Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Baustatik und Konstruktion, ETH Zürich

Master Thesis FS 2025

Towards Implementing and Dimensioning of Fiber Elements in Educational Software

Berner Fabio (19-926-013)

Zürich, 30.06.2025

Supervisors: Prof. Dr. Eleni Chatzi
Dr. Adrian Egger
Dr. Konstantinos Vlachas
Chair of Structural Mechanics and Monitoring

Abstract

In the past years, advanced numerical simulations have become crucial in civil engineering, especially in the context of performance-based design and seismic assessments. Fiber beam elements stand out by accurately modeling distributed nonlinear behavior like cracking and yielding and therefore significantly improve the predictions of structural responses beyond the elastic limits of the materials. However, existing commercial and open-source tools on one side lack transparency or on the other hand are difficult in usability or understanding, making them less suitable for educational purposes.

Responding to this gap, this thesis presents the development of a modular, transparent and flexibility-based fiber element solver. It integrates cross-sectional analysis tools, nonlinear material models which can easily be extended, and visualizations to provide students an accessible tool to deeply engage with complex structural behaviors. Validated against established software like FAGUS-9 for cross-sectional analysis, STATIK-9 for linear solutions, and OpenSees regarding nonlinear force-deformation curves, the solver ensures accurate simulations.

Zusammenfassung

In den letzten Jahren wurden fortgeschrittene numerische Simulationen im Bauingenieurwesen immer wichtiger, insbesondere im Zusammenhang mit leistungsbasiertem Bemessen von Strukturen und deren Beurteilung bezüglich Erdbebenbelastungen. Sogenannte "fiber beam elements", welche Balken mittels dünnen Fasern und einem uni-axialen Materialgesetz beschreiben, haben sich besonders bewährt, da sie in der Lage sind, das verteilte, nichtlineare Verhalten wie Rissbildung und Fliessen genau zu modellieren. Dadurch kann das Verhalten von Strukturen über das elastische Limit der Materialien hinaus deutlich besser vorhergesagt werden. Vorhandene kommerzielle und Open-Source-Softwares sind jedoch meist entweder wenig transparent oder schwer zu bedienen beziehungsweise zu verstehen, wodurch sie nicht wirklich für den Einsatz in der Bildung geeignet sind.

Diese Lücke versucht die vorliegende Arbeit mittels eines modularen, transparenten und sogenannten "flexibility-based fiber element solver" zu schliessen. Das entwickelte Python Framework integriert Werkzeuge zur Querschnittsanalyse, leicht erweiterbare nichtlineare Materialmodelle sowie anschauliche Visualisierungen, wodurch Studierenden ein einfach zugängliches Werkzeug geboten wird, um komplexe Strukturverhalten besser zu verstehen. Der entwickelte Solver wurde umfassend validiert, unter anderem mit der Software FAGUS-9 hinsichtlich Querschnittsanalysen, STATIK-9 bezüglich linearem Berechnen von Strukturen und OpenSees in Bezug auf nichtlineare Kraft-Deformations-Kurven.

Acknowledgments

I would like to express my sincere gratitude to Dr. Adrian Egger for his continuous support, insightful feedback and meetings throughout the duration of the thesis. His commitment and guidance played a crucial role in the development of my work.

Furthermore, I also wish to thank Prof. Dr. Eleni Chatzi for providing the academic framework and opportunity to carry out this thesis at the chair of Structural Mechanics and Monitoring. Her role as the official supervisor enabled this project to take place.

I am grateful to Dr. Konstantinos Vlachas for coordinating and scheduling matters, contributing to the smooth progress of the administrative aspects of the thesis as well as for his significant feedback at the intermediate presentations.

My deepest thanks go to my family and friends, whose encouragement and understanding provided me with the motivation and emotional support necessary to complete this journey.

Contents

List of Abbreviations

Latin Abbreviations

A_{fib}	cross-sectional area of a fiber
A_{fib}	cross-sectional area of a beam
A_{ij}	the i-th and j-th entry of the $[A]$ block-matrix in the $[ABD]$ matrix
$\mathbf{b}_{cs}$	force interpolation matrix of a cross section $[3 \times 5]$
B_{ij}	the i-th and j-th entry of the $[B]$ block-matrix in the $[ABD]$ matrix
$c_{y,fib}$	y-coordinate of a fiber's centroid
$c_{y,cs}$	y-coordinate of a cross section's centroid
$c_{z,fib}$	z-coordinate of a fiber's centroid
$c_{z,cs}$	z-coordinate of a cross section's centroid
D_{ij}	the i-th and j-th entry of the $[D]$ block-matrix in the $[ABD]$ matrix
DOF	degree of freedom
e_1	1st unit vector in global coordinates in the direction of the beam
e_2	2nd unit vector in global coordinates, perpendicular to e_1 and e_3
e_3	3rd unit vector in global coordinates, perpendicular to e_1 and e_2
E	young's modulus
$E_{t,fib}$	tangent modulus of a fiber's material
$\mathbf{F}$	force vector in the case of the linear solver
$\mathbf{F}_{beam}$	flexibility matrix of a beam $[5 \times 5]$
$\mathbf{f}_{cs}$	flexibility matrix of a cross section $[3 \times 3]$
f_c	concrete compressive strength
f_{sk}	characteristic yield strength of rebar B500B
f_{uk}	ultimate strength of rebar B500B
f_{yk}	characteristic yield strength of steel S235
$\mathbf{f}_{ext}$	external force vector
g^i	value of the constraint function used at the previous Newton-Raphson increment
$\mathbf{h}$	gradient of g with respect to $\mathbf{u}$, so $\mathbf{h} = \frac{\partial \mathrm{g}}{\partial \mathbf{u}}$
H	hardening modulus
$I_{y,fib}$	moment of inertia of a fiber around the local y-axis
$I_{y,cs}$	moment of inertia of a cross section around the local y-axis
$I_{z,fib}$	moment of inertia of a fiber around the local z-axis

$I_{z,cs}$	moment of inertia of a cross section around the local z-axis
J_{cs}	Jacobian in the beam element integration
$\mathbf{k}_{cs}$	stiffness matrix of a cross section $[3 \times 3]$
$\mathbf{K}_{beam,local}$	local stiffness matrix of a beam $[5 \times 5]$
$\mathbf{K}_{beam,global}$	global stiffness matrix of a beam $[12 \times 12]$
$\mathbf{K}_T$	tangent stiffness matrix
$\mathbf{L}$	transformation matrix for the expansion of DOFs $[12 \times 5]$
M_{xx}	bending moment in a lamina material around the x-axis
M_{xy}	twisting moment in a lamina material
M_y	bending moment around the local y-axis
M_{y1}	bending moment around the local y-axis at the 1st node of a beam
M_{y2}	bending moment around the local y-axis at the 2nd node of a beam
M_{yy}	bending moment in a lamina material around the y-axis
M_z	bending moment around the local z-axis
M_{z1}	bending moment around the local z-axis at the 1st node of a beam
M_{z2}	bending moment around the local z-axis at the 2nd node of a beam
n	number of nodes per fiber
N	normal force in a beam
N_{xx}	normal force in a lamina material in the x-direction
N_{xy}	shear force in a lamina material
N_{yy}	normal force in a lamina material in the y-direction
$\mathbf{Q}$	a ply's constitutive matrix in its main directions $[3 \times 3]$
$\mathbf{Q}_{beam}$	beam element forces
$\mathbf{Q}_{cs}(\xi)$	section forces
$\mathbf{Q}_{R,cs}(\xi)$	resisting section forces
$\mathbf{Q}_{U,cs}(\xi)$	unbalanced section forces
$\overline{\mathbf{Q}}$	a ply's constitutive matrix in x-y directions $[3 \times 3]$
$\overline{Q}_{ij}$	the i-th and j-th entry of the $\overline{\mathbf{Q}}$ matrix
$\mathbf{R}$	residual force vector
$\mathbf{r}_{beam}$	beam element deformation residuals
$\mathbf{r}_{cs}(\xi)$	section deformation residuals
$\mathbf{Rot}$	rotational matrix $[12 \times 12]$
s	derivative of g with respect to λ, so $s = \frac{\partial \mathrm{g}}{\partial \lambda}$
$\mathbf{T}$	a ply's rotational matrix $[3 \times 3]$
tol_{NR}	Newton Raphson convergence tolerance
tol_{CS}	cross section convergence tolerance
$\mathbf{U}$	displacement vector in the case of the linear solver
$\mathbf{u}^0$	initial displacements at the beginning of the Newton Raphson iteration
$\mathbf{u}_{cs}(\xi)$	section deformations
u_1	displacement in the local x-direction of the 1st node of a beam
u_2	displacement in the local x-direction of the 2nd node of a beam

v_1	displacement in the local y-direction of the 1st node of a beam
v_2	displacement in the local y-direction of the 2nd node of a beam
w_1	displacement in the local z-direction of the 1st node of a beam
w_2	displacement in the local z-direction of the 2nd node of a beam
y_{fib}	a fiber's y-coordinate (centroid)
y_i	y-coordinate of the i-th node
z_{fib}	a fiber's z-coordinate (centroid)
z_i	z-coordinate of the i-th node
z_k	the distance from the center of a lamina material to the center of the k-th ply

Greek Abbreviations

γ_{xy}	shear strain in a lamina material
Δ	deformation of a beam in its longitudinal direction
$\Delta\lambda$	load factor increment
$\Delta\lambda^p$	predictor step load factor increment
$\Delta\mathbf{u}$	displacement increment
$\Delta\mathbf{u}^p$	predictor step displacement increment
$\Delta\mathbf{u}_{cs}(\xi)$	section deformation increment
$\Delta\mathbf{u}_{beam}$	displacement increment in the beam element DOFs
$\Delta\mathbf{Q}_{beam}$	force increment in the beam element DOFs
$\Delta\mathbf{Q}_{cs}(\xi)$	section forces increment
ε	strain
ε_u	ultimate strain of rebar B500B
ε_{xx}	normal strain in the x-direction in a lamina material
ε_{yy}	normal strain in the y-direction in a lamina material
θ	rotation angle of a ply with respect to its main directions
θ_{x1}	rotation around the local x-axis at the 1st node of a beam (torsional DOF)
θ_{x2}	rotation around the local x-axis at the 2nd node of a beam (torsional DOF)
θ_{y1}	rotation around the local y-axis at the 1st node of a beam
θ_{y2}	rotation around the local y-axis at the 2nd node of a beam
θ_{z1}	rotation around the local z-axis at the 1st node of a beam
θ_{z2}	rotation around the local z-axis at the 2nd node of a beam
κ_{xx}	curvature around the x-axis in a lamina material
κ_{xy}	twisting curvature in a lamina material
κ_{yy}	curvature around the y-axis in a lamina material
λ^0	initial load factor at the beginning of the Newton Raphson iteration
ξ_{cs}	gauss-lobatto point
σ	stress
ω_{cs}	gauss-lobatto weight

1 Introduction and Motivation

The engineer's main task is the dimensioning of structures. This starts with the pre-dimensioning of beams and columns as well as the determination of distances and spans and ends with a detailed analysis of the structural behavior and its components. Nowadays, while manual calculations are still used to check for plausibility or pre-dimensioning, most of the calculations are carried out using numerical software. For most cases, linear-elastic calculations are sufficient, but especially regarding performance-based design or seismic assessment, the nonlinearity of the building materials has to be taken into account.

In these cases, so-called fiber beam-column elements play a crucial role and are widely used [1]. In a fiber element, consisting of multiple sections along the element, each cross-section is discretized into many fibers, each fiber having its own uniaxial stress-strain response. A schematic of a structure discretized into fibers is visible in Figure 1.1. This approach captures the spread of plasticity along a member, which is called distributed plasticity, as opposed to lumped plasticity models, where all the nonlinearity is concentrated at the beam element's end [1][2]. With the constitutive models of each fiber, most phenomena like cracking, yielding and even cyclic degradations can be captured to produce realistic results. In practice, fiber-based distributed plasticity models are considered more refined and reliable for simulating structural collapse mechanisms, though they are computationally heavier than hinge models that use pre-calibrated moment-rotation laws [1].

Analyzing a fiber element's nonlinear response (for example, determining a force-displacement curve of a cantilever) involves numerically integrating the contributions of the fibers across the section as well as integrating the section's response along the element length. In a typical fiber element formulation as proposed by Taucer et al. [2], the fiber's forces are summed to compute the section resultants (axial force N, bending moment M_y, M_z). Integration points, often using Gauss-Legendre or Gauss-Lobatto rules, are placed along the element to integrate these section forces and deformations over the member's length [3].

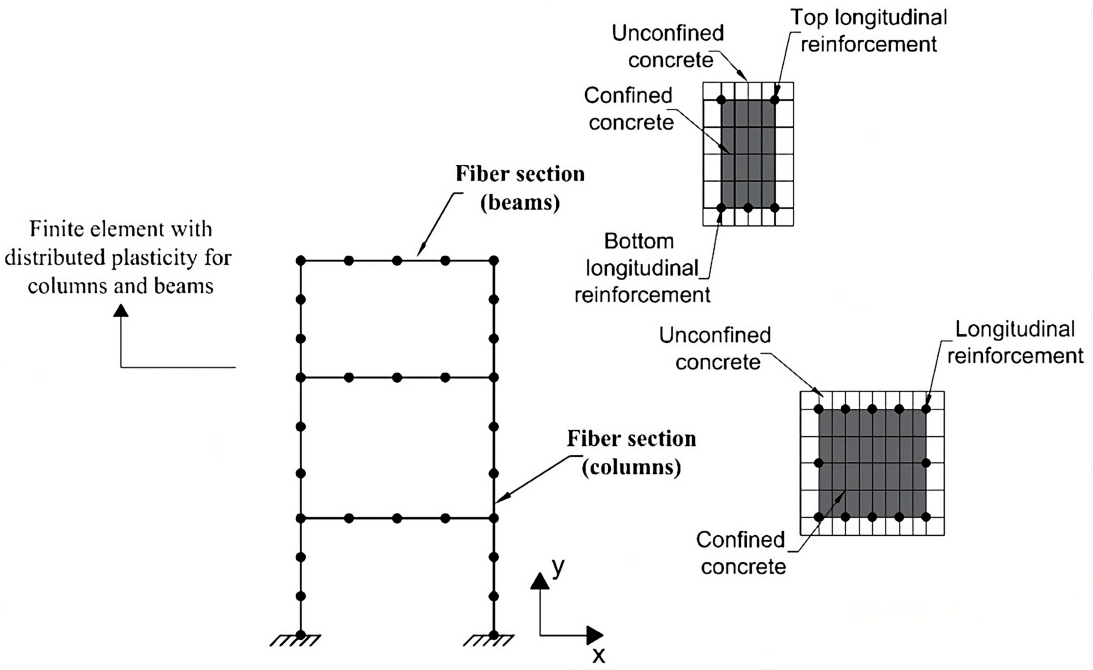


Figure 1.1: Schematic of a multi story frame discretized into fibers [4]

Two primary element formulations are used: a displacement-based (stiffness) and a force-based (flexibility) approach. On one hand, the displacement-based formulation, which uses assumed shape functions (polynomials) for the deflections, leads to a stiffness matrix. This approach may require a fine mesh to accurately capture the distributed plasticity [5]. Whereas, on the other hand, a force-based element, which uses the exact equilibrium interpolation (constant axial force and linear variation of the bending moments along the element), assembles a flexibility matrix. It enforces equilibrium and satisfies compatibility in an average sense, allowing a single element with several integration points to represent distributed yielding along the member [2].

This thesis forms the basis of a flexibility-based fiber element solver, with its idea originating from the work of Taucer and Filippou [2]. It serves multiple numerical methods: a) fiber integration, b) Gauss-Lobatto integration along the element, c) an iterative state determination of the cross sections as well as d) a Newton-Raphson solver at the structure level to ensure sectional and global equilibrium.

To support the full workflow of a practicing engineer, the framework also includes essential tools for cross-sectional analysis. These allow the calculation of detailed stress-strain responses, moment-curvature relationships, interaction diagrams, and cross-sectional properties such as area, centroid and moments of inertia. These form a package required in both preliminary design and detailed verification stages and therefore accompany an engineer throughout his day.

Sustainability and ecological responsibility are key considerations in modern structural engineering. To address these, the framework includes a lamina analysis tool based on the classical lamination theory and ABD-Matrices [6], enabling the simulation of layered sections such as tim-

ber laminates or retrofitted masonry walls reinforced with carbon-fiber grids. Timber laminates are widely used in construction, where the ecological part becomes important, while retrofitting existing structures plays a critical role in achieving sustainability goals by reducing demolition waste, lowering CO_2 emissions and encouraging the use of renewable materials. Both timber and retrofitted masonry typically involve very thin layers that are difficult to mesh individually. This is where the lamina analysis becomes essential by calculating equivalent material properties for a homogenized material, which can then be used as an input material law in the fiber element solver.

From an educational standpoint, the framework is implemented as an open, transparent, and modular Python tool. This makes it ideal for students to explore and understand the nonlinear structural behavior by interacting directly with the numerical algorithms as well as modifying the code to test new material laws or structures. Through its visual outputs, it promotes intuitive learning and bridges the gap between theory and application. Validation against sophisticated software such as FAGUS-9, STATIK-9 and OpenSees will be carried out to ensure its credibility and accuracy, making it not only a valuable teaching tool but also a functional environment for further research and development.

2 Theory

This section covers the most relevant theory that has been later implemented in the code shown in chapter 3. A deep focus lies on the cross-sections and their discretization regarding their geometrical properties, the integration of the fibers, as well as the cross-sectional analysis tool.

2.1 Cross Sectional Properties

As already mentioned in chapter 1, a fiber element consists of multiple cross-sections and each cross-section of multiple fibers. As section 3.2 will show, the cross-sections will be meshed with a Python library called *gmsh*. From this meshing, one can extract the nodes of each element. The mesh delivers rather quadrilateral or triangular elements. To calculate the cross-sectional properties consisting of these elements, or later called fibers, in a first manner, the properties of each fiber have to be calculated. Therefore, Borke (1997) [7] gives the following equations to calculate cross-sectional properties of polygons, assuming their nodes are ordered in a clockwise or counterclockwise orientation as seen in Figure 2.1.

