ME621 - Advanced Finite Element Methods
Assignment 3

Tommaso Bocchietti

A.Y. 2023/24 - W24

Contents

1 Requests **3**

2 Methodology **3**

3 Solution **4**

3.1 Mesh generation 4

3.2 Elemental matrices (for a 4-node rectangular element) 5

3.2.1 Shape functions 5

3.2.2 Compatibility matrix 5

3.2.3 Forces and mass matrix 6

3.3 Global matrices 6

3.4 Constitutive model 7

3.5 Flow Charts 7

4 Results **7**

4.1 Effect of mesh refinement 10

A Flow Charts **12**

List of Figures

1 Problem representation in its initial (dashed lines) and final (solid lines) configuration. 3

2 Grid structure 4

3 Node numbering (global indices) 4

4 Element numbering (element indices) 4

5 Average shear stress σ_{12} as a function of the shear strain γ. 8

6 Focus on the oscillation of the curve $\sigma_{12}(\gamma)$. 8

7 Initial and final configurations of the structure. 9

8 Coarse mesh. 10

9 Fine mesh. 10

10 Effect of mesh refinement on the results of the simulation. 10

11 Flowchart for the **Explicit Time Integration Algorithm**. The convergence criterion $\gamma < 0.07$ is relative to the given problem, while the rest of the algorithm is general. 12

12 Flowchart for the **Stress Update Algorithm**. The decision blocks in this case represent for loops over the elements and the integration points. 13

List of Tables

1 Parameters of the plate element. 3

2 Coordinates matrix for the example above. 4

3 Connectivity matrix for the example above (#n refers to the element node index). 4

4 Boundary conditions imposed on the structure. 7

5 Results of the simulation for different values of applied velocity v_{top} and density ρ. Here U_x and U_y are the displacements of the central node of the structure. When increasing the density of the material, the external body force (gravity) is increased as well. 10

6 Boundary conditions imposed on the structure. 10

7 Mesh sizes and CPU time of the simulations. 10

1 Requests

Use an Updated Lagrangian Formulation and four 4-node quadrilateral elements to simulate simple shear loading. The bottom edge of the specimen is fixed in place while a constant velocity is applied to the top in the $\vec{x}$ direction.

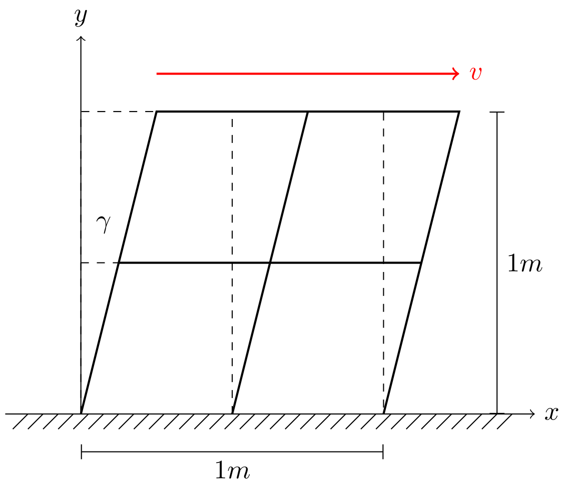

Figure 1: Problem representation in its initial (dashed lines) and final (solid lines) configuration.

The material is made out of aluminum with density $\rho = 2700\text{kg/m}^3$, Young's modulus $E = 70\text{GPa}$, and Poisson's ratio $\nu = 0.3$.
Under the loading condition of $v = 1\text{m/s}$ applied to the top three nodes:

- Plot the average shear stress σ_{12} as a function of shear strain until $\gamma = 0.07$. Use the **Truesdell objective rate** to update stress, $\boldsymbol{\sigma}^{oT} = \boldsymbol{C} : \boldsymbol{D}$.
- Plot the position of the nodes before and after loading the system.
- The loading speed, $v = 1\text{m/s}$, was intentionally high for the sake of computation time. If the velocity is more realistic, such as $v = 0.001\text{m/s}$, what can be done to further reduce the computation time?

Parameter	**Value**	**Unit**
E	70	GPa
ν	0.3	-
ρ	2700	kg/m^3
L_x	1	m
L_y	1	m

Table 1: Parameters of the plate element.

2 Methodology

To solve the given problem, we will use a similar approach adopted in the previous assignment. The main differences are:

- Adoption of the **Updated Lagrangian Formulation** instead of the Total Lagrangian Formulation.
- Different constitutive model (Truesdell objective rate).
- 2D problem instead of 1D.

