elastoplastic condition was to be conducted for a bimetal strip, fixed at one end. Finally, the results were compared to the solution obtained in MATLAB with respect to the displacement at the maximum temperature.

2.1 Task 2a - Implementation of FE problem

The analysis was conducted for thermo-elastoplastic behaviour of a bimetal strip fixed at one end. A suitable element type was chosen, all the material properties were assigned to different sections. Specified loading conditions were applied along with the temperature. The loading condition was applied for a time period of $1s$ over a step-time increment of $0.1s$. Finally, the job was run to obtain a contour plot for the Von Mises Stress distribution and a contour plot for plastic deformation.

- Import bimetalStrip3D.cae file in ABAQUS
- Specify material properties of two different material with thermo-elastoplastic conditions
- Assign section for each material
- Create load step for time period of $1s$
- Apply boundary condition for a fixed end and apply temperature load
- Choose and create a suitable element type and size
- Submit the job
- Visualize the results

A maximum vertical displacement was found to be $2.56mm$ at maximum temperature of $300°C$. A plot was obtained at the midplane of the bimetal strip.

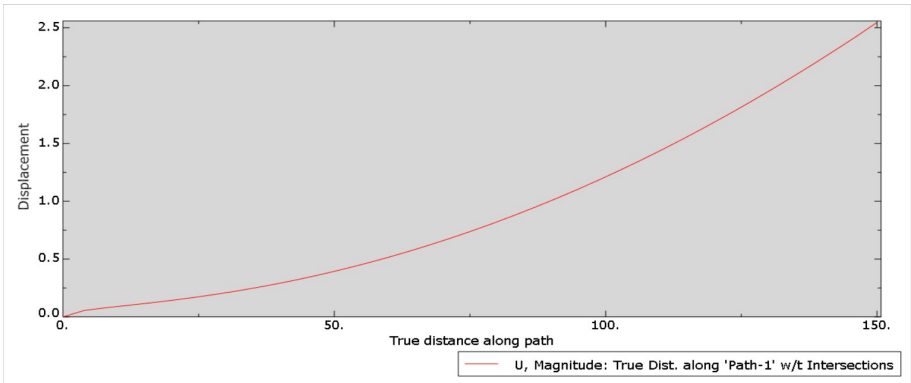

Figure 2.1: Vertical displacement plot at midplane

2.2 Task 2b - Results and discussion

At first, an element of type *tet* was chosen. The simulation was run for a temperature difference of $\Delta T = 300K$ over the time period for a varying total number of elements. One end of the bimetal strip was fixed using Encastre boundary condition and a thermal load was applied throughout the whole body. The thermal load was defined by varying amplitude over a the time period of $1s$, where exactly at $0.5s$, a maximum temperature of $\Delta T = 300K$ was attained in the system. The following deformation and Von Mises Stress plot is obtained for each global element size.

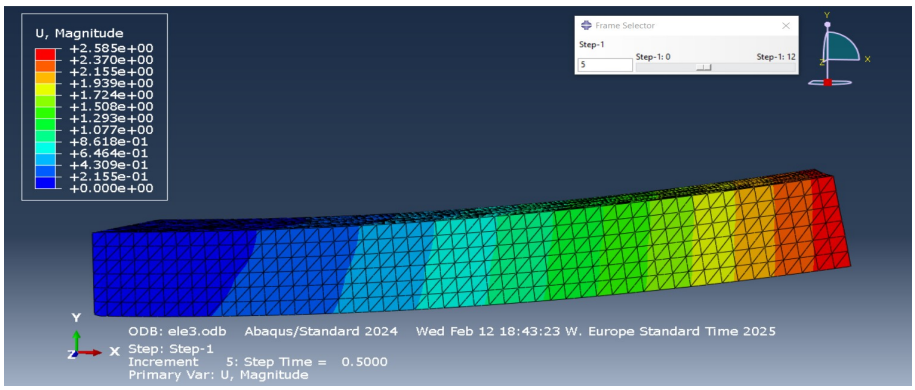

Figure 2.2: Deformed plot at maximum temperature for global element size 3

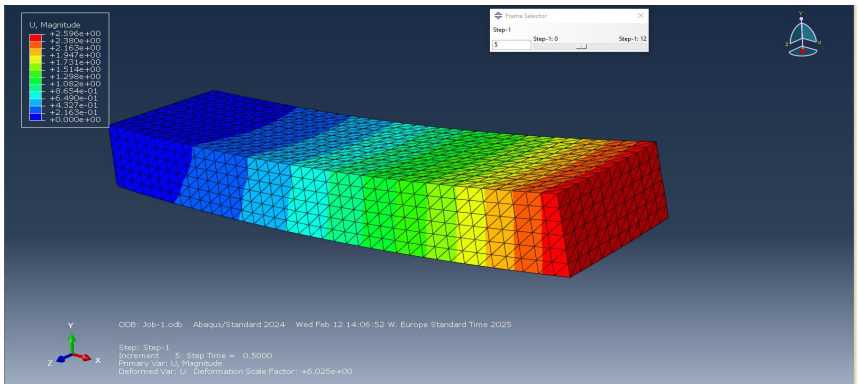

Figure 2.3: Deformed plot at maximum temperature for global element size 4

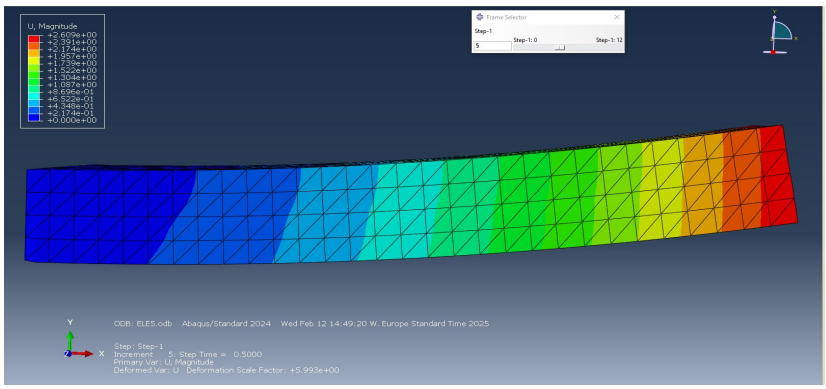

Figure 2.4: Deformed plot at maximum temperature for global element size 5

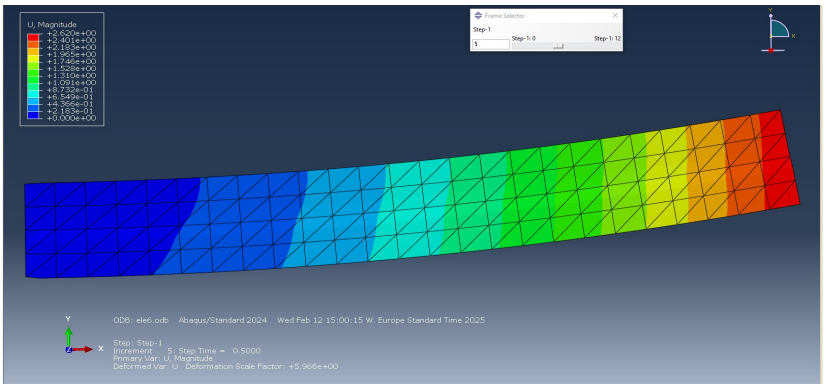

Figure 2.5: Deformed plot at maximum temperature for global element size 6

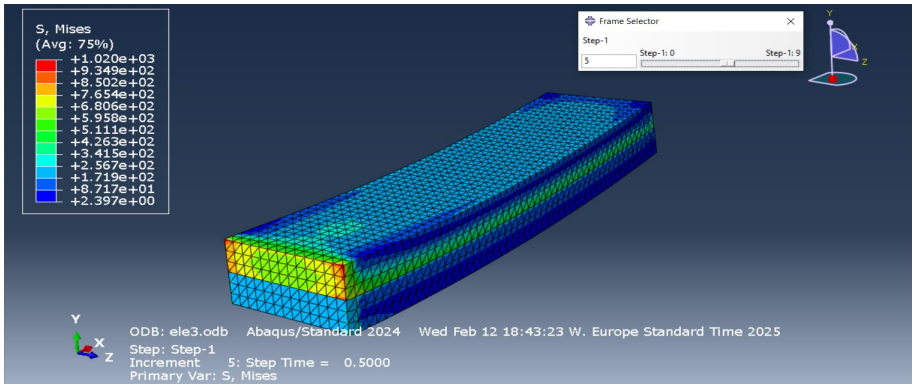

Figure 2.6: Von Mises Stress plot at maximum temperature for global element size 3