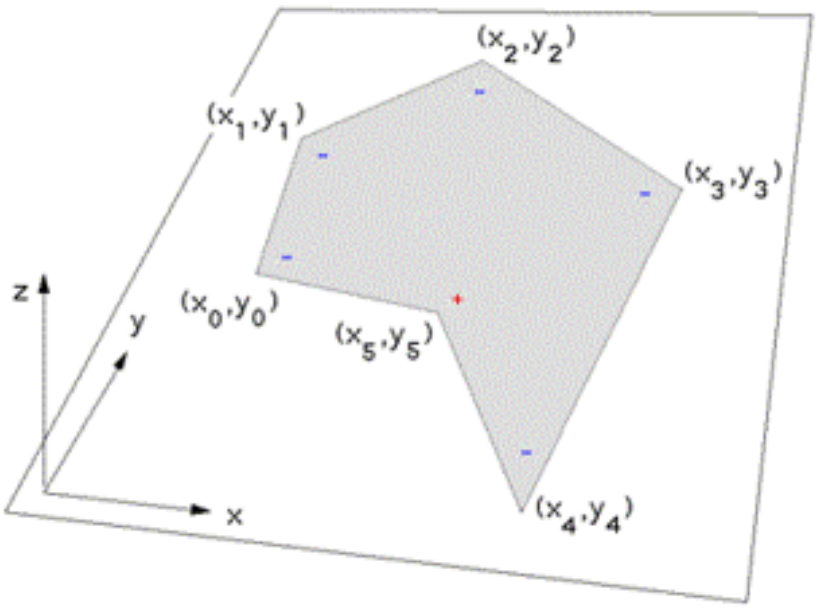


Figure 2.1: Visualization of a polygon in the x-y plane and its nodes $[x_i,y_i]$

It has to be mentioned that the node with index 0 has to be the same as the one with index N, so the polygon has to be closed. Inserting the node's coordinates of a polygon into Equation 2.1 gives the area of a fiber A_{fib}. In a difference to Borke (1997) [7], the axes' names are changed to y respectively z-axis because the x-axis belongs to the beam element's longitudinal axis.

$$A_{fib} = \frac{1}{2} \sum_{i=1}^{n} (y_{i+1} \cdot z_i - y_i \cdot z_{i+1}) \tag{2.1}$$

The coordinates of the centroid, also known as the center of gravity, are calculated using Equation 2.2 for the y-coordinate $c_{y,fib}$ respectively Equation 2.3 for the z-coordinate $c_{z,fib}$.

$$c_{y,fib} = \frac{1}{6 \cdot A_{fib}} \sum_{i=1}^{n} (y_i + y_{i+1}) \cdot (y_{i+1} \cdot z_i - y_i \cdot z_{i+1}) \tag{2.2}$$

$$c_{z,fib} = \frac{1}{6 \cdot A_{fib}} \sum_{i=1}^{n} (z_i + z_{i+1}) \cdot (y_{i+1} \cdot z_i - y_i \cdot z_{i+1}) \tag{2.3}$$

The moment of inertia of a fiber around the y-axis and around the z-axis are determined according to Equation 2.4 and Equation 2.5. The sum over the nodal coordinates results in the moment of inertia around the corresponding global axis of the mesh. To get the moment of inertia around the local axis which goes through the fiber's centroid, the parallel axis theorem is applied.

$$I_{y,fib} = \left[\frac{1}{12} \sum_{i=1}^{n} \left(z_i^2 + z_i \cdot z_{i+1} + z_{i+1}^2 \right) \cdot (y_{i+1} \cdot z_i - y_i \cdot z_{i+1}) \right] - A_{fib} \cdot c_{z,fib}^2 \tag{2.4}$$

$$I_{z,fib} = \left[\frac{1}{12} \sum_{i=1}^{n} \left(y_i^2 + y_i \cdot y_{i+1} + y_{i+1}^2 \right) \cdot (y_{i+1} \cdot z_i - y_i \cdot z_{i+1}) \right] - A_{fib} \cdot c_{y,fib}^2 \tag{2.5}$$

After all the individual fiber areas A_{fib} are computed, the cross-sectional area A_{cs} is just the sum over all fibers as in Equation 2.6.

$$A_{cs} = \sum_{fibers} A_{fib} \tag{2.6}$$

The centroid of a cross section $c_{y,cs}$ and $c_{z,cs}$ is determined as a weighted average of the fiber centroid weighted by the fibers' corresponding area.

$$c_{y,cs} = \frac{\sum_{fibers} A_{fib} \cdot c_{y,fib}}{A_{cs}} \tag{2.7}$$

$$c_{z,cs} = \frac{\sum_{fibers} A_{fib} \cdot c_{z,fib}}{A_{cs}} \tag{2.8}$$

To calculate the cross-sectional moment of inertia around the y and z-axis, the fibers' moment of inertia are summed up translated according to the parallel axis theorem to the centroid of the cross-section as visible in Equation 2.9 respectively Equation 2.10.

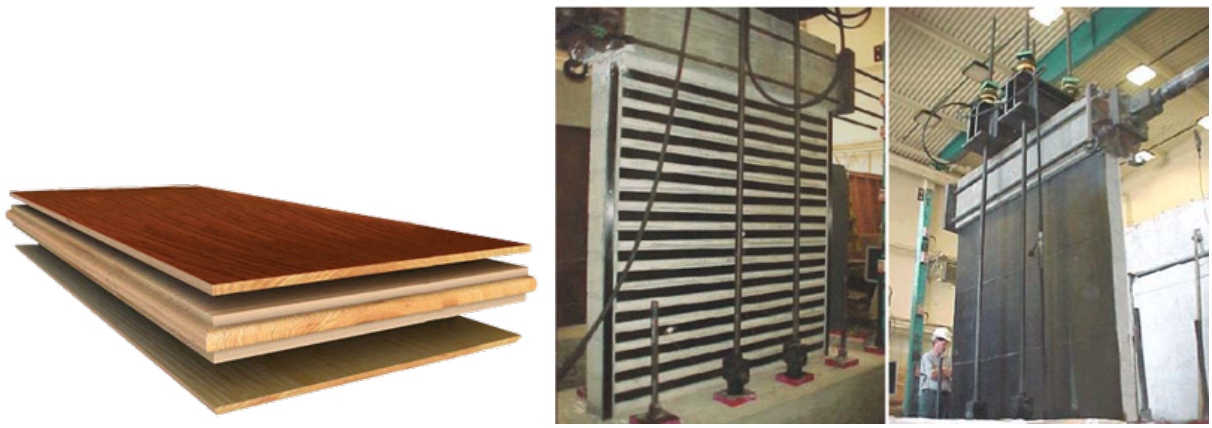
$$I_{y,cs} = \sum_{fibers} \left[I_{y,fib} + A_{fib} \cdot (c_{y,fib} - c_{y,cs})^2 \right] \tag{2.9}$$

$$I_{z,cs} = \sum_{fibers} \left[I_{z,fib} + A_{fib} \cdot (c_{z,fib} - c_{z,cs})^2 \right] \tag{2.10}$$

2.1.1 Lamina Analysis

For a composite material able to be discretized into fiber elements, each material needs a decent size. If, for example, in a wooden lamina section, one material is very thin compared to others, it is very difficult to mesh and depict the material properties of the individual layers accurately. That's where the so-called lamina analysis according to the classical lamina theory [6] comes into account.

As already mentioned in chapter 1, especially regarding ecological aspects, the lamina analysis plays a crucial role. Whether it is for a wood laminate, which acts as a renewable material in modern construction, or whether it is for reinforced masonry walls with carbon fibers, which are used for retrofitting existing buildings and therefore reducing demolition waste and enhancing the lifetime of existing buildings, thus reducing CO_2 emissions.



(a) (b)

Figure 2.2: (a) Example of a wooden laminate [8] and (b) a reinforced masonry wall with a carbon fiber grid [9]

The lamina analysis combines the material properties of each ply in the following so-called ABD-Matrix. This Matrix relates applied normal forces N_{xx}, N_{yy} and shear forces N_{xy} as well as bending moments M_{xx}, M_{yy} and twisting moments M_{xy} to the mid-sectional strains ε_{xx}, ε_{yy} and γ_{xy} as well as to the mid-sectional curvatures κ_{xx}, κ_{yy} and κ_{xy}.

$$\begin{bmatrix} N_{xx} \\ N_{yy} \\ N_{xy} \\ M_{xx} \\ M_{yy} \\ M_{xy} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} & B_{11} & B_{12} & B_{16} \\ A_{12} & A_{22} & A_{26} & B_{12} & B_{22} & B_{26} \\ A_{16} & A_{26} & A_{66} & B_{16} & B_{26} & B_{66} \\ B_{11} & B_{12} & B_{16} & D_{11} & D_{12} & D_{16} \\ B_{12} & B_{22} & B_{26} & D_{12} & D_{22} & D_{26} \\ B_{16} & B_{26} & B_{66} & D_{16} & D_{26} & D_{66} \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \\ \kappa_{xx} \\ \kappa_{yy} \\ \kappa_{xy} \end{bmatrix}$$

To build this ABD-Matrix, first, the **Q**-Matrix of each ply has to be assembled, which corresponds to the constitutive matrix with the following form:

$$\mathbf{Q} = \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix} = \begin{bmatrix} \frac{E_{11}^2}{E_{11} - \nu_{12}^2 \cdot E_{22}} & \frac{\nu_{12} \cdot E_{11} \cdot E_{22}}{E_{11} - \nu_{12}^2 \cdot E_{22}} & 0 \\ \frac{\nu_{12} \cdot E_{11} \cdot E_{22}}{E_{11} - \nu_{12}^2 \cdot E_{22}} & \frac{E_{11} \cdot E_{22}}{E_{11} - \nu_{12}^2 \cdot E_{22}} & 0 \\ 0 & 0 & G_{12} \end{bmatrix}$$

where E_{11} and E_{22} define the young's modulus in the main directions of an anisotropic material, G_{12} corresponds to the shear modulus and ν_{12} to the poisson's ratio.

Afterwards, if the specific ply is not an isotropic material and has a given rotation angle θ, the **Q**-Matrix has to be rotated, as seen in Equation 2.11, using the rotational matrix **T** depending on the ply's rotation angle θ.

$$\overline{\mathbf{Q}} = \mathbf{T} \cdot \mathbf{Q} \cdot \mathbf{T}^T \tag{2.11}$$

$$\mathbf{T} = \begin{bmatrix} \cos^2\theta & \sin^2\theta & 2 \cdot \sin\theta \cdot \cos\theta \\ \sin^2\theta & \cos^2\theta & -2 \cdot \sin\theta \cdot \cos\theta \\ -\sin\theta \cdot \cos\theta & \sin\theta \cdot \cos\theta & \cos^2\theta - \sin^2\theta \end{bmatrix}$$

After the rotation of the **Q**-Matrix, the entries of the ABD-Matrix are determined by a summation using the entries of the rotated matrix $\overline{\mathbf{Q}}_{ij}$ and the corresponding z_k, which corresponds to the z-coordinate of the ply's center measured from the center of the lamina.

The coefficients A_{ij} connect axial strains to axial forces, B_{ij} connect curvatures to axial forces and axial strains to bending moments and D_{ij} links curvatures to bending moments.

$$A_{ij} = \sum_{k=1}^{n_{ply}} \overline{Q}_{ij} \cdot (z_k - z_{k-1}) \tag{2.12}$$

$$B_{ij} = \frac{1}{2} \sum_{k=1}^{n_{ply}} \overline{Q}_{ij} \cdot \left(z_k^2 - z_{k-1}^2\right) \tag{2.13}$$

$$D_{ij} = \frac{1}{3} \sum_{k=1}^{n_{ply}} \overline{Q}_{ij} \cdot \left(z_k^3 - z_{k-1}^3\right) \tag{2.14}$$

From the constructed ABD-Matrix of a laminate material, the properties of an equivalent homogenized material can be obtained using the following formulas:

$$E_{xx} = A_{11}/h_{total} \qquad E_{yy} = A_{22}/h_{total} \qquad G_{xy} = A_{66}/h_{total} \qquad \nu_{xy} = A_{12}/A_{22}$$

By inverting the matrix to $[\text{ABD}]^{-1}$ and the application of some forces and bending moments, the laminate's response can be calculated. At first, this yields mid-plane strains and curvatures in the lamina material, which can then be computed to axial strains and stresses per lamina, as will be shown in section 4.1.1.

2.2 Assembly of the Structure

As previously stated and shown in Figure 1.1, a structure built of fiber elements is discretized in multiple steps. The following sections 2.2.1 to 2.2.4 deal with the assembly of the stiffness matrix $\mathbf{K}$ of such a structure, beginning on the smallest level of discretization: the fibers.

2.2.1 Fiber Level

The stiffness of a fiber is only defined by its tangent modulus $E_{t,fib}$. The tangent modulus of a fiber depends heavily on its assigned material and the corresponding stress-strain relationship. A few examples of these stress-strain relationships of the most common building materials such as Steel $S235$, Rebar $B500B$ and Concrete $C30/37$ are explained in section 2.5. Given the strain ε_{fib} of a fiber, the tangent modulus $E_{t,fib}$ can be read as the slope in the σ-ε-graph.

2.2.2 Cross Section Level

At the cross-sectional level, the axial stiffness of the fibers is integrated over the cross-section in the means of a summation. With each fiber's location, given by their centroid coordinates y_{fib} and z_{fib}, the fiber also contributes to a bending stiffness. The so-formed stiffness matrix of the cross-section $\mathbf{k}_{cs}$ is assembled as follows:

$$\mathbf{k}_{cs} = \begin{bmatrix} \sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot y_{fib}^2 & -\sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot y_{fib} \cdot z_{fib} & -\sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot y_{fib} \\ -\sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot y_{fib} \cdot z_{fib} & \sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot z_{fib}^2 & \sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot z_{fib} \\ -\sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot y_{fib} & \sum_{fibers} E_{t,fib} \cdot A_{fib} \cdot z_{fib} & \sum_{fibers} E_{t,fib} \cdot A_{fib} \end{bmatrix}$$

The first row and column correspond to the bending around the z-axis, the second around the y-axis and the third row and column link the axial strain and normal force.

At the beam level, that's where the flexibility-based approach comes into account. Thus, after the formation of the stiffness matrix, it has to be inverted as visible in Equation 2.15 to form the cross-sectional flexibility matrix $\mathbf{f}_{cs}$.

$$\mathbf{f}_{cs} = \mathbf{k}_{cs}^{-1} \tag{2.15}$$

2.2.3 Beam Level

At the beam level, the second integration happens. The cross-sectional flexibility has to be integrated along the multiple cross-sections along the beam. The higher the number of sections per beam element, the more accurate the results of the distributed plasticity are.

As an integrator over the beam element, the Gauss-Lobatto integration is chosen, which is further explained in section 2.2.3. First, the cross-sectional flexibility has to be multiplied by the so-called force interpolation matrix $\mathbf{b}_{cs}$, which depends on the location of the cross-section along the beam in an iso-parametric coordinate ξ. The following matrix shows that the chosen force interpolation matrix describes a linear dependency of the bending moment along each beam element for M_y and M_z and a constant value for the axial force N.

$$\mathbf{b}_{cs}(\xi) = \begin{array}{c} \\ M_z \\ M_y \\ N \end{array} \begin{array}{c} \begin{array}{ccccc} M_{z1} & M_{z2} & M_{y1} & M_{y2} & N \end{array} \\ \begin{bmatrix} \frac{\xi_{cs}+1}{2} & \frac{\xi_{cs}-1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{\xi_{cs}+1}{2} & \frac{\xi_{cs}-1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

As shown in Equation 2.16 the multiple cross-sectional flexibility matrices are summed up by multiplying them with the Gauss-Lobatto weight ω_{cs} and the determinant of the Jacobian $\det(J_{cs})$ which corresponds to half the beam element's length $L_{beam}/2$.

$$\mathbf{F}_{beam} = \sum_{sections} \left(\mathbf{b}_{cs}^{T}(\xi) \cdot \mathbf{f}_{cs} \cdot \mathbf{b}_{cs}(\xi)\right) \cdot \omega_{cs} \cdot \det(J_{cs}) \tag{2.16}$$

After the numerical integration, the beams flexibility matrix $\mathbf{F}_{beam}$ can be inverted to get the beam's local stiffness matrix $\mathbf{K}_{beam,local}$.

$$\mathbf{K}_{beam,local} = \mathbf{F}_{beam}^{-1} \tag{2.17}$$

Gauss-Lobatto Quadrature Rule

The Gaussian quadrature rules are used for numerical integration and they approximate an integral using a discrete set of points in the iso-parametric range $[-1, 1]$. The often used Gauss-Legendre quadrature rule does not include the end points -1 and 1. Therefore, the Gauss-Lobatto quadrature rule is more suited for this problem, as the formation of plastic deformations often starts at the nodes of a beam.

The Gauss-Lobatto integration is well suited for polynomials up to a degree of $2n-3$, where n is the number of the Gauss points. The middle points ξ_2 to ξ_{n-1} of the Gauss-Lobatto integration are defined as the zero points of the first derivative of the Legendre Polynomial P_{n-1} [10]. The Gauss-Lobatto weights are then calculated using Equation 2.18.

$$\omega_i = \frac{2}{n(n-1) \cdot P_{n-1}^{2}(\xi_i)} \tag{2.18}$$

A few of the first Gauss-Lobatto integration points and weights are shown in Table 2.1.

Table 2.1: example of the first few Gauss-Lobatto points ξ and weights ω

n	2	3		4		5		
ξ	± 1	0	± 1	$\pm\sqrt{1/5}$	± 1	0	$\pm\sqrt{3/7}$	± 1
ω	1	4/3	1/3	5/6	1/6	32/45	49/90	1/10

2.2.4 Structure Level

To get from the local beam stiffness matrix $\mathbf{K}_{beam,local}$ to the global beam stiffness matrix $\mathbf{K}_{beam,global}$, there are two necessary steps. First, the global beam coordinates θ_{z1}, θ_{z2}, θ_{y1}, θ_{y2} and Δ have to be expanded to the six global DOFs per node respectively 12 DOFs per beam element. This is done by multiplying the local beam stiffness matrix $\mathbf{K}_{beam,local}$ by the transformation matrix $\mathbf{L}$ defined as follows:

$$\mathbf{L} = \begin{array}{c|ccccc} & \theta_{z1} & \theta_{z2} & \theta_{y1} & \theta_{y2} & \Delta \\ u_1 & 0 & 0 & 0 & 0 & -1 \\ v_1 & -\frac{1}{l} & -\frac{1}{l} & 0 & 0 & 0 \\ w_1 & 0 & 0 & \frac{1}{l} & \frac{1}{l} & 0 \\ \theta_{x1} & 0 & 0 & 0 & 0 & 0 \\ \theta_{y1} & 0 & 0 & 1 & 0 & 0 \\ \theta_{z1} & 1 & 0 & 0 & 0 & 0 \\ \hline u_2 & 0 & 0 & 0 & 0 & 1 \\ v_2 & \frac{1}{l} & \frac{1}{l} & 0 & 0 & 0 \\ w_2 & 0 & 0 & -\frac{1}{l} & -\frac{1}{l} & 0 \\ \theta_{x2} & 0 & 0 & 0 & 0 & 0 \\ \theta_{y2} & 0 & 0 & 0 & 1 & 0 \\ \theta_{z2} & 0 & 1 & 0 & 0 & 0 \end{array}$$

The second step is the rotation of the local stiffness matrix to the orientation of the global coordinates. The rational matrix **Rot** is a block-diagonal matrix composed of the three column vectors e_1, e_2, and e_3. These vectors are all normal and unit vectors. e_1 corresponds to the beam's local x-direction. e_2 is defined as the beam's local y-direction and is generated as a cross-product of e_1 and the global z-axis, as long as the beam's x-axis is not equal to the global z-axis. The column vector e_3 is constructed as a cross-product of e_1 and e_2 and is the beam's local z-axis.

