UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Computational aerospace engineering

Assingment 3

Elastostatic Analysis of a Cantilever Beam

Andreu Craus Vidal

Roger Tatjé i Ramos

Greta- Group 1

Professor

Joaquín A. Hernández Ortega
Raúl Rubio Serrano

Friday 3rd January, 2025

Contents

List of Figures

Statement of the Assignment

1. Consider the cantilever box beam depicted in Figure 1. The length of the beam is $L_X = 2\,\text{m}$, its width and height $h_y = h_z = 0.25\,\text{m}$, and its thickness $e = 0.05\,\text{m}$. The material is isotropic, with $E = 70 \times 10^6\,\text{kN/m}^2$ and $\nu = 0.3$. The beam is subjected to a uniformly distributed load of value $t(2) = -500\,\text{kN/m}^2$ on its top surface ($y = y_{\max}$).

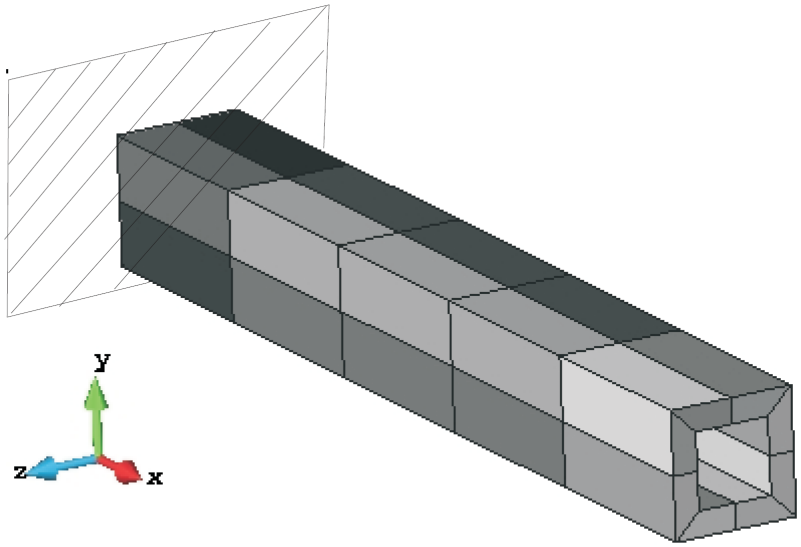

Figure 1: Cantilever box beam.

   Assess the convergence upon mesh refinement by launching 4 different analyses with increasing number of finite elements (hexahedral). In particular, use semi-structured meshes with:

   - **MESH 1:** $n_{ex} = 2$ divisions in the x-direction, and typical size in the y-z plane of $0.1\,\text{m}$.
   - **MESH 2:** $n_{ex} = 10$ divisions in the x-direction, and typical size in the y-z plane of $0.05\,\text{m}$.
   - **MESH 3:** $n_{ex} = 15$ divisions in the x-direction, and typical size in the y-z plane of $0.05\,\text{m}$.
   - **MESH 4:** $n_{ex} = 20$ divisions in the x-direction, and typical size in the y-z plane of $0.025\,\text{m}$.

   For these four cases, plot the distribution of vertical displacements (in the y-direction) along one of the edges on the top surface (parallel to the x-axis). Check also whether the maximum displacement at the tip of the beam converges to the analytical predictions provided by the theory of Strength of Materials.

2. Once the performance of the code has been properly assessed, study, for **MESH 4**, the finite element solution corresponding to the load state in which a torque $T = 100\,\text{kN m}$ is applied at the free end.

3. Code a MATLAB routine able to automatically compute the resultant of the reaction forces at the fixed end (R_x, R_y, R_z as well as M_x, M_y, and M_z). Check that such reactions are in equilibrium with the prescribed forces.

1 Part 1: Basic Tasks

1.1 Mesh Generation

The initial phase of this assignment involves defining the geometry of the problem and generating a mesh within that geometry. A critical aspect to consider is the orientation of the coordinate axes, as they play a fundamental role in specifying all boundary conditions.

The process begins with the creation of the cross-sectional area (Figure 2). This section is then extruded to form the 3D geometry of the beam (Figure 3). Once the beam geometry is established, material properties and boundary conditions are applied. Finally, various semi-structured meshes composed of hexahedral elements are generated, adhering to the specifications outlined in the assignment statement.

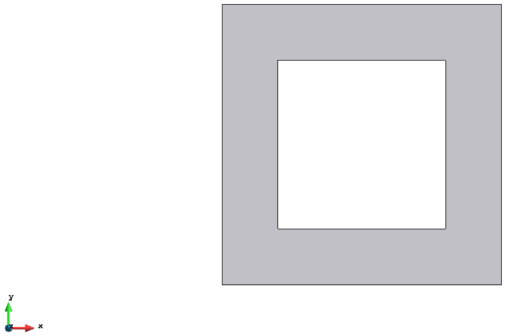

Figure 2: Cross-sectional area of the beam geometry.

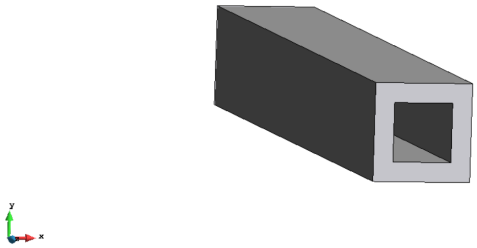

Figure 3: 3D geometry of the beam after extrusion.

1.2 Development of the Finite Element Program

Similar to Assignment 2, the majority of the functions required for the finite element program were pre-defined. However, there were specific components that needed to be implemented to complete the program. For this assignment, the necessary developments included assembling the stiffness matrix $\mathbf{K}$, implementing the shape function routine for trilinear hexahedral elements, and coding the solution process for the final system of equations.

The codes developed for these tasks, along with other relevant auxiliary functions, are included in the Appendix of this report. In the following subsections, the MATLAB implementations will be described in detail.

1.2.1 Assembly of the Stiffness Matrix K

The full implementation for this section can be found in Appendix A.1. The global stiffness matrix $\mathbf{K}$ was assembled by integrating the stiffness submatrices calculated for each element using the `ComputeKeMatrix` function.

This assembly process was carried out through a loop iterating over all elements in the mesh. Unlike Assignment 2, this assignment involves a 3D geometry, meaning each node is associated with three degrees of freedom. This added complexity required adjustments in the assembly logic to account for the additional dimensions.

1.2.2 Implementation of Shape Function Routine for Trilinear Hexahedral Elements

The implementation of the routine for this task is detailed in Appendix A.2. This routine builds upon the function `Quadrilateral4InPoint`, introduced in Assignment 2, but extends its application to trilinear hexahedral elements. The function computes the weights and positions for 2D Gaussian quadrature points, which are subsequently used to calculate the element shape functions N^e and B^e for these Gauss points.