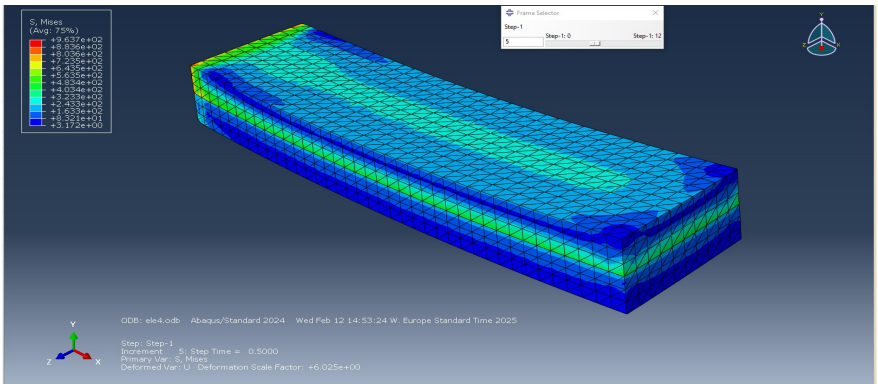

Figure 2.7: Von Mises Stress plot at maximum temperature for global element size 4

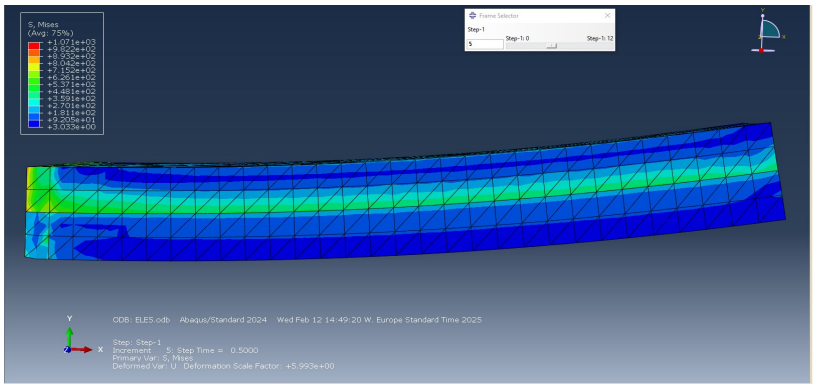

Figure 2.8: Von Mises Stress plot at maximum temperature for global element size 5

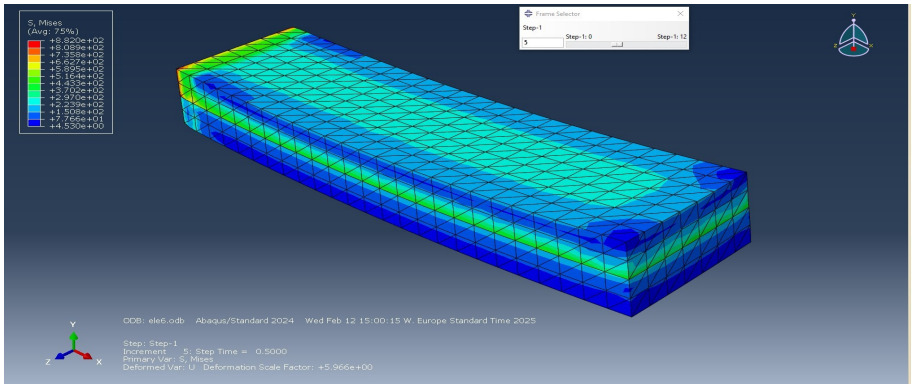

Figure 2.9: Von Mises Stress plot at maximum temperature for global element size 6

The following table was obtained for the element type *tet* in terms of the maximum deformation and Von Mises stress.

Global Element Size	**No. of elements**	**Deformation** (mm)	**Max.Von Mises Stress** (MPa)
6	5243	2.62	8.82×10^2
5	8491	2.609	1.07×10^3
4	15248	2.596	9.637×10^2
3	30455	2.585	1.02×10^3

Table 2.1: Results for element type *tet*

Further, a different element type was considered i.e, an element of type *Hex* was chosen this time. As in the previous step, the results were obtained for different global element sizes, the deformation as well as the Von Mises Stress results were plotted.