As the transformation matrix $\mathbf{L}$, to expand the beam's coordinates, and the rotational matrix **Rot**, to rotate the local to global coordinates, are defined, the beam's global stiffness matrix $\mathbf{K}_{beam,global}$ can be calculated according to Equation 2.19.

$$\mathbf{K}_{beam,global} = \mathbf{Rot}^T \cdot \left(\mathbf{L} \cdot \mathbf{K}_{beam,local} \cdot \mathbf{L}^T\right) \cdot \mathbf{Rot} \tag{2.19}$$

$$
\mathbf{Rot} = \begin{array}{c|ccc|ccc|ccc|ccc}
 & u_1 & v_1 & w_1 & \theta_{x1} & \theta_{y1} & \theta_{z1} & u_2 & v_2 & w_2 & \theta_{x2} & \theta_{y2} & \theta_{z2} \\
\hline
u_1 & | & | & | & & & & & & & & & \\
v_1 & e_1 & e_2 & e_3 & & \mathbf{0} & & & \mathbf{0} & & & \mathbf{0} & \\
w_1 & | & | & | & & & & & & & & & \\
\hline
\theta_{x1} & & & & | & | & | & & & & & & \\
\theta_{y1} & & \mathbf{0} & & e_1 & e_2 & e_3 & & \mathbf{0} & & & \mathbf{0} & \\
\theta_{z1} & & & & | & | & | & & & & & & \\
\hline
u_2 & & & & & & & | & | & | & & & \\
v_2 & & \mathbf{0} & & & \mathbf{0} & & e_1 & e_2 & e_3 & & \mathbf{0} & \\
w_2 & & & & & & & | & | & | & & & \\
\hline
\theta_{x2} & & & & & & & & & & | & | & | \\
\theta_{y2} & & \mathbf{0} & & & \mathbf{0} & & & \mathbf{0} & & e_1 & e_2 & e_3 \\
\theta_{z2} & & & & & & & & & & | & | & |
\end{array}
$$

Last but not least, each beam's global stiffness matrix $\mathbf{K}_{beam,global}$ can be added together at the specific degrees of freedom of the beams to form the structural stiffness matrix $\mathbf{K}_{struc}$.

$$\mathbf{K}_{struc}[DOF_{beam,i}] = \mathbf{K}_{beam,i,global} \tag{2.20}$$

2.3 Linear Solution Process

To solve a structure's linear response to some applied forces, Equation 2.21 has to be solved. The assembly of the structural stiffness matrix $\mathbf{K}$ has been explained before in section 2.2. The only remaining needed tasks are the assembly of the applied force vector $\mathbf{F}$ as well as the application of the natural boundary conditions to the structure [11].

$$\mathbf{F} = \mathbf{K} \cdot \mathbf{U} \tag{2.21}$$

To apply the natural boundary conditions, the following procedure is applied for all fixed degrees of freedom. If a fixed DOF is called f, and an iterable variable i represents all DOFs, then all entries $\mathbf{K}_{fi}$ and $\mathbf{K}_{if}$ of the stiffness matrix need to be zero, except $\mathbf{K}_{ff}$, which has to be assigned a value a few orders of magnitude smaller than the other entries in the stiffness matrix. The external force vector $\mathbf{F}$ has all 0 entries, except at the DOF where an external load should be applied. This DOF takes the value of the applied load.

If the boundary conditions are applied correctly, the stiffness matrix $\mathbf{K}$ can be inverted and the following equation can be solved to determine the structure's deformations $\mathbf{U}$.

$$\mathbf{U} = \mathbf{K}^{-1} \cdot \mathbf{F} \tag{2.22}$$

By the multiplication of the structures' deformation $\mathbf{U}$ with the initial stiffness matrix $\mathbf{K}$, before applying the boundary conditions, the reaction forces can be calculated. An example of this is shown in section 4.2.

2.4 Nonlinear Solution Process

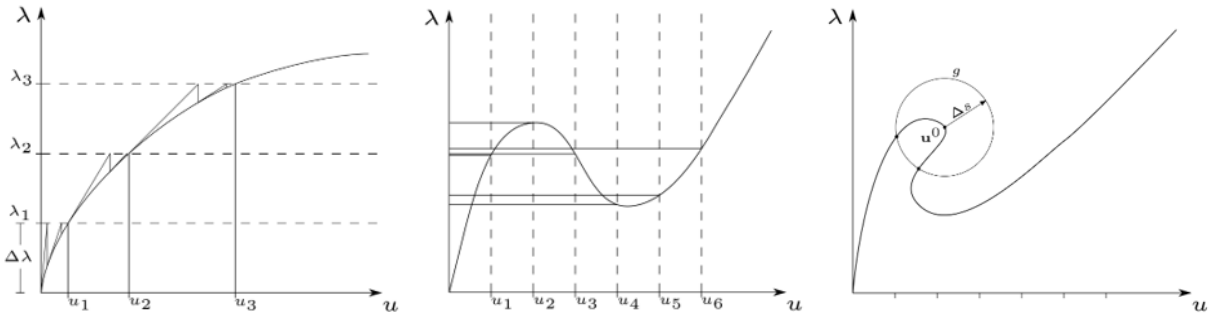
The nonlinear solution process involves three iteration loops in total. The most outer loop, indicated with the iteration variable k is the increasing load factor λ. The middle iteration loop is a Newton-Raphson iteration using an iteration variable i. Per Newton-Raphson iteration, for each beam element and for each cross-section, there is the most inner loop, indicated with j, which is the cross-sectional state determination. Section 2.4.1 deals with the Newton-Raphson iteration, whereas section 2.4.2 explains in detail the cross-sectional state-determination loop.

2.4.1 Newton Raphson

The procedure of the, in the framework implemented, Newton-Raphson iteration follows strictly the scheme taught in the lecture "Method of Finite Elements II" at ETH by Prof. Dr. Eleni Chatzi [12]. In general, at each load step k, the following equation has to be fulfilled.

$$\begin{bmatrix} \mathbf{K}_T & -\mathbf{f}_{ext} \\ \mathbf{h}^T & s \end{bmatrix} \begin{bmatrix} \Delta\mathbf{u} \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -\mathbf{R}^i \\ -\mathrm{g}^i \end{bmatrix}$$

The Newton-Raphson algorithm can be applied to different path-following constraints. A load-controlled approach increases with each increment the load factor, whereas a displacement-based approach increments the displacement of a specific DOF per iteration. There are also more complex methods like an arc-length method or even the Riks method.



(a) (b) (c)

Figure 2.3: Visualization of different Newton-Raphson constraints: (a) Load controlled (b) Displacement controlled and (c) Arc-Length method.

Figure 2.3 shows in a schematic way the procedure of different Newton-Raphson constraints. A load-controlled, a displacement-controlled approach, and the arc-length method are implemented in the framework as shown in chapter 3.

The iterative process of the Newton-Raphson requires the following steps:

1. Get the initial values of the load factor λ^0 and of the displacements $\mathbf{u}^0$. At the first load step these are 0 respectively the values from the last iteration at further load steps.

2. Get the residual force vector $\mathbf{R}$, tangent stiffness $\mathbf{K}_T$ and the external force vector $\mathbf{f}_{ext}$ from the structure.

3. Prediction of the displacement increment $\Delta\mathbf{u}^p$ using a specific Newton-Raphson constraint.

$$\Delta\mathbf{u}^p = \mathbf{K}_T^{-1} \cdot \mathbf{f}_{ext} \tag{2.23}$$

4. Prediction of the load factor increment $\Delta\lambda^p$, where κ is used for the definition of the sign of the increment and Δs is the increment.

$$\kappa = \frac{\mathbf{f}_{ext}^T \cdot \Delta\mathbf{u}^p}{\Delta\mathbf{u}^{pT} \cdot \Delta\mathbf{u}^p} \tag{2.24}$$

$$\Delta\lambda^p = \text{sign}(\kappa) \cdot \frac{\Delta s}{||\Delta\mathbf{u}^p||} \tag{2.25}$$

5. Update of the load factor and displacements

$$\lambda^1 = \lambda^0 + \Delta\lambda^p \qquad \mathbf{u}^1 = \mathbf{u}^0 + \Delta\lambda^p \cdot \Delta\mathbf{u}^p$$

6. Update of the residual force vector $\mathbf{R}$ and the tangent stiffness matrix $\mathbf{K}_T$

7. Start of the iteration procedure

8. Evaluation of the constraints g^i, $\mathbf{h}$ and s depending of the chosen constraints.

9. Solving of the block systems

$$\Delta\mathbf{u}^I = \mathbf{K}_T^{-1} \cdot \mathbf{f}_{ext} \qquad \Delta\mathbf{u}^{II} = -\mathbf{K}_T^{-1} \cdot \mathbf{R}^i$$

10. Calculation of the displacement and load factor increments, $\Delta\mathbf{u}$ respectively $\Delta\lambda$.

$$\Delta\lambda = -\frac{\mathrm{g}^i + \mathbf{h}^T \cdot \Delta\mathbf{u}^{II}}{s + \mathbf{h}^T \cdot \Delta\mathbf{u}^I} \qquad \Delta\mathbf{u} = \Delta\lambda \cdot \Delta\mathbf{u}^I + \Delta\mathbf{u}^{II}$$

11. Update of the residual force vector $\mathbf{R}$ and the tangent stiffness matrix $\mathbf{K}_T$ depending on the load factor increment $\Delta\lambda$ and displacement increment $\Delta\mathbf{u}$. During this step, the, in the next section 2.4.2 explained, cross-sectional state-determination is taken into account.

12. update of the solution variables λ, $\mathbf{u}$

$$\lambda^{i+1} = \lambda^i + \Delta\lambda \qquad \mathbf{u}^{i+1} = \mathbf{u}^i + \Delta\mathbf{u}$$

13. Check for convergence. The norm of the residual force vector $\mathbf{R}$ needs to be below a certain tolerance tol_{NR}. Go back to step 7. if the convergence criteria is not fulfilled.

$$||\mathbf{R}|| \leq tol_{NR} \tag{2.26}$$

2.4.2 Fiber Beam Element Iteration

The cross-sectional state determination plays a crucial role in a fiber element solver. The Newton-Raphson iteration returns the displacement increment $\Delta\mathbf{u}$. For these displacements, each section in each beam has to be in an applicable force-strain state, meaning the integration of the fiber's stresses, resulting from the strains at the section, has to be in equilibrium with the section forces resulting from the beam's deformation and the local stiffness of the section.

The following iterative process, which is implemented in the framework, follows the ideas of Taucer et al. [2] and Zidan [10]. In detail, these are the steps:

1. The Newton-Raphson algorithm calculates the global displacement increment $\Delta\mathbf{u}$. For each beam element in the structure, the displacement increments of the specific beam DOFs have to be transformed into the beams coordinates by

$$\Delta\mathbf{u}_{beam} = \mathbf{L}^T \left(\mathbf{Rot} \cdot \Delta\mathbf{u}\right) \tag{2.27}$$

2. The iteration on the beam level starts here, where from the displacement increment $\Delta\mathbf{u}_{beam}$ the force increment $\Delta\mathbf{Q}^j_{beam}$ is calculated using the stiffness matrix from the last Newton-Raphson iteration.

$$\Delta\mathbf{Q}^j_{beam} = \mathbf{K}^{j-1}_{beam,local} \cdot \Delta\mathbf{u}^j_{beam} \tag{2.28}$$

3. As a next step, the beam element forces $\mathbf{Q}^j_{beam}$ are updated.

$$\mathbf{Q}^j_{beam} = \mathbf{Q}^{j-1}_{beam} + \Delta\mathbf{Q}^j_{beam} \tag{2.29}$$

4. At this point starts the cross-sectional iteration process. For each cross-section in a beam element, the section force increments are calculated $\Delta\mathbf{Q}_{cs}^{j}(\xi)$, using the force-interpolation matrix $\mathbf{b}_{cs}(\xi)$ at the specific Gauss-Lobatto point ξ_{cs}, and the section forces $\mathbf{Q}_{cs}^{j}(\xi)$ are updated.

$$\Delta\mathbf{Q}_{cs}^{j}(\xi) = \mathbf{b}_{cs}(\xi) \cdot \Delta\mathbf{Q}_{beam}^{j} \tag{2.30}$$

$$\mathbf{Q}_{cs}^{j}(\xi) = \mathbf{Q}_{cs}^{j-1}(\xi) + \Delta\mathbf{Q}_{cs}^{j}(\xi) \tag{2.31}$$

5. The section deformations increments $\Delta\mathbf{u}_{cs}^{j}(\xi)$ (strain and curvature increments) are computed as a summation of the residual section deformations from the previous iteration $\mathbf{r}_{cs}^{j-1}(\xi)$ and the deformations caused by the force increment $\mathbf{Q}_{cs}^{j}(\xi)$. The residual deformations at the first iterations are $\mathbf{r}_{cs}^{0}(\xi) = \mathbf{0}$.

$$\Delta\mathbf{u}_{cs}^{j}(\xi) = \mathbf{r}_{cs}^{j-1}(\xi) + \mathbf{f}_{cs}^{j-1}(\xi) \cdot \mathbf{Q}_{cs}^{j}(\xi) \tag{2.32}$$

6. The section deformations are updated.

$$\mathbf{u}_{cs}^{j}(\xi) = \mathbf{u}_{cs}^{j-1}(\xi) + \Delta\mathbf{u}_{cs}^{j}(\xi) \tag{2.33}$$

7. Knowing these section deformations $\mathbf{u}_{cs}^{j}(\xi) = [\theta_y, \theta_z, \varepsilon]^T$ the tangent stiffness matrix $\mathbf{k}_{cs}^{j}(\xi)$ can be updated using the defined material law per fiber as mentioned in section 2.2.1.

8. With the new section stiffness matrix $\mathbf{k}_{cs}^{j}(\xi)$ the section resisting forces $\mathbf{Q}_{R,cs}^{j}(\xi)$ can be calculated

$$\mathbf{Q}_{R,cs}^{j}(\xi) = \mathbf{k}_{cs}^{j}(\xi) \cdot \mathbf{u}_{cs}^{j}(\xi) \tag{2.34}$$

9. Afterwards, the sections unbalanced forces $\mathbf{Q}_{U,cs}^{j}(\xi)$ and the residual deformations $\mathbf{r}_{cs}^{j}(\xi)$ can be calculated to

$$\mathbf{Q}_{U,cs}^{j}(\xi) = \mathbf{Q}_{cs}^{j}(\xi) - \mathbf{Q}_{R,cs}^{j}(\xi) \tag{2.35}$$

$$\mathbf{r}_{cs}^{j}(\xi) = \mathbf{f}_{cs}^{j}(\xi) \cdot \mathbf{Q}_{U,cs}^{j}(\xi) \tag{2.36}$$

10. If for all cross sections the norm of the unbalanced forces lies below a certain value tol_{cs}, the corresponding beam element has converged and the stiffness matrix of the beam element $\mathbf{K}_{beam,local}$ is calculated.

$$||\mathbf{Q}_{U,cs}^{j}(\xi)|| \leq tol_{cs} \tag{2.37}$$

11. If this is not the case, the residual beam element deformations $\mathbf{r}^j_{beam}$ need to be calculated and the beam iteration variable increases to $j+1$. The iteration is then started again at step 2. but this time using $\Delta\mathbf{u}^j_{beam} = -\mathbf{r}^j_{beam}$.

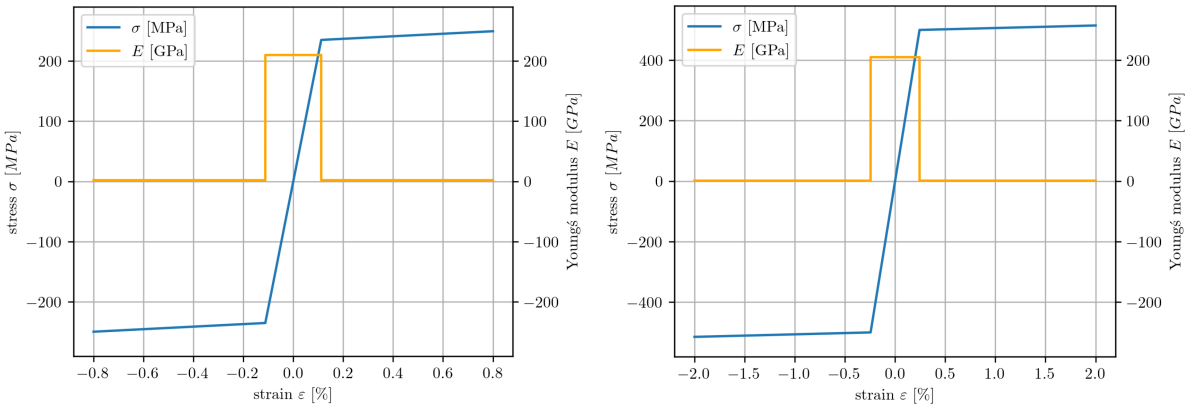
$$\mathbf{r}^j_{beam} = \sum_{sections} \mathbf{b}_{cs} \cdot \omega_{cs} \cdot \mathbf{r}^j_{cs} \cdot \det\left(J_{cs}\right) \tag{2.38}$$

12. After convergence of all the beam elements, the structural stiffness matrix $\mathbf{K}_{struc}$ and the internal force vector $\mathbf{R}$ can be assembled as described in section 2.2

2.5 Material Laws

There are four different material laws implemented, one for steel $S235$, one for rebar $B500B$ and two for concrete $C30/37$. One of the force-displacement relationships for concrete is more simple and one more sophisticated.

