CHALMERS
UNIVERSITY OF TECHNOLOGY

TME245 – Finite element method - structures
Department: Mechanics and Maritime Sciences
Mobility Engineering, Master of Science Programme

Computer Assignment 1

Pietro Nardon

7/02/2024

TABLE OF CONTENTS

1 TASK 1: ELASTOPLASTIC ANALYSIS USING ABAQUS

The elastoplastic analysis of the square-shaped steel block through the program Abaqus is going to be focused on two different load cases:

- Uniaxial in-plane loading in the x-direction (From here on called "LC1")
- Uniaxial in-plane loading in the y-direction (From here on called "LC2")

1.1 BOUNDARY CONDITIONS (BC)

In order to achieve the uniaxial in-plane loading it was necessary to utilize precise boundary conditions. A roller was introduced along the whole side of the plate, in the direction of the loading, and a single roller was applied to one of the cornering nodes. This set of boundary conditions was applied to achieve the uniaxial in-plane loading, while at the same time it doesn't allow for a rigid translation in the perpendicular direction, without over-constraining the system. The rigid rotation of the whole plate was locked thanks to the already mentioned set of rollers along the side. Furthermore, to impose the increasing displacement on the plates a displacement boundary condition was selected along the studied direction, on the opposite side of the plate respect to where the set of rollers was positioned, in this way a uniaxial loading case could be described.

This translates into:

- **LC1** Figure 1:
    1. Set of rollers along the x direction, on the left side of the plate.
    2. Single roller along the y direction, on the upper left node of the plate.
    3. Imposed displacement along the positive x direction, on the right side of the plate.

- **LC2** Figure 2:
    1. Set of rollers along the y direction, on the bottom side of the plate.
    2. Single roller along the x direction, on the bottom right node of the plate.
    3. Imposed displacement along the positive y direction, on the upper side of the plate.

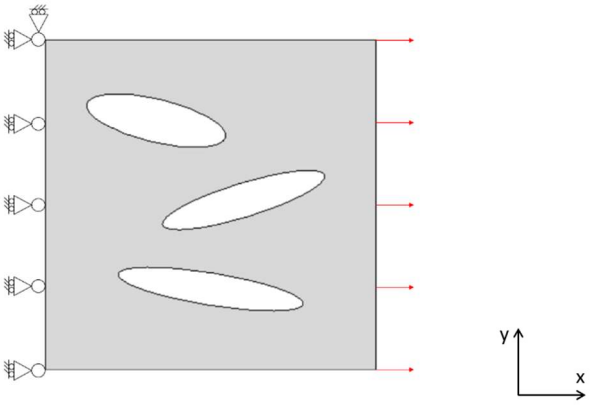

Figure 1 – Boundary conditions of Load Case 1

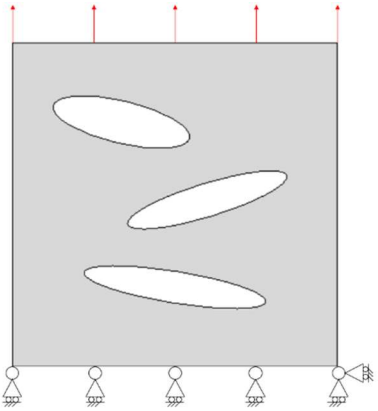

Figure 2 - Boundary conditions of Load Case 2

1.2 DEFINITION OF THE FE MODEL

Starting from the given geometrical model, the complete finite element model was created.

- **Material definition:**

  The material elastic behaviour was defined by inserting into Abaqus:

  $$\mathrm{E} = 210 \cdot 10^3\ MPa \text{ and } \nu = 0.3$$

  While the plastic properties were defined by inserting into Abaqus:

  $$Plastic\ Strain = 0 \quad at \quad \sigma_{yield} = 500\ MPa$$

  $$Plastic\ Strain = 0.2 \quad at \quad \sigma_{yield} = 4500\ MPa$$

- **Section:**

  A section was created and defined as "Solid, Homogeneous" and with a Plane stress/strain thickness equal to the plate's thickness of $10mm$.

- **Assembly:**

  The plate was defined as an instance and was added to an assembly through the "Create Instance" command.

- **Static Load:**

  The static load for each LC was created through the command "step" in Abaqus and was defined with 100 time steps, each one of a constant value of 0.01. This allowed to impose a displacement starting from $0mm$ to the maximum of $1mm$ in 100 steps.

- **Boundary Conditions:**

  The boundary conditions were imposed with the "Create Boundary Condition" command and the type "Displacement/Rotation" was selected with uniform distribution and to all of them was assigned the step already created. Then the desired direction between x, y and rotation was selected and put to $0mm$ for the rollers or $1mm$ for the imposed displacement side, accordingly to the LC studied and the BC defined before.

- **Domain Discretisation**

  The domain was discretised with three different meshes using the same element type, one coarser, one medium and one finer to study how convergence and precision of the results is dependent on the mesh type and size.
  The element type chosen is the "6-node quadratic plane strain triangle", since it's one of the most versatile element type available and fits well in the study, providing a good balance between accuracy, consistency, and computational time. Moreover, is also useful to compare the results obtained through ABAQUS with the ones obtained through MATLAB since it's the same element type that is implemented in the given meshes.

In order to ease the calculation, plane strain approximation is chosen since the thickness of the plate is of comparable magnitude and is not considerably smaller compared to the other two dimensions.
The size of the elements has been imposed through the "Seed Part" command, choosing different global size as seen in Table 1. The choice of this numbers was dictated by the number of elements that the mesh would have in the end. The mesh with size 110 is generated with 486 elements which is a comparable with the given mesh for the MATLAB part called "quadraticFine2024" which has 485 elements. The finer mesh was generated to have a better view and general understanding of the experiment since it was visually better and clearer to understand and study.

Element Global Size	Element Number
110	486
60	766
10	2640

Table 1 – Element dimensions

1.3 LOAD CASES

To check if the solution has converged with respect to the spatial discretisation both the reaction force and the stress distribution were analysed. Three different meshes were created with the same element type to study if the values were consistent and tended to the same results regardless of the mesh's element size and number.

In Figure 3 the Reaction Force-Displacement relation for the LC2 is plotted for the three different meshes. It is clear that the different solutions tend to converge to the same value with very small differences.

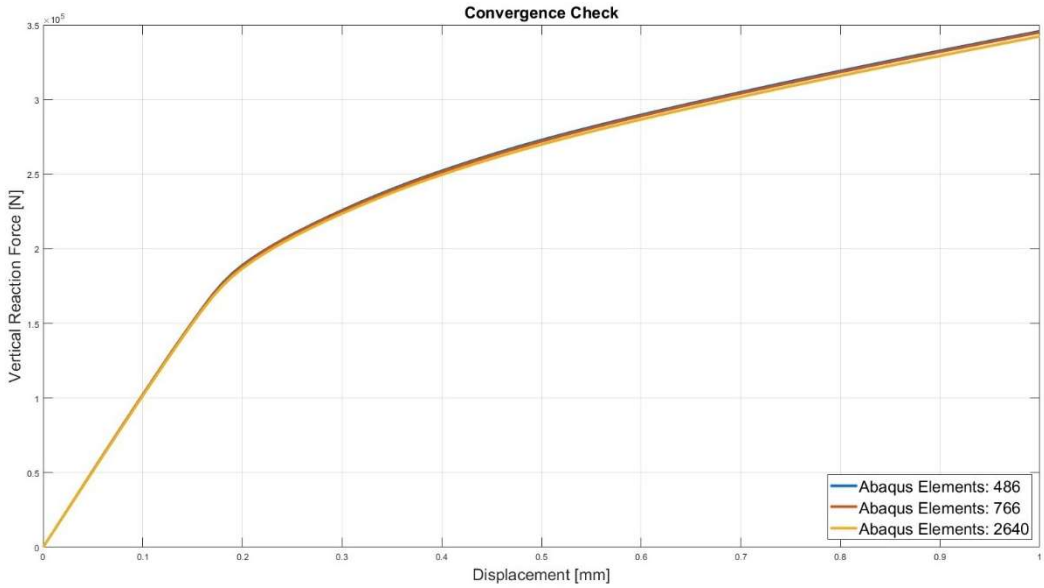

Figure 3 – Force convergence of the mesh

In Figure 4 is shown the Von Mises stress distribution in the LC2 for the three different meshes in the same colorscale, increasing the number of elements from left to right. The plots show the same distribution of the stresses with the same values in all the regions, regardless of the number of elements of the mesh.