The expressions for the shape functions in this case are given as follows:

$$N^e(1{:}4) = \frac{1}{8}\begin{bmatrix}(1-\xi)\cdot(1-\eta)\cdot(1-\zeta) & (1+\xi)\cdot(1-\eta)\cdot(1-\zeta)\\ (1+\xi)\cdot(1+\eta)\cdot(1-\zeta) & (1-\xi)\cdot(1+\eta)\cdot(1-\zeta)\end{bmatrix}$$

$$N^e(5{:}8) = \frac{1}{8}\begin{bmatrix}(1-\xi)\cdot(1-\eta)\cdot(1+\zeta) & (1+\xi)\cdot(1-\eta)\cdot(1+\zeta)\\ (1+\xi)\cdot(1+\eta)\cdot(1+\zeta) & (1-\xi)\cdot(1+\eta)\cdot(1+\zeta)\end{bmatrix}$$

The matrix B^e is defined as:

$$B^e(:,1{:}4) = \begin{bmatrix}-(1-\eta)\cdot(1-\zeta) & (1-\eta)\cdot(1-\zeta) & (1+\eta)\cdot(1-\zeta) & -(1+\eta)\cdot(1-\zeta)\\ -(1-\xi)\cdot(1-\zeta) & -(1+\xi)\cdot(1-\zeta) & (1+\xi)\cdot(1-\zeta) & (1-\xi)\cdot(1-\zeta)\\ -(1-\xi)\cdot(1-\eta) & -(1+\xi)\cdot(1-\eta) & (1+\xi)\cdot(1+\eta) & -(1-\xi)\cdot(1+\eta)\end{bmatrix}$$

$$B^e(:,5{:}8) = \begin{bmatrix}-(1-\eta)\cdot(1+\zeta) & (1-\eta)\cdot(1+\zeta) & -(1-\eta)\cdot(1+\zeta) & (1+\eta)\cdot(1+\zeta)\\ -(1-\xi)\cdot(1+\zeta) & -(1+\xi)\cdot(1+\zeta) & (1+\xi)\cdot(1+\zeta) & (1-\xi)\cdot(1+\zeta)\\ -(1-\xi)\cdot(1-\eta) & (1+\xi)\cdot(1-\eta) & (1+\xi)\cdot(1+\eta) & (1-\xi)\cdot(1+\eta)\end{bmatrix}$$

The shape functions N and B are evaluated using the appropriate weights at each Gaussian quadrature point.

1.2.3 Solution of the Final System of Equations

The MATLAB code used to solve the final system of equations is provided in Appendix A.3. The routine, implemented in the function `SolveELAS`, computes the solution for the system of equations represented by the following matrix expression:

$$\begin{bmatrix}K_{RR} & K_{RL}\\ K_{LR} & K_{LL}\end{bmatrix}\begin{bmatrix}d_R\\ d_L\end{bmatrix} = \begin{bmatrix}f_R\\ f_L\end{bmatrix} \tag{1}$$

The specific operation carried out by the function to compute the unknowns is given by:

$$d_L = K_{LL}^{-1}\cdot(f_L - K_{LR}\cdot d_R) \tag{2}$$

1.3 Simulation Results

Once the required functions were implemented to make the code fully operational, the results from the postprocessing stage are presented in this section. These results, obtained using the GID CIMNE software based on the outputs of the MATLAB code, are analyzed for the various meshes specified in the assignment statement. The objective is to examine the convergence of the results as the number of discretization elements in the beam geometry increases.

1.3.1 Simulation for the First Mesh (2 Divisions in X Direction, Typical Size of 0.1m in Plane YZ)

The first simulation involves a semi-structured mesh characterized by 2 divisions along the X-direction and a typical element size of 0.1 m in the YZ plane. The mesh for this case is depicted in Figure 4.

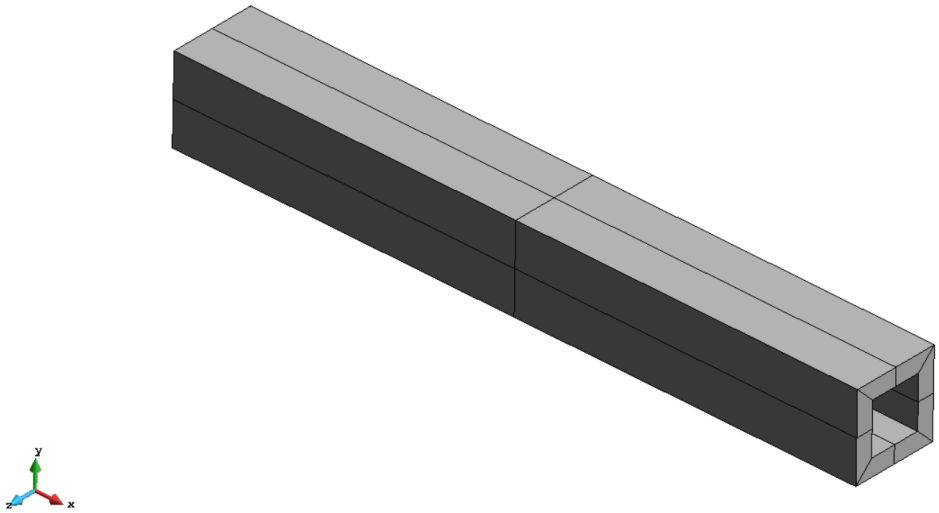

Figure 4: Generated mesh for the case with 2 divisions in the X-direction.

After solving the displacement system using the finite element program, a `.res` file was created and subsequently processed using the postprocessing features of GID CIMNE software. The results of the simulation, focusing on the vertical displacements of the beam, are illustrated below:

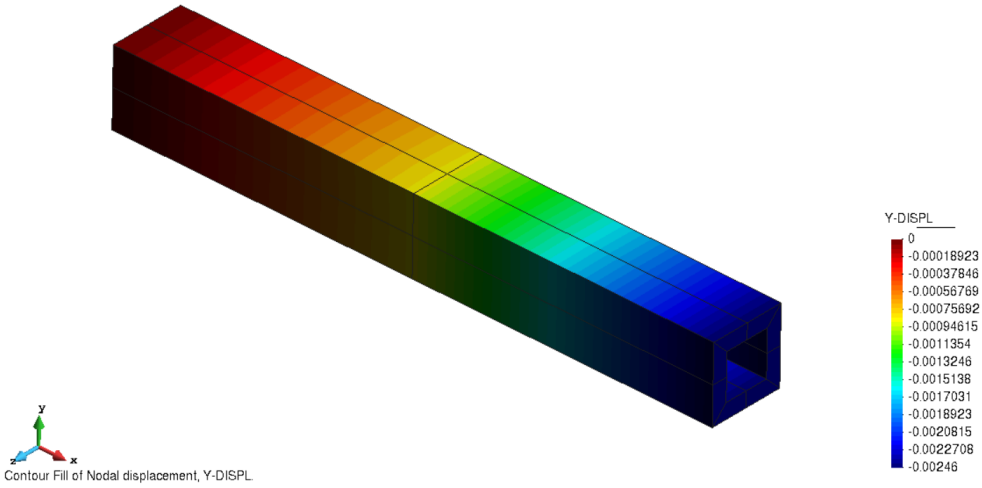

Figure 5: Postprocessed results showing vertical displacements for the first mesh.

The analysis of the particular vertical displacements on the upper surface of the beam for this discretization will be provided later in the report in a comparison graph with the other mesh results.

1.3.2 Simulation for the Second Mesh (10 Divisions in X Direction, Typical Size of 0.05m in Plane YZ)

The second simulation examines a semi-structured mesh of hexahedral elements, similar in configuration to the first mesh. However, this mesh features 10 divisions along the X-direction and a typical element size of 0.05 m in the YZ plane. The generated mesh for this configuration is shown in Figure 6.