As for the code implementation, a skeleton of the code may be summarized as follows:

1. Definition of parameters for the problem (material properties, geometry of the plate, boundary conditions...).
2. Mesh generator for the 2D plate (domain discretization, node numbering and connectivity dictionary).

3. Explicit integration scheme with Updated Lagrangian Formulation.
4. Plotting and post-processing of the results.

Note: given the advice from the professor in the last lecture, for this assignment we have changed a bit the structure and the way of do things, so to rely less on the `MATLAB` built-in functions and more on the custom implementation of the code. This will allow us to have a better understanding of the underlying mechanics of the problem and the numerical methods used to solve it.

3 Solution

In the following, we present and explain the main steps involved in the solution of the given problem.

3.1 Mesh generation

After having defined all the data of the problem in adequate data structures, the next step is to generate the mesh. Generating a mesh means to discretize the domain into a finite number of elements, creating a grid of locations at which we will evaluate the approximate solution of the problem.
In particular, the algorithm has to return the following data structures:

- `coordinates`: a matrix of size $N_{MeshNodes} \times N_{DoF}$ containing the (x, y) coordinates of the nodes of the mesh.
- `connectivity`: a matrix of size $N_{Element} \times N_{NodesPerElement}$ containing the indices of the nodes that belong to each element of the mesh.

Moreover, we can analyze an example of mesh generation applied to a simple domain 2D domain composed of 2 rectangular-4-nodes elements, as shown in Figures 2, 3, and 4.

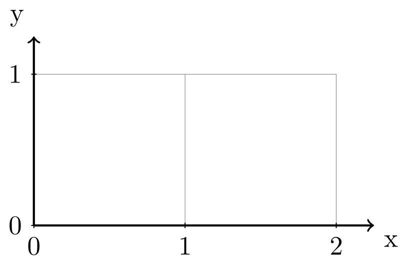

Figure 2: Grid structure

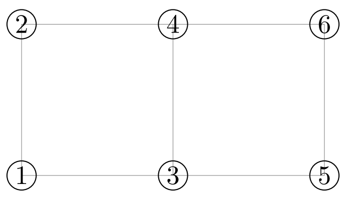

Figure 3: Node numbering (global indices)

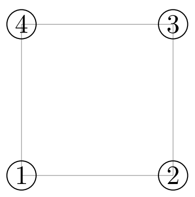

Figure 4: Element numbering (element indices)

At code level, all the information about the mesh numbering and space discretization are stored separately in the `coordinates` and `connectivity` matrices as follows:

Node index (global)	**X (m)**	**Y (m)**
1	0.0	0.0
2	0.0	1.0
3	1.0	0.0
4	1.0	1.0
5	2.0	0.0
6	2.0	1.0

Table 2: Coordinates matrix for the example above.

Element index	**#1**	**#2**	**#3**	**#4**
1	1	3	4	2
2	3	5	6	4

Table 3: Connectivity matrix for the example above (#n refers to the element node index).

Once the mesh has been generated, the next step is to compute the elemental matrices (in both the matrix and voigt form as needed).

3.2 Elemental matrices (for a 4-node rectangular element)

Similar to what we have done in previous assignments, we will now compute the elemental matrices for the problem at hand.
The major difference in this case is that we are dealing with a 2D problem. In order to work with some higher order tensorial quantities (i.e. tensor of order 4), we will need to use the voigt notation which allows us to represent a tensor of order 4 as a tensor of order 2 (easily manageable from a coding perspective).

3.2.1 Shape functions

The first step is to compute the shape functions for the problem at hand and in particular for the element we are considering.
Given that we are working with a rectangular-4-node element, we can use a simple `Mathematica` script to compute the shape functions:

```
Clear["Global`*"]

Displachment=coef0+coef1*\[Xi]+coef2*\[Eta]+coef3*\[Xi]*\[Eta];

Equation1=Displachment/.{\[Xi]->-1,\[Eta]->-1};
Equation2=Displachment/.{\[Xi]->+1,\[Eta]->-1};
Equation3=Displachment/.{\[Xi]->+1,\[Eta]->+1};
Equation4=Displachment/.{\[Xi]->-1,\[Eta]->+1};

sol=Solve[
{Equation1==u1,Equation2==u2,Equation3==u3,Equation4==u4},
{coef0,coef1,coef2,coef3}
];
{coef0,coef1,coef2,coef3} = {coef0,coef1,coef2,coef3}/.sol[[1]];

N1=Coefficient[Displachment,u1];
N2=Coefficient[Displachment,u2];
N3=Coefficient[Displachment,u3];
N4=Coefficient[Displachment,u4];

NShapeFunctions = {N1, N2, N3, N4};
```

Listing 1: Mathematica script to compute the shape functions for a rectangular-4-node element.