The material law for steel $S235$ is, as well as the one for rebar $B500B$, modeled as a bilinear material law. While the initial young's modulus E for steel is set to $210'000$ N/mm^2, for the rebar it has a value of $E_{B500B} = 205'000$ N/mm^2. After reaching the yield stress of $f_{yk} = 235$ N/mm^2, the steel $S235$ is modeled with a hardening modulus 1% of the initial young's modulus $H_{S235} = 0.01 \cdot E_{S235}$. According to Kaufmann [13], the rebars limit after yielding at $f_{sk} = 500$ N/mm^2 is set to $f_{uk} = 1.08 \cdot f_{sk}$ at a failure strain of $\varepsilon_u = 5\%$ which leads to a hardening modulus of $H_{B500B} = 840$ N/mm^2 $= 0.4\% \cdot E_{B500B}$. The stress-strain relationship for steel $S235$ and rebar $B500B$ are visible in Figure 2.4.



(a) (b)

Figure 2.4: Implemented stress-strain relationship of (a) steel S235 and (b) rebar B500B

In compression, concrete has a highly nonlinear behavior. Hulail and Hamza [14] managed to determine a stress-strain relationship for concrete under compression, which matches with experimental data. The fundamental law is given by Equation 2.39.

$$\sigma = f_c \cdot \left(\frac{\varepsilon}{\varepsilon_0}\right)^{\beta} \tag{2.39}$$

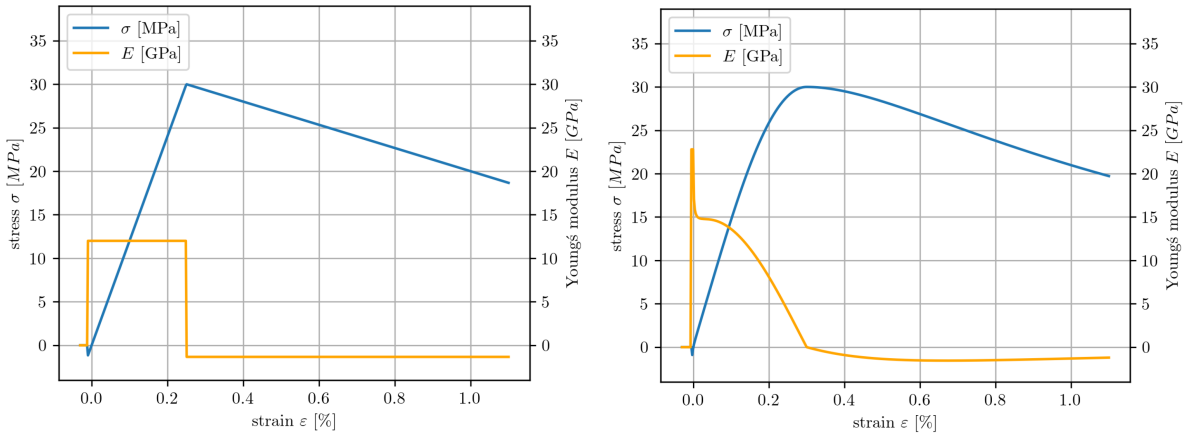
The exponent β depends if the strain ε is on the ascending branch $\varepsilon \leq \varepsilon_0$ or if the strain ε lies on the descending branch $\varepsilon \geq \varepsilon_0$.

$$\beta = \frac{a\left(1 - \frac{\varepsilon}{\varepsilon_0}\right)}{\left(1 + b \cdot \frac{\varepsilon}{\varepsilon_0}\right)} \text{ for } \varepsilon \leq \varepsilon_0 \qquad \beta = \frac{c^{\frac{\varepsilon_0}{\varepsilon}} \cdot \left[1 - \left(\frac{\varepsilon}{\varepsilon_0}\right)^c\right]}{\left[1 + \left(\frac{\varepsilon}{\varepsilon_0}\right)^c\right]} \text{ for } \varepsilon \geq \varepsilon_0$$

where the coefficients a, b and c only depend on the concrete's compressive strength f_c.

$$a = 0.7 \cdot f_c^{\frac{1}{15}} \qquad b = -0.02 \cdot f_c^{0.8} \geq -0.95 \qquad c = 0.02 \cdot f_c$$

For the concrete type of $C30/37$ with a compressive strength of $f_c = 30$ N/mm^2 at a strain of $\varepsilon_0 = 2.5‰$, Figure 2.5b shows the stress-strain relation as well as the tangent modulus E_t. A simplified concrete material law has been achieved with a bilinear model using the averaged tangent modulus.



(a) (b)

Figure 2.5: Implemented stress-strain relationship of concrete as a (a) bilinear material law and (b) a more complex material law according to Hulail and Hamza [14]

It has to be mentioned that both of the concrete material laws were extended to negative strains using the tangent modulus at $\varepsilon = 0$ to a tensile stress of $f_{ct} = -1.28$ N/mm^2.

2.6 Cross Section Analysis

The cross-sectional analysis relies mostly on the geometry, respectively the fiber mesh, and the used material laws. The main idea behind the cross-sectional analysis is that some axial strain ε and some curvatures around the y-axis $\chi_{y,cs}$ and z-axis $\chi_{z,cs}$ can be applied. From strains and curvatures, the strain of each fiber can be calculated using Equation 2.40 where $c_{y,fib}$ and $c_{z,fib}$ refer to the fiber's centroid coordinates.

$$\varepsilon_{fib} = \varepsilon_{cs} + \chi_{y,cs} \cdot c_{z,fib} + \chi_{z,cs} \cdot c_{y,fib} \tag{2.40}$$

After the calculation of the fiber strains ε_{fib}, using their uniaxial material law,the fiber stresses σ_{fib} can be determined. These stresses are summed up to calculate the corresponding section forces:

$$N = \sum_{fibers} A_{fib} \cdot \sigma_{fib} \tag{2.41}$$

$$M_y = \sum_{fibers} A_{fib} \cdot \sigma_{fib} \cdot c_{z,fib} \tag{2.42}$$

$$M_z = \sum_{fibers} A_{fib} \cdot \sigma_{fib} \cdot c_{y,fib} \tag{2.43}$$

By a linear variation of these strains, some bending moment - curvature diagrams can be plotted as shown in section 4.4.

It is also possible to do an inverse analysis, meaning to apply section forces N, M_y and M_z and calculate the corresponding strain ε_{cs} and curvatures $\chi_{y,cs}$ as well as $\chi_{z,cs}$. By defining a residual function such as the following linear system of equations, an optimization algorithm is able to solve for the strain and curvatures. The currently implemented framework for the cross-sectional uses a local optimization algorithm that needs some initial values that are always set to $[\varepsilon_{cs} \;\; \chi_{y,cs} \;\; \chi_{z,cs}] = [0 \;\; 0 \;\; 0]$. Global optimization algorithms, such as genetic algorithms, were first considered but then dropped due to the scope of the work.

$$N_{target} - N(\varepsilon_{cs}) \approx 0 \qquad M_{y,target} - M_y(\varepsilon_{cs},\ \chi_{y,cs}) \approx 0 \qquad M_{z,target} - M_z(\varepsilon_{cs},\ \chi_{z,cs}) \approx 0$$

With the tool of an optimization algorithm and therefore the possibility to get a cross-section's response to some given section forces, it was possible to generate some N-M interaction plots as well as M_y-M_z interaction plots visible in section 4.4.

3 Methods

This chapter deals with the implementation of the framework, which incorporates the fiber element solver, the cross-sectional analysis, and the lamina analysis tool. The framework is implemented as an object-oriented Python project. It was developed using Python 3.13. To be able to run the code or the examples found in the appendix A, first the following Python libraries have to be installed:

Table 3.1: List of the python libraries used by the framework.

library	usage
abc	for the definition of an abstract class *Constraint(abc)* with abstract methods to be able to extend the Newton-Raphson constraints.
collections	using *defaultdict* for grouping the fibers into material groups.
gmsh	used for the discretization of the cross-sections into the fibers.
matplotlib	needed for all the produced plots and visualizations.
mpl_toolkits	*Poly3DCollection* is used during the plotting of the lamina to display the plies.
numba	imported for the use of the decorator *@jit* which is used by the computation of the strains and stresses and compiles the code just-in-time into machine code for faster performance.
numpy	all arrays and matrices are *numpy* array for faster computation.
pandas	used for a clean console output of displacements and forces.
scipy	*optimize* is used for the inverse cross sectional analysis as mentioned in section 2.6 and *legendre* is used for the computation of the Gauss-Lobatto points and weights.
tabulate	needed for the clean output of the cross-sectional properties
tqdm	delivers the possibility of a progress-bar used by the nonlinear solver

In the following sections, the code structure and the most important features of the Python-based framework are highlighted.

3.1 Code Structure

The Python code of the fiber element solver is distributed into three packages, the *Structure* itself, the *Solver* and the *Materials*. Figure 3.1 shows the frameworks structure using the UML (unified modeling language). The code of the *Structure* is structured using the same principle as mentioned in the theory section 2.2 of how a structure, modeled from fiber elements, is assembled.

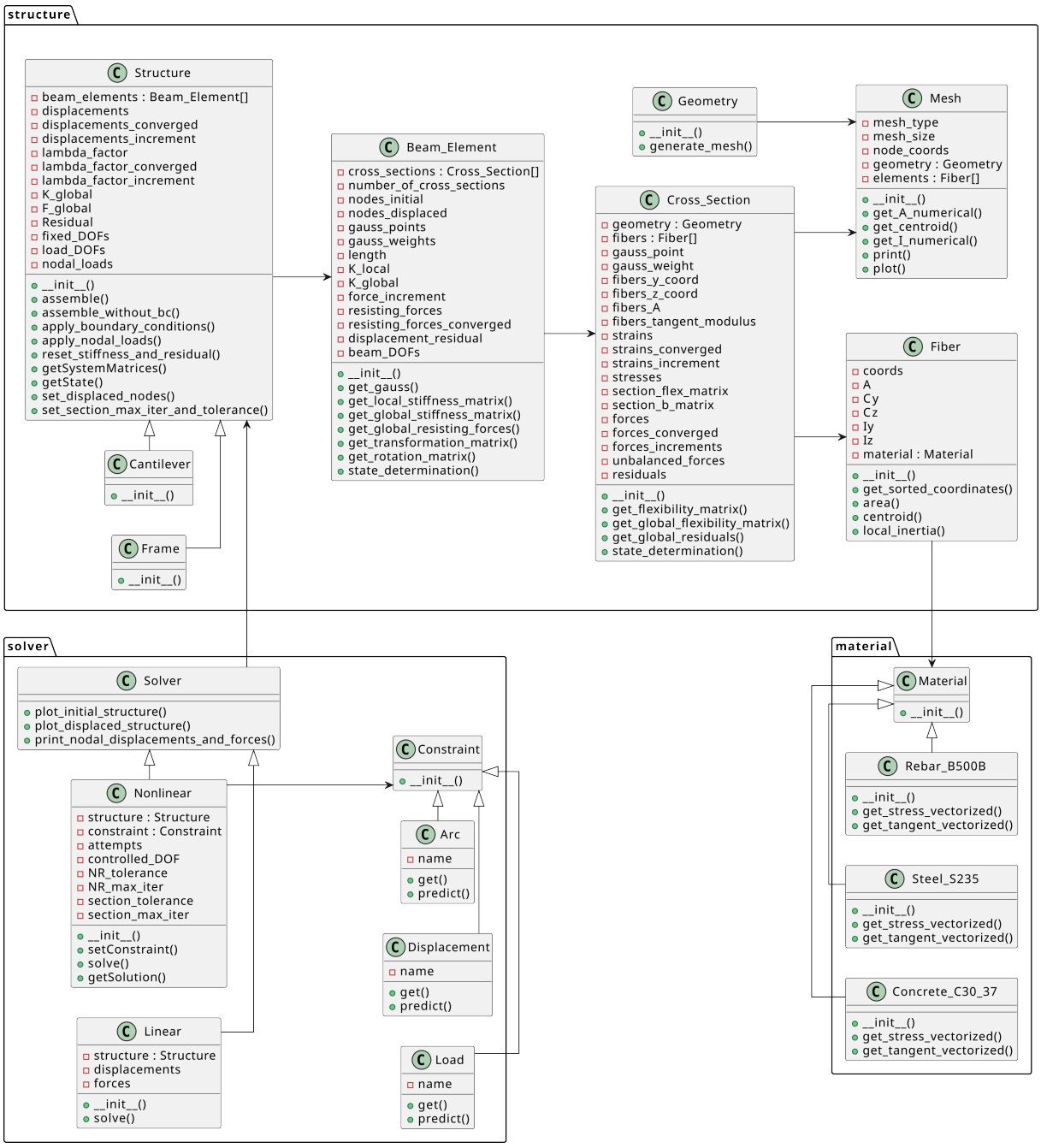


Figure 3.1: UML of the fiber element solver

The Solver incorporates a linear, as well as a nonlinear solver, where for a nonlinear solver it can be chosen to use one constraint from a load-controlled, displacement-controlled, or an arc-length method. The last package is formed by the *Material*. As for now, the material laws mentioned in section 2.5 are implemented in the framework.

3.2 Meshing with gmsh

A crucial part in a fiber element solver is the discretization of the cross-section. In this framwork, this is done with the library *gmsh*. *Gmsh* is an open-source finite element mesh generator. Its goal is to provide a fast and user-friendly meshing tool with parametric input capabilities [15]. *Gmsh* is accessed by simply installing *gmsh* and importing it in Python. The documentation can be found here.

The framework offers multiple already defined geometries. Each geometry consists of its constructor and the *generate_mesh()* method. The *generate_mesh()* takes two arguments, the *type*, which corresponds to the element type where one can choose between a triangular and a quadrilateral mesh, and the *size* that is the approximate size of the mesh respectively the approximate distance between the nodes. Examples of the generation of multiple predefined meshes can be found in appendix A.1.

The meshing with the *gmsh* library is a very straightforward process, which is shown at the example of a simple rectangle in the following code snippets.

1. First, the libraby has to be imported and then initialized

```
import gmsh
gmsh.initialize()
```

2. Then, a geometrical model can be added with a corresponding name

```
gmsh.model.add("geometry's name")
```

3. The points, that are used to describe the geometry need to be defined. Afterwards, the points can be connected to lines using the points IDs.

```
# gmsh.model.geo.addPoint(x_coord, y_coord, z_coord, mesh_size, point_ID)
gmsh.model.geo.addPoint(-self.width/2, -self.height/2, 0, size, 1)
gmsh.model.geo.addPoint( self.width/2, -self.height/2, 0, size, 2)
gmsh.model.geo.addPoint( self.width/2,  self.height/2, 0, size, 3)
gmsh.model.geo.addPoint(-self.width/2,  self.height/2, 0, size, 4)
```

```
# gmsh.model.geo.addLine(point_ID_1, point_ID_2, line_ID)
gmsh.model.geo.addLine(1, 2, 1)
gmsh.model.geo.addLine(2, 3, 2)
gmsh.model.geo.addLine(3, 4, 3)
gmsh.model.geo.addLine(4, 1, 4)
```

4. The lines are then assembled to a closed loop. It has to be ensured, that the lines follow a continuus direction either clock or counterclockwise. A lines drection can be changed using a minus sign in front of its ID. The closed loop is then transformed into a surface and the *gmsh* model is synchronized.

```
# gmsh.model.geo.addCurveLoop([list_of_lines], loop_ID)
gmsh.model.geo.addCurveLoop([1, 2, 3, 4], 1)

# gmsh.model.geo.addPlaneSurface([list_of_loops], surface_ID)
gmsh.model.geo.addPlaneSurface([1], 1)

gmsh.model.geo.synchronize()
```

5. If the element type was chosen to quadrilateral elements, the triangular elements can be recombined to quadrilaterals. Afterwards, the mesh can be generated, where the argument in the function is the definition of the dimensions of the mesh. As here a planar cross-section will be meshed, the argument is set to 2.

```
if type == "quadrilateral":
    gmsh.model.mesh.setRecombine(2,1)

gmsh.model.mesh.generate(2)
```

6. Once the mesh is generated, using the methods *getNodes()* and *getElements()*, the nodal coordinates as well as the node IDs per element can be extracted from the *gmsh* model. The element nodes vector also has to be reshaped considering the element type, where *2* represents triangular elements and *3* quadrilateral elements.

```
node_tags, node_coords, _ = gmsh.model.mesh.getNodes()
node_coords = np.array(node_coords).reshape(-1, 3)[:, :2]

elem_types, elem_tags, elem_nodes = gmsh.model.mesh.getElements()
elements = []
for i, e_type in enumerate(elem_types):
    if e_type == 2:
        elements = np.array(elem_nodes[i]).reshape(-1, 3) - 1
    if e_type == 3:  # Type 3 = Quadrilateral
        elements = np.array(elem_nodes[i]).reshape(-1, 4) - 1
```

7. The last step is to finalize and close gmsh as well as construct the fibers for each generated triangular or quadrilateral element.

```
gmsh.finalize()

fibers = [Fiber(coords=node_coords[elem], nodes=elem) for elem in elements]
```

Gmsh also offers curved lines and a lot more possibilities. It also features many different meshing algorithms that could be further analysed but were not considered further in relation to the scope and aim of this thesis. Using multiple surfaces, something like a reinforced concrete section can be created where each surface has another ID which can then be assigned to a specific material. The application of the material law to the fibers happens at the end of the *generate_mesh()* function. Depending on the material a fiber should be assigned to, another name has to be inserted into *Fiber(coords, nodes, mat)*, where "mat" stands for the material's name which is resolved in the fiber's constructor.

With this strong tool, any other geometries that are not implemented so far can be expanded and used in the fiber element solver or investigated in a cross-sectional analysis.

3.3 Extensibility of the code

A big advantage of the object-oriented programming of the framework is its extensibility. Besides the mentioned possible extensibility of the cross-sections that can be discretized, the structures, the material laws as well as the Newton-Raphson constraints can be extended, which allows for maximum flexibility.

3.3.1 Structures

Each structure such as the implemented n-story frame or the cantilever is a sub-class of the super-class *Structure*. The super-class deals with the initialization of all needed arrays in its constructor and comes along with all the needed methods such as *assemble()*, *apply_boundary_conditions()* or *getSystemMatrices()*.