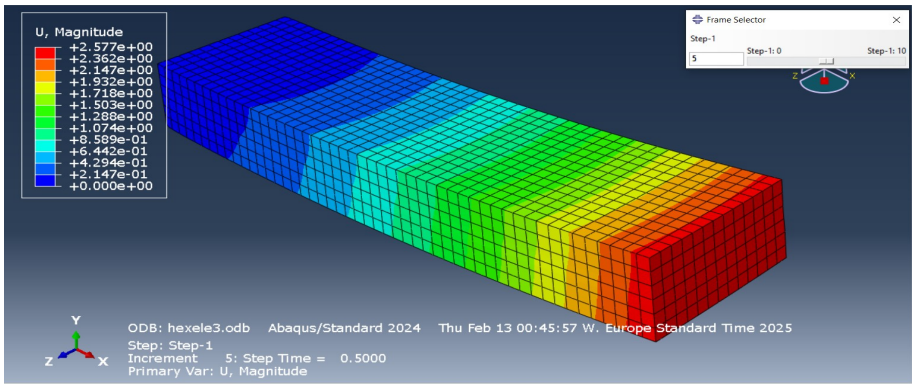

Figure 2.10: Deformation plot at maximum temperature for global element size 3

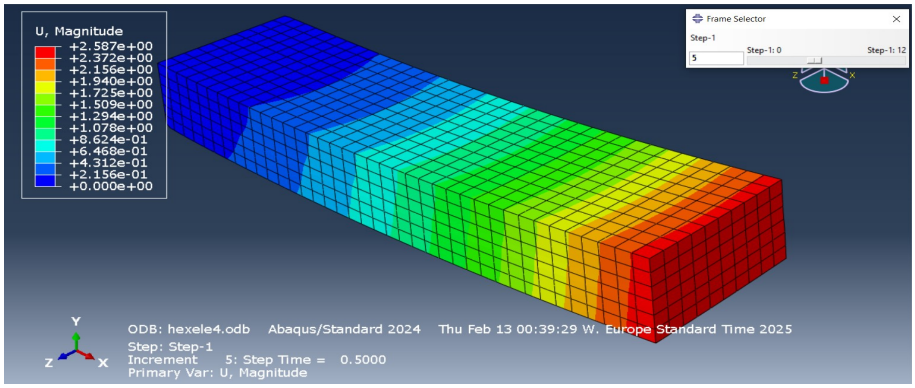

Figure 2.11: Deformation plot at maximum temperature for global element size 4

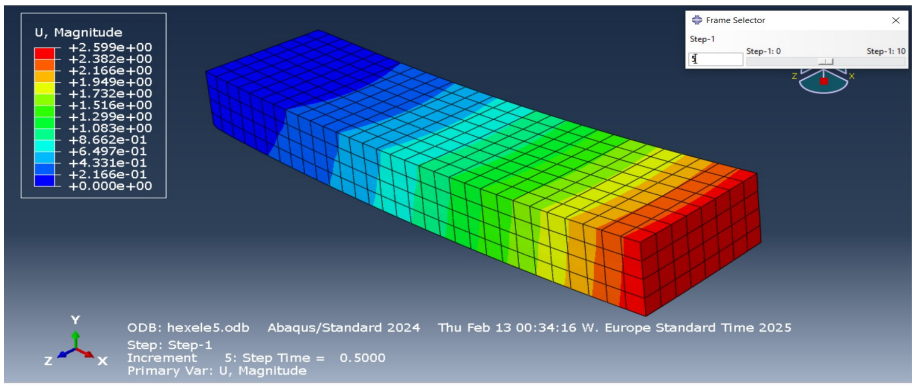

Figure 2.12: Deformation plot at maximum temperature for global element size 5

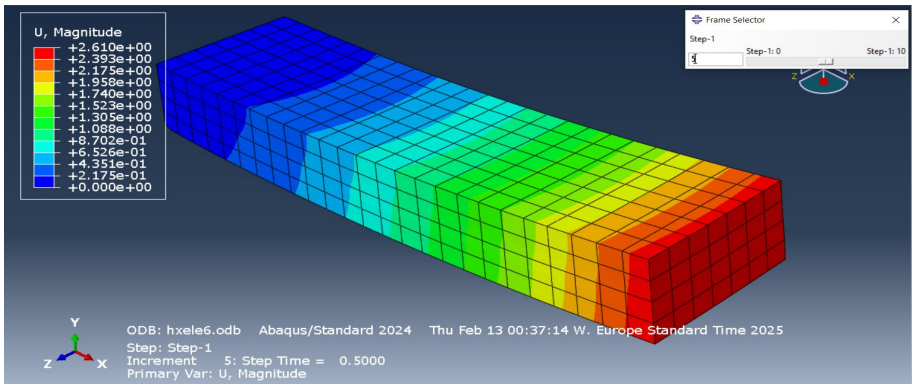

Figure 2.13: Deformation plot at maximum temperature for global element size 6

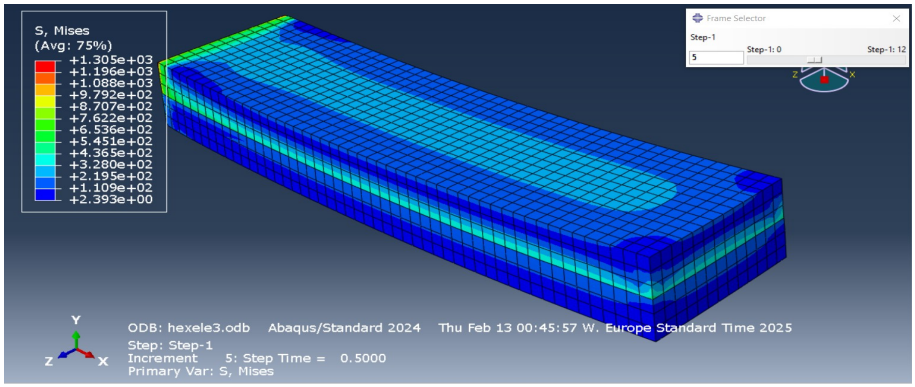

Figure 2.14: Von Mises stress plot at maximum temperature for global element size 3

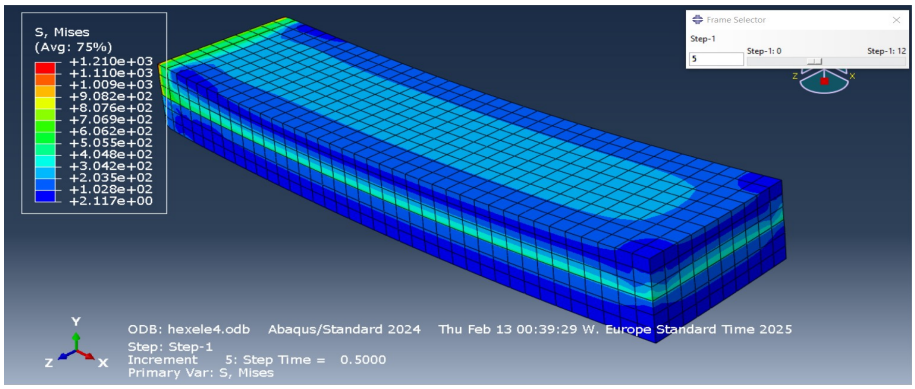

Figure 2.15: Von Mises stress plot at maximum temperature for global element size 4

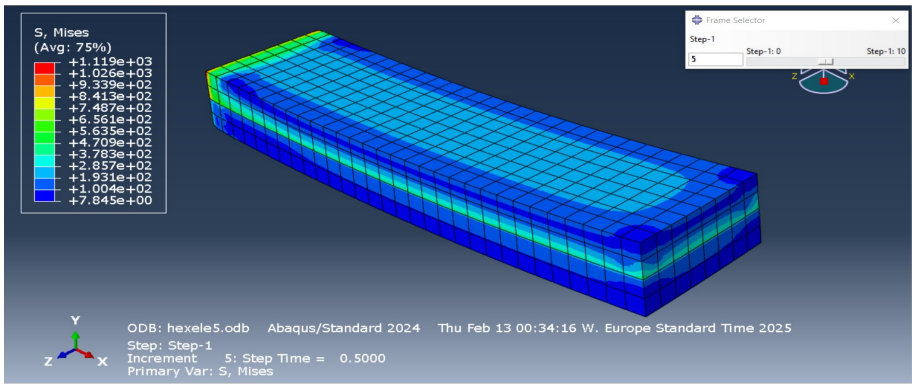

Figure 2.16: Von Mises stress plot at maximum temperature for global element size 5

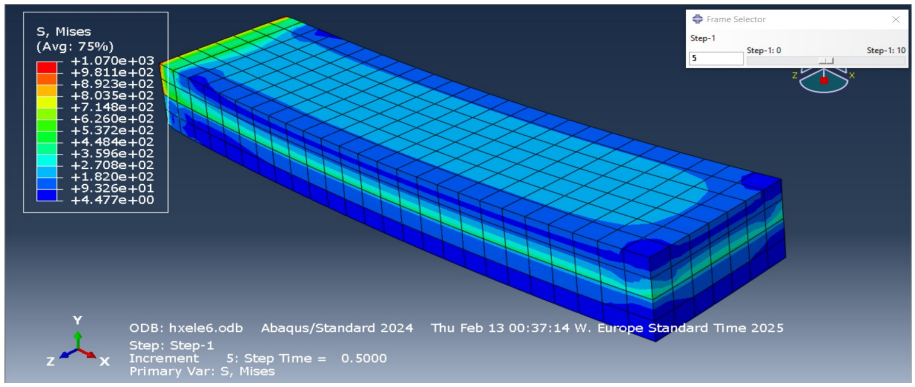

Figure 2.17: Von Mises stress plot at maximum temperature for global element size 6

The following table was obtained for element type 'Hex' in terms of maximum deformation and Von Mises stress

Global Element Size	**No. of elements**	**Deformation** (mm)	**Max.Von Mises Stress** (MPa)
6	700	2.61	1.07×10^3
5	960	2.599	1.119×10^3
4	2280	2.587	1.21×10^3
3	3900	2.577	1.305×10^3

Table 2.2: Results for element type *Hex*

First observation made was that the element type *tet* and *Hex* both give similar results in terms of deformation. As the number of elements increases, both element type come close to the final value and converge after few iteration. It was observed that there is not much of a difference in maximum deformation between each global element size.
One major drawback observed for the element type *tet* was that the stress values would not converge for the element size and further investigation was needed by increasing the number of elements which demands high computational effort and was noted that it is a very time consuming process. However, the stress converges after a few iterations and increasing the number of elements, this is obtained by the element type *Hex*.
The *Hex* element type converged the deformation and stress values for a significantly less number of elements compared to *tet* as seen in the Table 2.1 and Table 2.2. It was also

observed that the computational effort required to obtain the results was significantly less compared to *tet*.
It came down to a conclusion that the element type *Hex* was much better and efficient way to obtain result at least for this test case/task and it safe to say that the results have converged.

The figures below gives the plots at the end time when the temperature is returned to $\Delta T = 0K$.