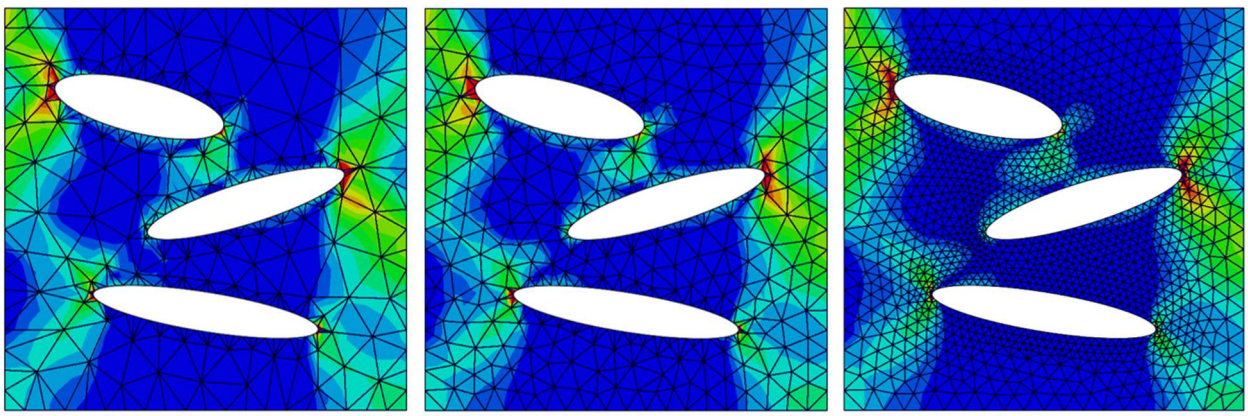

Figure 4 – Stress Convergence of the mesh

Figure 3 and Figure 4 demonstrate that the solution has converged with respect to the spatial distribution since, regardless of the number of elements of the mesh, the solutions were always equal. The same results and conclusions could be found by analysing the LC1.

1.3.1 Load Case 1 Results

The results here shown are obtained with the mesh containing 2640 elements and the colours limits and scales were adjusted to have a better visibility. To ensure comparability between the two Load Cases it was decided to use the same colour-scales with the same upper and lower limits for the different plots.

1.3.1.1 Von Mises stress distribution

In Figure 5 is displayed the Von Mises stress distribution regarding the LC1. As it can be expected the higher stresses are located in proximity of the plate's holes, in particular in the regions where the vertical distances between each hole is smaller. In these regions there is less material along the vertical direction and, since the prescribed displacement is along the horizontal direction, this will result into a higher stiffness and higher stress concentrations.

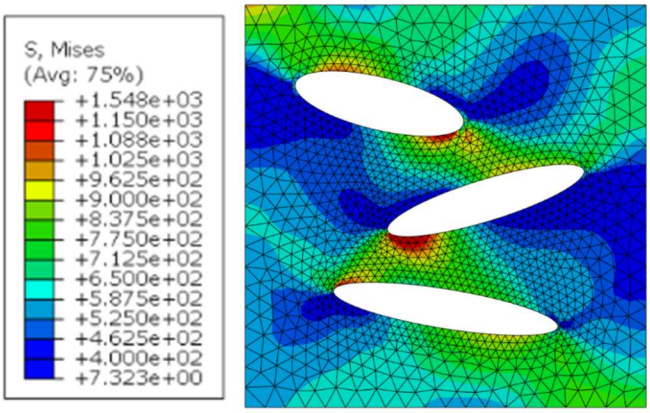

Figure 5 – LC1: Von Mises Stress

1.3.1.2 Equivalent Plastic Strain (PEEQ)

In Figure 6 is displayed the Equivalent Plastic Strain distribution regarding the LC1. It is clearly visible that the direction of the plastic deformation depends on the orientation of the plate's holes, and it is present in particular in the zones where there is less material between one hole and the next one, since we have the higher stresses as described in (1.3.1.1).

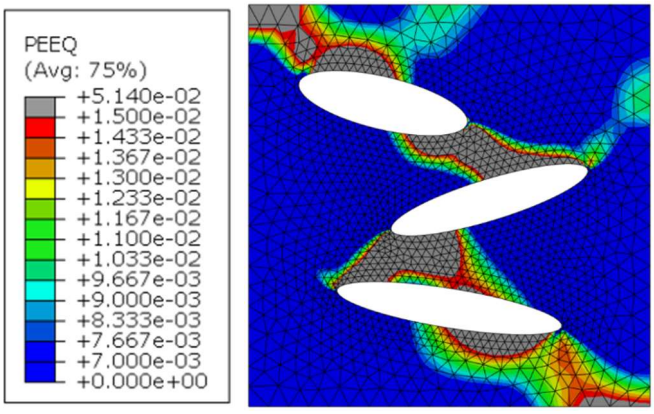

Figure 6 – LC1: PEEQ

1.3.1.3 Force-Displacement Relation

In Figure 7 is shown the relationship between the reaction force at the boundary where the varying prescribed displacement is imposed. The classic elastoplastic behaviour of the material can be seen in the graph, at around $0.2mm$ the elastic phase ends and the plastic deformation begins.

The value of the reaction force at the maximum displacement is: $RF1 = 572168\ N$

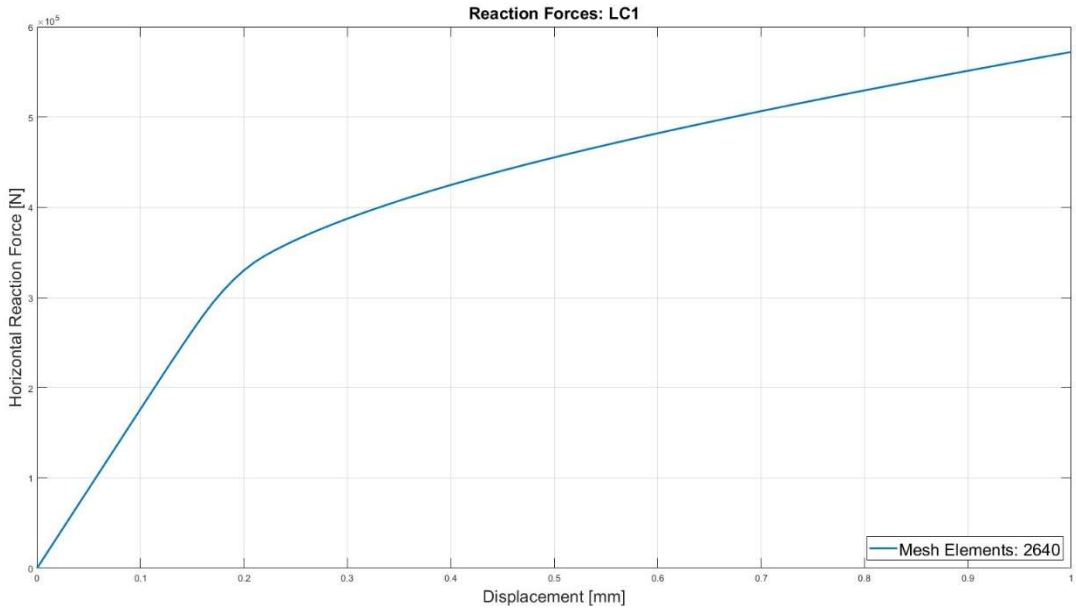

Figure 7 – LC1: Force-Displacement Relation

1.3.2 Load Case 2 Results

The results here shown are obtained with the mesh containing 2640 elements and the colours limits and scales were adjusted to have a better visibility.

1.3.2.1 Von Mises stress distribution

In Figure 8 is displayed the Von Mises stress distribution regarding the LC2. The stresses are mainly concentrated in the areas where the holes are closer to the edges of the plate. In these areas the material on the horizontal direction is less and, since we are imposing a vertical displacement, these areas will be subjected to more stress since the stiffness will be higher.

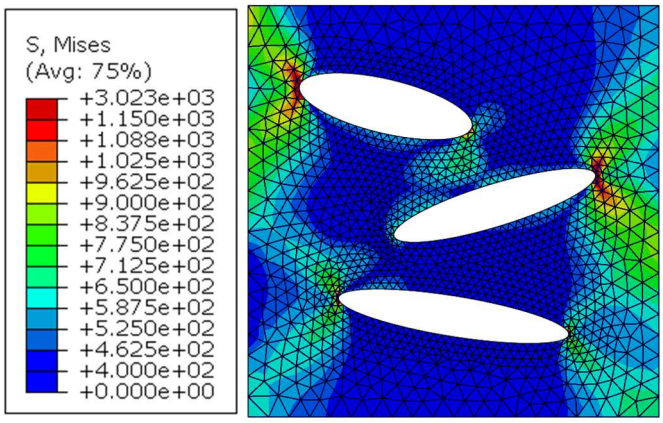

Figure 8 – LC2: Von Mises Stress

1.3.2.2 Equivalent Plastic Strain (PEEQ)

In Figure 9 the Equivalent Plastic Strain distribution is represented and, similarly to the LC1 case, the regions where there is more deformation are the ones subjected to the higher Von Mises stresses, where there is less material.