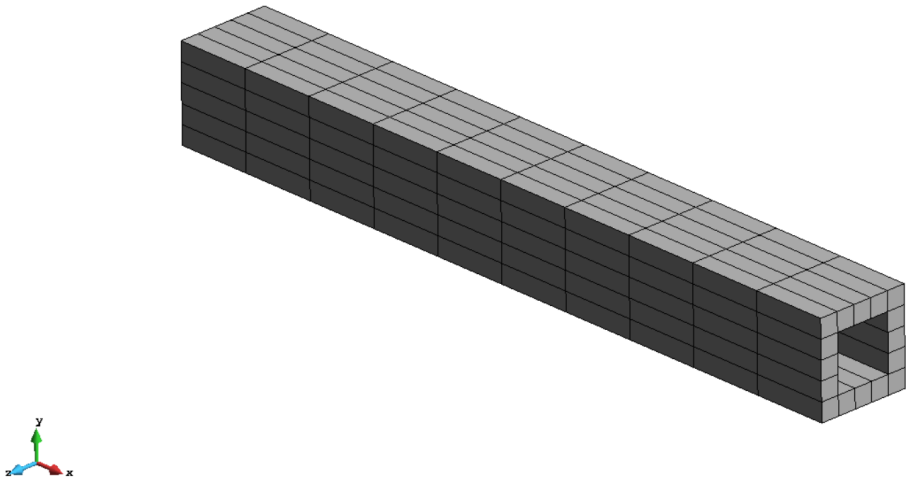

Figure 6: Generated mesh for the case with 10 divisions in the X-direction.

Using the MATLAB code, the computed results were processed with GID's postprocessing tools. The field of vertical displacements for the beam in this simulation is presented in Figure 7.

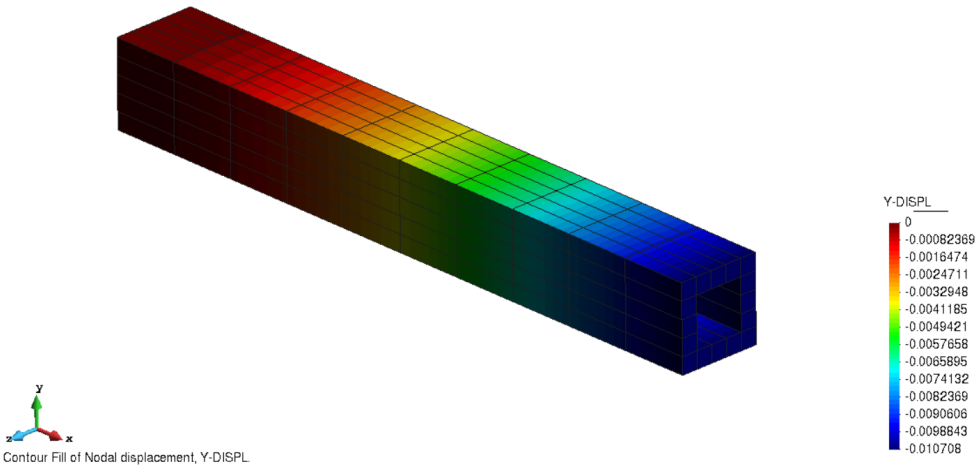

Figure 7: Postprocessed results showing vertical displacements for the second mesh.

1.3.3 Simulation for the Third Mesh (15 Divisions in X Direction, Typical Size of 0.05m in Plane YZ)

The third simulation involves a semi-structured mesh of hexahedral elements. Similar to the second mesh, it maintains a typical element size of 0.05 m in the YZ plane, but the number of divisions along the X-direction has been increased to 15. The mesh generated for this configuration

is shown in Figure 8.

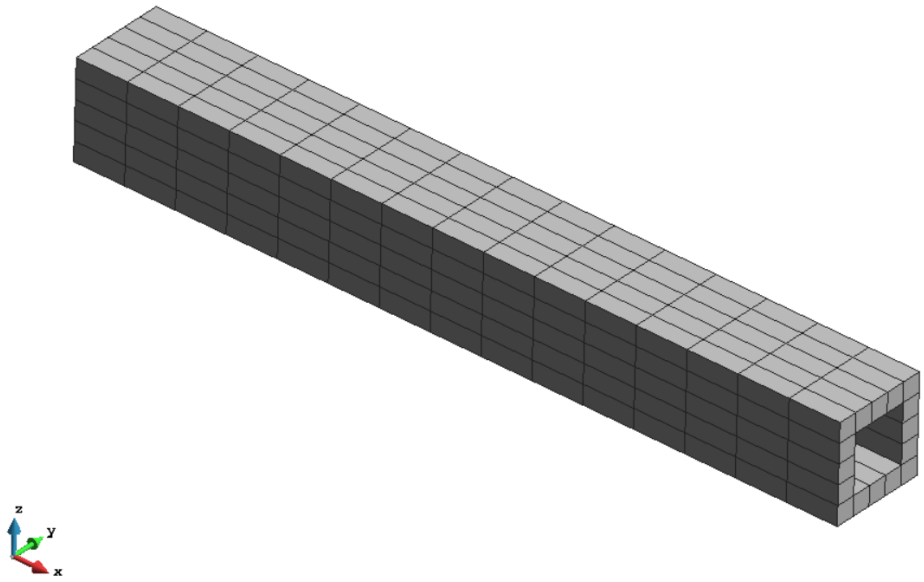

Figure 8: Generated mesh for the case with 15 divisions in the X-direction.

Postprocessing the results from the MATLAB finite element program, the vertical displacement field for the beam was obtained using GID. The distribution of vertical displacements for this simulation is displayed in Figure 9.

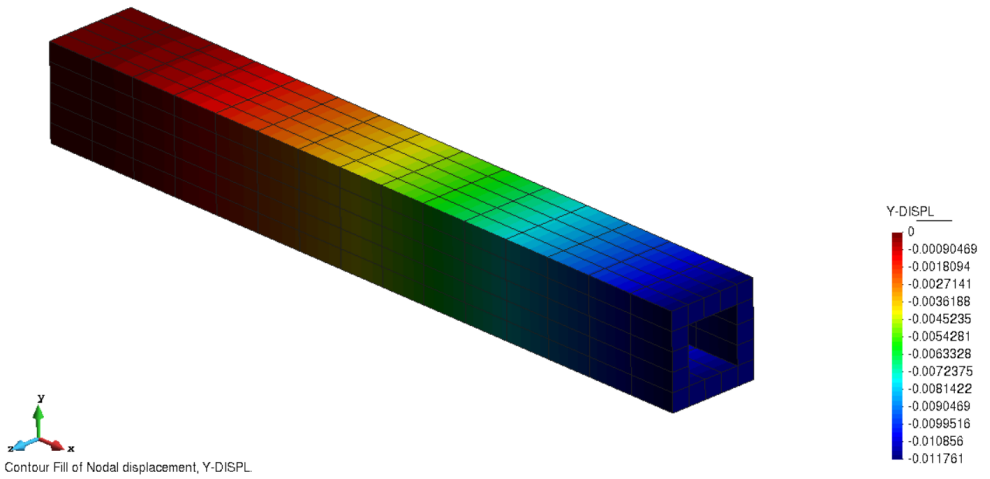

Figure 9: Postprocessed results showing vertical displacements for the third mesh.

1.3.4 Simulation for the Fourth Mesh (20 Divisions in X Direction, Typical Size of 0.025m in Plane YZ)

The final simulation evaluates a semi-structured hexahedral mesh with 20 divisions along the X-direction and a typical element size of 0.025 m in the YZ plane. The generated mesh for this configuration, created using GID software, is shown in Figure 10.