A structure's sub-class needs a constructor that implements its nodes' coordinates as well as the beam elements using a predefined cross-section geometry, the implemented nodal coordinates of the start and end-node per beam and the indices of these nodal DOFs. The last thing that needs to be handled in a structure's subclass is the definition of the fixed DOFs. For example if the structure is clamped at the first node, the fixed DOFs are $[0, 1, 2, 3, 4, 5]$.

```python
class Any_Structure(Structure):
    def __init__(self, cs_geometry, number_of_sections_per_element,
                       load_DOFs, nodal_loads, *args):

        self.number_of_nodes = ...
        self.number_of_DOFs  = 6 * self.number_of_nodes

        # --- Generate nodes ---
        # array of the form [[x1, y1, z1], [x2, y2, z2], ...]
        self.structure_nodes_initial = ...

        #--- Generate Beam Elements ---
        # array that holds all the Beam_Element(cs_geometry,
        #                                       number_of_sections,
        #                                       nodes, beam_DOFs)
        self.beam_elements = ...

        #--- Apply Boundary Conditions and prepare iteration variables ---
        # fixed_DOFs is an array, that holds the indices of all fixed DOFs
        # e.g. clamped at the first node fixed_DOFs = [0,1,2,3,4,5]
        fixed_DOFs = ...
        super().__init__(fixed_DOFs, load_DOFs, nodal_loads)
```

3.3.2 Materials

To implement a new material law, respectively a new stress-strain relationship, the constructor looks like the following:

```python
class Any_Material(Material):
    def __init__(self):
        # define the r,g,b values for the material's color
        self.color = (r, g, b, 0.5)

        # define the materials name, used for the grouping of fibers
        self.name  = "any_material"
```

The color of the material is used when plotting a meshed cross-section. It is important to mention here that to apply a new material law to a cross-section respectively to fibers, two considerations have to be made:

1. in the *generate_ mesh()* method of the geometry where the fibers are constructed, the name of the material has to be given into the constructor.

```
fibers = [Fiber(coords = node_coords[elem],
                nodes  = elem,
                mat    = "any_material") for elem in elements]
```

2. in the Fiber class, the existing if-else-statement has to be expanded by the new material

```
if mat == "Concrete_C30_37":
    self.material = Concrete_C30_37()
elif mat == "Steel_S235":
    self.material = Steel_S235()
elif mat == "Rebar_B500B":
    self.material = Rebar_B500B()
elif mat == "any_material":
    self.material = Any_Material()
else:
    self.material = Unknown()
```

In addition to the assignment of the material to the fiber, the methods *get_ stress_ vectorized()* and *get_ tangent_ vectorized()* have to be implemented. Both of these methods receive a *numpy*-array of strains ε as an input and have to return either the stresses σ or the tangent modulus E_t to the corresponding strains as a *numpy*-array. It is recommended to do the calculations vectorized and using the *@jit* decorator for faster computation.

3.3.3 Newton Raphson Constraints

Besides the implemented Newton-Raphson constraints like the load-controlled, the displacement controlled and the arc-length method, there are others like, for example, the Riks-method. The constraints are built as an abstract class, enforcing the two methods *predict()* and *get()*.

```
class Constraint(ABC):
    @abstractmethod
    def get(self):
        pass

    @abstractmethod
    def predict(self):
        pass
```

As the name of the methods already tells, the *predict()* function predicts the increment of the load factor $\Delta\lambda^p$ and the increment of the deformation vector $\Delta\mathbf{u}^p$ and the method *get()* evaluates the constraint and returns g^i, $\mathbf{h}$ and s for the given input as described in section 2.4.1.

4 Results

This chapter summarizes all the results that can be produced from the framework and the examples introduced in the appendix A.

4.1 Cross Sectional Properties

The example in appendix A.1 shows how all the implemented cross sections can be set up. In addition, with the command *mesh.print()* the cross-sectional properties of the introduced cross-section can be generated. For a few selected H and L profiles, the produced values are printed in Table 4.1

Table 4.1: Cross section properties of some specific steel profiles

profile	area	centroid		moment of inertia	
	$A\ [mm^2]$	$c_y\ [mm]$	$c_z\ [mm]$	$I_y\ [mm^4]$	$I_z\ [mm^4]$
LNP 60/8	902.63	17.63	17.63	0.29	0.29
LNP 100/65/9	1412.39	15.93	33.18	1.40	0.47
HEA 200	5105.00	0.00	0.00	35.10	13.30
HEB 300	14282.00	0.00	0.00	242.00	85.50

4.1.1 Lamina Analysis

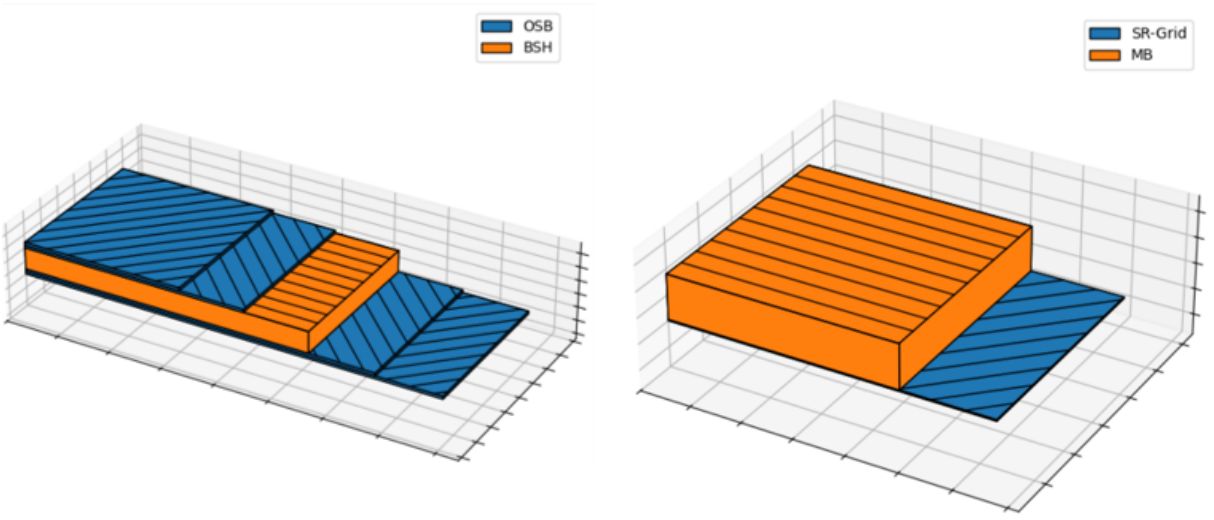
The analysis of a laminate material was introduced primarily due to the determination of the properties of a corresponding homogeneous material. The implemented tool of the lamina analysis, as mentioned in section 2.1.1, also includes a stress-strain analysis at the ply's level.

As an example, the produced visualizations of two laminate materials are shown in Figure 4.1. Figure 4.1a displays the following wooden lamina and its material properties [16]:

```
layers = [
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta':  45, 'thickness': 10},
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta': -45, 'thickness': 10},
    {'name': "BSH", 'E1': 9100, 'E2':  250, 'G12': 540, 'v12': 0.3,
     'theta':   0, 'thickness': 80},
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta': -45, 'thickness': 10},
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta':  45, 'thickness': 10}
]
```

The lamina in Figure 4.1b corresponds to a reinforced masonry wall using a carbon fiber grid [17] [18]: Besides the names of each layer, also the orientation of the ply's main axis is shown.

```
layers = [
    {'name': "SR-Grid", 'E1':  215e3, 'E2': 215e3, 'G12':     0, 'v12': 0.30,
     'theta': 45, 'thickness':   5},
    {'name':      "MB", 'E1': 10.8e3, 'E2': 5.4e3, 'G12': 4.5e3, 'v12': 0.26,
     'theta':  0, 'thickness': 175}
]
```



(a) (b)

Figure 4.1: Example of two lamina inputs. (a) a wood lamina and (b) a reinforced masonry wall.

Using the *LaminateLoadAnalysis()* as the second example (appendix A.2), the stresses and strains in the wooden lamina are displayed under the application of a normal force $N_{xx} = 100$ kN. It is visible that due to the orientations of the plies, some stresses are resulting in the y-direction.

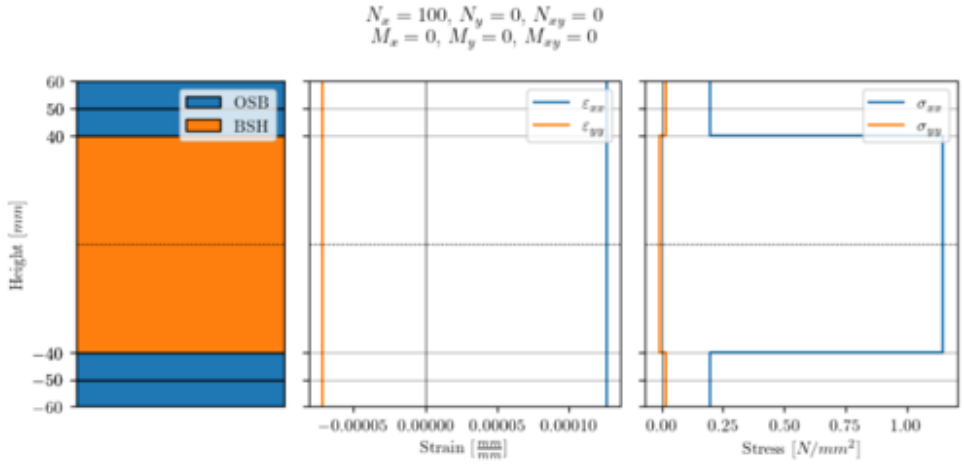


Figure 4.2: Wood Lamina Analysis Results under an axial force of $N_x = 100$ kN

Figure 4.3 displays the results of the stresses and strains acting on each layer in numerical values. While the strains are distributed linearly over the plies due to continuity conditions, it is visible that the BSH, which has a higher young's modulus than the OSB plates, takes the larger proportion of the applied force.

strains per layer:

layer	e_xx	e_yy	e_xy
1	0.000127	-7.17e-05	-3.55e-23
2	0.000127	-7.17e-05	-2.9e-23
3	0.000127	-7.17e-05	0
4	0.000127	-7.17e-05	2.9e-23
5	0.000127	-7.17e-05	3.55e-23

(a)

stresses per layer:

layer	s_xx	s_yy	t_xy
1	0.0353	0.002	-0.00161
2	0.0325	0.00198	0.00171
3	0.115	-0.000845	0
4	0.00734	0.0014	0.0003
5	0.00458	0.00138	-0.000406

(b)

Figure 4.3: (a) Strains and (b) stress resultants at each ply in a wood lamina analysis under a normal force $N_x = 100$ kN

The method *print_results()* returns besides the ABD-Matrix also the material properties of an equivalent and homogenized material. The corresponding values to the lamina show in Figure 4.1a are: $E_x = 6890$ MPa, $E_y = 976$ MPa, $G_{xy} = 1120$ MPa and $\nu_{xy} = 0.57$ MPa.

4.2 Linear Structural Analysis

The example of the linear solver in appendix A.3 solves a clamped frame under a horizontal load acting at the upper left node in the global x-direction of $F_{2,x} = 100$ kN. The columns correspond to *HEB 300* beams and the horizontal beam to a *HEA 200*. The initial and displaced structures with a scale factor can be plotted using *plot_displaced_structure(linear_solver, scale=50.0)* and are shown in Figure 4.4.

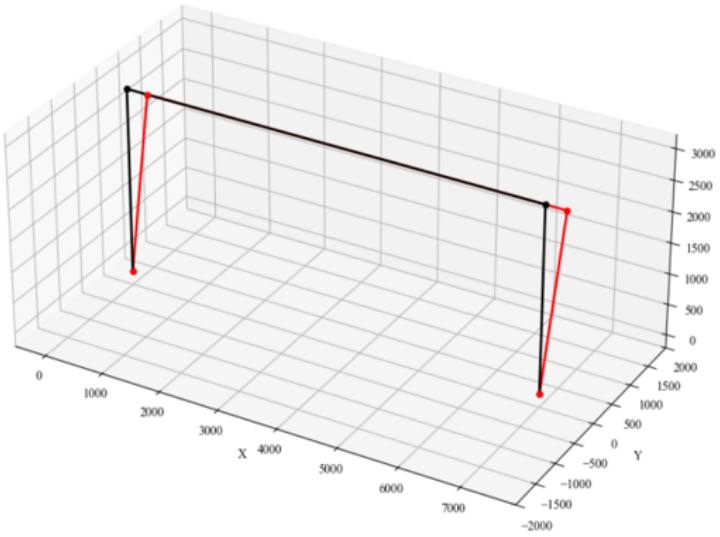


Figure 4.4: Linearly solved, displaced clamped one story frame after the application of a lateral force

The nodal displacements and reaction forces in the clamped nodes are visible in 4.2. The numbering of the nodes starts at the bottom left, then bottom right, afterwards the top left and so on. As the clamped nodes are not displaced at all and as there are no reaction forces at the upper nodes, these values are neglected in the table.

Table 4.2: Numerical results (displacements and reaction forces) per node of the clamped one story frame under a lateral load of $F_{2,x} = 100$ kN

node	displacements						rection forces					
	u	v	w	θ_x	θ_y	θ_z	F_x	F_y	F_z	M_x	M_y	M_z
0	–	–	–	–	–	–	-50.98	0.00	-5.82	0.00	-132.42	0.00
1	–	–	–	–	–	–	-49.02	0.00	5.82	0.00	-126.85	0.00
2	7.22	0.00	0.00	0.00	3.31	0.00	–	–	–	–	–	–
3	6.89	0.00	0.00	0.00	3.15	0.00	–	–	–	–	–	–

4.3 Nonlinear Structural Analysis

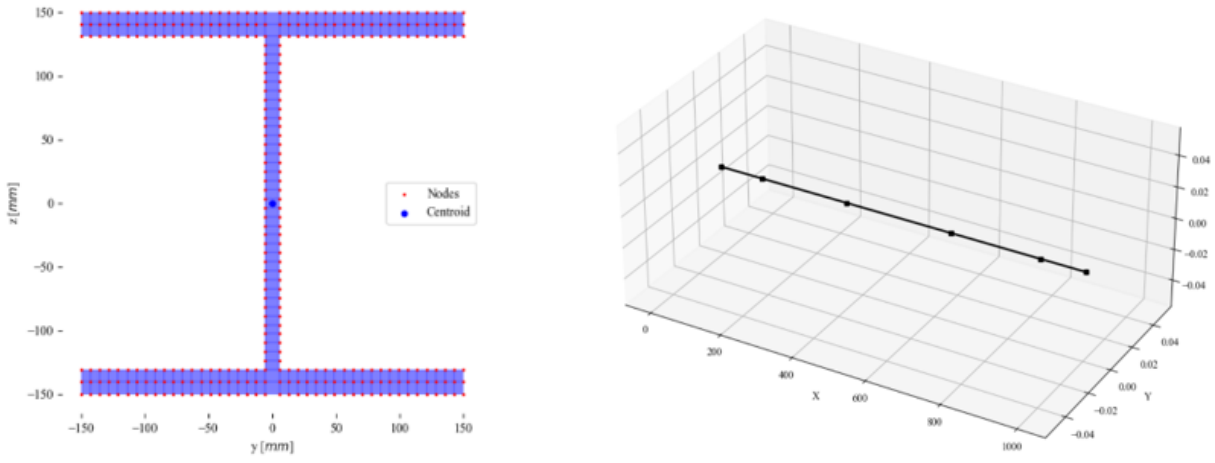
The main purpose of this thesis was the implementation of the fiber element solver, which is able to solve structures using nonlinear uniaxial material laws for each fiber. The nonlinearity of a structure can be most easily shown using the load-displacement curve. This section shows the results of multiple load-displacement curves of cantilevers as well as a clamped one-story frame. In addition, a sensitivity analysis was done. On one hand analyzing the influence of the number of sections per beam element and on the other hand the analysis of the sensitivity on the number of elements in the mesh.

4.3.1 Load-Displacement Curve

Cantilever

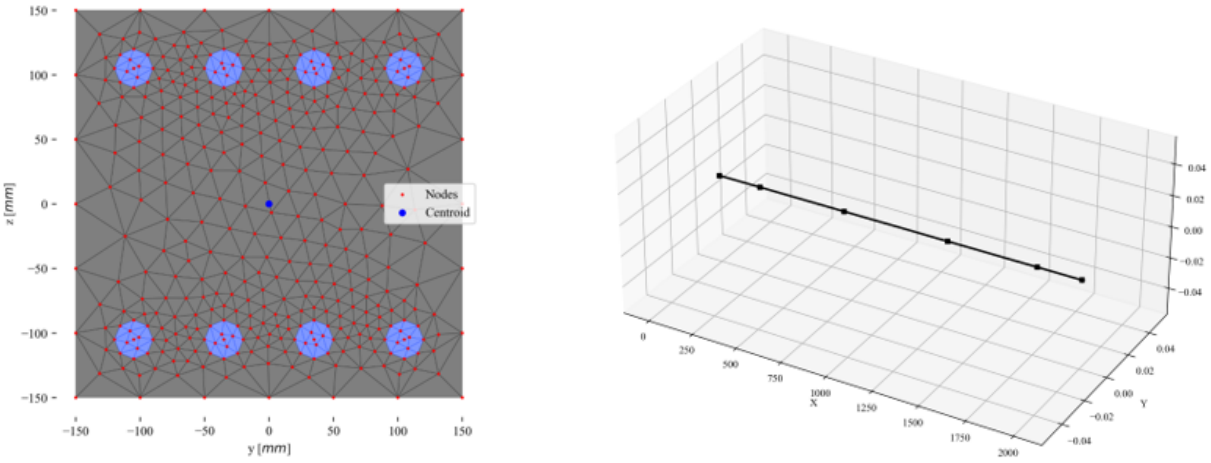
The first two load displacement curves have been generated for a *HEB 300* cantilever with the length of 1 m and clamped at one end, which is shown in Figure 4.5 and a reinforced concrete cantilever with a length of 2 m. Each of these cantilevers is loaded with an increasing lateral load acting in the global z-direction causing bending around the y-axis. The reinforced conncrete section used is:

```
RC_section = ReinforcedConcreteColumn(width  = 300,
                                      height = 300,
                                      concrete_cover = 30,
                                      rebar_diameter = 30,
                                      rebar_spacing  = 50)
```



(a) (b)

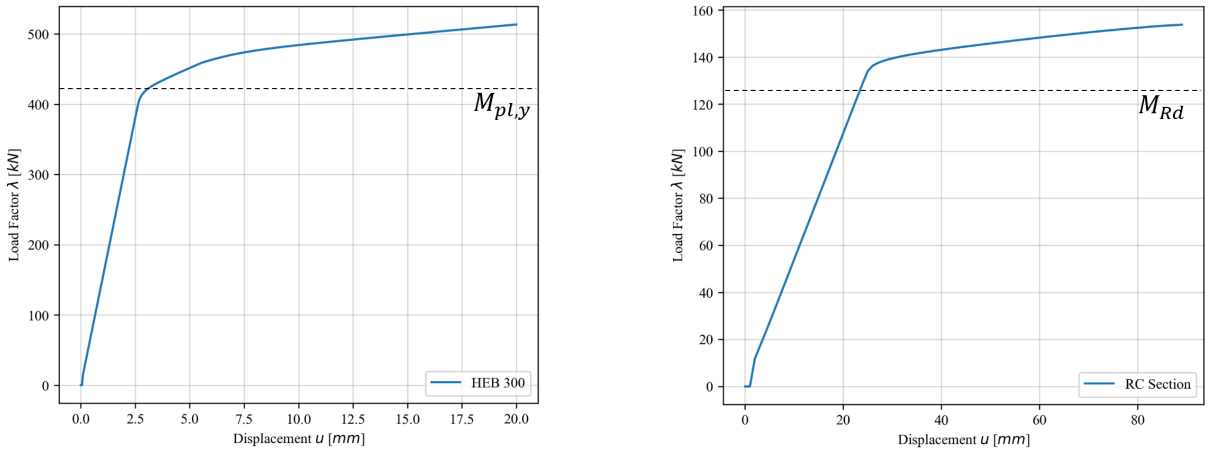
Figure 4.5: Steel beam HEB 300 with cross-section (a) and cantilever structure (b) of length 1 m



(a) (b)

Figure 4.6: Reinforced concrete beam 300·300 mm with cross-section (a) and cantilever structure (b) of length 2 m

Figure 4.7a and Figure 4.7b show the produced load-displacement curves of the steel beam and the reinforced concrete beam, respectively. The stiffness degradation of the *HEB 300* is clearly visible after reaching the plastic moment resistance, which is $M_{pl,y} = 418.2$ kNm corresponding to the SZS C5 table [19].