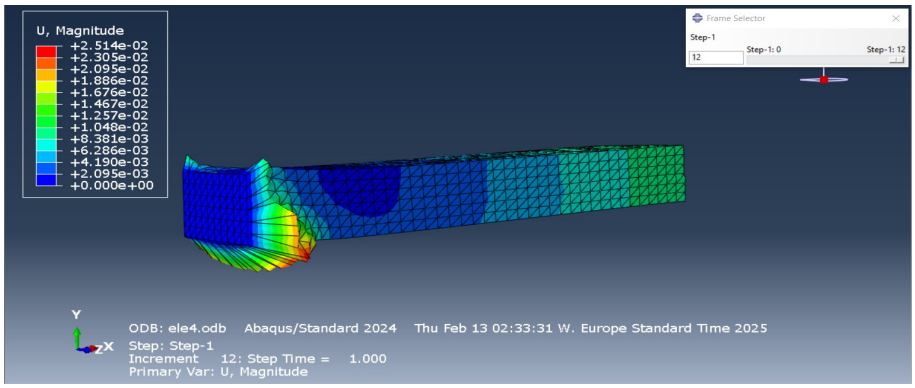

Figure 2.18: Deformation plot at end time $T = 1s$

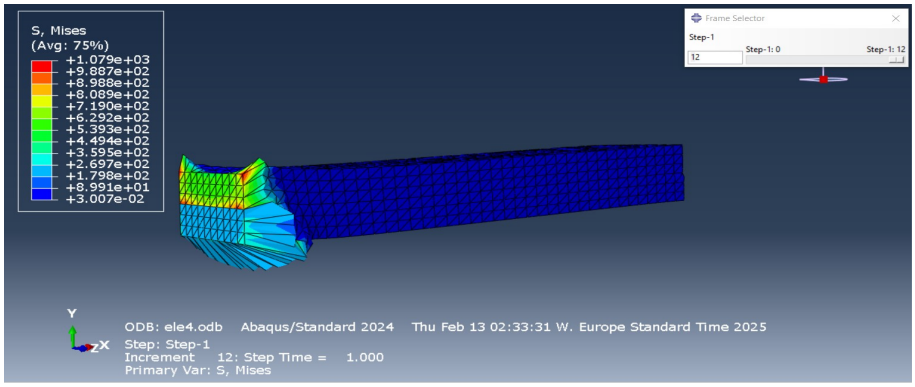

Figure 2.19: Stress plot at end time $T = 1s$

The figure below demonstrates the domain which has experienced plastic deformation after the temperature has reached the starting point again. A maximum limit was set to be as minimal as possible and was of the order 0.001.

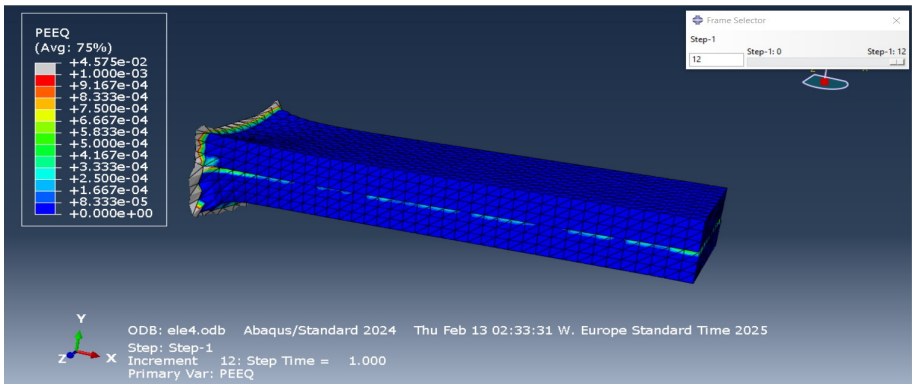

Figure 2.20: Plot for PEEQ at endtime $T = 1s$

The plastic deformation occurs near the fixed end. This is due to the expansion of the material due to heat and thus the stresses are relieved. This can be observed near the fixed end, there are very few stresses.

2.2.1 Task 2b - Conclusion

The maximum displacement at the maximum temperature for Matlab was found to be $3.2mm$ The maximum displacement at a maximum temperature for Abaqus was found to be $2.61mm$. This specific element size was chosen because the number elements are 700 for Abaqus and for Matlab was found 240.
The difference in the deformation is most likely due to the coarse mesh in the Matlab function. Having a coarse mesh will not provide a sufficiently accurate solutions. Nevertheless is the order of magnitude for the deformation already very close to the solution calculated by Abaqus.
Another reason would be the different element types used in both simulations. In Matlab, second order triangles were used, which are defined through quadratic shape functions where as in Abaqus first order tetrahedron elements were use. These only consist of four linear shape functions.

References

[GF07] K.-H. Grote and J. Feldhusen. *Dubbel* -. Wiesbaden: Springer Berlin Heidelberg, 2007. ISBN: 978-3-540-68191-5.

A

Appendix: Code listing

A.1 Task 1a - Main File

```
%% Material model parameters
g = 9.81; % Gravitational acceleration
mpar.Emod=210e9; %Pa
mpar.Sy=400e6; %Pa
mpar.Hmod=mpar.Emod/20;
nu = 0.3;
dens = 7850;

% Gravitation (1=yes; 0=no)
grav = 0;

% External force P acting on node 3
P = -20000; % Force in [N]

% D matrix of material properties for plane strain
De = (mpar.Emod / ((1 + nu) * (1 - 2*nu))) * [1-nu, nu, 0; nu, 1-nu, 0;
    0, 0, (1-2*nu)/2];

% Length
L = 2;

% Height
h = 2;

% Depth
d = 1;

% Element connectivity Edof
Edof = [1 1 2 5 6 7 8;2 1 2 3 4 5 6];

% No. of elements
nelem = size(Edof,1);

% Position of nodes
Coord = [0 0; L 0; L h; 0 h];
nnodes = size(Coord,1);
ndofs = 2*nnodes;

% DoFs in each node
dof = [1 2; 3 4; 5 6; 7 8];

% Compute Ex and Ey from CALFEM routine
[Ex,Ey]=coordxtr(Edof,Coord,dof,3);
```

```
43
44 % eldraw2(Ex,Ey);
45 % xlabel('x-coordinate [m]');
46 % ylabel('y-coordinate [m]')
47
48 scatter(Ex,Ey,"k");
49 hold on;
50 xlabel('x-coordinate [m]');
51 ylabel('y-coordinate [m]')
52 % plot([0,2,0,2,0,0,0,0,2,2],[0,2,2,2,0,2,2,2,0,2],"k")
53 plot([0,2,0,0,2,2],[0,2,2,0,0,2],"k")
54 xlim([-0.5, 2.5]);
55 ylim([-0.5 2.5]);
56
57 % Initialize displacements
58 a=zeros(ndofs,1);
59 aold=zeros(ndofs,1);  %old displacements (from previous timestep)
60 da=a-aold;
61
62
63 % Define free dofs and constrained dofs
64 dof_F=[1:ndofs];
65 dof_C=[1 2 6 7 8];
66 dof_F(dof_C) = []; %removing the prescribed dofs from dof_F
67
68 % Time stepping
69 ntime=100; %number of timesteps
70 tend=ntime; %end of time [s]
71 t=linspace(0,tend,ntime);
72
73 % Test run
74 a_total = zeros(ndofs,ntime);
75
76 % Displacement control
77 amax = 1E-3;
78 aa = linspace(0,amax,ntime);
79
80 % Initialize variables for post processing
81 K = spalloc(ndofs,ndofs,20*ndofs); % defines K as a sparse matrix and
    sets the size
82                                    % to (ndof x ndof) with initial zero
    value
83                                    % No. of estimated non-zero entries
    is 20*ndofs
84
85 % Initialize internal and external force
86 fint = zeros(ndofs,1);
87 fext = fint;
88 F=zeros(size(aa));
89
90 % Vector of applied external forces
91 f_ext = [0;0;0;0;0;P;0;0];
92
93 %tolerance value for Newton iteration
94 tol=1e-6;
95
96 % Initializing stress and strain matrices
97 Et = zeros(nelem,3); % Strain
```

```
98  Es = zeros(nelem,3); % Stress
99
100 niter_total = 0;
101
102 %-----------------------------------------------------
103 % Newton iteration for solving Non-Linear problem
104 %-----------------------------------------------------
105
106 for i=1:ntime
107     % Initial guess of unknown displacement field
108     a(dof_F)=aold(dof_F)+da(dof_F);
109
110     a(6) = -aa(i);
111
112     % Newton iteration to find unknown displacements
113     unbal=1e10; niter=0;
114     while unbal > tol
115         K=K.*0;
116         fint=fint.*0; %nullify
117         % fext = f_ext;
118     fext = f_ext;
119
120         % Loop over elements
121         for iel=1:nelem
122
123             % Extract element diplacements
124             ed=a(Edof(iel,2:end));
125
126             % Element calculations for CST
127             [Ae,Be,Ke,fe_int,fe_ext] = CST_Calc(ed,iel,Ex,Ey,De,d,dens,g
    ,grav);
128
129             % Assembling
130             fint(Edof(iel,2:end))=fint(Edof(iel,2:end))+fe_int;
131             fext(Edof(iel,2:end))=fext(Edof(iel,2:end))+fe_ext;
132
133             K(Edof(iel,2:end),Edof(iel,2:end))=...
134                 K(Edof(iel,2:end),Edof(iel,2:end))+Ke;
135
136             % Calculation of strain and stress
137             Et(iel,:) = Be*a(Edof(iel,2:end)); % Strain
138             Es(iel,:) = De*Be*a(Edof(iel,2:end)); % Stress
139
140         end
141         % Unbalance equation
142         g_F=fint(dof_F)-fext(dof_F);
143
144         unbal=norm(g_F);
145         if unbal > tol
146             % Newton update
147             a(dof_F)=a(dof_F)-K(dof_F,dof_F)\g_F;
148         end
149
150         niter=niter+1; %update iter counter
151         [niter unbal]  %print on screen
152
153         if niter>20
154             disp('no convergence in Newton iteration')
```

```
155             break
156         end
157 
158         f_extC = fint(dof_C)- fext(dof_C);
159 
160 
161     end
162 
163     niter_total = cat(1,niter_total,niter);
164     F(i)=-fint(6);
165 
166     da=a-aold;
167     aold = a;
168     a_total(:,i) = a;
169     plot(aa,F,'-') %for plotting during simulation
170     drawnow
171 end
172 
173 
174 close all
175 plot(aa,F,'linewidth',2)
176 set(gca,'FontSize',14,'fontname','Times New Roman')
177 xlabel('$a$ [m]','FontSize',16,'interpreter','latex')
178 ylabel('$F$ [N]','FontSize',16,'interpreter','latex')
179 grid on
180 
181 niter_total(1) = [];
182 plot(t,niter_total)
183 xlabel('Time step')
184 ylabel('Number of iterations')
185 
186 P = F(end);
187 
188 % Analytical solution of cantilever beam
189 % https://www.engineeringtoolbox.com/cantilever-beams-d_1848.html
190 I = 1/12 * d*h^3;
191 defl = abs(P)*L^3 /(3*mpar.Emod*I);
192 
193 P = (amax * 3*mpar.Emod*I)/L^3;
```