The script above will generate the following shape functions:

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\ N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \frac{1}{4}(1+\xi)(1+\eta) \\ N_4 &= \frac{1}{4}(1-\xi)(1+\eta) \end{aligned} \tag{1}$$

Notice that the shape functions are defined in terms of the natural coordinates ξ and η, and also notice that the index of the shape function corresponds to the elemental node number (counterclockwise starting from the bottom left corner).
Our shape functions can be represented in matrix form as well as using voigt notation:

$$N_{matrix} = [N_1, N_2, N_3, N_4] \tag{2}$$

$$N_{voigt} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix} \tag{3}$$

3.2.2 Compatibility matrix

Having the shape functions at hand, we can now compute the compatibility matrix B. From its definition, we have that:

$$B = \frac{\partial N}{\partial \vec{x}} = \frac{\partial N}{\partial \vec{\xi}} \frac{\partial \vec{\xi}}{\partial \vec{x}}, \quad \text{where} \quad \vec{\xi} = \begin{bmatrix} \xi \\ \eta \end{bmatrix} \tag{4}$$

By using the chain rule, we have decomposed the compatibility matrix into two parts: the derivative of the shape functions with respect to the natural coordinates $\vec{\xi}$ and the Jacobian matrix $\frac{\partial \vec{\xi}}{\partial \vec{x}}$. We can proceed by computing the two parts separately.
The derivative of the shape functions with respect to the natural coordinates is given by:

$$\frac{\partial N}{\partial \vec{\xi}} = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_4}{\partial \eta} \end{bmatrix} = \frac{1}{4} \begin{bmatrix} -(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta) \\ -(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi) \end{bmatrix} \tag{5}$$

The Jacobian matrix is given by:

$$\frac{\partial \vec{\xi}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} \tag{6}$$

Given that we can represent $\vec{x}$ in terms of the shape functions and the nodal coordinates, we can write:

$$\frac{\partial \vec{\xi}}{\partial \vec{x}} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{bmatrix} \frac{1}{4} \begin{bmatrix} -(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta) \\ -(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi) \end{bmatrix}^T \tag{7}$$

By using Equations 5 and 7 in Equation 4, we can compute the compatibility matrix B in matrix form as well as in voigt notation:

$$B_{matrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \end{bmatrix} \tag{8}$$

$$B_{voigt} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} \end{bmatrix} \tag{9}$$

3.2.3 Forces and mass matrix

The forces and mass matrix can be computed similarly to what we have done in previous assignments (see Assignment 2, Section 3.2) even if we are now working in the *Updated Lagrangian* framework. For this reason we will not go into the details of the computation and, instead, we will provide just the final results[1]:

$$f^e_{int} = \int_\Omega B^{eT} \sigma d\Omega \approx \sum_{\xi=-1}^{+1} \sum_{\eta=-1}^{+1} w_\xi w_\eta B(\xi,\eta,u)^{eT} \sigma(\xi,\eta,u) J(\xi,\eta,u) \tag{10}$$

$$f^e_{ext} = \int_\Omega N^{eT} \rho b d\Omega \approx \sum_{\xi=-1}^{+1} \sum_{\eta=-1}^{+1} w_\xi w_\eta N(\xi,\eta)^{eT} \rho b J(\xi,\eta,u) \tag{11}$$

$$M^e = \int_\Omega N^{eT} \rho N^e d\Omega \approx \sum_{\xi=-1}^{+1} \sum_{\eta=-1}^{+1} w_\xi w_\eta N(\xi,\eta)^{eT} \rho N(\xi,\eta)^{eT} J(\xi,\eta,u) \tag{12}$$

Notice that the approximations of the integrals are done using a Gauss quadrature rule with a given number of point for each direction. In our solution, we have chosen to use a 3-point rule in both directions.

3.3 Global matrices

Global matrices are obtained by assembling the elemental matrices based on the `connectivity matrix` generated during the mesh generation phase.
In particular, given that f^e_{int}, f^e_{ext}, M^e have been computed using voigt notation, we can assemble the global matrices knowing that each row and columns of the global matrices correspond to a particular degrees of freedom of the system. By summing for each element the contribution of the elemental matrices to the corresponding positions of the global matrices, we obtain the global matrices.

[1] In the formulation of f^e_{ext}, we have neglected the contribution of any traction applied to the boundary or any other point force given that for our problem they are not present.