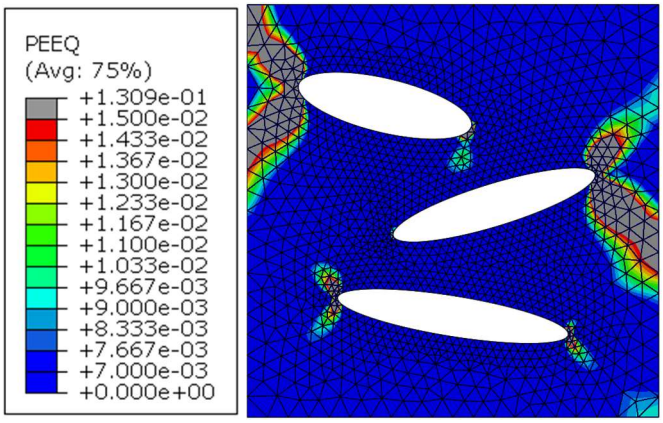

Figure 9 – LC2: PEEQ

1.3.2.3 Force-Displacement Relation

In Figure 10 is shown the relation between the reaction Force on the displaced boundary and the Displacement. Similarly to the LC1 the trend is the typical one for an elastoplastic material, but the final value is considerably smaller, where

At maximum displacement the Reaction Force is: $RF2 = 342317\ N$

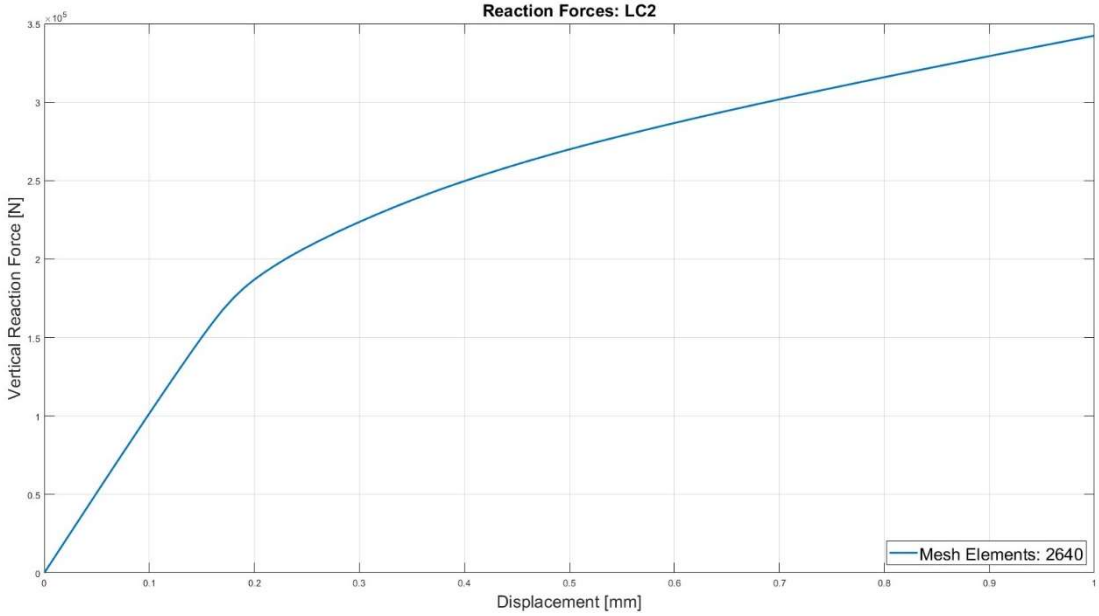

Figure 10 – LC2: Force-Displacement Relation

The reason why the two reaction forces for the two different Load Cases are so much different is to be investigated in how the holes are positioned. The holes' orientation changes considerably the quantity of material that will oppose to the prescribed displacement, thus it will change the stiffness of the plate in that particular direction. Since the orientation of the holes is mainly horizontal, in the LC2 the material will have a lower stiffness, resulting in lower reaction forces and higher maximum Von Mises stresses and higher maximum deformations all concentrated in smaller areas of the plate, respect to the LC1.

2 TASK 2: ELASTIC AND ELASTOPLASTIC ANALYSIS USING MATLAB

An elastic and elastoplastic analysis was carried on using MATLAB. The meshes of the plate were given and the analysis was focused only on the Load Case 2, so only the case of vertical imposed displacement was considered.

The assumption of plane strain condition was done, as also described in (1.2). In the different scripts this conditions was implemented either by introducing directly the corresponding $[3x3]$ Constitutive Matrix as shown for example in (3.1) or by using the CALFEM function **`hooke`** and selecting **`ptype = 2`** as for example in (3.4).

In order to compute all the calculations for every mesh type a **`for`** cycle was introduced at the beginning of the script. This loop extends for almost the entire script and allows to use every mesh available at the same time and have all the results at the same moment to analyse them faster.

At the beginning of every script, inside the **`for`** cycle to compute all the meshes, an **`if`** and **`elseif`** structure is present to load the correct mesh and its name. Subsequently a few lines of code are utilized to create the **`Ex`** and **`Ey`** matrices that contain the coordinates of every degree of freedom associated with every element.

The weight force was imposed null in every study.

The forces along the vertical degree of freedom of the nodes from the top side of the plate are extracted from the vector of the external forces and then are all summed up into a single value.

2.1 LINEAR ELASTIC BEHAVIOUR

To compute the behaviour of the material assuming it to be linear elastic the old code from the CE1 was modified. Apart from the implementations previously described the main difference respect to the older code is the definition of the boundary conditions. Like in the LC2 Abaqus part the nodes at the plate's bottom side were constrained in the vertical direction, and the bottom right node was also constrained in the horizontal direction. To describe the prescribed displacement the top nodes position was imposed fixed at $1mm$. Then an iterative cycle passing for each element builds the stiffness matrix and load vector (which will be all zeros since we don't consider the weight force). After the cycle, the displacements were found through the resolution of the following system of equation:

$$\begin{cases} \underline{a}_F = K_{FF}^{-1}(\underline{f}_F - K_{FC}\underline{a}_C) \\ \underline{f}_c = K_{CF}\underline{a}_f + K_{CC}\underline{a}_c \end{cases}$$

The Reaction Force computed as described before are summarised in Table 2:

Mesh	Vertical Reaction Force
Coarse (209)	1189916.9825 N
Medium (320)	1112485.1256 N

Table 2 – Reaction Forces Linear

In Figure 11 is shown the deformation of the plate with the coarse mesh while in Figure 12 is shown the deformation of the plate with the medium mesh. The black contious line represents the orignal mesh and the red dashed line represents the deformed mesh after imposing the prescribed displacement (Scale Factor = 1.5).

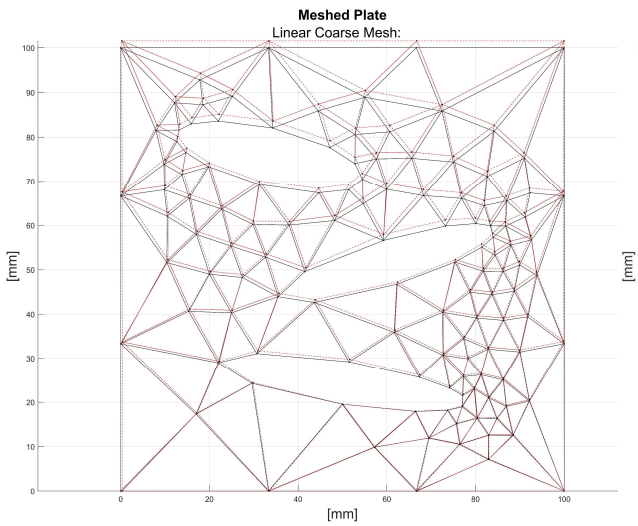

Figure 11 – Linear Coarse Mesh Deformation

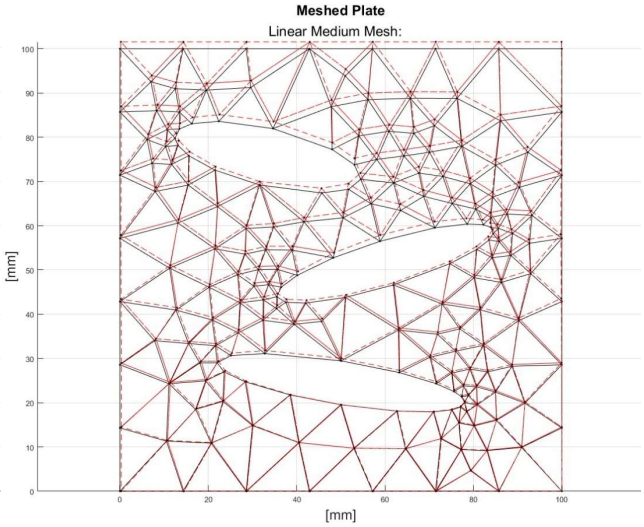

Figure 12– Linear Medium Mesh Deformation

2.2 QUADRATIC ISOPARAMETRIC 6-NODED TRIANGULAR ELEMENTS

To implement the isoparametric 6-noded triangular elements into the code it was necessary to change the routine to compute the element Stiffness Matrix and the element Load Vector. This was done through the function **`node6_isop_stiff_load`** (3.2.1). This function computes the Stiffness Matrix and the Load Vector of the element through a 3-point Gauss integration over the quadratic isoparametric 6-noded triangular elements. In order to compute the B_e matrix and the determinant of the $F_{Isotropic}$ matrix the function **`Be_6node_func`** (3.2.2) is called. This function is a symbolic MATLAB function that computes $det\ (F_{Isotropic})$, B_e matrix, and N_e matrix of the element. Inside it contains the shape functions of the isoparametric 6-noded triangular element and its derivatives.

The results of the Vertical Reaction Forces is shown in Table 3

Mesh	Vertical Reaction Force
Coarse (209)	1046885.1042 N
Medium (320)	1027985.0759 N
Fine (485)	1025432.1412 N

Table 3 – Reaction Forces 6-noded

The results are different from the previous point (2.1) and the reason is to be found in the nature of the shape function of the elements used, since the number of elements is equal. The linear triangular element uses a linear shape function, while the quadratic isoperimetric 6-noded triangle has a quadratic shape function. The higher the order of the shape function, the more precise the approximation is, that's why the values of the reaction forces are different between these two elements. Furthermore, changing the element number of the mesh means changing the accuracy of the results, that's why, even if the element type is the same, the values of the forces are different for different elements number.