A.1.1 Task 1a - Function files

Area function.

```
function A = Area(x1, y1, x2, y2, x3, y3)

A = 1/2 * det([x1, y1, 1; x2, y2, 1; x3, y3, 1]);

end
```

CSTCalc function.

```
function [Ae,Be,Ke,fe_int,fe_ext] = CST_Calc(ed,iel,Ex,Ey,De,d,dens,g,
    grav)

% Area of element
Ae=Area(Ex(iel,1),Ey(iel,1),Ex(iel,2),Ey(iel,2),Ex(iel,3),Ey(iel,3));

% B-Matrix of element
Be=Be_cst_func([Ex(iel,1) Ey(iel,1)]',[Ex(iel,2) Ey(iel,2)]',...
    [Ex(iel,3) Ey(iel,3)]');

% Element stiffness matrix
Ke=Be'*De*Be*Ae*d;

% Internal force vector of element
fe_int = Ke*ed;

% Gravitational force vector of element
fe_ext = zeros(6,1);
if grav == 1
    fe_ext = d*dens*Ae*g*1/3 *[0;-1;0;-1;0;-1];
end

end
```

A.2 Task 1b - Main File

```
%% Material model parameters
g = 9.81; % Gravitational acceleration
mpar.Emod=210e9; %Pa
mpar.Sy=400e6; %Pa
mpar.Hmod=mpar.Emod/20;
nu = 0.3;
dens = 7850;

% Gravitation (1=yes; 0=no)
grav = 0;

% External force P acting on node 3
P = 0; % Force in [N]

% D matrix of material properties for plane strain
% ptype=1: plane stress ||| 2: plane strain ||| 3:axisym ||| 4: 3d
ptype=2;
De =hooke(ptype,mpar.Emod,nu); % Constitutive matrix - plane strain
% De = (mpar.Emod / ((1 + nu) * (1 - 2*nu))) * [1-nu, nu, 0; nu, 1-nu,
    0; 0, 0, (1-2*nu)/2];
De(3,:) = [];
De(:,3) = [];

% Length
L = 2;

% Height
h = 2;

% Depth
d = 1;

% Element connectivity Edof
Edof = [1 1 2 5 6 13 14 11 12 15 16 17 18;...
    2 1 2 3 4 5 6 7 8 9 10 11 12];

% No. of elements
nelem = size(Edof,1);

% Position of nodes
Coord = [0 0; L 0; L h; L/2 0; L h/2; L/2 h/2; 0 h; L/2 h; 0 h/2];
nnodes = size(Coord,1);
ndofs = 2*nnodes;

% DoFs in each node
dof = zeros(nnodes,2);
for i = 1:nnodes
    dof(i,1) = 2*i -1;
    dof(i,2) = 2*i;
end

% Compute Ex and Ey from CALFEM routine
[Ex,Ey]=coordxtr(Edof,Coord,dof,6);

scatter(Ex,Ey,"k");
```

```
55 hold on;
56 xlabel('x-coordinate [m]');
57 ylabel('y-coordinate [m]')
58 % plot([0,2,0,2,0,0,0,0,2,2],[0,2,2,2,0,2,2,2,0,2],"k")
59 plot([0,2,0,0,2,2],[0,2,2,0,0,2],"k")
60 xlim([-0.5, 2.5]);
61 ylim([-0.5 2.5]);
62
63 % Initialize displacements
64 a=zeros(ndofs,1);
65 aold=zeros(ndofs,1);  %old displacements (from previous timestep)
66 da=a-aold;
67
68
69 % Define free dofs and constrained dofs
70 dof_F=[1:ndofs];
71 dof_C=[1 2 6 13 14 17 18];
72 dof_F(dof_C) = []; %removing the prescribed dofs from dof_F
73
74 % Time stepping
75 ntime=100; %number of timesteps
76 tend=ntime; %end of time [s]
77 t=linspace(0,tend,ntime);
78
79 % Test run
80 a_total = zeros(ndofs,ntime);
81
82 % Displacement control
83 amax = 1E-3;
84 aa = linspace(0,amax,ntime);
85
86 % Initialize variables for post processing
87 K = spalloc(ndofs,ndofs,20*ndofs); % defines K as a sparse matrix and
    sets the size
88                                    % to (ndof x ndof) with initial zero
    value
89                                    % No. of estimated non-zero entries
    is 20*ndofs
90
91 % Initialize internal and external force
92 fint = zeros(ndofs,1);
93 fext = fint;
94 F=zeros(size(aa));
95
96 % Vector of applied external forces
97 f_ext = fint;
98 f_ext(6) = P;
99
100 %tolerance value for Newton iteration
101 tol=1e-6;
102
103 % Initializing stress and strain matrices
104 Et = zeros(nelem,3); % Strain
105 Es = zeros(nelem,3); % Stress
106
107 niter_total = 0;
108
109 %-------------------------------------------------
```

```
% Newton iteration for solving Non-Linear problem
%-----------------------------------------------------

for i=1:ntime
    % Initial guess of unknown displacement field
    a(dof_F)=aold(dof_F)+da(dof_F);

    a(6) = -aa(i);

    % Newton iteration to find unknown displacements
    unbal=1e10; niter=0;
    while unbal > tol
        K=K.*0;
        fint=fint.*0; %nullify
        fext = f_ext;

        % Loop over elements
        for iel=1:nelem

            % Extract element diplacements
            ed=a(Edof(iel,2:end));

            % Element calculations for CST
            [fe_int,Ke,fe_ext] = QuadTriang(ed,Ex(iel,:),Ey(iel,:),De,d)
    ;

            % Assembling
            fint(Edof(iel,2:end))=fint(Edof(iel,2:end))+fe_int;
            fext(Edof(iel,2:end))=fext(Edof(iel,2:end))+fe_ext;

            K(Edof(iel,2:end),Edof(iel,2:end))=...
                K(Edof(iel,2:end),Edof(iel,2:end))+Ke;

            % Calculation of strain and stress
            % Et(iel,:) = Be*a(Edof(iel,2:end)); % Strain
            % Es(iel,:) = De*Be*a(Edof(iel,2:end)); % Stress

        end
        % Unbalance equation
        g_F=fint(dof_F)-fext(dof_F);

        unbal=norm(g_F);
        if unbal > tol
            % Newton update
            a(dof_F)=a(dof_F)-K(dof_F,dof_F)\g_F;
        end

        niter=niter+1; %update iter counter
        [niter unbal]  %print on screen

        if niter>400
            disp('no convergence in Newton iteration')
            break
        end

        % f_extC = fint(dof_C)- fext(dof_C);