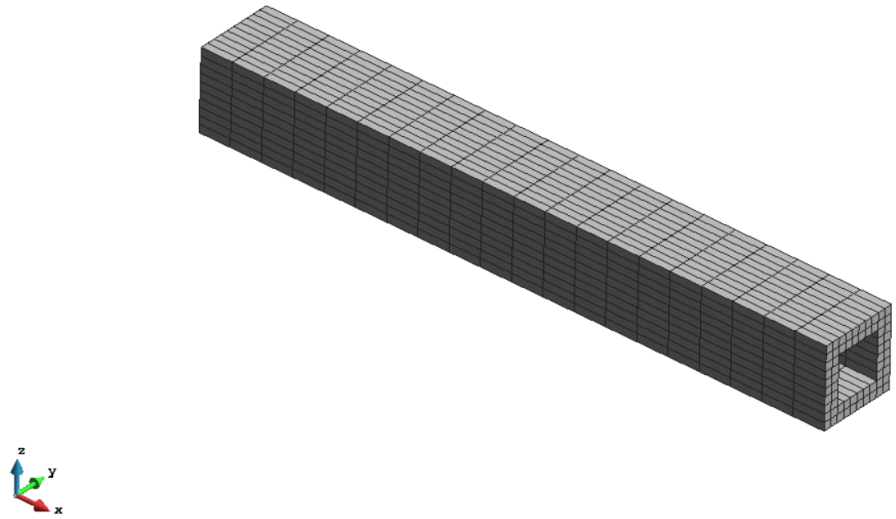

Figure 10: Generated mesh for the case with 20 divisions in the X-direction.

For this mesh, the vertical displacement field of the beam, as computed by the MATLAB finite element program and postprocessed with GID, is displayed in Figure 11.

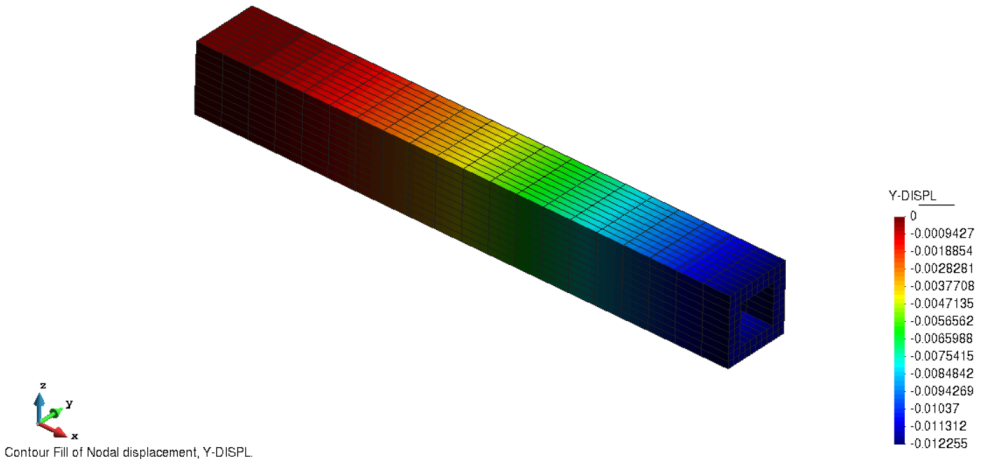

Figure 11: Postprocessed results showing vertical displacements for the fourth mesh.

1.4 Theoretical Calculation of Tip Deflection

To evaluate the convergence of the simulation results discussed earlier, the theoretical deflection at the beam's tip is derived in this section, relying on the geometric and physical parameters provided in the problem statement.

The process begins with constructing the force and moment diagrams for the cantilever beam, as depicted in Figure 12. In this context, q denotes the uniformly distributed load acting along the beam's length.

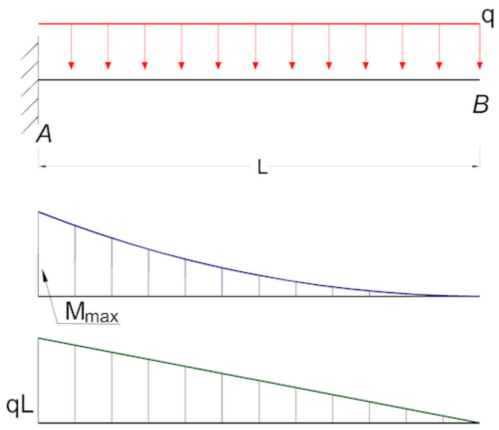

Figure 12: Force distribution, shear, and bending moment along the cantilever beam. Source: The Engineering Toolbox.

Based on these diagrams, the following expressions are obtained to describe the internal forces and moments at any point along the beam:

$$T(x) = q \cdot h \cdot (L - x) \tag{3}$$

$$M(x) = \int T(x)\, dx = q \cdot h \cdot \left(L \cdot x - \frac{x^2}{2} \right) \tag{4}$$

Here, h refers to the beam's cross-sectional dimension. The vertical deflection $\delta(x)$ of the beam is computed using the following double integral:

$$\delta(x) = \iint \frac{q \cdot h}{E \cdot I} \left(L \cdot x - \frac{x^2}{2} \right) dx\, dx \tag{5}$$

Imposing the boundary conditions, $\delta(0) = 0$ at $x = 0$ and $\theta(L) = 0$ at $x = L$ (where θ represents the slope), the deflection at any point along the beam can be expressed as:

$$\delta(x) = -\frac{h \cdot q \cdot x^2}{24 \cdot E \cdot I} \cdot \left(6 \cdot L^2 - 4 \cdot L \cdot x + x^2 \right) \tag{6}$$

For a square cross-section, the moment of inertia I is computed as:

$$I = \frac{1}{12} \left[h^4 - (h - 2 \cdot e)^4 \right] \tag{7}$$

Substituting the given parameters yields:

$$I = 2.83 \times 10^{-4}\, \mathrm{m}^4 \tag{8}$$

Finally, by inserting this value of I into Equation 6, the theoretical tip deflection $\delta(L)$ is found to be:

$$\delta(L) \approx -0.0125\, \mathrm{m} \tag{9}$$

With the theoretical vertical deflection at the beam's tip established, the next section will assess the convergence of the finite element simulation results.

1.5 Convergence Analysis of the Finite Element Program Results

This section examines the convergence of the finite element program by comparing the simulated results with the theoretical displacements provided in Equation 6. The convergence is depicted in Figure 13, which illustrates the relationship between the number of mesh elements and the accuracy of the vertical displacement predictions. The MATLAB code used to compute the convergence results can be found in Appendix A.3.1.

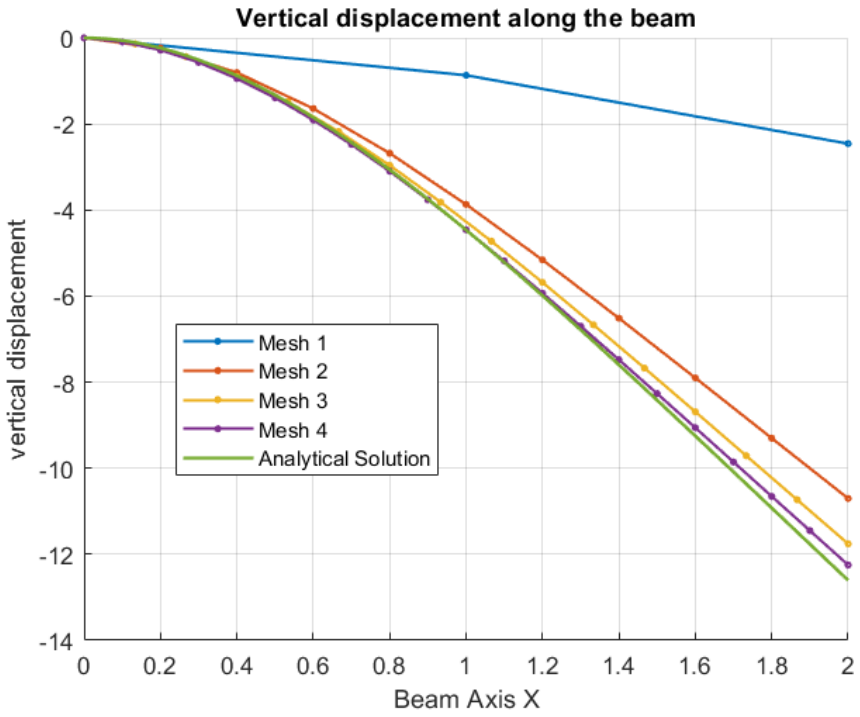

Figure 13: Convergence of finite element program results with theoretical vertical displacements.