In Figure 13, Figure 14 and Figure 15 the mesh deformations for the different mesh sizes are plotted (Scale Factor = 1.5)..

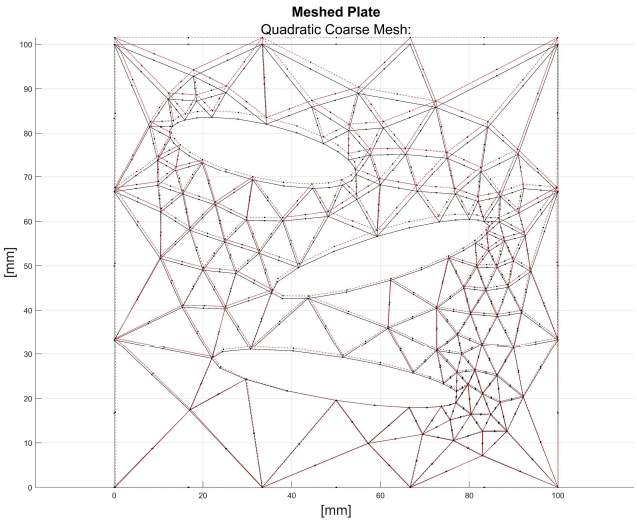

Figure 13 – Quadratic Coarse Mesh Deformation

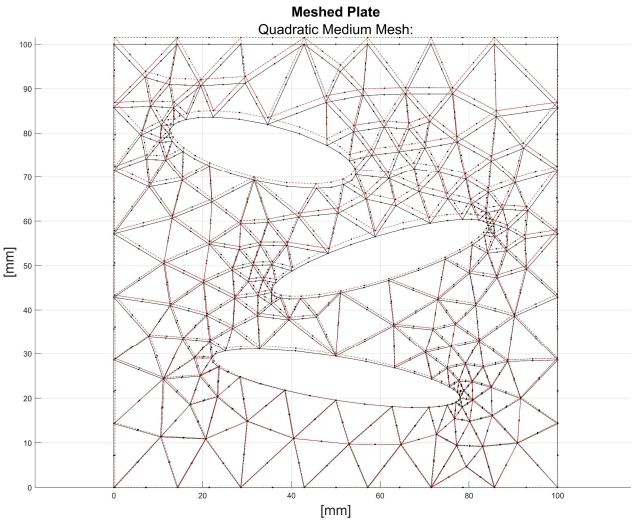

Figure 14 – Quadratic Medium Mesh Deformation

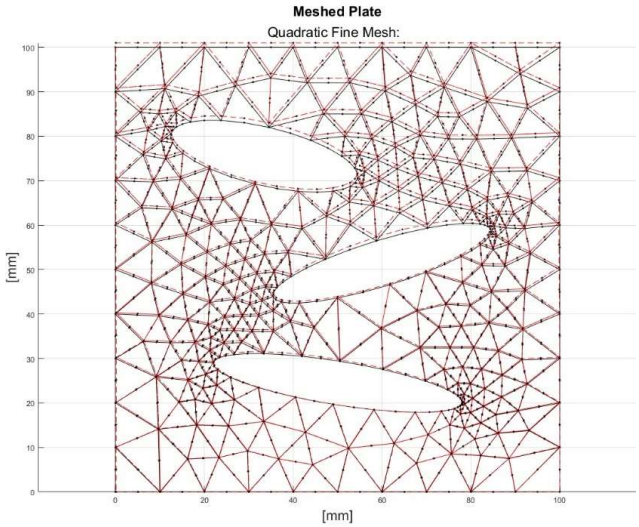

Figure 15 – Quadratic Fine Mesh Deformation

2.3 NONLINEAR FE PROGRAM

To rewrite the elasticity code as a nonlinear element program it was necessary to include a step-wise increment of the displacement at the prescribed boundary and then implement a Newton iterative cycle to update, for each load step, the free displacement of the nodes.

The step wise increment for both the time and the displacement was implemented by inserting the appropriate values in the function **`linspace`** and as can be seen in (3.3).

Then after the necessary initializations a **`for`** cycle was implemented to pass for each time step and, after it, the initial guess for the unknown displacement field was computed. Subsequently the displacements of the prescribed degrees of freedom were updated with the value of the prescribed displacement at that time step.

The Newton iterations were implemented thanks to a **`while`** cycle that stopped when the unbalance was smaller than an arbitrary tolerance. The unbalance was firstly defined as an arbitrary big number and then was computed as the norm of the difference between the internal and external forces, evaluated in the free degrees of freedom at that time step. Inside the **`while`** loop after nullifying the needed variables, another **`for`** cycle was implemented to loop over all the elements of the mesh and compute the Stiffness Matrix and External Force Vector at that particular time step with the function **`node6_isop_stiff_load`** (3.2.1). After the loop, the unbalance was computed, as described before. If the unbalance was bigger than the chosen tolerance then displacements of the free degrees of freedom were updated.

After the **`while`** cycle, the External Forces vector applied to the constrains was computed as described in (2.2) using the displacements of the free degrees of freedom computed with Newton, after it the number of iterations and displacements variable was updated.

The Newton iterations, for every kind of mesh, converge in less than 2 iteration per load step, regardless of the number of load steps you consider. This can be seen in Figure 16, the number of iterations is the same for all the three meshes.

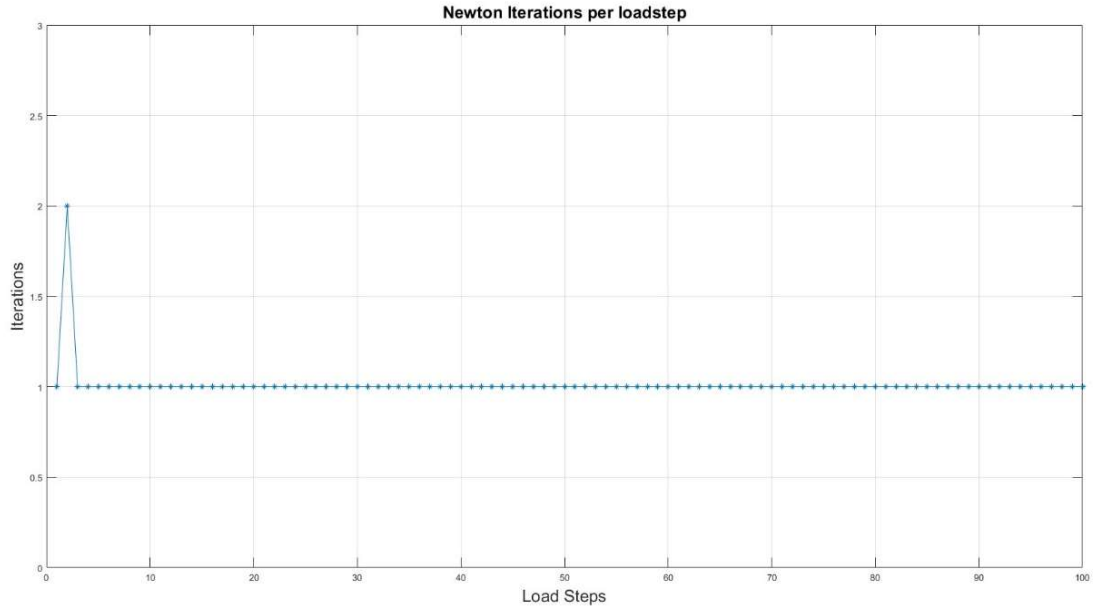

Figure 16 - Number of Newton iterations

In Table 4 is shown the differences between the various methods to compute the vertical reaction force. It's also clear that the Newton iteration can converges well even in just one step since the solution is equal to the one computed with 100 step and with the elastic code. The solutions computed with (2.2) and (2.3) are equal since the material is still supposed elastic in both cases. The difference between the results of the (2.3) with the (2.1) could be explained in the same way as the difference in the results in (2.2) and (2.1).

Mesh	Elastic VRF	NonLinear VRF (100 step)	NonLinear VRF (1 step)
Coarse (209)	1046885.1042 N	1046885.1042 N	1046885.1042 N
Medium (320)	1027985.0759 N	1027985.0759 N	1027985.0759 N
Fine (485)	1025432.1412 N	1025432.1412 N	1025432.1412 N

Table 4 – Comparison between Vertical Reaction Forces

2.4 ELASTOPLASTIC BEHAVIOUR

To account for elastoplastic behaviour of the material the code (3.4) had to be modified. First of all, the introduction of new variables was needed to store the state and the stress experienced by the material and to implement the presence of the yield stress. Then the code is similar to (3.3) until the **`for`** cycle that loops over the elements, in this cycle to compute the element Stiffness Matrix and the element Load Vector a new function called **`node6_isop_non_lin`** (3.4.1) is used.