```

```
167     end
168 
169     niter_total = cat(1,niter_total,niter);
170     F(i) = -2*fint(6);
171 
172     da=a-aold;
173     aold = a;
174     a_total(:,i) = a;
175     plot(aa,F,'-') %for plotting during simulation
176     drawnow
177 end
178 
179 close all
180 plot(aa,F,'linewidth',2)
181 set(gca,'FontSize',14,'fontname','Times New Roman')
182 xlabel('$a$ [m]','FontSize',16,'interpreter','latex')
183 ylabel('$F$ [N]','FontSize',16,'interpreter','latex')
184 grid on
185 
186 niter_total(1) = [];
187 plot(t,niter_total)
188 xlabel('Time step')
189 ylabel('Number of iterations')
190 
191 P = F(end);
192 
193 
194 % Analytical solution of cantilever beam
195 % https://www.engineeringtoolbox.com/cantilever-beams-d_1848.html
196 I = 1/12 * d*h^3;
197 defl = abs(P)*L^3 /(3*mpar.Emod*I);
198 
199 P = (amax * 3*mpar.Emod*I)/L^3;
```

A.2.1 Task 1b - Function files

SymQuadTriang function

```
% Element routine for quadratic triangular elements

% Define isoparametric coordinates
xi=sym('xi',[2,1],'real');

% Define six shape functions for quadratic elements
N1 = (1-xi(1)-xi(2))*(1-2*xi(1)-2*xi(2));
N2 = xi(1)*(2*xi(1)-1);
N3 = xi(2)*(2*xi(2)-1);
N4 = 4*xi(1)*(1-xi(1)-xi(2));
N5 = 4*xi(1)*xi(2);
N6 =4*xi(2)*(1-xi(1)-xi(2));

% Derivatives of shape functions
dN1_dxi=gradient(N1,xi);
dN2_dxi=gradient(N2,xi);
dN3_dxi=gradient(N3,xi);
dN4_dxi=gradient(N4,xi);
dN5_dxi=gradient(N5,xi);
dN6_dxi=gradient(N6,xi);

% Define node positions symbolically
xe1=sym('xe1',[2,1],'real');
xe2=sym('xe2',[2,1],'real');
xe3=sym('xe3',[2,1],'real');
xe4=sym('xe4',[2,1],'real');
xe5=sym('xe5',[2,1],'real');
xe6=sym('xe6',[2,1],'real');

% Define spatial coordinate as function of isoparam. coord.
x=N1*xe1 + N2*xe2 + N3*xe3 + N4*xe4 + N5*xe5 + N6*xe6;

% Compute Jacobian
Fisopeeee=jacobian(x,xi);

% Use chain rule to compute spatial derivatives
dN1 = simplify(inv(Fisopeeee)'*dN1_dxi);
dN2 = simplify(inv(Fisopeeee)'*dN2_dxi);
dN3 = simplify(inv(Fisopeeee)'*dN3_dxi);
dN4 = simplify(inv(Fisopeeee)'*dN4_dxi);
dN5 = simplify(inv(Fisopeeee)'*dN5_dxi);
dN6 = simplify(inv(Fisopeeee)'*dN6_dxi);

%----------------------------------------
% Calculate element B-matrix
%----------------------------------------
% H = 1/6; % Integration weight
IP = [1/6, 1/6; 1/6, 2/3; 2/3, 1/6].'; % Integration points

% Structure of B-matrix
B = [dN1(1), 0, dN2(1), 0, dN3(1), 0, dN4(1), 0, dN5(1), 0, dN6(1), 0 ;
0, dN1(2), 0, dN2(2), 0, dN3(2), 0, dN4(2), 0, dN5(2), 0, dN6(2);
dN1(2), dN1(1), dN2(2), dN2(1), dN3(2), dN3(1), dN4(2), dN4(1), dN5(2),
    dN5(1), dN6(2), dN6(1)];

```

```
% Initializing 3 empty matrices for 3 gauss points
Be = sym(zeros(size(B,1),size(B,2),3));
Fisop = sym(zeros(size(Fisopeeee,1),size(Fisopeeee,2),3));

% Computing symbolic B-matrix for all three gauss points
for i = 1:3
    Be(:,:,i) = subs(B,xi,IP(:,i));
    Fisop(:,:,i) = subs(Fisopeeee,xi,IP(:,i));
end

% Creating function file
matlabFunction(Be,Fisop,'File','Be_quad_func','Vars',{xe1,xe2,xe3,xe4,
   xe5,xe6});
```

QuadTriang function

```
function [fe_int,Ke,fe_ext] = QuadTriang(ed,Ex,Ey,De,d)

Ke = zeros(12);
fe_int = zeros(12,1);
H = 1/6;

[Be,Fisop] = Be_quad_func([Ex(1) Ey(1)]',[Ex(2) Ey(2)]',...
    [Ex(3) Ey(3)]',[Ex(4) Ey(4)]',[Ex(5) Ey(5)]',...
    [Ex(6) Ey(6)]');

for i = 1:3
    % Calculation of strain and stress
    % Et = Be(:,:,i)*ed;
    % Es = De*Be(:,:,i) * ed;

    Ke = Ke + Be(:,:,i).' * De*Be(:,:,i)*H*d*det(Fisop(:,:,i));
    % fe_int = fe_int + Be(:,:,i).'*Es * d* det(Fisop(:,:,i))*H;
end

fe_int = Ke*ed;
fe_ext = zeros(12,1);

% Calculate D-matrix using CALFEM
% mpar.Emod = 200.e3; % Youngs modulus [MPa]
% mpar.v = 0.3; % Poisson's ratio [-]
% ptype=1; %ptype=1: plane stress 2: plane strain, 3:axisym, 4: 3d
% D=hooke(ptype,mpar.Emod,mpar.v); % Constitutive matrix - plane stress

end
```