The graph clearly shows that the relative error between the simulation results and the theoretical values diminishes as the number of discretization elements in the mesh increases. This trend confirms the principle that refining the mesh—by increasing the number of elements—reduces numerical error, thus demonstrating convergence in the simulation results. This behavior aligns with the findings discussed in the previous assignment, highlighting the importance of mesh refinement for accurate numerical analysis.

1.6 Additional Results from Mesh 4

To complement the analysis, further significant results have been extracted from the simulation using Mesh 4.

1.6.1 Stress Distribution Along the X-Axis

The stress distribution along the X-axis (σ_{XX}) for Mesh 4 is shown in Figure 14. This result represents the normal stresses caused by the uniformly distributed load applied on the beam's top surface.

The stress contours reveal a linear variation of the stresses along the beam's length, as expected

for a cantilever beam subjected to bending. Maximum tensile stresses appear at the top edge near the fixed end, while compressive stresses are observed at the bottom edge. These results align with the theoretical behavior of cantilever beams under loading conditions, where bending moments generate a stress gradient across the cross-section.

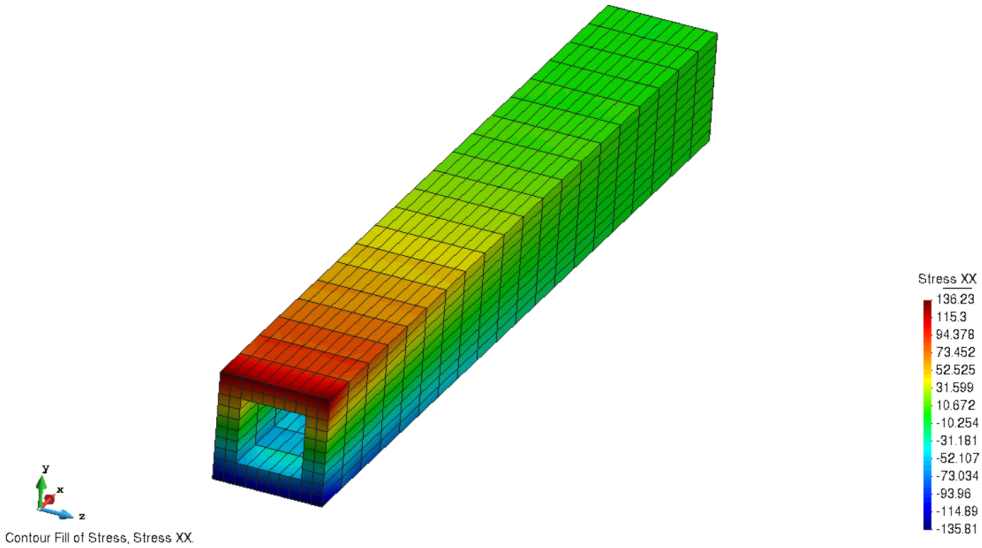

Figure 14: Stress distribution (σ_{XX}) along the cantilever beam for Mesh 4.

The analysis confirms that the stress distribution is symmetric across the neutral axis and highlights the regions of maximum stress concentration, which are critical for evaluating structural integrity.

1.6.2 Nodal Reactions at the Fixed Edge

The nodal reactions at the fixed edge of the cantilever beam have been analyzed to verify equilibrium with the applied forces. The results are presented in Figure 15, which shows both the scalar values of the nodal reactions and their vector representations.

The left image illustrates the magnitude of the nodal reactions, highlighting the regions where the highest reaction forces are concentrated, particularly along the beam's upper and lower edges. The right image provides a vector representation of the nodal reactions, scaled for better visualization. It clearly depicts the direction and distribution of the reaction forces at the fixed edge.

As expected, the nodal reactions counterbalance the applied uniformly distributed load, ensuring static equilibrium. The forces are symmetrically distributed across the fixed edge, while the vector directions align with the load's opposite effect.

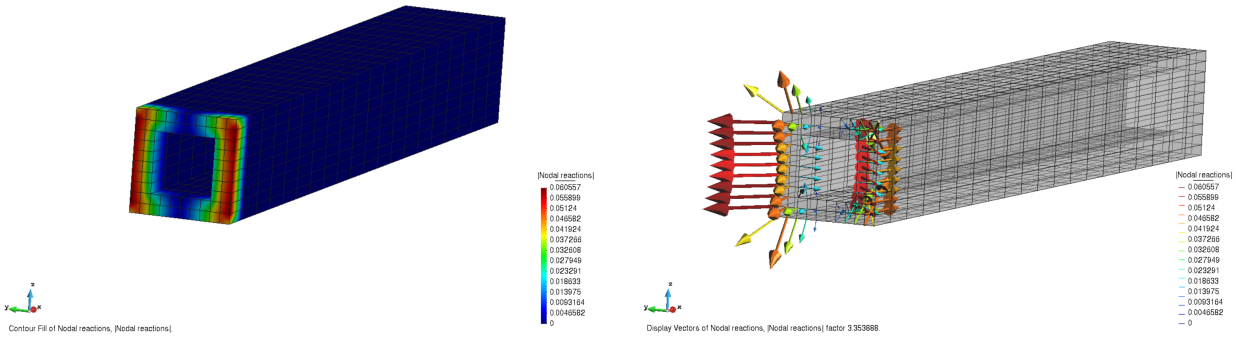

Figure 15: Nodal reactions at the fixed edge of the cantilever beam.

1.6.3 Verification of Resultant Forces and Moments

The nodal reactions obtained at the fixed edge are summarized as follows:

- $F_x = 0.00\,\text{MN}$
- $F_y = 0.25\,\text{MN}$
- $F_z = 0.00\,\text{MN}$
- $M_x = 0.00\,\text{MNm}$
- $M_y = 0.00\,\text{MNm}$
- $M_z = 0.25\,\text{MNm}$

The results confirm that the applied vertical load generates a vertical reaction force (F_y) and a moment (M_z) at the fixed edge. These values are consistent with the problem statement, where a uniformly distributed load $t(2) = 500\,\text{kN/m}^2$ is applied on the beam's top surface.

1.6.3.1 Calculation of the Resultant Force

The uniformly distributed load acts over the beam's top surface with dimensions $L_x = 2\,\text{m}$ and $h_y = 0.25\,\text{m}$. The total resultant force F_y is calculated as:

$$F_y = t(2) \cdot A_{\text{top}} = 500\,\text{kN/m}^2 \cdot (L_x \cdot h_y)$$

Substituting the values:

$$F_y = 500 \cdot (2 \cdot 0.25) = 500 \cdot 0.5 = 250\,\text{kN} = 0.250\,\text{MN}.$$

1.6.3.2 Calculation of the Resultant Moment About the Z-Axis (M_z)

The moment M_z is generated by the vertical load acting on the beam's top surface. The moment arm is half the beam's length ($L_x/2$). The resultant moment is given by:

$$M_z = F_y \cdot \left(\frac{L_x}{2}\right).$$

Substituting the values:

$$M_z = 250\,\text{kN} \cdot \left(\frac{2}{2}\right) = 250\,\text{kN} \cdot 1 = 250\,\text{kN} \cdot \text{m} = 0.250\,\text{MNm}.$$

1.6.3.3 Consistency with the Problem Statement

The calculated values for the resultant force F_y and moment M_z perfectly align with the nodal reactions obtained from the finite element analysis. The zero values for F_x, F_z, M_x, and M_y indicate no horizontal force or torsional moment, as expected for a uniformly distributed vertical load. This agreement validates both the theoretical assumptions and the numerical solution.