3.4 Constitutive model

Our final major step in the solution of the problem at hand is the definition of the constitutive model.
For the given problem, we are asked to use Truesdell objective stress rate to define the constitutive model.
In continuum mechanics, an objective stress tensor (as for example the Truesdell one) is used to describe the rate of change of stress within the material undergoing deformation in a way that it doesn't depend on the observer's frame of reference.
The Truesdell objective stress tensor is defined as:

$$\sigma^{oT} = C : D = \dot{\sigma} - L\sigma - \sigma L^T + tr(L)\sigma \tag{13}$$

where σ is the Cauchy stress tensor, C is the stiffness or elasticity tensor, D is the (symmetric) rate of deformation, and L is the (asymmetric) velocity gradient tensor.
In particular, the following definition can be used inside Equation 13:

$$C = \begin{bmatrix} 2\mu+\lambda & \lambda & 0 \\ \lambda & 2\mu+\lambda & 0 \\ 0 & 0 & \mu \end{bmatrix} = \frac{E}{(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \quad \text{Considering plane stress} \tag{14}$$

$$L = \vec{\nabla} \otimes \vec{v} = \frac{\partial \vec{v}}{\partial \vec{x}} = \frac{\partial(\vec{v}_{node}N)}{\partial \vec{x}} = \vec{v}_{node}B^T \tag{15}$$

$$D = \frac{1}{2}(L + L^T) = B\vec{v}_{node} \tag{16}$$

where μ is the shear modulus, λ is the Lamé constant, E is the Young's modulus, and ν is the Poisson's ratio.
Given all the definitions above, we can observe that the only unknown in Equation 13 is the rate of Cauchy stress tensor $\dot{\sigma}$, that can be computed as:

$$\dot{\sigma} = C : D + L\sigma + \sigma L^T - tr(L)\sigma \tag{17}$$

The result from Equation 17 can be used to compute the stress tensor at the next time step in the simulation given that:

$$\sigma_{n+1} = \sigma_n + \dot{\sigma}_{n+1/2}\Delta t_{n+1/2} \tag{18}$$

3.5 Flow Charts

To better understand the flow used in the solution of the problem, we leave in the Appendix section (Section A) a couple of flow charts that represent the main steps involved in the solution of the problem.

4 Results

The numerical results for the problem described in Section 1 are reported in the following.
When not specified, the results are obtained with the following boundary conditions imposed on the structure:

	U_x m	U_y m	V_x m/s	V_y m/s
Bottom nodes	Fixed	Fixed	0	0
Top nodes	-	Fixed	1	0

Table 4: Boundary conditions imposed on the structure.

Average $\sigma_{12}(\gamma)$ for element #1 To answer the first question of the assignment, we have decided to consider for the computation of the average shear stress σ_{12} just the first element of the mesh, which is the one in the bottom-left corner of the structure. Moreover, given that σ_{12} get computed in every Gaussian point of the element, to simplify the computation, we have decided to consider just the Gaussian point in elemental coordinates $(\eta, \xi) = (-1/\sqrt{3}, -1/\sqrt{3})$, which is the one in the bottom-left corner of the element. This choice is justified by the fact that γ, which is the shear strain of the element, is usually referred to the bottom-left corner of the element with the hypotheses of null gradient all over the element (i.e., $\gamma_{element} = \gamma_{bottom-left} \rightarrow \gamma(x, y) = \gamma(0, 0)$).
With the hypothesis of null or almost null $\nabla\gamma$ in the element, we report in Figure 5 the shear stress σ_{12} as a function of the shear strain γ until $\gamma = 0.07$.

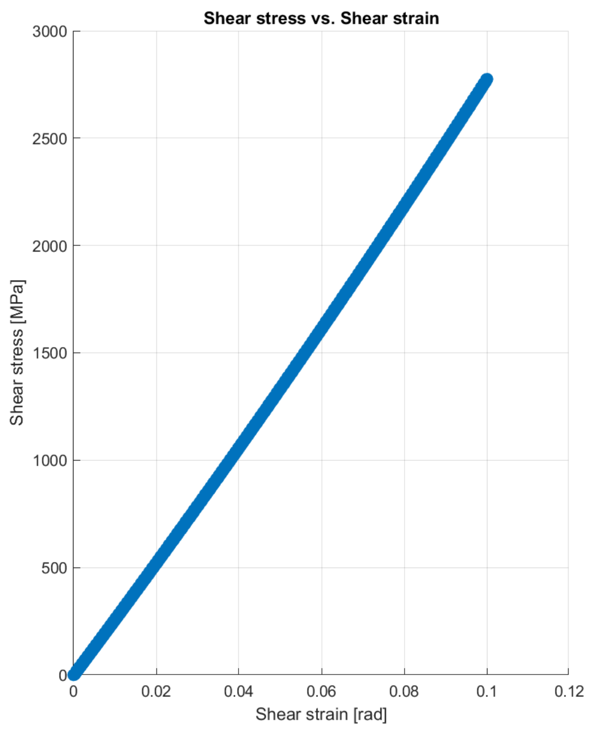

Figure 5: Average shear stress σ_{12} as a function of the shear strain γ.