This function contains a **`for`** cycle that for each load step loops over the Gauss integration points, recalls the function **`Be_6node_func`** (3.2.2) and computes the increment in strain and the trial stress. Then through the function **`mises`** it updates the stresses and the state, and through **`dmises`** computes the derivative of the stress over the strain to be able to finally compute the Element Stiffness Matrix and the Element Internal Force Vector thanks to the Gauss integrations. After assembling K and f_{int} the code is equal to (3.3) except for the computation of the constrain forces. The constrain forces are computed thanks to the unbalance vector that after the Newton iterations will be null.

$$\underline{g}\left(\begin{bmatrix}\underline{a}_F\\ \underline{a}_c\end{bmatrix}\right) = \underline{f}^{int} - \underline{f}^{ext} = \underline{0} \quad \rightarrow \quad \underline{f}^{ext} = \underline{f}^{int} \quad \rightarrow \quad \underline{f}_c = \underline{f}^{ext}(\underline{a}_c)$$

The easier and sufficiently robust way of increasing the boundary displacement is through a linear step, that allows for a constant increment of the displacement and a good consistency in the overall study. Other ways of increasing the boundary displacement are possible and suitable but were not investigated in this study.

To define a suitable number of load steps several trials were made using the finer mesh since it would be the hardest one to compute, the minimum number of time steps to achieve Newton convergence in maximum 20 iterations is 127. Increasing the number of steps meant that the number of Newton iterations needed for each load step diminishes considerably, as can be seen in Figure 17, but due to the increased number of load steps the code gets slower as can be seen in Table 5. In the end all the calculations were carried out at 250 steps.

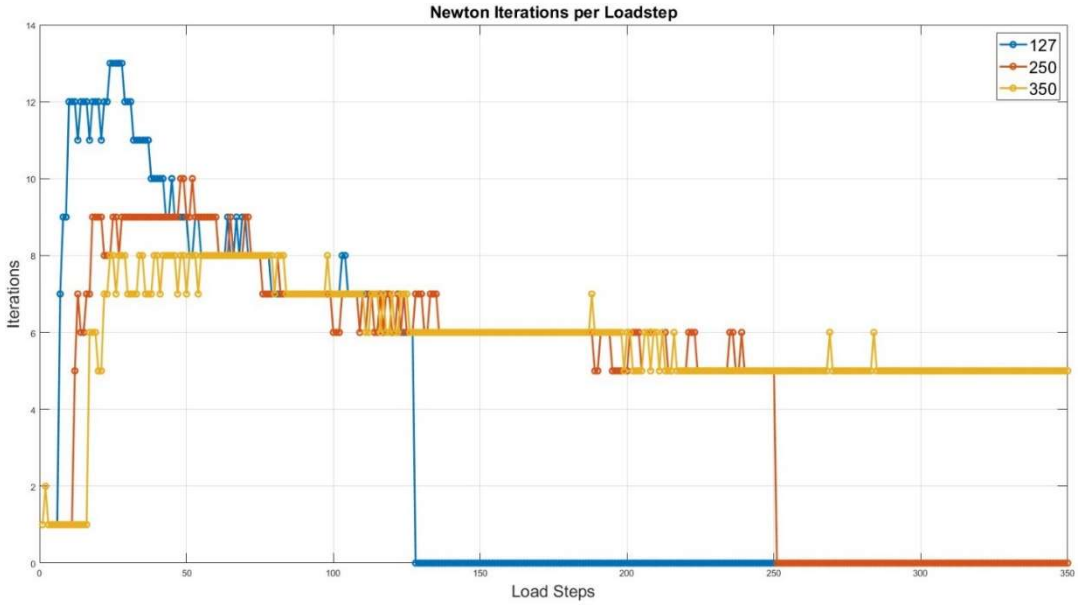

Figure 17 – Newton Iterations for different steps

Steps	Run Time [s]
127	120.764562
130	124.170443
180	147.198045
250	191.131553
350	239.193481
1250	587.040628

Table 5 – Run time

In Figure 18 is shown how the reaction force develops as a function of the increasing prescribed displacement. As already said in (1.3.1.3) the classic elastoplastic behaviour of the material can be seen in the graph, at around $0.2mm$ the elastic phase ends and the plastic deformation begins.
The curve is the same for every kind of given mesh with very little difference in values, as shown in Figure 3 and Table 6. Since the material is now elastoplastic it shows a big drop in the values of the reaction force due to the increasing plasticization. As in all the other points the values of the forces change depending on the element numbers since the accuracy increases.

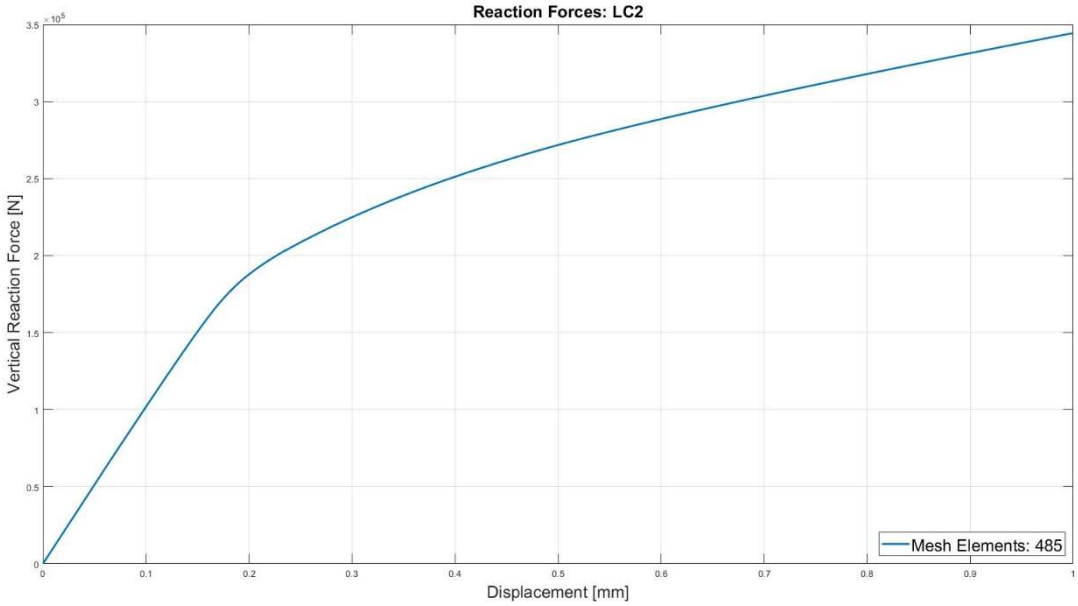

Figure 18 – Vertical Reaction Force

Mesh	Vertical Reaction Force
Coarse (209)	352469.3382 N
Medium (320)	345670.4413 N
Fine (485)	344373.0915 N

Table 6 – Elastoplastic Reaction Forces

These results found in the elastoplastic analysis with MATLAB are coherent with the result found with Abaqus in (1.3.2.3) with the same mesh, as can be seen in Figure 19 thus the results are deemed acceptable and satisfactory for the study.