(a) (b)

Figure 4.7: Load deformation curve of the (a) steel and the (b) reinforced concrete cantilever in comparison to their plastic moment resistance $M_{pl,y}$ respectively M_{Rd}

The bending resistance M_{Rd} of the reinforced concrete section can be calculated with the following equation [20].

$$M_{Rd} = A_s \cdot f_{sd} \cdot d \cdot \left(1 - \frac{\omega}{2}\right) = 249.4 \text{ kNm} \tag{4.1}$$

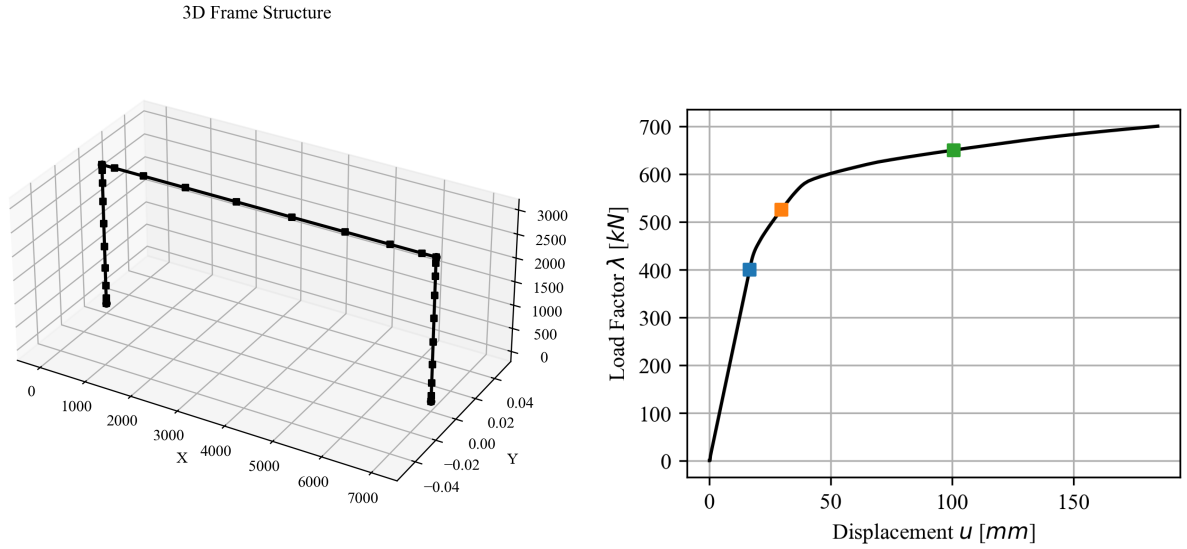
where $A_s = 4 \cdot \frac{(30mm)^2 \cdot \pi}{4} = 2827 \text{ mm}^2$ is the area of reinforcement, $f_{sd} = 500 \text{ N/mm}^2$ is the strength of the reinforcement, $d = 300mm - 30mm - 15mm = 255$ mm corresponds to the static height and ω defines the mechanic reinforcement content.

$$\omega = \frac{A_s \cdot f_{sd}}{d \cdot b \cdot f_{cd}} = \frac{2827 \text{ mm}^2 \cdot 500 \text{ N/mm}^2}{255 \text{ mm} \cdot 300 \text{ mm} \cdot 30 \text{ N/mm}^2} = 0.616 \tag{4.2}$$

With the length of the reinforced concrete cantilever of 2 m this moment resistance is reached with an applied load of $F_z = 124.7$ kN. Despite that, the implemented cross-section starts yielding at a higher load factor of about $\lambda = 135 - 140$ kN.

Frame

The nonlinear response of a clamped one-story frame, consisting of all *HEB 300* beams as in the example in appendix A.4 is shown in Figure 4.8b. There are two bends in the curve, the first one represents the formation of plastic hinges in the lower, clamped ends of the column, whereas the second bend corresponds to the formation of a plastic hinge in the frame's corner.



(a) (b)

Figure 4.8: (b) Load-displacement curve of (a) a one-story clamped frame under lateral loading at the upper left node.

4.3.2 Section Forces

Not only the load factors λ and the deformations $\mathbf{u}$ can be extracted from the nonlinear solver, but also the section forces at each cross-section and the corresponding curvatures. These curvatures can be integrated along the beam to get the a beam's rotation and these in turn can be integrated to the displacements. Figure 4.9 shows the curvature κ, the rotation θ, the displacements u as well as the bending moment along the steel cantilever from Figure 4.5. The formation of plastic hinges at the clamping is clearly visible under increasing horizontal load. This figure also highlights the distributed plasticity along the beam in contrast to a lumped plasticity model.

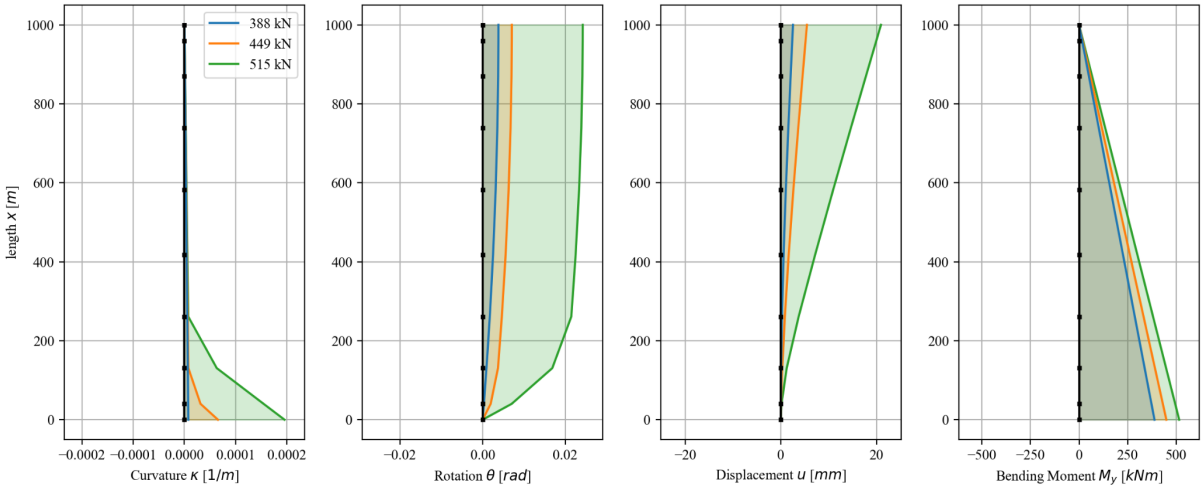


Figure 4.9: Moment, curvature and deformation relation under lateral loading $F = 388$ kN, 449 kN and 515 kN of the steel cantilever shown in Figure 4.5

Figure 4.10 displays the curvature κ, the rotation θ, the displacements u as well as the bending moment along the left column of the introduced steel frame in Figure 4.8a. A closer look at the curvature shows the formation of the plastic hinges. At a horizontal load of 400 kN, the frame still behaves linearly as it is also visible in the load-deformation curve 4.8b marked as the blue dot. As the load increases to 525 kN, the first plastic hinges are formed at the clamping of the columns. On a further increase of the horizontal load to 650 kN, a second plastic hinge is formed at the upper end of the column. Also visible is the redistribution of the bending moment from the lower to the upper end of the column. The bending moment on the upper end increases more than on the clamped end, that was already in a plastic state.

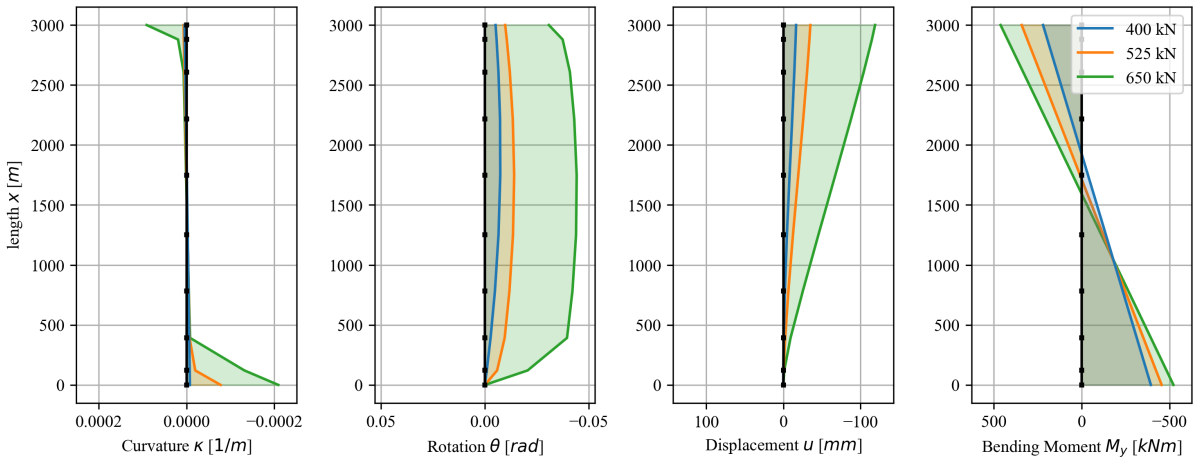
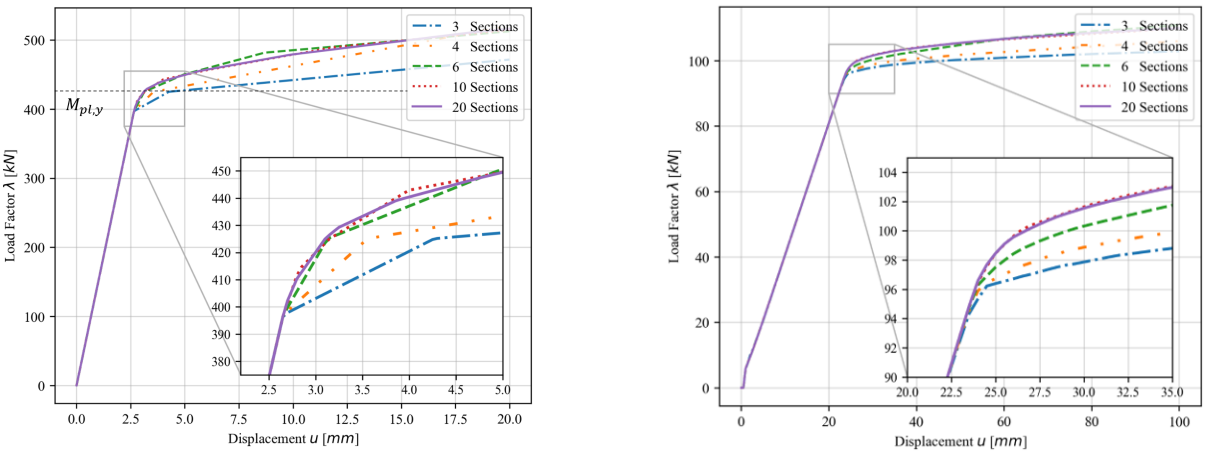


Figure 4.10: Moment, curvature and deformation relation under lateral loading $F_{2,x} = 400$ kN, 525 kN and 650 kN of the steel frame shown in Figure 4.8a

4.3.3 Sensitivity Analysis

Sensitivity on the number of sections per beam

A sensitivity analysis on the number of cross sections was performed on a steel as well as on a reinforced concrete cantilever. The results are shown in Figure 4.11a respectively in Figure 4.11b.



(a) (b)

Figure 4.11: Load deformation curve of the (a) steel and the (b) reinforced concrete cantilever for different numbers of sections along the beam

It can be seen that there is quite a dependency of the number of sections along a beam regarding the plastic deformations. Only from a number of sections above 10, there are no more visible differences in the beam's response to lateral loading.

Sensitivity on the number of elements in the mesh

Regarding the sensitivity on the mesh-size respectively the number of fibers per cross-section, the following four cross sections were compared in a load-deformation curve of a 1 m long cantilever.

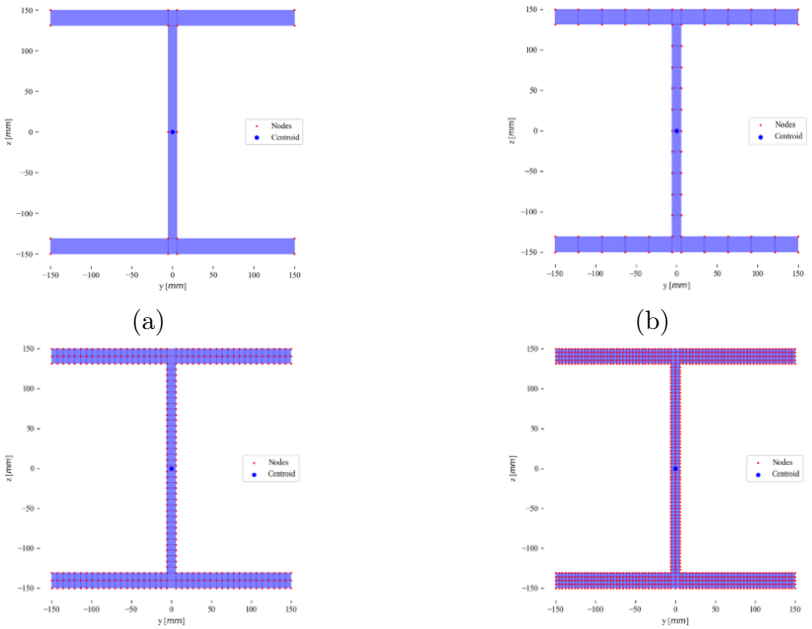


Figure 4.12: Steel HEB 300 cross-sections with different discretizations regarding the number of element per cross-section

Surprisingly, already the cross-section from Figure 4.12b with 32 elements generates the same result as the cross-section with 942 fibers. If the computational time of the solver should be decreased, first the number of fibers per element should be reduced rather than the number of cross-sections along the beam.

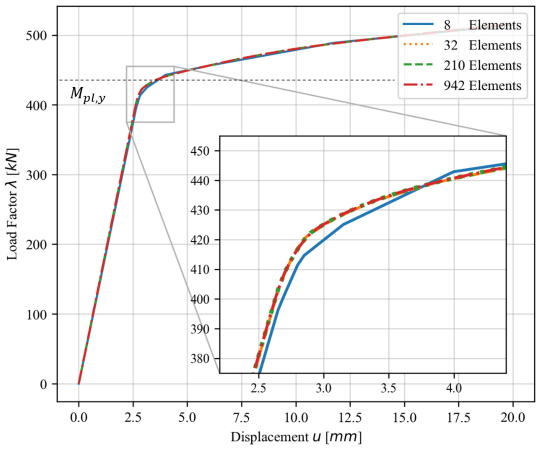
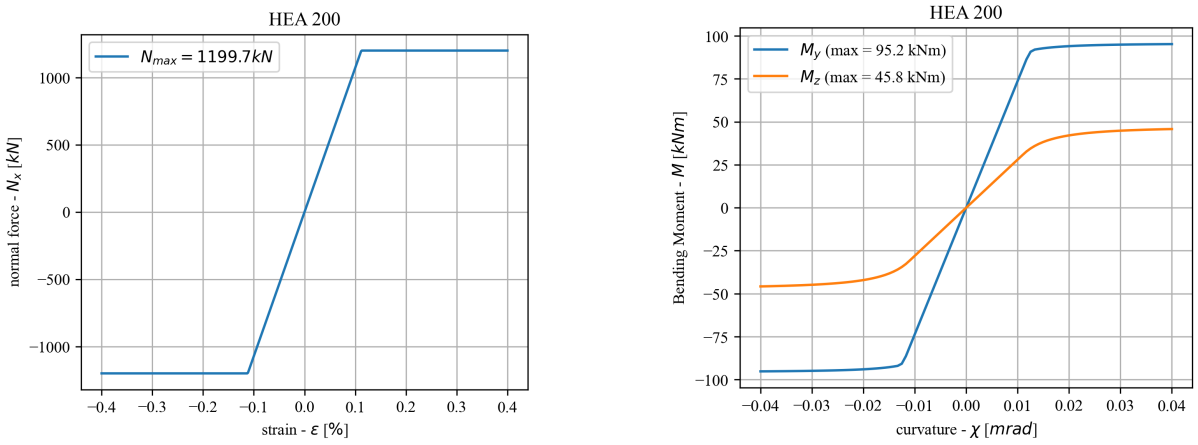


Figure 4.13: load deformation curve of the steel cantilever for different number of elements per cross section as mentioned in Figure 4.12a to Figure 4.12d.