1.7 Finite Element Program with a Torque Applied to the Beam Tip

The finite element program developed in the previous sections is designed to handle forces but does not directly compute the effects of torques. To analyze the reactions of the beam when a torque is applied at its tip, Saint Venant's principle is utilized. According to this principle, the effects of a torque at the beam tip can be replicated by applying two forces in opposite directions at specific nodes along the edges of the beam.

The equivalent force, derived from the torque, is calculated as:

$$F = \frac{T}{h} = \frac{100}{0.25} = 400\,\text{kN} \tag{10}$$

For this analysis, the fourth mesh is used, and the nodes where the equivalent forces are applied are identified as nodes 31 and 192. These nodes, located on the half of the left and right side edges of the beam, are highlighted in Figure 16.

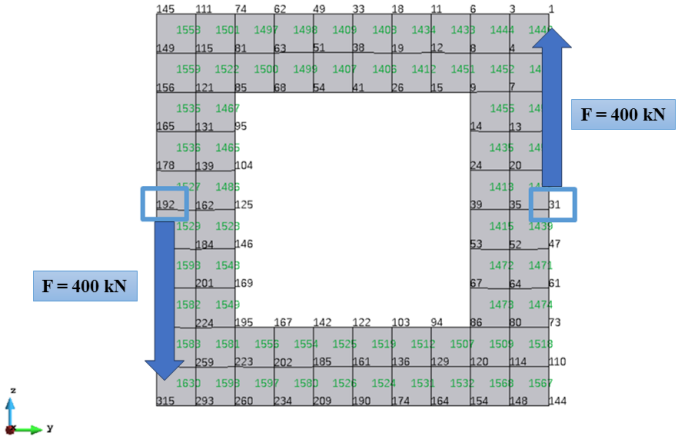

Figure 16: Location of nodes 31 and 192 where equivalent forces are applied to simulate a torque at the beam tip.

For enhanced visualization of the beam's response under torque, the figure depicts an exaggerated representation of the deformation, scaled by a factor of 30. This amplified view emphasizes the torsional effect on the cantilever beam, illustrating the resulting displacements and deformations induced by the applied torque at the free end. Such a scaled visualization aids in better understanding the structural behavior and deformation patterns that might otherwise appear negligible at real-world scales.

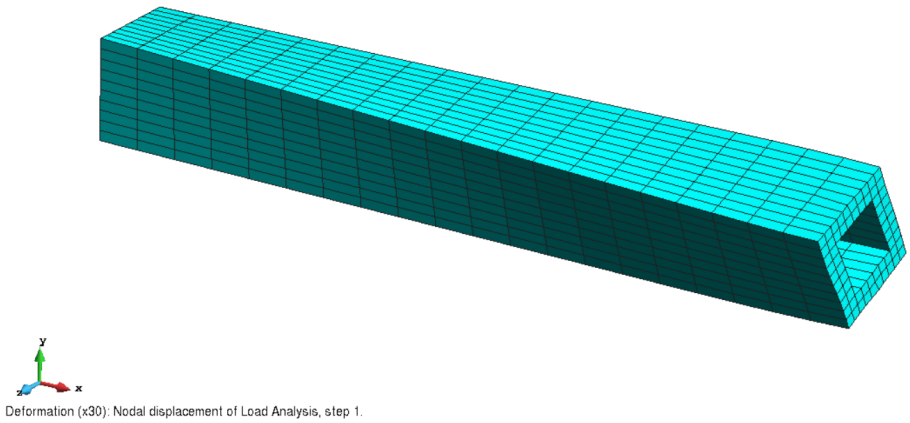

Figure 17: Exaggerated deformation of the cantilever beam under applied torque

The following images display the displacements of the cantilever beam in the Y- and Z-directions due to the applied torque at its free end. As expected, no displacement occurs in the X-direction since the torque acts about the X-axis, and the deformation is primarily observed as torsional effects perpendicular to this axis.

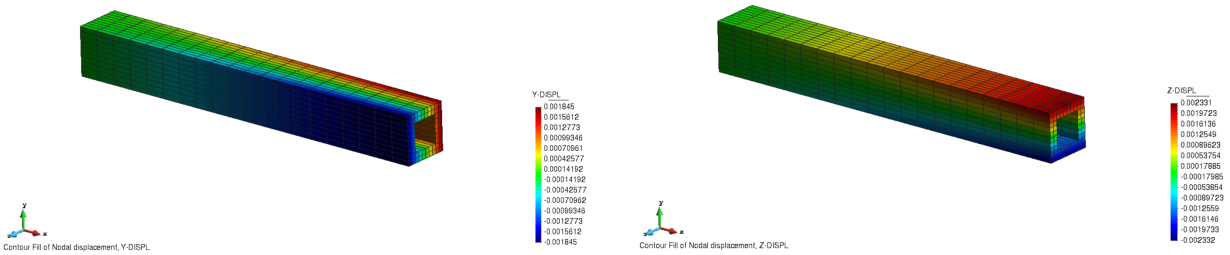

Figure 18: Displacements in the Y- and Z-directions caused by the applied torque.

1.7.1 Calculation of Reactions at the Fixed End

In addition to analyzing the displacements caused by the applied torque, the reactions at the fixed end of the cantilever beam have been calculated. As expected, the results show the presence of a moment in the X-axis, acting in the opposite direction to the applied torque. The generated forces cancel each other out, resulting in no reaction forces in the Y- or Z-directions. The MATLAB code used to calculate these reactions can be found in Appendix A.3.2.

$$M_x = -100 \; kN \cdot N \tag{11}$$

2 Part 2 (Advanced) - Thermoelasticity

2.1 Statement

Let $\Omega = \bigcup_{e=1}^{n_{el}} \Omega^e$ be a given finite element discretization. Suppose we solve the heat conduction problem explained in chapter 2 under given boundary conditions, and obtain the vector of temperatures, θ, at the nodes of the discretization (with respect to the reference temperature T_0). With this vector at our disposal, we can interpolate the relative temperature at any point within an element Ω^e by the corresponding shape functions:

$$\Delta T(x) = \mathcal{N}^e(x)\theta^e, \quad x \in \Omega^e \tag{12}$$

where θ^e denotes the vector of nodal temperatures of element Ω^e. With this consideration in mind, derive the discrete system of equilibrium equations arising from the variational principle when the stresses depend on the temperature.