A.3 Task 1c - Main File

```
%% Material model parameters
g = 9.81; % Gravitational acceleration
mpar.Emod=210e9; % Young's modulus [Pa]
mpar.Sy=500e6; % Sigma yield [Pa]
mpar.Hmod=40000E6; % H-modulus [Pa]
mpar.nu = 0.3;
dens = 7850;
alpha = 10e-6; % Thermal expansion coefficient (K^-1)


% Gravitation (1=yes; 0=no)
grav = 0;

% External force P acting on node 3
P = 0; % Force in [N]

% D matrix of material properties for plane strain
% ptype=1: plane stress ||| 2: plane strain ||| 3:axisym ||| 4: 3d
ptype=2;
De =hooke(ptype,mpar.Emod,mpar.nu); % Constitutive matrix - plane strain
% De = (mpar.Emod / ((1 + nu) * (1 - 2*nu))) * [1-nu, nu, 0; nu, 1-nu,
    0; 0, 0, (1-2*nu)/2];

% Length
L = 2;

% Height
h = 2;

% Depth
d = 1;

% Element connectivity Edof
Edof = [1 1 2 5 6 13 14 11 12 15 16 17 18;...
    2 1 2 3 4 5 6 7 8 9 10 11 12];

% No. of elements
nelem = size(Edof,1);

% Position of nodes
Coord = [0 0; L 0; L h; L/2 0; L h/2; L/2 h/2; 0 h; L/2 h; 0 h/2];
nnodes = size(Coord,1);
ndofs = 2*nnodes;

% DoFs in each node
dof = zeros(nnodes,2);
for i = 1:nnodes
    dof(i,1) = 2*i -1;
    dof(i,2) = 2*i;
end

% Compute Ex and Ey from CALFEM routine
[Ex,Ey]=coordxtr(Edof,Coord,dof,6);

% Initialize displacements
```

```
a=zeros(ndofs,1);
aold=zeros(ndofs,1);  %old displacements (from previous timestep)
da=a-aold;


% Define free dofs and constrained dofs
dof_F=[1:ndofs];
dof_C=[1 2 3 4 5 6 7 8 9 10 13 14 15 16 17 18];
dof_F(dof_C) = []; %removing the prescribed dofs from dof_F

% Time stepping
ntime=100; %number of timesteps
tend=ntime; %end of time [s]
t=linspace(0,tend,ntime);

% Test run
a_total = zeros(ndofs,ntime);

% Initialize variables for post processing
K = spalloc(ndofs,ndofs,20*ndofs); % defines K as a sparse matrix and sets the size
                                   % to (ndof x ndof) with initial zero value
                                   % No. of estimated non-zero entries is 20*ndofs

% Initialize internal and external force
fint = zeros(ndofs,1);
fext = fint;
F=zeros(1,ntime);

% Vector of applied external forces
f_ext = fint;
f_ext(6) = P;

% Temperature control
Tmin = 0;
Tmax = 10;
T = linspace(Tmin,Tmax,ntime);
dT = 0;

%tolerance value for Newton iteration
tol=1e-6;

%initialize state variables
no_state=3;
state=zeros(no_state, 3, nelem);
state_old=zeros(no_state, 3, nelem); %old state variables
state_old(2,:,:) = mpar.Sy;

% Initializing stress and strain matrices
stress_old = zeros(4, 3, nelem);
stress = stress_old;

niter_total = 0;

%-------------------------------------------------
% Newton iteration for solving Non-Linear problem
```

```
110 %-----------------------------------------------------
111
112 for i=1:ntime
113     % Initial guess of unknown displacement field
114     a(dof_F)=aold(dof_F)+da(dof_F);
115
116     if i>1
117         dT = T(i)-T(i-1);
118     end
119
120
121     % Newton iteration to find unknown displacements
122     unbal=1e10; niter=0;
123     while unbal > tol
124         K=K.*0;
125         fint=fint.*0; %nullify
126         fext = f_ext;
127         % fext(6) = PP(i);
128
129         % Loop over elements
130         for iel=1:nelem
131
132             % Extract element diplacements
133             ed=a(Edof(iel,2:end));
134             d_ae = a(Edof(iel,2:end)) - aold(Edof(iel,2:end));
135
136             % Element calculations for
137             [fe_int, Ke,fe_ext, state_new, stress_new] = ThermQuadTriang
    (ed,...
138                 d_ae,Ex(iel,:),Ey(iel,:),De,d,state_old(:,:,iel),...
139                 stress_old(:,:,iel), mpar, T(i),dT,alpha);
140
141             % Assembling
142             fint(Edof(iel,2:end))=fint(Edof(iel,2:end))+fe_int;
143             fext(Edof(iel,2:end))=fext(Edof(iel,2:end))+fe_ext;
144
145             K(Edof(iel,2:end),Edof(iel,2:end))=...
146                 K(Edof(iel,2:end),Edof(iel,2:end))+Ke;
147
148
149             % temporarily updating state
150             state(:,:,iel) = state_new;
151             stress(:,:,iel) = stress_new;
152
153         end
154         % Unbalance equation
155         g_F=fint(dof_F)-fext(dof_F);
156
157         unbal=norm(g_F);
158         if unbal > tol
159             % Newton update
160             a(dof_F)=a(dof_F)-K(dof_F,dof_F)\g_F;
161         end
162
163         niter=niter+1; %update iter counter
164         [niter unbal]  %print on screen
165
166         if niter>400
```

```
167             disp('no convergence in Newton iteration')
168             break
169         end
170 
171         % f_extC = fint(dof_C)- fext(dof_C);
172 
173 
174     end
175 
176     niter_total = cat(1,niter_total,niter);
177     F(i) = -fint(5);
178 
179     stress_old = stress;
180     state_old = state;
181     da=a-aold;
182     aold = a;
183     a_total(:,i) = a;
184     % plot(T,F,'-') %for plotting during simulation
185     % drawnow
186 end
187 
188 close all
189 plot(T,F,'linewidth',2)
190 set(gca,'FontSize',14,'fontname','Times New Roman')
191 xlabel('$\Delta T$ [K]','FontSize',16,'interpreter','latex')
192 ylabel('$F$ [N]','FontSize',16,'interpreter','latex')
193 grid on
194 %
195 % P = F(end);
196 
197 stress_total = zeros(4,1,nelem);
198 stress_avg = zeros(nelem,1);
199 for i = 1:nelem
200     stress_total(:,:,i) = (stress(:,1,i) + stress(:,2,i) + stress(:,3,i)
    )/3;
201     stress_avg(i,1) = (stress_total(1,1,i)+stress_total(2,1,i)+
    stress_total(3,1,i))/3;
202 end
203 
204 
205 % Analytical solution of cantilever beam
206 % https://www.engineeringtoolbox.com/cantilever-beams-d_1848.html
207 sigma = De * alpha * (Tmax-Tmin)*[1;1;1;0];
208 F_anal = sigma * h*d;
209 F_FE = fint(5) + fint(9) + fint(3);
```

A.3.1 Task 1c - Function files

ThermQuadTriang

```
function [fe_int,Ke,fe_ext,state,stress] = ThermQuadTriang(ed,da,Ex,Ey,De,...
    d,state_old,stress_old, mpar, T, dT,alpha)

% Gauss weights for quadratic triangles
H = 1/6;

% Initializing a bunch of variables
Ke = zeros(12);
fe_int = zeros(12,1);
state = zeros(size(state_old));
stress = zeros(size(stress_old));

% Calculating area of element
% Ae=Area(Ex(1),Ey(1),Ex(2),Ey(2),Ex(3),Ey(3));

% Determining B-matrix and Fisop-matrix for all three gauss points
[Be,Fisop] = Be_quad_func([Ex(1) Ey(1)]',[Ex(2) Ey(2)]',...
    [Ex(3) Ey(3)]',[Ex(4) Ey(4)]',[Ex(5) Ey(5)]',...
    [Ex(6) Ey(6)]');
sizeBe = zeros(1,size(Be(:,:,1),2));

for i = 1:3

    Be_temp = [Be(1:2,:,i);sizeBe;Be(3,:,i)];
    strain = Be_temp * ed;
    dstrain = Be_temp * da;

    % Compute trial stress (and add zero zz-strain component using D as 4x4 matrix
    % stress_trial = stress_old(:,i) + De*[dstrain(1:3);0];
    stress_trial = stress_old(:,i) + De*(dstrain -alpha*dT*[1;1;1;0]);

    % Compute updated stress and updated state variables from trial stress
    [stress(:,i),deps,state(:,i)]=mises(2,[mpar.Emod;mpar.nu;mpar.Hmod],stress_trial',...
        state_old(:,i)');

    % Compute tangent material behaviour
    dstress_dstrain = dmises(2,[mpar.Emod mpar.nu mpar.Hmod],stress(:,i)',state(:,i)');

    % Ke = Ke + Be(:,:,i).' * De*Be(:,:,i)*H*d*det(Fisop(:,:,i));

    fe_int = fe_int + Be_temp.'* stress(:,i)*d*H*det(Fisop(:,:,i));
    Ke = Ke + Be_temp.' * dstress_dstrain * H * Be_temp * d* det(Fisop(:,:,i));

end

fe_ext = zeros(12,1);

end
```