4.4 Cross Sectional Analysis

4.4.1 Steel Section

As mentioned in section 2.6, the cross-sectional analysis tool can produce axial strain ε - normal force N_x diagrams as well as curvature $\chi_{y/z}$ - bending moment $M_{y/z}$ diagrams by a linear variation of the strains or curvatures and computing the corresponding normal forces and moments. As an example, these diagrams are shown in Figure 4.14 for a *HEA 200* beam of the quality $S235$.



(a) (b)

Figure 4.14: Relation between (a) the axial strain ε and the normal force N and (b) the curvature χ and the bending moments M_y and M_z in an HEA 200 beam.

Whereas the full capacity of the bending moment around the y-axis M_y is higher than M_z, the transition between the elastic to the plastic state of the cross-section is basically abrupt around the y-axis because all the fibers contributing to the moment resistance M_y reach the plastic state at the same curvature.

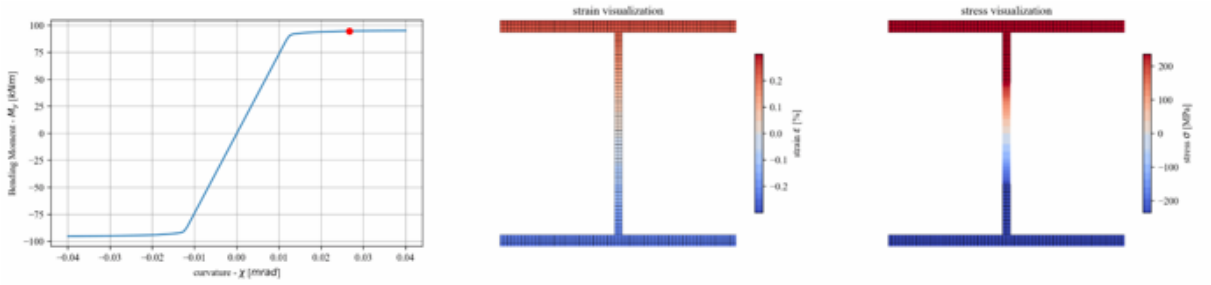


Figure 4.15: Resulting strains and stresses in a steel HEA 200 section by the application of a given curvature χ_y

Figure 4.15 and Figure 4.16 display the resulting strains and stresses for the *HEA 200* beam at a given M-χ state. The fibers that are elongated past their yield strength are marked with black edges in the stress visualization.

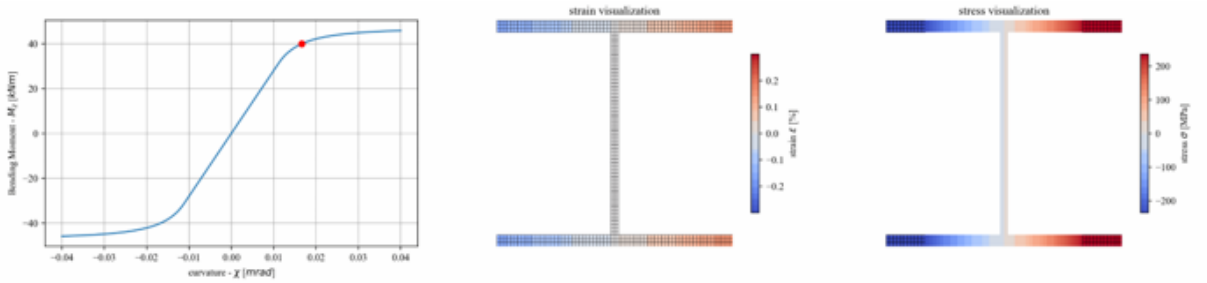


Figure 4.16: Resulting strains and stresses in a steel HEA 200 section by the application of a given curvature χ_z

Using the inverse analysis by solving the system of equations mentioned in section 2.6, the influence of the normal force N_x on the bending moment capacities of M_y and M_z can be considered as in Figure 4.17. Hereby, the normal force is fixed to a specific value of N_{target} and the bending moment M_y or M_z is varied along a predefined interval where for each tuple the corresponding curvature χ_y or χ_z is found. The increase of the normal force N_x reduces the cross-section's bending moment capacity.

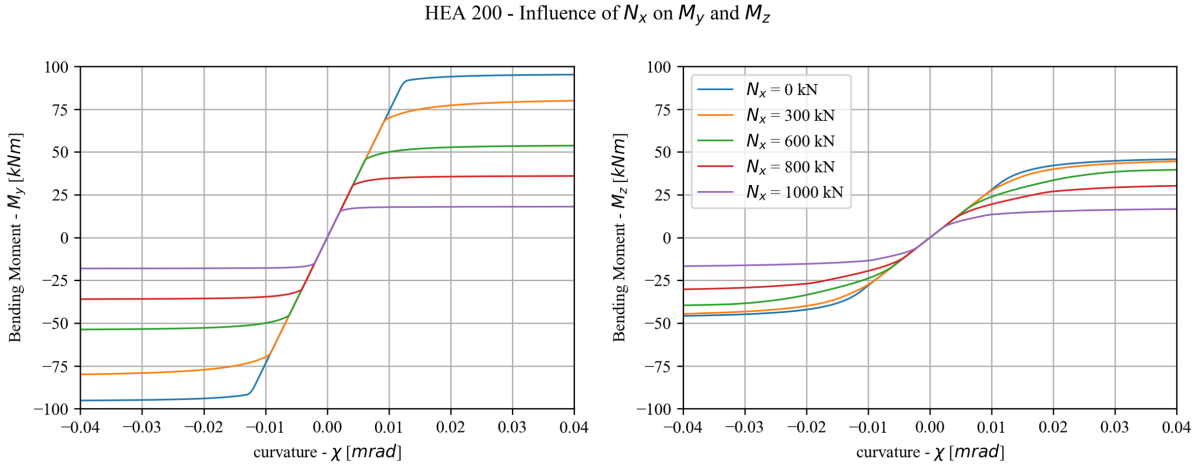


Figure 4.17: Influence of a given normal force N on the bending moment capacity M_y and M_z of an HEA 200 section.

In the same way as described before, with a fixed value of M_z and a linear varying value of M_y, the influence of the skewed bending can be measured and visualized in Figure 4.18.

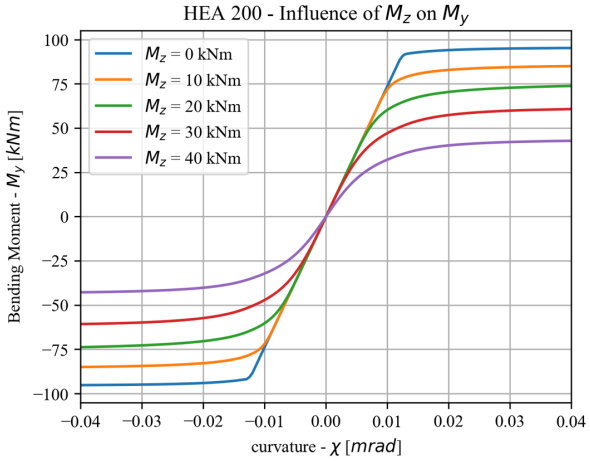
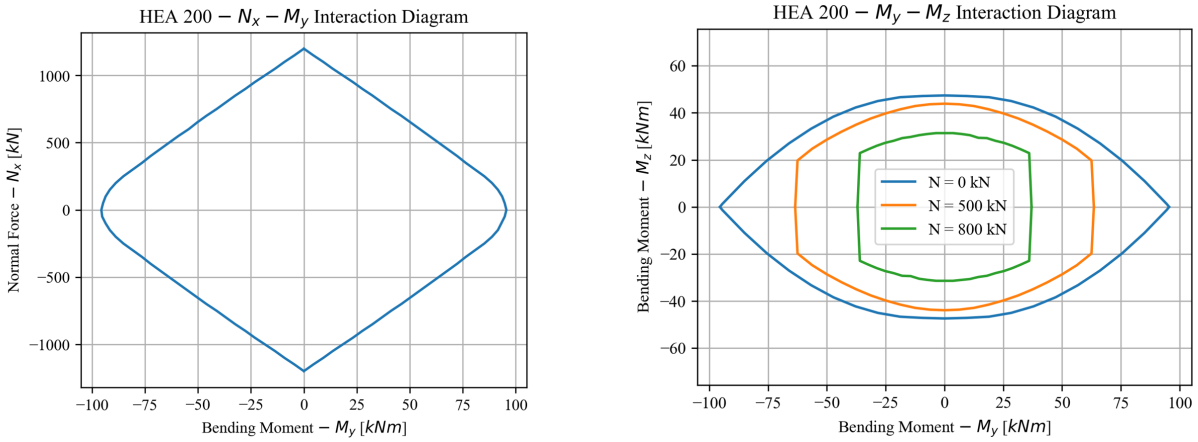


Figure 4.18: Influence of a given bending moment M_y on the bending moment capacity M_z of an HEA 200 section.

By consideration of always the highest value for the influence of the normal force N_x on the bending moment capacity M_y in Figure 4.17, these values can be put together to form a N-M_y interaction diagram for the steel *HEA 200* section in Figure 4.19a as well as a M_y-M_z interaction diagram under different normal forces N_x in Figure 4.19b. Because the steel *HEA 200* section and its material law are perfectly symmetric, so is the N-M_y interaction diagram.



(a) (b)

Figure 4.19: (a) $N - M_y$ interaction diagram and (b) $M_y - M_z$ interaction diagram under given normal forces $N = 0$ kN, 500 kN and 800 kN for an HEA 200 section.

4.4.2 Reinforced Concrete Section

As a second example, the cross-sectional analysis was also performed for a reinforced concrete section. The figures 4.20 and 4.21 show the stress-strain states of the concrete section under an applied curvature χ_y respectively χ_z. These figures show, besides the linear strain variation of the strains along the cross-sections, also the stresses of the rebars as well as for the concrete fibers. The pink marked fibers are concrete fibers that are in tension. If the strains reach a certain value corresponding to the concrete's tensile strength f_{ct}, the concrete cracks, which can be seen by no more visualized stresses.

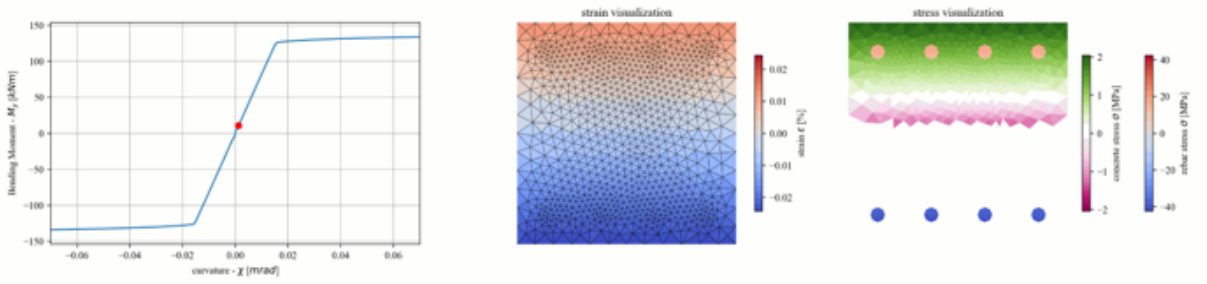


Figure 4.20: Resultant strains and stresses in a reinforced concrete section by the application of a given curvature χ_y

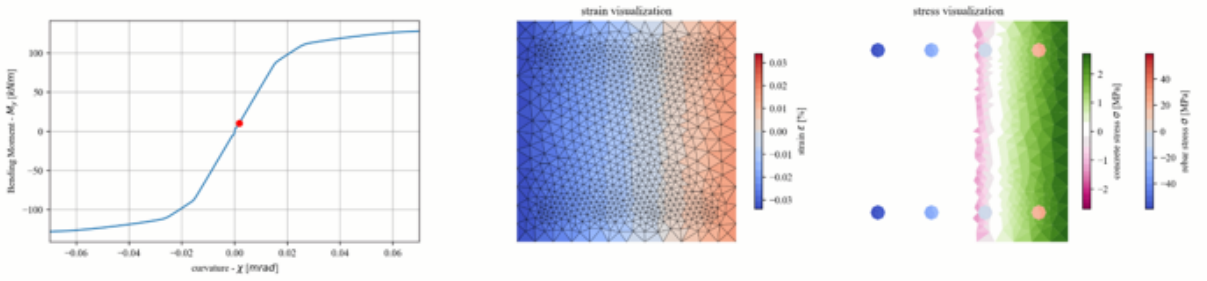


Figure 4.21: Resultant strains and stresses in a reinforced concrete section by the application of a given curvature χ_z

As for the steel cross-section, the influence of the normal force N_x on the bending moment capacities $M_{y/z}$ as well as the influence of M_y on M_z can be determined and visualized in the figures 4.22 and 4.23. In difference to the steel section, an increase in the normal force N_x also leads to an increase of the bending moment capacity M_y and M_z.

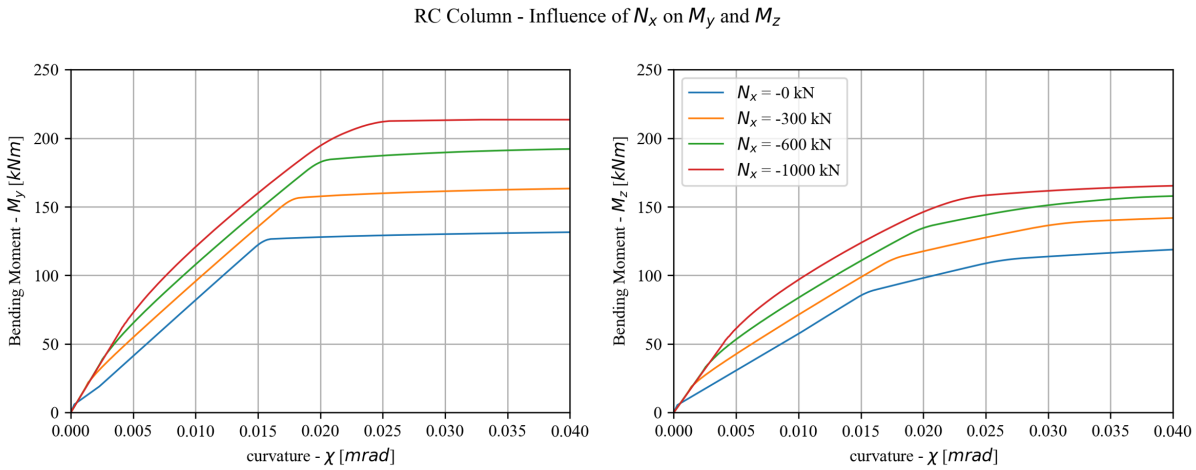


Figure 4.22: Influence of a given normal force N on the bending moment capacity M_y and M_z of a reinforced concrete section.

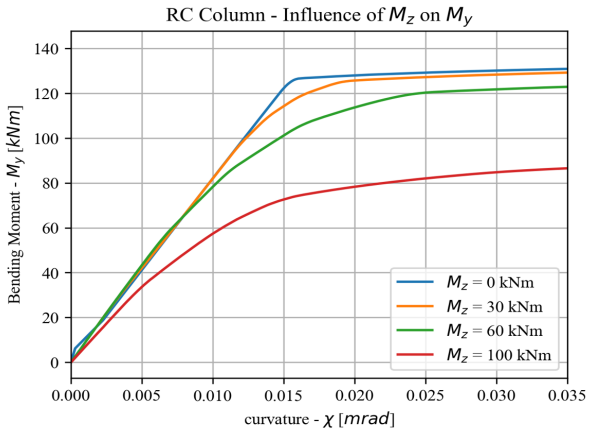
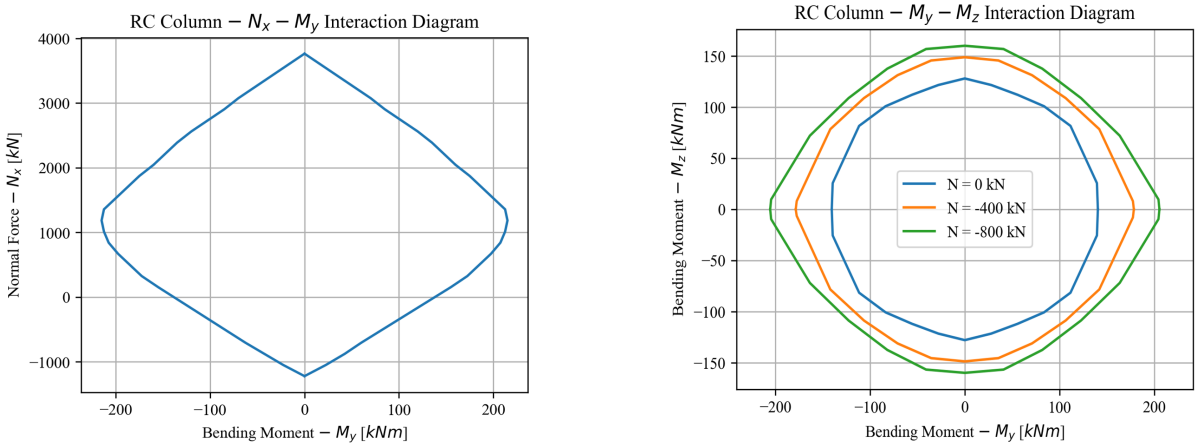


Figure 4.23: Influence of a given bending moment M_y on the bending moment capacity M_z of a reinforced concrete section.

The increase of the bending moment capacity with an increased normal force N_x can been seen in Figure 4.24a. Up to a normal force of approximately $N_x = 1150$ kN, the bending moment capacity increases from around $M_{y,Rd} = 150$ kNm at $N_x = 0$ kN up to $M_{y,Rd} = 215$ kNm. Rather than the normal force, skewed bending in the reinforced concrete section isn't beneficial regarding the bending moment capacities as Figure 4.24b shows.



(a) (b)

Figure 4.24: (a) $N - M_y$ interaction diagram and (b) $M_y - M_z$ interaction diagram under given normal forces $N = 0$ kN, 500 kN and 800 kN for a reinforced concrete section.