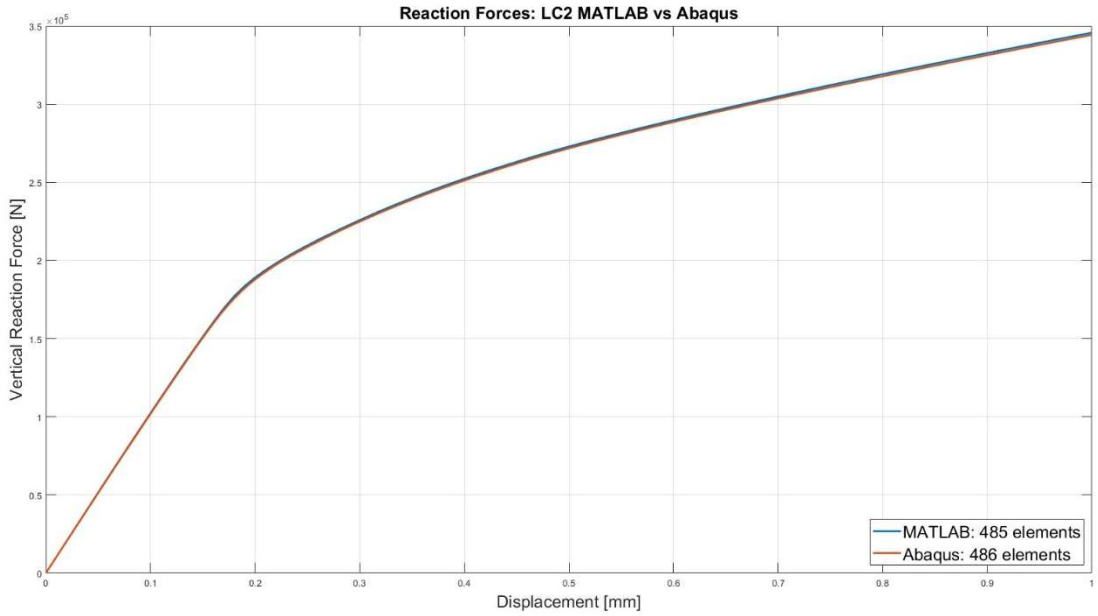

Figure 19 – MATLAB versus Abaqus

3 MATLAB CODE

3.1 LINEAR ELASTIC BEHAVIOUR

```
clc
clear
close all

% Datas
data_output = DataGen_CA1;

L = data_output.L;
H = data_output.H;
h = data_output.h;
E = data_output.E;
nu = data_output.poisson_ratio;
sigma_yield = data_output.sigma_yield;
H_coeff = data_output.H_coeff;

% Vector to plot to screen the results
F_React_LinearFE = zeros(5,1);

% Constitutive Matrix (Plane Strain)
D_Mat = E/((1 + nu)*(1 - 2*nu))*[1 - nu,   nu   ,      0;
                                 nu   , 1 - nu,      0;
                                 0    ,   0    , (1 - 2*nu)/2];

% Choose which mesh to load
for mesh = 1:2      % Select only the linear meshes

if mesh == 1
    load("linearCoarse2024.mat");
    name = {'Linear Coarse Mesh:   '};
elseif mesh == 2
    load("linearMedium2024.mat");
    name = {'Linear Medium Mesh:   '};
elseif mesh == 3
    load("quadraticCoarse2024.mat");
    name = {'Quadratic Coarse Mesh:'};
elseif mesh == 4
    load("quadraticMedium2024.mat");
    name = {'Quadratic Medium Mesh:'};
elseif mesh == 5
    load("quadraticFine2024.mat");
    name = {'Quadratic Fine Mesh:  '};
end


% Element nodes Coordinates
Coord_x = Coord(:,1);
Edof_x = Edof(:, 2:2:end);
Ex = Coord_x(ceil(Edof_x/2));

Coord_y = Coord(:, 2);
```

```
Edof_y = Edof(:, 3:2:end);
Ey = Coord_y(floor(Edof_y/2));


% Show Mesh
figure('WindowState','maximized')
plotpar = [1 1 0];
eldraw2_ext(Ex,Ey,plotpar); grid on
xlabel('[mm]'); ylabel('[mm]');
title('Structure Mesh'); subtitle(name);


% Degrees of Freedom/Constrained (LC2)
n_elem = size(Edof, 1);
n_dof = ndofs;


% Boundary conditions
n_constr = 1 + length(bottomNodes) + length(topNodes);
Dof_Constr = [Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []), reshape(Dof(topNodes,
2)', 1, [])];
Dof_Free = reshape(Dof', 1, []);
Dof_Free([Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []), reshape(Dof(topNodes, 2)',
1, [])])= [];


% Initializations of Stiffness Matrix, Load Vector and Displacement Vector
K = spalloc(n_dof, n_dof, 20*n_dof);
f_ext = zeros(n_dof, 1);
a_constr = zeros(n_constr,1);
a_constr(length(bottomNodes) + 2:end) = 1;          % Imposing a Displacement of 1mm on all the
constrained nodes after the bottom ones

% Iterative cycle passing for each element
for el = 1:n_elem
    % Computing the K and f for each element
    [K_el, f_el] = CST_stiff_load([Ex(el, 1), Ex(el,2), Ex(el,3); Ey(el,1), Ey(el,2), Ey(el,3)],
D_Mat, 0, h);

    % Assemblying
    K(Edof(el, 2:end), Edof(el, 2:end)) = K(Edof(el, 2:end),Edof(el, 2:end)) + K_el;
    f_ext(Edof(el, 2:end)) = f_ext(Edof(el, 2:end)) + f_el;
end


% Solving the system equation
a_free = K(Dof_Free, Dof_Free)\(f_ext(Dof_Free) - K(Dof_Free, Dof_Constr)*a_constr);
f_ext_constr = K(Dof_Constr, Dof_Free)*a_free + K(Dof_Constr, Dof_Constr)*a_constr -
f_ext(Dof_Constr);


% Total Vertical Reaction Force
F_React_LinearFE(mesh) = sum(f_ext_constr((length(bottomNodes) +2): end));
fprintf(['For the ', name{1}, ' Total Vertical Reaction Force is: ',
num2str(F_React_LinearFE(mesh)), ' N\n'])
```

```
% Assemblying solutions
a(Dof_Free) = a_free;
a(Dof_Constr) = a_constr;

Q = zeros(n_dof,1);
Q(Dof_Constr) = f_ext_constr;


% Plot Deformed Shape
Ed = extract_dofs(Edof,a);         % Extract element displacements for plotting
plotpar=[2 4 0];
sfac = 1;                          % Magnification
eldisp2(Ex,Ey,Ed,plotpar,sfac); grid on; title('Meshed Plate');
xlabel('[mm]'); ylabel('[mm]'); axis equal


end
```

3.2 QUADRATIC ISOPARAMETRIC 6-NODED TRIANGULAR ELEMENTS

```
clc
clear
close all

% Datas
data_output = DataGen_CA1;

L = data_output.L;
H = data_output.H;
h = data_output.h;
E = data_output.E;
nu = data_output.poisson_ratio;
sigma_yield = data_output.sigma_yield;
H_coeff = data_output.H_coeff;

% Constitutive Matrix (Plane Strain)
D_Mat = E/((1 + nu)*(1 - 2*nu))*[1 - nu,   nu   ,      0;
                                 nu   , 1 - nu,      0;
                                 0    ,   0    , (1 - 2*nu)/2];

% Choose which mesh to load
for mesh = 3:5


    if mesh == 1
        load("linearCoarse2024.mat");
        name = {'Linear Coarse Mesh:   '};
    elseif mesh == 2
        load("linearMedium2024.mat");
        name = {'Linear Medium Mesh:   '};
    elseif mesh == 3
        load("quadraticCoarse2024.mat");
        name = {'Quadratic Coarse Mesh:'};
    elseif mesh == 4
        load("quadraticMedium2024.mat");
```

```
        name = {'Quadratic Medium Mesh:'};
    elseif mesh == 5
        load("quadraticFine2024.mat");
        name = {'Quadratic Fine Mesh:  '};
    end


    % Element nodes Coordinates
    Coord_x = Coord(:,1);
    Edof_x = Edof(:, 2:2:end);
    Ex = Coord_x(ceil(Edof_x/2));

    Coord_y = Coord(:, 2);
    Edof_y = Edof(:, 3:2:end);
    Ey = Coord_y(floor(Edof_y/2));


    % Degrees of Freedom/Constrained (LC2)
    n_elem = size(Edof, 1);
    n_dof = ndofs;
    n_constr = 1 + length(bottomNodes) + length(topNodes);
    Dof_Constr = [Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []),
reshape(Dof(topNodes, 2)', 1, [])];
    Dof_Free = reshape(Dof', 1, []);
    Dof_Free([Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []), reshape(Dof(topNodes,
2)', 1, [])])= [];


    % Show Mesh
    figure('WindowState','maximized')
    plotpar = [1 1 0];
    eldraw2_ext(Ex,Ey,plotpar); grid on
    xlabel('[mm]', 'FontSize', 18); ylabel('[mm]', 'FontSize', 18);
    title('Structure Mesh', 'FontSize', 18); subtitle(name, 'FontSize', 18);


    % Stiffness matrix and load vector
    K = spalloc(n_dof, n_dof, 20*n_dof);
    f_ext = zeros(n_dof, 1);
    a_constr = zeros(n_constr,1);
    a_constr(length(bottomNodes) + 2:end) = 1;          % Imposing a Displacement of 1mm on all the
constrained nodes after the bottom ones


    % Gravity Load
    F_Body = zeros(n_elem, 2);    % In this case is absent;


    % Iterative cycle to find K and fe_ext
    for el = 1:n_elem
        % Computing the K and f for each element
        [K_el, f_el] = node6_isop_stiff_load(Ex(el,:), Ey(el,:), D_Mat, F_Body(el, :)', h);

        % Assemblying
        K(Edof(el, 2:end), Edof(el, 2:end)) = K(Edof(el, 2:end), Edof(el, 2:end)) + K_el;
        f_ext(Edof(el, 2:end)) = f_ext(Edof(el, 2:end)) + f_el;
    end
```

```
    % Solving the system equation
    a_free = K(Dof_Free, Dof_Free)\(f_ext(Dof_Free) - K(Dof_Free, Dof_Constr)*a_constr);
    f_ext_constr = K(Dof_Constr, Dof_Free)*a_free + K(Dof_Constr, Dof_Constr)*a_constr -
f_ext(Dof_Constr);


    % Assemblying a and Q
    a(Dof_Free) = a_free;
    a(Dof_Constr) = a_constr;
    Q = zeros(n_dof,1);
    Q(Dof_Constr) = f_ext_constr;


    % Plot Deformed Shape
    Ed = extract_dofs(Edof,a);            % Extract element displacements for plotting
    plotpar=[2 4 0];
    sfac = 1;                           % Magnification
    eldisp2_ext(Ex,Ey,Ed,plotpar,sfac); grid on; title('Meshed Plate');
    xlabel('[mm]'); ylabel('[mm]'); axis equal


    % Total Vertical Reaction Force
    F_React_LinearFE(mesh) = sum(f_ext_constr((length(bottomNodes) + 2): end));
    fprintf(['For the ', name{1}, ' Total Vertical Reaction Force is: ',
num2str(F_React_LinearFE(mesh)), ' N\n'])

end
```