A.4 Task 1d - Main File

```
%% Material model parameters
mpar.Emod=210e9; % Young's modulus [Pa]
mpar.Sy=500e6; % Sigma yield [Pa]
mpar.Hmod=40000E6; % H-modulus [Pa]
mpar.nu = 0.3;
alpha_s = 10e-6; % Thermal expansion coefficient (K^-1)

mpar2.Emod = 80e9;
mpar2.Sy = 200e6;
mpar2.nu = 0.2;
mpar2.Hmod = 1000e6;
alpha_a = 20e-6;

load("bimetal_coarse.mat");

Coord = 1E-3*Coord;

% External force P acting on node 3
P = 0; % Force in [N]

% D matrix of material properties for plane strain
% ptype=1: plane stress ||| 2: plane strain ||| 3:axisym ||| 4: 3d
ptype=2;
Ds =hooke(ptype,mpar.Emod,mpar.nu); % Constitutive matrix - plane strain
Da =hooke(ptype,mpar2.Emod,mpar2.nu); % Constitutive matrix - plane
    strain

% Length
L = 0.15;

% Height
h = 0.01;

% Depth
d = 0.04;

% No. of elements
nelem = nel;

% DoFs in each node
dof = Dof;

% Compute Ex and Ey from CALFEM routine
[Ex,Ey]=coordxtr(Edof,Coord,dof,6);

% Initialize displacements
a=zeros(ndofs,1);
aold=zeros(ndofs,1);  %old displacements (from previous timestep)
da=a-aold;


% Define free dofs and constrained dofs
dof_F=[1:ndofs];
dof_C = zeros(1,2*size(leftNodes,2));
for i =1: size(leftNodes,2)
```

```
55      dof_C(2*i-1) = 2*leftNodes(i)-1;
56      dof_C(2*i) = 2*leftNodes(i);
57  end
58  dof_F(dof_C) = []; %removing the prescribed dofs from dof_F
59
60  % Time stepping
61  % -----------------------------------------------------
62  ntime=100; %number of timesteps
63  tend=ntime; %end of time [s]
64  t=linspace(0,tend,ntime);
65  % -----------------------------------------------------
66
67  % Test run
68  a_total = zeros(ndofs,ntime);
69
70  % Initialize variables for post processing
71  K = spalloc(ndofs,ndofs,20*ndofs); % defines K as a sparse matrix and
        sets the size
72                                     % to (ndof x ndof) with initial zero
        value
73                                     % No. of estimated non-zero entries
        is 20*ndofs
74
75  % Initialize internal and external force
76  fint = zeros(ndofs,1);
77  fext = fint;
78  F=zeros(1,ntime);
79
80  % Vector of applied external forces
81  f_ext = fint;
82
83  % Temperature control
84  Tmin = 0;
85  Tmax = 300;
86  RiseFall = 1;
87  if RiseFall == 1
88      T1 = linspace(Tmin,Tmax,ntime/2);
89      T2 = linspace(Tmax,Tmin,ntime/2);
90      T = cat(2,T1,T2);
91  else
92      T = linspace(Tmin,Tmax,ntime);
93  end
94  dT = 0;
95
96  %tolerance value for Newton iteration
97  tol=1e-6;
98
99  %initialize state variables
100 no_state=3;
101 state=zeros(no_state, 3, nelem);
102 state_old=zeros(no_state, 3, nelem); %old state variables
103 state_old(2,:,:) = mpar.Sy;
104
105 for i = 1:nelem
106     if max(Ey(i,:))>0.01
107         state_old(2,:,i) = mpar.Sy;
108     else
109         state_old(2,:,i) = mpar2.Sy;
```

```
    end
end

% Initializing stress and strain matrices
stress_old = zeros(4, 3, nelem);
stress = stress_old;

niter_total = 0;

%-----------------------------------------------------
% Newton iteration for solving Non-Linear problem
%-----------------------------------------------------

for i=1:ntime
    % Initial guess of unknown displacement field
    a(dof_F)=aold(dof_F)+da(dof_F);

    if i>1
        dT = T(i)-T(i-1);
    end


    % Newton iteration to find unknown displacements
    unbal=1e10; niter=0;
    while unbal > tol
        K=K.*0;
        fint=fint.*0; %nullify
        fext = f_ext;
        % fext(6) = PP(i);

        % Loop over elements
        for iel=1:nelem

            % Extract element diplacements
            ed=a(Edof(iel,2:end));
            d_ae = a(Edof(iel,2:end)) - aold(Edof(iel,2:end));

            % Element calculations for
            [fe_int, Ke,fe_ext, state_new, stress_new] =
   BiThermQuadTriang(ed,...
                d_ae,Ex(iel,:),Ey(iel,:),Ds,Da,d,state_old(:,:,iel),...
                stress_old(:,:,iel), mpar, mpar2,dT,alpha_s,alpha_a);

            % Assembling
            fint(Edof(iel,2:end))=fint(Edof(iel,2:end))+fe_int;
            fext(Edof(iel,2:end))=fext(Edof(iel,2:end))+fe_ext;

            K(Edof(iel,2:end),Edof(iel,2:end))=...
                K(Edof(iel,2:end),Edof(iel,2:end))+Ke;


            % temporarily updating state
            state(:,:,iel) = state_new;
            stress(:,:,iel) = stress_new;

        end
        % Unbalance equation
        g_F=fint(dof_F)-fext(dof_F);
```

```
167
168         unbal=norm(g_F);
169         if unbal > tol
170             % Newton update
171             a(dof_F)=a(dof_F)-K(dof_F,dof_F)\g_F;
172         end
173
174         niter=niter+1; %update iter counter
175         [niter unbal]  %print on screen
176
177         if niter>40
178             disp('no convergence in Newton iteration')
179             break
180         end
181
182         % f_extC = fint(dof_C)- fext(dof_C);
183
184
185     end
186
187     niter_total = cat(1,niter_total,niter);
188     F(i) = a(10);
189
190     stress_old = stress;
191     state_old = state;
192     da=a-aold;
193     aold = a;
194     a_total(:,i) = a;
195     % plot(T,F,'-') %for plotting during simulation
196     % drawnow
197 end
198
199 close all
200 plot(T(1:end/2),F(1:end/2),'linewidth',2)
201 hold on;
202 plot(T(end/2:end),F(end/2:end),'linewidth',2)
203 set(gca,'FontSize',14,'fontname','Times New Roman')
204 xlabel('$\Delta T$ [K]','FontSize',16,'interpreter','latex')
205 ylabel('$a$ [m]','FontSize',16,'interpreter','latex')
206 grid on
207
208 stress_total = zeros(4,1,nelem);
209 stress_avg = zeros(nelem,1);
210 for i = 1:nelem
211     stress_total(:,:,i) = stress(:,1,i) + stress(:,2,i) + stress(:,3,i);
212     stress_avg(i,1) = (stress_total(1,1,i)+stress_total(2,1,i)+
    stress_total(3,1,i))/3;
213 end
214
215 figure
216 niter_total(1) = [];
217 plot(t,niter_total)
218 xlabel('Time step')
219 ylabel('Number of iterations')
220
221 figure
222 plot(t,T)
223 xlabel('Time step','FontSize',16,'interpreter','latex')
```

```
224 ylabel('$\Delta T$ [K]','FontSize',16,'interpreter','latex')
225
226 maxDisp = zeros(1,ntime);
227 for i = 1:ntime
228     maxDisp(i) = max(abs(a_total(:,i)));
229 end
230
231 figure
232 plot(T,maxDisp,'linewidth',2)
233 set(gca,'FontSize',14,'fontname','Times New Roman')
234 xlabel('$\Delta T$ [K]','FontSize',16,'interpreter','latex')
235 ylabel('$a$ [m]','FontSize',16,'interpreter','latex')
236 grid on
```

A.4.1 Task 1d - Function files

```
function [fe_int,Ke,fe_ext,state,stress] = BiThermQuadTriang(ed,da,Ex,Ey
    ,Ds,Da,...
    d,state_old,stress_old, mpar1, mpar2, dT,alpha_s,alpha_a)


if max(Ey)>0.01
    mpar = mpar1;
    alpha = alpha_s;
    De = Ds;
else
    mpar = mpar2;
    alpha = alpha_a;
    De = Da;
end

% Gauss weights for quadratic triangles
H = 1/6;

% Initializing a bunch of variables
Ke = zeros(12);
fe_int = zeros(12,1);
state = zeros(size(state_old));
stress = zeros(size(stress_old));

% Calculating area of element
% Ae=Area(Ex(1),Ey(1),Ex(2),Ey(2),Ex(3),Ey(3));

% Determining B-matrix and Fisop-matrix for all three gauss points
[Be,Fisop] = Be_quad_func([Ex(1) Ey(1)]',[Ex(2) Ey(2)]',...
    [Ex(3) Ey(3)]',[Ex(4) Ey(4)]',[Ex(5) Ey(5)]',...
    [Ex(6) Ey(6)]');
sizeBe = zeros(1,size(Be(:,:,1),2));

for i = 1:3

    Be_temp = [Be(1:2,:,i);sizeBe;Be(3,:,i)];
    strain = Be_temp * ed;
    dstrain = Be_temp * da;

    % Compute trial stress (and add zero zz-strain component using D as
    4x4 matrix
    % stress_trial = stress_old(:,i) + De*[dstrain(1:3);0];
    stress_trial = stress_old(:,i) + De*(dstrain -alpha*dT*[1;1;1;0]);

    % Compute updated stress and updated state variables from trial
    stress
    [stress(:,i),deps,state(:,i)]=mises(2,[mpar.Emod;mpar.nu;mpar.Hmod],
    stress_trial',...
        state_old(:,i)');

    % Compute tangent material behaviour
    dstress_dstrain = dmises(2,[mpar.Emod mpar.nu mpar.Hmod],stress(:,i)
    ',state(:,i)');

    % Ke = Ke + Be(:,:,i).' * De*Be(:,:,i)*H*d*det(Fisop(:,:,i));

```

```
52      fe_int = fe_int + Be_temp.'* stress(:,i)*d*H*det(Fisop(:,:,i));
53      Ke = Ke + Be_temp.' * dstress_dstrain * H * Be_temp* d* det(Fisop
        (:,:,i));
54
55  end
56
57  fe_ext = zeros(12,1);
58
59  end
```