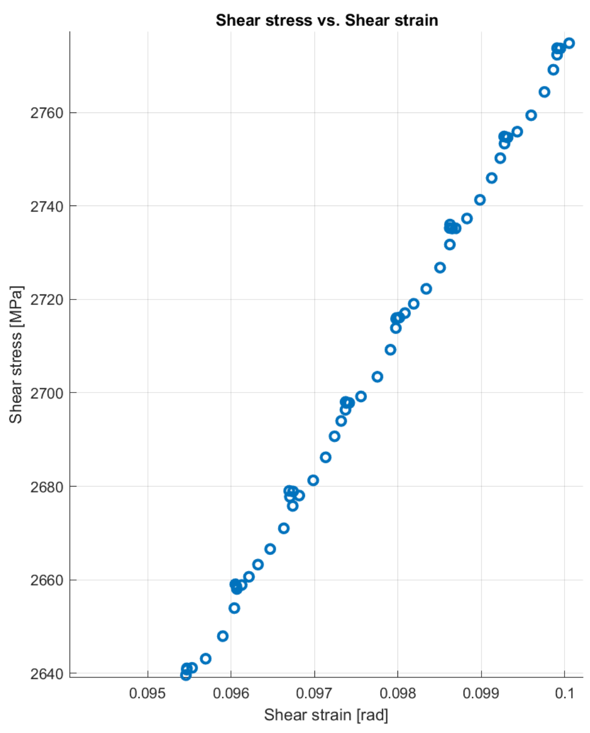

Figure 6: Focus on the oscillation of the curve $\sigma_{12}(\gamma)$.

From the plot, it's clearly visible a general linear trend of the stress-strain curve, which is typical for a linear elastic material. In particular, the curve well approximate the ideal behavior having a slope almost equal to the shear modulus $G = \frac{E}{2(1+\nu)} = 26.923\text{GPa}$, characteristic of the elastic region of the material.
However, because of the inertia effects and the high loading speed, the curve is not perfectly linear, and it shows some oscillations. When the loading speed is reduced to a more realistic value, such as $v = 0.001\text{m/s}$, the inertia effects are strongly reduced, and the oscillatory behavior of the curve is almost completely removed.

Initial and final positions of the structure In Figure 7, we can observe the structure in both its initial configuration (shown in blue), and in its final configuration (shown in red).

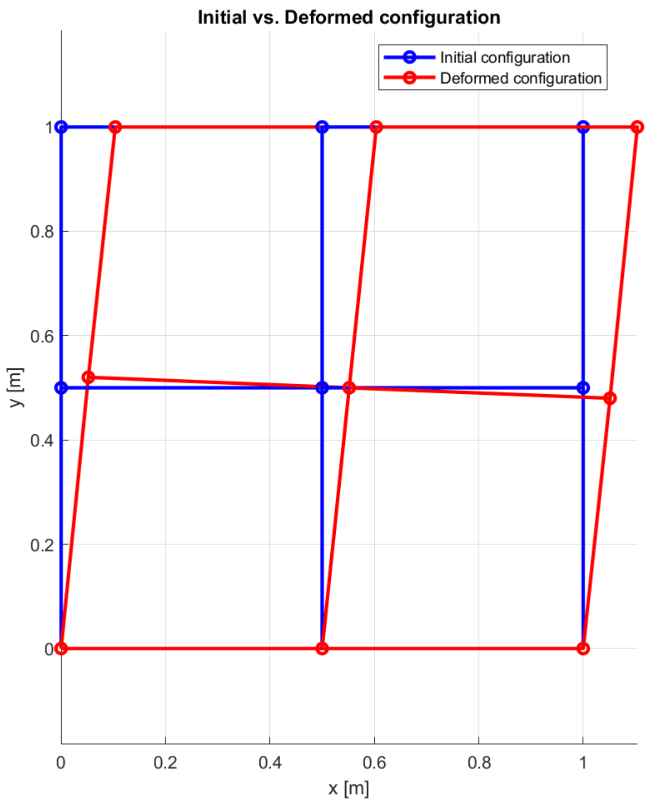

Figure 7: Initial and final configurations of the structure.