3.2.1 node6_isop_stiff_load

```
function [K_el, fel_ext] = node6_isop_stiff_load(ex, ey, D_Mat, F_Body, t)

% xi-coordinates Vector creation
xi_vect = [1/6 1/6 2/3;
           1/6 2/3 1/6];

% Integration Weights vector
H_vect = 1/6*ones(1,3);

% Initializing
K_el = zeros(12);
fel_ext = zeros(12,1);

% Loop over integration points
for ii = 1:3
    H = H_vect(ii);
    xi = xi_vect(:, ii);
    [Be, det_F_Isop, Ne] = Be_6node_func(xi, [ex(1);ey(1)], [ex(2);ey(2)], [ex(3);ey(3)],
[ex(4);ey(4)], [ex(5);ey(5)], [ex(6);ey(6)]);
    K_el = K_el + Be'*D_Mat*Be*det_F_Isop*t*H;
    fel_ext = fel_ext + Ne'*F_Body*det_F_Isop*t*H;
```

```
end
end
```

3.2.2 Be_6node_func

```
% Define Variable
xi = sym('xi', [2, 1], 'real');

% Define shape function
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));

%define N-matrix for the element
Ne = [N1 0 N2 0 N3 0 N4 0 N5 0 N6 0;
      0 N1 0 N2 0 N3 0 N4 0 N5 0 N6];

% Differentiate 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);

% Introduce node position
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');

% Introduce Spatial coordinates as function
x = N1*xe1 + N2*xe2 + N3*xe3 + N4*xe4 + N5*xe5 + N6*xe6;

% Compute Jacobian
F_Isop = jacobian(x, xi);
det_F_Isop = det(F_Isop);

% Spatial Derivatives (Chain Rule)
dN1_dx = simplify(inv(F_Isop)'*dN1_dxi );
dN2_dx = simplify(inv(F_Isop)'*dN2_dxi );
dN3_dx = simplify(inv(F_Isop)'*dN3_dxi );
dN4_dx = simplify(inv(F_Isop)'*dN4_dxi );
dN5_dx = simplify(inv(F_Isop)'*dN5_dxi );
dN6_dx = simplify(inv(F_Isop)'*dN6_dxi );

% B-Matrix of the element
Be = [dN1_dx(1),     0,     dN2_dx(1),     0,     dN3_dx(1),     0,     dN4_dx(1),     0,
dN5_dx(1),     0,     dN6_dx(1),     0;
          0,     dN1_dx(2),     0,     dN2_dx(2),     0,     dN3_dx(2),     0,     dN4_dx(2),
0,     dN5_dx(2),     0,     dN6_dx(2);
```

```
      dN1_dx(2), dN1_dx(1), dN2_dx(2), dN2_dx(1), dN3_dx(2), dN3_dx(1), dN4_dx(2), dN4_dx(1),
dN5_dx(2), dN5_dx(1), dN6_dx(2), dN6_dx(1)];


% Creation of the function
matlabFunction(Be, det_F_Isop, Ne, 'File','Be_6node_func','Vars',{xi,xe1,xe2,xe3,xe4,xe5,xe6});
```

3.3 NonLinear FE Program

```
clc
clear
close all

% Datas
data_output = DataGen_CA1;

L = data_output.L;
H = data_output.H;
h = data_output.h;
E = data_output.E;
nu = data_output.poisson_ratio;
sigma_yield = data_output.sigma_yield;
H_coeff = data_output.H_coeff;

F_React_NonLinearFE = zeros(5, 1);
num_iter = zeros(100,3);

% Constitutive Matrix (Plane Strain)
D_Mat = E/((1 + nu)*(1 - 2*nu))*[1 - nu,   nu   ,      0;
                                 nu   , 1 - nu,      0;
                                 0    ,   0    , (1 - 2*nu)/2];

% Choose which mesh to load
for mesh = 3:5


    if mesh == 1
            load("linearCoarse2024.mat");
            name = {'Linear Coarse Mesh:   '};
    elseif mesh == 2
            load("linearMedium2024.mat");
            name = {'Linear Medium Mesh:   '};
    elseif mesh == 3
            load("quadraticCoarse2024.mat");
            name = {'Quadratic Coarse Mesh:'};
    elseif mesh == 4
            load("quadraticMedium2024.mat");
            name = {'Quadratic Medium Mesh:'};
    elseif mesh == 5
            load("quadraticFine2024.mat");
            name = {'Quadratic Fine Mesh:  '};
    end
```

```
    % Element nodes Coordinates
    Coord_x = Coord(:,1);
    Edof_x = Edof(:, 2:2:end);
    Ex = Coord_x(ceil(Edof_x/2));

    Coord_y = Coord(:, 2);
    Edof_y = Edof(:, 3:2:end);
    Ey = Coord_y(floor(Edof_y/2));


    % Degrees of Freedom/Constrained (LC2)
    n_elem = size(Edof, 1);
    n_dof = ndofs;
    n_constr = 1 + length(bottomNodes) + length(topNodes);
    Dof_Prescribed = reshape(Dof(topNodes, 2)', 1, []);
    Dof_Constr = [Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []),
reshape(Dof(topNodes, 2)', 1, [])];
    Dof_Free = reshape(Dof', 1, []);
    Dof_Free([Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []), reshape(Dof(topNodes,
2)', 1, [])])= [];


    % Initialize Displacements
    a = zeros(n_dof, 1);
    a_old = zeros(n_dof, 1);
    da = a - a_old;
    a_constr = zeros(n_constr,1);



    % Time Stepping
    n_time = 100;               % Number of timesteps
    t_end = 100;
    t = linspace(0, t_end, n_time);

    % Displacement Control
    U_max = 1;                  %[mm]
    Uu = linspace(0, U_max, n_time);

    % Initialize Variables for post processing
    K = spalloc(n_dof, n_dof, 20*n_dof);
    F_react = zeros(size(Uu));

    % Inizialize force vectors
    f_int = zeros(n_dof, 1);
    f_ext = f_int;

    % Gravity Load
    F_Body = zeros(n_elem, 2);    % In this case is absent

    % Tolerance for Newton iterations
    toll = 1e-6;

    figure('WindowState','maximized')
    for ii = 1: n_time
```

```
        % Initial Guess of unknown Displecement field
        a(Dof_Free) = a_old(Dof_Free) + da(Dof_Free);

        % Update Prescribed dof
        a(Dof_Prescribed) = Uu(ii);

        % Iterations with Newton
        unbal = 1e10;         % It's the error
        n_iter = 0;

        while unbal > toll
            % Nullifying
            K = K.*0;
            f_int = f_int.*0;

            % Loop over the elements
            for el = 1: n_elem
                % Compute K_el and fel_ext
                [K_el, f_el] = node6_isop_stiff_load(Ex(el,:), Ey(el,:), D_Mat, F_Body(el, :)', h);

                % Assemblying
                K(Edof(el, 2:end), Edof(el, 2:end)) = K(Edof(el, 2:end), Edof(el, 2:end)) + K_el;
                f_ext(Edof(el, 2:end)) = f_ext(Edof(el, 2:end)) + f_el;
            end
            % Compute Unbalance
            f_int = K*a;
            g_F = f_int(Dof_Free) - f_ext(Dof_Free);
            unbal = norm(g_F);

            if unbal > toll
                % Newton Update
                a(Dof_Free) = a(Dof_Free) - K(Dof_Free, Dof_Free)\g_F;
            end

            % Update iterations
            n_iter = n_iter + 1;

            if n_iter > 20
                error('Newton cannot converge')
            end
        end

        % Solving the system equation
        f_ext_constr = K(Dof_Constr, Dof_Free)*a(Dof_Free) + K(Dof_Constr,
Dof_Constr)*a(Dof_Constr) - f_ext(Dof_Constr);

        % Save Data for postprocessing
        F_react(ii) = sum(f_ext_constr((length(bottomNodes) + 2): end));

        % Save New displacement
        da = a - a_old;
        a_old = a;

        % Store Iterations
        num_iter(ii, mesh - 2) = n_iter;

        % Plots
```

```
        plot(Uu, F_react, '-', 'LineWidth',2); grid on;
        xlabel('Load Time Stepping [s]'); ylabel('Reaction Force [N]');
        title('Vertical Reaction developement over Prescribed Displacement')
        subtitle(name); ylim([0 12e5]);
        drawnow
    end
    % Screen Display
    fprintf(['For the ', name{1}, ' Total Vertical Reaction Force is: ', num2str(F_react(end)), '
N\n'])
    F_React_NonLinearFE(mesh) = F_react(end);

    % Plotting Deformations
    figure('WindowState','maximized')
    plotpar = [1 1 0];
    eldraw2_ext(Ex,Ey,plotpar); grid on
    xlabel('[mm]'); ylabel('[mm]');
    title('Structure Mesh'); subtitle(name);
    Ed = extract_dofs(Edof,a);
    plotpar=[2 4 0];
    sfac = 1;
    eldisp2_ext(Ex,Ey,Ed,plotpar,sfac);

% End of the for loop to do more mesh at same time
end
```