2.2 Derivation of thermo-mechanical equilibrium equations

The derivation of the discrete system of equilibrium equations incorporates the thermo-mechanical constitutive equation, where the stress field is temperature-dependent:

$$\sigma(T) = \mathbf{C} \cdot (\nabla^s \mathbf{u} - \alpha \Delta T) = \mathbf{C} \cdot \nabla^s \mathbf{u} - \beta \Delta T. \tag{13}$$

The variational formulation of the boundary value problem is given by:

$$\int_\Omega \nabla^s \mathbf{v}^T \cdot \sigma \, d\Omega = \int_\Omega \mathbf{v}^T \mathbf{f} \, d\Omega + \int_{\Gamma_\sigma} \mathbf{v}^T \mathbf{t} \, d\Gamma. \tag{14}$$

Substitution of the constitutive equation into the variational formulation yields:

$$\int_\Omega \nabla^s \mathbf{v}^T (\mathbf{C} \nabla^s \mathbf{u} - \beta \Delta T) \, d\Omega = \int_\Omega \mathbf{v}^T \mathbf{f} \, d\Omega + \int_{\Gamma_\sigma} \mathbf{v}^T \mathbf{t} \, d\Gamma. \tag{15}$$

Rearranging terms leads to:

$$\int_\Omega \nabla^s \mathbf{v}^T \mathbf{C} \nabla^s \mathbf{u} \, d\Omega - \int_\Omega \nabla^s \mathbf{v}^T \beta \Delta T \, d\Omega = \int_\Omega \mathbf{v}^T \mathbf{f} \, d\Omega + \int_{\Gamma_\sigma} \mathbf{v}^T \mathbf{t} \, d\Gamma. \tag{16}$$

The relative temperature field is expressed as:

$$\Delta T(x) = \mathbf{N}^e(x)\theta^e, \quad x \in \Omega. \tag{17}$$

Displacements and test functions are approximated using:

$$\mathbf{u}(x) = \mathbf{N}\mathbf{d}, \quad \mathbf{v}(x) = \mathbf{N}\mathbf{c}, \tag{18}$$

with the derivative matrix $\mathbf{B} = \nabla^s \mathbf{N}$. Substitution into the variational equation produces:

$$\int_\Omega (\mathbf{B}^T \mathbf{c}^T) \mathbf{C} \mathbf{B} \mathbf{d} \, d\Omega = \int_\Omega \mathbf{c}^T (\mathbf{N}^T \mathbf{f} + \mathbf{B}^T \beta \mathbf{N} \theta) \, d\Omega + \int_{\Gamma_\sigma} \mathbf{N}^T \mathbf{c}^T \mathbf{t} \, d\Gamma. \tag{19}$$

The factor $\mathbf{c}^T$ is eliminated to obtain:

$$\mathbf{d}\int_\Omega \mathbf{B}^T\mathbf{C}\mathbf{B}\,d\Omega = \int_\Omega (\mathbf{N}^T\mathbf{f} + \mathbf{B}^T\beta\mathbf{N}\theta)\,d\Omega + \int_{\Gamma_\sigma} \mathbf{N}^T\mathbf{t}\,d\Gamma. \tag{20}$$

Rearranging terms results in:

$$\mathbf{d}\int_\Omega \mathbf{B}^T\mathbf{C}\mathbf{B}\,d\Omega - \int_\Omega (\mathbf{N}^T\mathbf{f} + \mathbf{B}^T\beta\mathbf{N}\theta)\,d\Omega - \int_{\Gamma_\sigma} \mathbf{N}^T\mathbf{t}\,d\Gamma = 0. \tag{21}$$

Comparison with the matrix structure $\mathbf{K}\mathbf{d} - \mathbf{F} = 0$ identifies the thermal force vector as:

$$\mathbf{F}^e_{\text{th}} = \int_{\Omega^e} (\mathbf{N}^T\mathbf{f} + \mathbf{B}^T\beta\mathbf{N}\theta^e)\,d\Omega + \int_{\Gamma^e_\sigma} \mathbf{N}^T\mathbf{t}\,d\Gamma. \tag{22}$$

Appendix A Code

A.1 Code for section 1.2.1

```
    function K = ComputeK(COOR,CN,TypeElement, celasglo) ;
%%%%
% This subroutine   returns the global stiffness matrix K (ndim*nnode x ndim*nnode)
% Inputs:   COOR: Coordinate matrix (nnode x ndim), % CN: Connectivity matrix (nelem x nnodeE), % TypeElement: Type of finite element (quadrilateral,...), celasglo (nstrain x nstrain x nelem)  % Array of elasticity matrices
% Dimensions of the problem
if nargin == 0
    load('tmp1.mat')
end
nnode = size(COOR,1); ndim = size(COOR,2); nelem = size(CN,1); nnodeE = size(CN,2) ;
% nstrain = size(celasglo,1) ;
% Shape function routines (for calculating shape functions and derivatives)
TypeIntegrand = 'K';
[weig,posgp,shapef,dershapef] = ComputeElementShapeFun(TypeElement,nnodeE,TypeIntegrand) ;
% Assembly of matrix K
% ----------------
K = sparse([],[],[],nnode*ndim,nnode*ndim,nnodeE*ndim*nelem) ;
%K = zeros(nnode*ndim) ;
for e = 1:nelem
    celas = celasglo(:,:,e) ;  % Stiffness matrix of element "e"
    CNloc = CN(e,:) ;   % Coordinates of the nodes of element "e"
    Xe = COOR(CNloc,:)' ;     % Computation of elemental stiffness matrix
    Ke = ComputeKeMatrix(celas,weig,dershapef,Xe) ;

       %error('You should program the assembly of the stiffness matrix !!!! ')
       % ------------------------------------------------------------------------
    % Assembly process
    indices = zeros(nnodeE * 3, 1);
    for node = 1:nnodeE
        indices((node-1)*3+1,1) = (CN(e,node)-1)*3+1;
        indices((node-1)*3+2,1) = (CN(e,node)-1)*3+2;
        indices((node-1)*3+3,1) = (CN(e,node)-1)*3+3;
    end
    K(indices, indices) = K(indices, indices) + Ke;

    if mod(e,10)==0  % To display on the screen the number of element being assembled
        disp(['e=',num2str(e)])
    end
end
```

A.2 Code for section 1.2.2

```
    function [weights, gaussPoints, shapeFuncs, shapeFuncDerivs] = Hexaedra8NInPoints()

% Define dimensions and constants
numDims = 3;
numNodesPerElem = 2^numDims;
weights = ones(1, numNodesPerElem);
numGaussPoints = length(weights);

% Define signs for Gauss points
signMatrix = [-1  1  1 -1 -1  1  1 -1;   % Signs for xi
```

```
            -1 -1  1  1 -1 -1  1  1;   % Signs for eta
            -1 -1 -1 -1  1  1  1  1];  % Signs for zeta

gaussPoints = (1 / sqrt(3)) * signMatrix;

% Initialize shape functions and derivatives
shapeFuncs = zeros(numGaussPoints, numNodesPerElem);
shapeFuncDerivs = zeros(numDims, numNodesPerElem, numGaussPoints);

% Compute shape functions and their derivatives at each Gauss point
for pointIdx = 1:numGaussPoints
    xiCoord = gaussPoints(1, pointIdx);
    etaCoord = gaussPoints(2, pointIdx);
    zetaCoord = gaussPoints(3, pointIdx);

    % Compute shape functions
    shapeFuncs(pointIdx, :) = (1 / numNodesPerElem) * [
        (1 - xiCoord) * (1 - etaCoord) * (1 - zetaCoord), ...
        (1 + xiCoord) * (1 - etaCoord) * (1 - zetaCoord), ...
        (1 + xiCoord) * (1 + etaCoord) * (1 - zetaCoord), ...
        (1 - xiCoord) * (1 + etaCoord) * (1 - zetaCoord), ...
        (1 - xiCoord) * (1 - etaCoord) * (1 + zetaCoord), ...
        (1 + xiCoord) * (1 - etaCoord) * (1 + zetaCoord), ...
        (1 + xiCoord) * (1 + etaCoord) * (1 + zetaCoord), ...
        (1 - xiCoord) * (1 + etaCoord) * (1 + zetaCoord)
    ];

    % Compute derivatives of shape functions
    shapeFuncDerivs(:, :, pointIdx) = (1 / numNodesPerElem) .* [
        signMatrix(1, :) .* [(1 - etaCoord) * (1 - zetaCoord), (1 - etaCoord) *
    (1 - zetaCoord), ...
    (1 + etaCoord) * (1 - zetaCoord), (1 + etaCoord) * (1 - zetaCoord), ...
    (1 - etaCoord) * (1 + zetaCoord), (1 - etaCoord) * (1 + zetaCoord), ...
    (1 + etaCoord) * (1 + zetaCoord), (1 + etaCoord) * (1 + zetaCoord)];
     signMatrix(2, :) .* [(1 - xiCoord) * (1 - zetaCoord), (1 + xiCoord) * (1 -
    zetaCoord), ...
    (1 + xiCoord) * (1 - zetaCoord), (1 - xiCoord) * (1 - zetaCoord), ...
    (1 - xiCoord) * (1 + zetaCoord), (1 + xiCoord) * (1 + zetaCoord), ...
    (1 + xiCoord) * (1 + zetaCoord), (1 - xiCoord) * (1 + zetaCoord)];
     signMatrix(3, :) .* [(1 - xiCoord) * (1 - etaCoord), (1 + xiCoord) * (1 -
    etaCoord), ...
    (1 + xiCoord) * (1 + etaCoord), (1 - xiCoord) * (1 + etaCoord), ...
    (1 - xiCoord) * (1 - etaCoord), (1 + xiCoord) * (1 - etaCoord), ...
    (1 + xiCoord) * (1 + etaCoord), (1 - xiCoord) * (1 + etaCoord)]
    ];
end

end
```