The structure is deformed in the $\vec{x}$ direction, as expected, and the deformation is more pronounced in the top part of the structure where the velocity constrains is applied. The deformation is almost linear, and the structure is not showing any sign of instability or excessive deformation.
The well known behavior of compression on the right side and tension on the left side of the lower half of the structure is clearly visible by observing the central line which is horizontal in the initial configuration and shows a clear clockwise rotation in the final configuration.

Computation time reduction Finally, a possible way to reduce the computation time when the loading speed is reduced to a more realistic value, such as $v = 0.001\text{m/s}$, is to increase the density property of the material (ρ).
This choice is motivated by the fact that the time step of the simulation is usually limited by the smallest element of the mesh, which is the one that requires the smallest time step to satisfy convergence and stability conditions. In particular:

$$\Delta t = \frac{2}{\omega_{max}} \tag{19}$$

where ω_{max} is the maximum frequency of the system, computable as the largest eigenvalues of the equation of motion for the element (i.e., $[M]\ddot{U} + [K]U = 0 \rightarrow det([M]\omega^2 - [K]) = 0 \rightarrow \omega_{max}$). By solving the equation, we obtain the following expression for the time step:

$$\Delta t = \frac{L_{min}}{c} = \frac{L_{min}}{\sqrt{\frac{E}{\rho}}} = \frac{L_{min}}{\sqrt{E}}\sqrt{\rho} \tag{20}$$

where L_{min} is the smallest characteristic length of the element, and c is the speed of sound in the material.
From Equation 20, it's clear that by increasing the density of the material, the time step of the simulation is increased as well, and the computation time is decreased.
One may wonder if the choice of increasing the density of the material may affect the results of the simulation. For this purpose, we report in Table 5 the results of the simulation for two different values of applied velocity v_{top} and density ρ.

	v_{top} m/s	ρ kg/m^3	CPU time s	U_x m	U_y m
Simulation 1	1	2700	9.64	0.5518	0.5000
Simulation 2	0.01	2700	662.17	0.5518	0.5000
Simulation 3	0.01	27000000	7.68	0.5518	0.4992

Table 5: Results of the simulation for different values of applied velocity v_{top} and density ρ. Here U_x and U_y are the displacements of the central node of the structure. When increasing the density of the material, the external body force (gravity) is increased as well.

From Table 5, it's clear that the results of the simulation are not affected by the choice of increasing the density of the material (as long as the mesh is sufficiently fine) and that the computation time is significantly reduced by this choice.

4.1 Effect of mesh refinement

It's worth mentioning that the results of the simulation are affected by the choice of the mesh size.
In Figure 10, we report the results of the simulation runned with same boundary conditions (Table 6) but different mesh sizes (Table 7).

	U_x m	U_y m	V_x m/s	V_y m/s
Bottom nodes	Fixed	Fixed	0	0
Top nodes	-	Fixed	1000	0

Table 6: Boundary conditions imposed on the structure.

Mesh size	**Number of elements**	**CPU time (s)**
Coarse	2x2	0.20
Fine	4x20	8.58

Table 7: Mesh sizes and CPU time of the simulations.

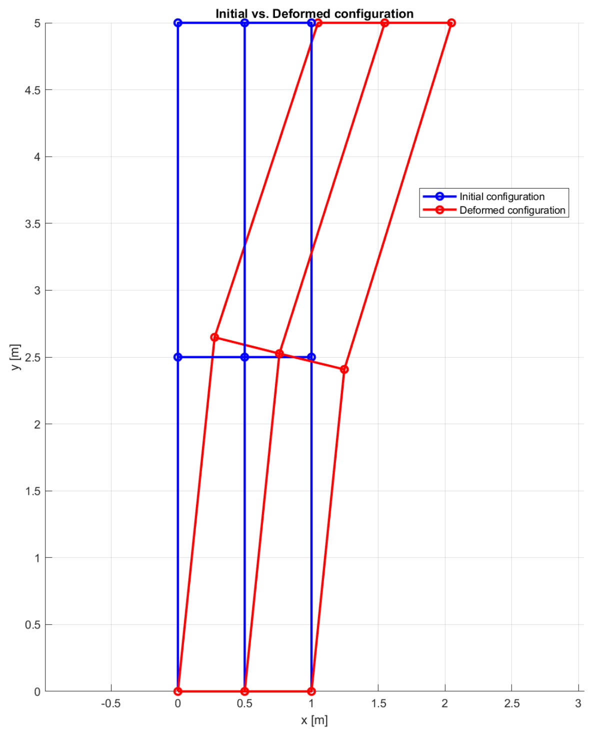

Figure 8: Coarse mesh.