3.4 ELASTOPLASTIC BEHAVIOUR

```
clc
clear
close all

% Datas
data_output = DataGen_CA1;

L = data_output.L;
H = data_output.H;
h = data_output.h;
E = data_output.E;
nu = data_output.poisson_ratio;
sigma_yield = data_output.sigma_yield;
H_coeff = data_output.H_coeff;

F_React_NonLinearFE = zeros(5, 1);
num_iter = zeros(100,3);


% Input vectors for CALFEM (They won't change in the loop)
ptype = 2;
Mat_Prop = [E, nu, H_coeff];
D_Mat = hooke(ptype, E, nu);

% Choose which mesh to load
for mesh = 3:5
```

```
    if mesh == 1
        load("linearCoarse2024.mat");
        name = {'Linear Coarse Mesh:   '};
    elseif mesh == 2
        load("linearMedium2024.mat");
        name = {'Linear Medium Mesh:   '};
    elseif mesh == 3
        load("quadraticCoarse2024.mat");
        name = {'Quadratic Coarse Mesh:'};
    elseif mesh == 4
        load("quadraticMedium2024.mat");
        name = {'Quadratic Medium Mesh:'};
    elseif mesh == 5
        load("quadraticFine2024.mat");
        name = {'Quadratic Fine Mesh:  '};
    end



    % Element nodes Coordinates
    Coord_x = Coord(:,1);
    Edof_x = Edof(:, 2:2:end);
    Ex = Coord_x(ceil(Edof_x/2));

    Coord_y = Coord(:, 2);
    Edof_y = Edof(:, 3:2:end);
    Ey = Coord_y(floor(Edof_y/2));



    % Degrees of Freedom/Constrained (LC2)
    n_elem = size(Edof, 1);
    n_dof = ndofs;
    n_constr = 1 + length(bottomNodes) + length(topNodes);
    Dof_Prescribed = reshape(Dof(topNodes, 2)', 1, []);
    Dof_Constr = [Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []),
reshape(Dof(topNodes, 2)', 1, [])];
    Dof_Free = reshape(Dof', 1, []);
    Dof_Free([Dof(bottomNodes(1),1), reshape(Dof(bottomNodes, 2)', 1, []), reshape(Dof(topNodes,
2)', 1, [])])= [];


    % Initialize Displacements
    a = zeros(n_dof, 1);
    a_old = zeros(n_dof, 1);
    d_a = a - a_old;
    a_constr = zeros(n_constr,1);

    % Initialize State Variables
    n_state = 3;
    ngp = 3;                            % Number of Gauss integration points
    stress_old = zeros(4, ngp, n_elem);
    stress = stress_old;
    state = zeros(n_state, ngp, n_elem);
    state_old = zeros(size(state));

    % Imposing yield stress
```

```
    state_old(2,:,:) = sigma_yield;

    % Time Stepping
    n_time = 250;               % Number of timesteps
    t_end = 100;
    t = linspace(0, t_end, n_time);

    % Displacement Control
    U_max = 1;                  %[mm]
    Uu = linspace(0, U_max, n_time);

    % Initialize Variables for post processing
    K = spalloc(n_dof, n_dof, 20*n_dof);
    F_react = zeros(size(Uu));

    % Inizialize force vectors
    f_int = zeros(n_dof, 1);
    f_ext = f_int;

    % Gravity Load
    F_Body = zeros(n_elem, 2);    % In this case is absent

    % Tolerance for Newton iterations
    toll = 1e-6;

    figure('WindowState','maximized')
    for ii = 1: n_time

        % Initial Guess of unknown Displecement field
        a(Dof_Free) = a_old(Dof_Free) + d_a(Dof_Free);

        % Update Prescribed dof
        a(Dof_Prescribed) = Uu(ii);

        % Iterations with Newton
        unbal = 1e10;         % It's the error
        n_iter = 0;
        stress_trial = zeros(4,1);

        while unbal > toll
            % Nullifying
            K = K.*0;
            f_int = f_int.*0;

            % Loop over the elements
            for el = 1: n_elem
                % Find dof of the element
                dof = Edof(el, 2:end);

                % Compute incrementral element displacement
                d_ae = a(dof) - a_old(dof);

                % Compute Stiffness Mat, Force vect, stress and state vect
                [K_el, f_el, stress(:,:, el), state(:,:, el)] = node6_isop_non_lin(Ex(el,:),
Ey(el,:), d_ae, h, D_Mat, ptype,Mat_Prop,stress_old(:,:,el),state_old(:,:,el));

                % Assemblying
```

```
                K(Edof(el, 2:end), Edof(el, 2:end)) = K(Edof(el, 2:end), Edof(el, 2:end)) + K_el;
                f_int(Edof(el, 2:end)) = f_int(Edof(el, 2:end)) + f_el;

            end
            % Compute Unbalance
            g_F = f_int(Dof_Free) - f_ext(Dof_Free);
            unbal = norm(g_F);

            % Newton Update
            if unbal > toll
                a(Dof_Free) = a(Dof_Free) - K(Dof_Free, Dof_Free)\g_F;
            end

            % Update Iterations
            n_iter = n_iter + 1;

            % Divergence Check
            if n_iter > 20
                error('Newton cannot converge')
            end

        end

        % Computing external Forces
        f_ext_constr = f_int(Dof_Constr);


        % Save Data on Reaction Force
        F_react(ii) = sum(f_ext_constr((length(bottomNodes) + 2): end));

        % Save New displacement
        d_a = a - a_old;
        a_old = a;

        % Storing
        num_iter(ii, mesh - 2) = n_iter;
        state_old = state;
        stress_old = stress;


        % % Plots
        % plot(Uu, F_react, '-', 'LineWidth',2); grid on;
        % xlabel('Load Time Stepping [s]'); ylabel('Reaction Force [N]');
        % title('Vertical Reaction developement over Prescribed Displacement')
        % subtitle(name); ylim([0 4e5]);
        % drawnow
    end
    % Screen Display
    fprintf(['For the ', name{1}, ' Total Vertical Reaction Force is: ', num2str(F_react(end)), '
N\n'])
    F_React_NonLinearFE(mesh) = F_react(end);

    % Plots
    plot(Uu, F_react, '-', 'LineWidth',2); grid on;
    xlabel('Load Time Stepping [s]'); ylabel('Reaction Force [N]');
    title('Vertical Reaction developement over Prescribed Displacement')
    subtitle(name); ylim([0 4e5]);
```

```
    % Plots
    figure('WindowState','maximized')
    plotpar = [1 1 0];
    eldraw2_ext(Ex,Ey,plotpar); grid on
    xlabel('[mm]'); ylabel('[mm]');
    title('Structure Mesh'); subtitle(name);
    Ed = extract_dofs(Edof,a);
    plotpar=[2 4 0];
    sfac = 1;
    eldisp2_ext(Ex,Ey,Ed,plotpar,sfac); grid on;


% End of the for loop to do more mesh at same time
end
```

3.4.1 node6_isop_non_lin

```
function [K_el,f_el_int,stress,state] = node6_isop_non_lin(ex, ey, d_ae, t, D_Mat,
ptype,Mat_Prop,stress_old,state_old)

% Define integration point and weight
ngp = 3;

% xi-coordinates Vector creation
xi_vect = [1/6 1/6 2/3;
           1/6 2/3 1/6];

% Integration Weights vector
H_vect = 1/6*ones(1,3);

% Initializing
K_el = zeros(12,12);
f_el_int = zeros(12, 1);
state = zeros(size(state_old));
stress = zeros(size(stress_old));

% Loop over Gauss Points
for  ii = 1:ngp
    % Define location and weight of integration
    H = H_vect(ii);
    xi = xi_vect(:, ii);

    % Find Be matrix
    [Be, det_F_Isop] = Be_6node_func(xi, [ex(1);ey(1)], [ex(2);ey(2)], [ex(3);ey(3)],
[ex(4);ey(4)], [ex(5);ey(5)], [ex(6);ey(6)]);

    % Compute strain increment
    delta_strain = Be*(d_ae);

    % Find Stress Trial
    stress_trial = stress_old(:, ii) + D_Mat*[delta_strain(1:2);0;delta_strain(3)];

    % Compute updated stress and updated state variables from trial stress
    [stress(:, ii), delta_strain, state(:, ii)] = mises(ptype, Mat_Prop, stress_trial',
state_old(:, ii)');
```

```
    % Compute tangent material
    dstress_dstrain = dmises(ptype, Mat_Prop, stress(:, ii)', state(:, ii)');

    % Stiffness Matrix
    K_el = K_el + Be'*dstress_dstrain([1 2 4], [1 2 4])*Be*t*det_F_Isop*H;

    % Force Vector
    f_el_int = f_el_int + Be'*stress([1 2 4], ii)*t*det_F_Isop*H;

end
end
```

3.5 DataGen Function

This function is created to improve the flexibility of the scripts, it recalls the same data in all the different scripts. If it's needed to modify a value, by just modifying it in this function, it will modify automatically in each one of the scripts that use this function without the need to manually modify the value in each script.

```
function data_output = DataGen_CA1()

% Geometrical
L = 100;                % [mm]
H = 100;                % [mm]
h = 10;                 % [mm]

% Material
E = 210e3;               % [MPa]
poisson_ratio = 0.3;     % [1]
sigma_yield = 500;       % [MPa]
H_coeff = 20000;         % [MPa]


% Output
data_output.L = L;
data_output.H = H;
data_output.h = h;
data_output.E = E;
data_output.poisson_ratio = poisson_ratio;
data_output.sigma_yield = sigma_yield;
data_output.H_coeff = H_coeff;

end
```