A.3 Code for section 1.2.3

```
    function [d strainGLO stressGLO  React posgp] = SolveELAS(K,Fb,Ftrac,dR,DOFr,
    COOR,CN,TypeElement,celasglo,...
    typePROBLEM,celasgloINV,DATA) ;
% This function returns   the (nnode*ndim x 1) vector of nodal displacements (d),
% as well as the arrays of stresses and strains
%%% points  (qheatGLO)
% Input data
% K = Global stiffness matrix   (nnode*ndim x nnode*ndim)
% Fb = External force vector due to  body forces  (nnode*ndim x 1)
% Ftrac = External force vector due to  boundary tractions    (nnode*ndim x 1)
% DOFr = Set of restricted DOFs
% dR = Vector of prescribed displacements
```

```
% ----------------------
if nargin == 0
    load('tmp.mat')
end
nnode = size(COOR,1); ndim = size(COOR,2); nelem = size(CN,1); nnodeE = size(CN,2) ;     %
% Solution of the system of FE equation
% Right-hand side
F = Fb + Ftrac ;
% Set of nodes at which temperature is unknown
DOFl = 1:nnode*ndim ;
DOFl(DOFr) = [] ;


%error('You should implement the solution of the system of equations, as well as the computation of nodal reaction forces')
% To be completed ....
% dL =  K^{-1}*(Fl .Klr*dR)
% ***


d = zeros(nnode*ndim,1) ; % Nodal displacements (initialization)
React = zeros(size(d)) ;  %  REaction forces  (initialization)


d(DOFl, 1) = K(DOFl, DOFl) \ (F(DOFl, 1) - K(DOFl, DOFr) * dR);
React(DOFr) = K(DOFr, DOFr) * d(DOFr, 1) + K(DOFr, DOFl) * d(DOFl, 1) - F(DOFr, 1);

%%%% COmputation of strain and stress vector at each gauss point
disp('Computation of stress and strains at each Gauss point')
[strainGLO stressGLO posgp]= StressStrains(COOR,CN,TypeElement,celasglo,d,typePROBLEM,celasgloINV,DATA) ;
end
```

A.3.1 Code for section 1.5

```
    clear
close all
% Data for Mesh_1
x1 = [0, 1, 2];
y1 = [0, -0.000868, -0.002458];

% Data for Mesh_2
x2 = [0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2];
y2 = [0, -0.000239, -0.000805, -0.001642, -0.002681, -0.003869, ...
      -0.005159, -0.006513, -0.007901, -0.009302, -0.010703];

% Data for Mesh_3
x3 = [0, 0.133333, 0.266667, 0.4, 0.533333, 0.666667, 0.8, 0.933333, ...
      1.06667, 1.2, 1.33333, 1.46667, 1.6, 1.73333, 1.86667, 2];
y3 = [0, -0.000136, -0.000437, -0.000895, -0.001482, -0.002177, ...
      -0.002961, -0.003816, -0.004728, -0.005682, -0.006667, ...
      -0.007671, -0.008688, -0.009711, -0.010734, -0.011756];

% Data for Mesh_4
x4 = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, ...
      1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2];
y4 = [0, -9.4e-05, -0.000281, -0.00057, -0.000944, -0.001392, ...
      -0.001907, -0.002478, -0.003099, -0.003762, -0.004459, ...
      -0.005185, -0.005933, -0.006699, -0.007478, -0.008266, ...
      -0.009059, -0.009856, -0.010654, -0.011451, -0.012248];

```

```
%Theoretical displacements
L = 2;                          % Beam longitude
h = 0.25;                       % Width and height
e = 0.05;                       % Thickness
E = 70e9;                       % Young modulus
I = 1/12 * (h^4-(h-2*e)^4);     % Inertia
t = 5e5;                        % Vertical force
xT = 0:0.001:2;
yT = -h*t*xT.^2/(24.*E.*I).*(xT.^2+6.*L^2-4*L.*xT);


% Plot all Mesh data
figure(1);
hold on;
plot(x1, y1*1e3, '-o', 'LineWidth', 1.2, 'MarkerSize', 2, 'DisplayName', 'Mesh 1');
plot(x2, y2*1e3, '-o', 'LineWidth', 1.2, 'MarkerSize', 2, 'DisplayName', 'Mesh 2');
plot(x3, y3*1e3, '-o', 'LineWidth', 1.2, 'MarkerSize', 2, 'DisplayName', 'Mesh 3');
plot(x4, y4*1e3, '-o', 'LineWidth', 1.2, 'MarkerSize', 2, 'DisplayName', 'Mesh 4');
plot(xT, yT*1e3, 'LineWidth', 1.4, 'MarkerSize', 2, 'DisplayName', 'Analytical Solution');

hold off;

% Graph labels and legend
xlabel('Beam Axis X');
ylabel('vertical displacement');
title('Vertical displacement along the beam');
legend('show');
grid on;
```

A.3.2 Code for section 1.7.1

```
    % Load data from the provided file
load('INFO_FE.mat'); % Replace with the correct filename if needed

% Initialize reaction forces
Fx = 0;
Fy = 0;
Fz = 0;

% Initialize moments
Mx = 0;
My = 0;
Mz = 0;

% Reshape reaction forces into a matrix [n x 3]
Fr = [React(1:3:end).'; React(2:3:end).'; React(3:3:end).'].';

% Reference point for moment calculation
ref_point = [0 .25/2 -0.25/2];

% Calculate resultant reactions and moments
for i = 1:size(Fr, 1)
    Fx = Fx + Fr(i, 1);
    Fy = Fy + Fr(i, 2);
    Fz = Fz + Fr(i, 3);

    displacement = [(COOR(i, 1) - ref_point(1)), ...
                    (COOR(i, 2) - ref_point(2)), ...
```

```
                    (COOR(i, 3) - ref_point(3))];
    moment = cross(displacement, Fr(i, :));

    Mx = Mx + moment(1);
    My = My + moment(2);
    Mz = Mz + moment(3);
end

% Display results
fprintf("\nComputed Reactions:\n\n");
fprintf("  * Fx = %f MN\n", Fx);
fprintf("  * Fy = %f MN\n", Fy);
fprintf("  * Fz = %f MN\n", Fz);
fprintf("  * Mx = %f MNm\n", Mx);
fprintf("  * My = %f MNm\n", My);
fprintf("  * Mz = %f MNm\n", Mz);

% Final reactions vector
Reactions = [Fx, Fy, Fz, Mx, My, Mz];
```