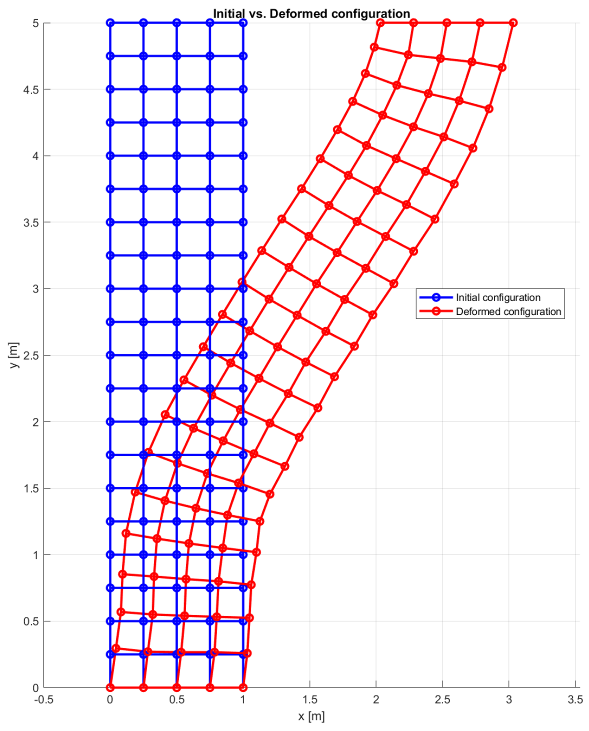

Figure 9: Fine mesh.

Figure 10: Effect of mesh refinement on the results of the simulation.

From Figure 10, it's clear that the results of the simulation are affected by the choice of the mesh size. In particular, given the high loading speed and the inertia effects, the results of the simulation are more accurate when a fine mesh is used, where the mass distribution is more accurate. In the coarse mesh, the mass distribution is not accurate (on the central node an equivalent 25% of the total mass is applied), and the inertia effects are not well represented, leading to a not realistic slow behavior of the node.

A Flow Charts

The following flow chart mimics the structure of the MATLAB code used to solve the problem.
The code is structured in two nested loops: the main loop (**Explicit Time Integration Algorithm**) and the inner loop (**Stress Update Algorithm**).
The Explicit Time Integration Algorithm is the main loop that iterates until the convergence criterion is met. Inside the main loop, the Stress Update Algorithm is called to update the stress state of each element (at each integration point).

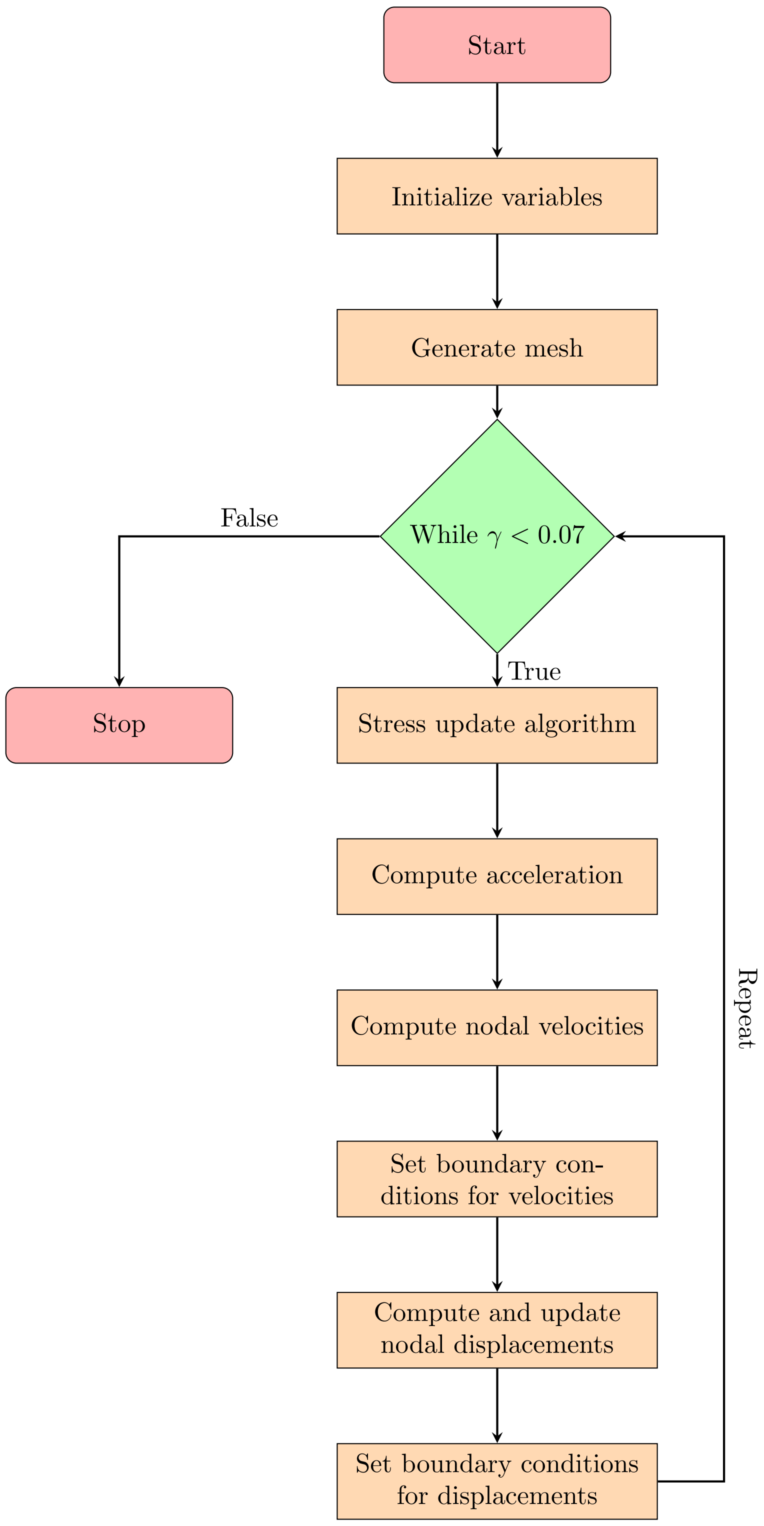

Figure 11: Flowchart for the **Explicit Time Integration Algorithm**. The convergence criterion $\gamma < 0.07$ is relative to the given problem, while the rest of the algorithm is general.

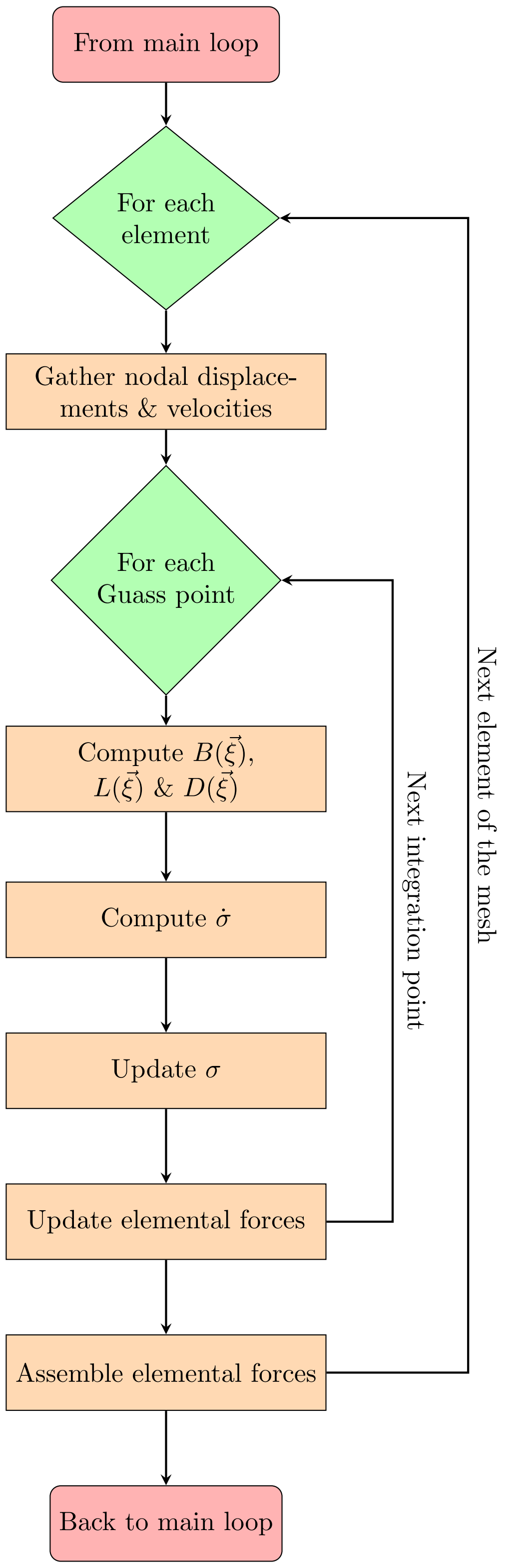

Figure 12: Flowchart for the **Stress Update Algorithm**. The decision blocks in this case represent for loops over the elements and the integration points.