5 Discussion

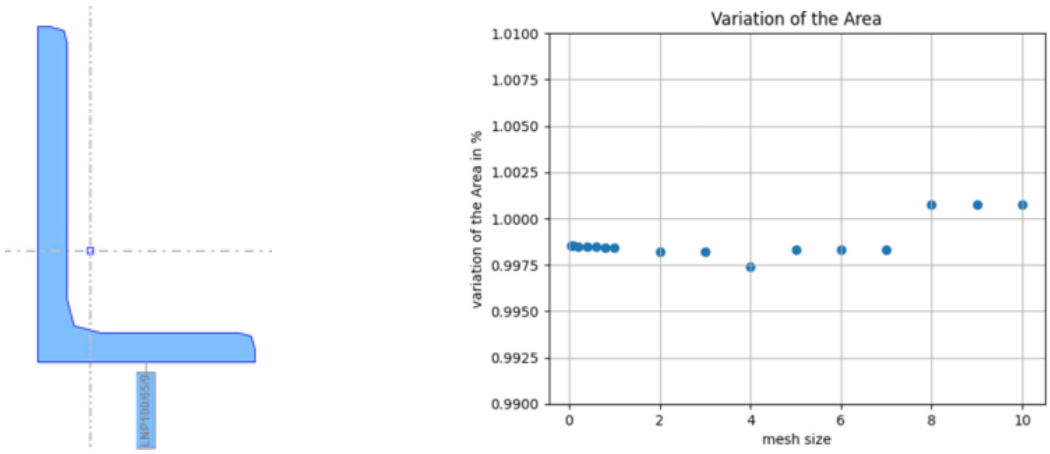
In this section, the produced results in chapter 4 are critically assessed. The cross-sectional properties, as well as the cross-sectional analysis results, will be compared to the software FAGUS-9 from CUBUS AG. To evaluate and verify the linear structural analysis, the software STATIK-9 will be used. The results from the nonlinear analysis will be compared to a simulation using the Open System for Earthquake Engineering Simulation (OpenSees).

In addition, some of the limitations of the implemented framework will be mentioned.

5.1 Comparisons

5.1.1 Cross Section Properties vs. FAGUS-9

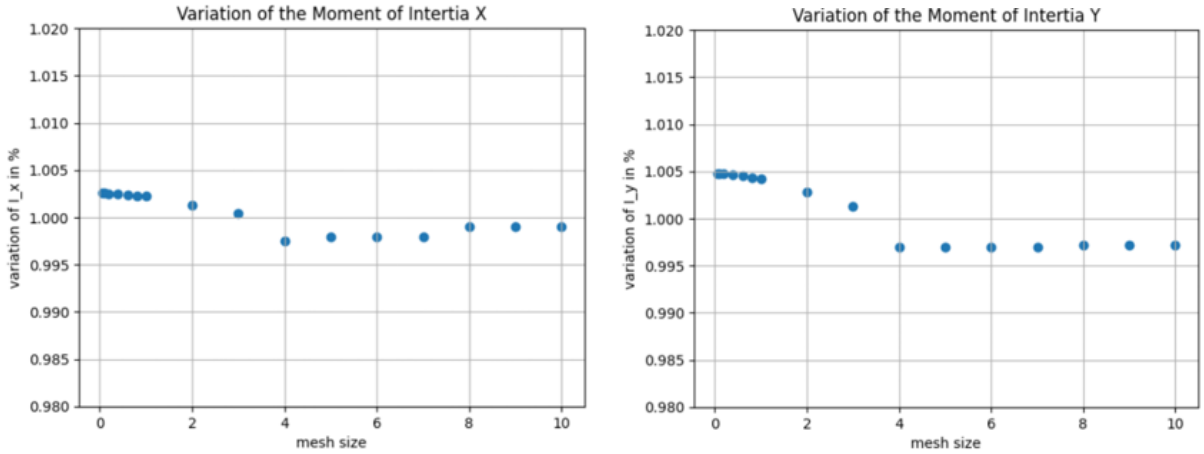
Regarding the cross-sectional properties like the area A_{cs} or the moments of inertia I_y and I_z, these were verified using the software FAGUS-9. FAGUS is a software developed by CUBUS AG with the purpose of cross-sectional analysis, primarily in the areas of reinforced and prestressed concrete [21]. During the introduction of a specific cross-section in FAGUS, one can extract multiple cross-sectional properties.



(a) LNP 100/65/9 (b) Normalized area A_{cs}/A_{FAGUS} vs. mesh size

Figure 5.1: (a) The LNP 100/65/9 steel profile in FAGUS-9 and (b) the variation of the normalized area A_{cs}/A_{FAGUS} on the approximate mesh size.

Figure 5.1a shows the introduced *LNP 100/65/9* steel cross-section in FAGUS. Depending on the mesh size, respectively the approximate distance between two nodes, the values of the cross-sectional area A_{cs} of the implemented framework were compared to the cross-sectional area resulting from FAGUS A_{FAGUS} in Figure 5.1. It is visible that even with a mesh size higher than the wall thickness of 9 mm, the value of the area differs by less than 0.3 %.



(a) (b)

Figure 5.2: The variation of the normalized moment of inertia I_{cs}/I_{FAGUS} on the approximate mesh size around (a) the y-axis and (b) the z-axis.

Nearly the same phenomenon is visible for the moments of inertia I_y and I_z in Figure 5.2. Whereas a convergence is visible for small mesh sizes, already the biggest mesh size delivers a value in the range of a deviation less than 0.5 % from the value delivered by FAGUS.

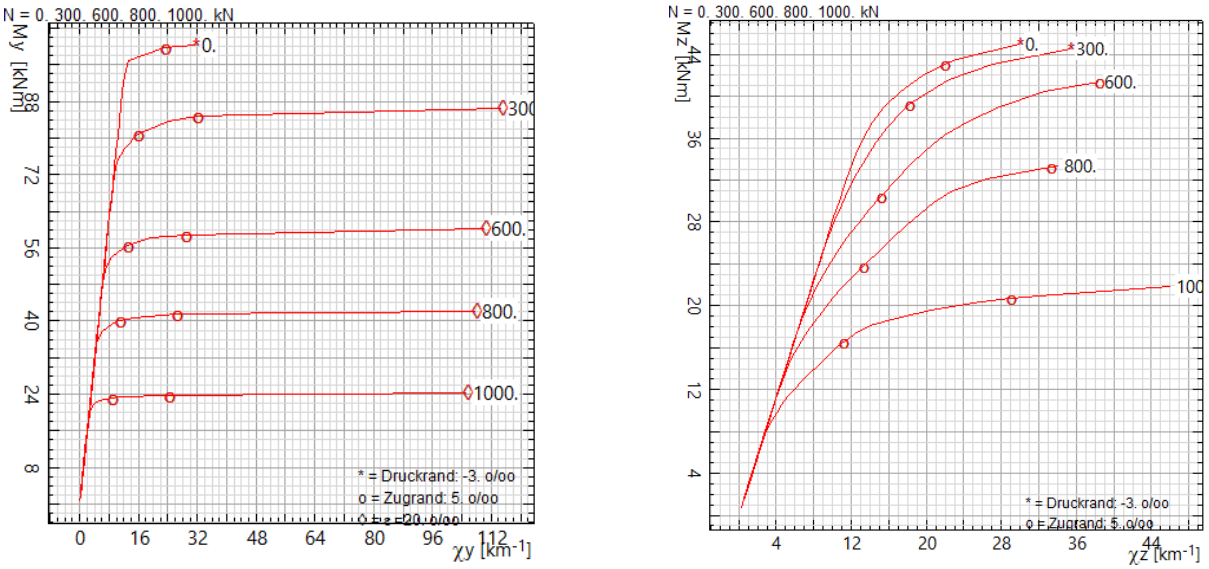
This, along with the sensitivity analysis on the mesh size in section 4.3.3, supports the statement that a coarse mesh is completely sufficient for most applications.

5.1.2 Cross Section Analysis vs. FAGUS-9

Besides the cross-sectional properties, also the results from the cross-sectional analysis in section 4.4 are compared to FAGUS. This will be done for the *HEA 200* steel cross-section as well as for the reinforced concrete cross-section.

Steel Beam - HEA 200

Figure 5.3 shows the influence of the normal force N_x on the bending moment capacities (a) M_y and (b) M_z. To produce these values, the analysis parameter *!GZG* has been chosen. Table 5.1 shows the numerical results from FAGUS compared to the results from the implemented cross-sectional analysis tool.



(a) (b)

Figure 5.3: Moment-curvature diagram of a steel *HEA 200* cross-section produced by FAGUS using *!GZG*.

The bending moment capacity around the y-axis from FAGUS produced higher values compared to the ones from the implemented framework. This is caused by the rounding between the web and the flanges of the steel beam, which is neglected in the framework. It is interesting to see that the values yielded by FAGUS are constantly about 5 kNm higher.

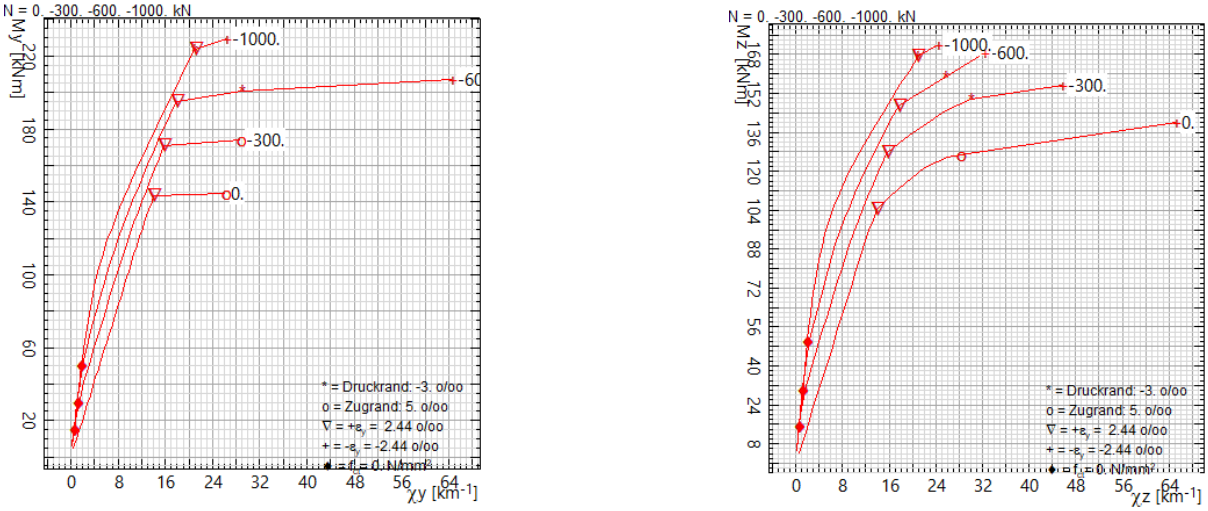
Around the z-axis, the bending moment capacities M_z delivered by FAGUS are lower for small normal loads N_x, which is caused by the limitation of strains in FAGUS. For higher normal loads, the behavior is the same as for M_y.

Table 5.1: Bending Moment capacity depending on the applied normal force for a HEA 200 steel section from the implemented framework and FAGUS.

N_x $[kN]$		0	300	600	800	1000
M_y $[kNm]$	FAGUS	100.3	86.5	60.1	42.2	24.2
	framework	96.6	81.8	55.8	37.8	19.0
M_z $[kNm]$	FAGUS	45.0	44.5	41.4	33.2	21.8
	framework	46.8	45.2	40.8	31.2	17.6

RC Column

For the analysis of the reinforced concrete section, in FAGUS the analysis parameter *!GZG* has been chosen. Figure 5.4 shows the moment-curvature plot of the reinforced concrete section generated using FAGUS.



(a) (b)

Figure 5.4: Moment-curvature diagram of a reinforced concrete section produced by FAGUS using *!GZG*.

The numerical values in Table 5.2 show the values from FAGUS being 3.5 % - 14.0 % higher. As the implemented geometry is exactly the same, the error has to come from the either the discretization of the cross-section in the implemented framework or the material laws, which are not necessarily the exact same, especially using the more sophisticated material law for the concrete shown in Figure 2.5b.

Table 5.2: Bending Moment capacity depending on the applied normal force for a reinforced concrete section from the implemented framework and FAGUS.

N_x $[kN]$		0	-300	-600	-1000
M_y $[kNm]$	FAGUS	144.4	174.0	207.1	229.5
	framework	132.0	164.5	193.0	213.5
M_z $[kNm]$	FAGUS	140.3	155.4	168.6	172.1
	framework	123.0	143.0	157.5	166.0

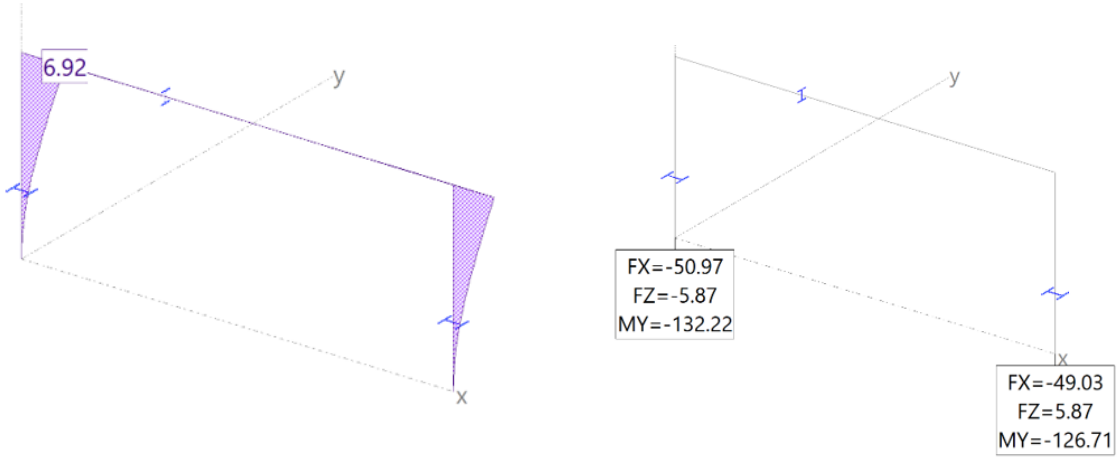
Regarding the reinforced concrete cantilever in section 4.3, which starts yielding at a load factor $\lambda = 135 - 140$ kN, wether at already $\lambda = 124.7$ kN as the analytical formula suggests, is caused by the upper reinforcement in the pressure zone, which is neglected by the analytical formula for M_{Rd}.

5.1.3 Linear Structural Analysis vs. STATIK-9

The linear structural analysis is validated with the software STATIK-9 from CUBUS AG. STATIK is a software for the linear-elastic analysis of general beam structures according to 1st and 2nd order theory [22].

As well as the frame in section 4.2, the frame introduced in STATIK consists of two 3 m high *HEB 300* columns and a 7 m long *HEA 200* beam. The columns were clamped at the bottom and a horizontal force of $F_x = 100$ kN was applied at the upper left node.

The frame displaces 6.92 mm to the side as shown in Figure 5.5 compared to the calculated 7.22 mm in section 4.2. The difference comes from the loss in the stiffness in the implemented geometry, which neglects the rounding. The reaction forces, compared to the ones from Table 4.2, have a maximum deviation of 0.2 %.



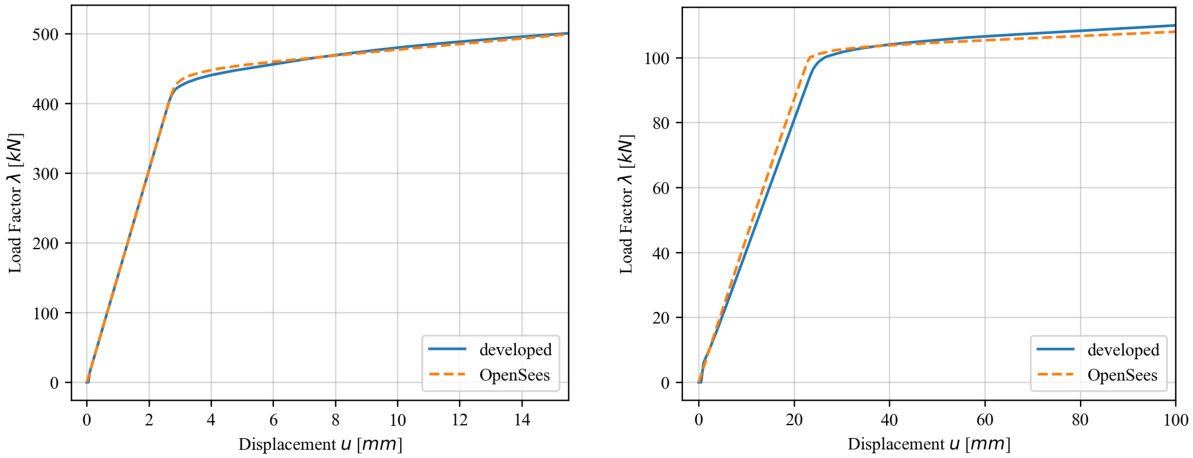
(a) (b)

Figure 5.5: (a) Displacements d_x and (b) reaction forces of the clamped one story frame obtained from STATIK-9.

5.1.4 Nonlinear Structural Analysis vs. OpenSees

To verify the accuracy of the developed nonlinear fiber element solver, the simulations were compared to the results obtained from OpenSeesPy, which is the Python interface to OpenSees. The elements chosen in the OpenSees simulations were also flexibility-based fiber elements using a Gauss-Lobatto integration. The two cantilevers mentioned in section 4.3 were used for comparison. The Python code used for the OpenSees simulations is shown in appendix B.

The resulting load-deformation curves in Figure 5.6 show excellent agreement in both regions, in the linear-elastic and in the plastic region. The start of the yielding matches closely. Minor deviations at higher load factors can be caused by different numerical tolerances or convergence criteria, as well as slightly different material laws.



(a) Steel Beam - HEB 300 (b) RC Column

Figure 5.6: Load-deformation corves obtained from the implemented framework and OpenSeesPy

5.2 Limitations

The implemented framework for a nonlinear fiber-element analysis as well as cross-sectional analysis performs well compared to established software. But there are at the moment several limitations that remain:

- Only nodal loads are supported. The current solver does not yet allow for distributed axial or lateral loads along the elements.
- There is no support for dynamic respectively seismic loading. The materials laws can be expanded with a history variable and therefore some hysteretic material laws such as Kent-Park or Mengetto-Pinto could be implemented for quasi-static simulations [10].
- The current fiber formulation uses a linear force interpolation function regarding the bending moments and a constant one regarding the axial force. For the implementation of distributed loads, this formulation of the force-interpolation matrix has to be revised.

These constraints are primarily due to scope limitations of the thesis. The object-oriented design of the framework should allow for relatively simple integration of these functionalities in future development.

6 Conclusion

This thesis presented the development and the theory behind a transparent and modular flexibility-based fiber element solver. Implemented in Python, the solver enables linear and nonlinear structural analysis of general three-dimensional beam structures based on fiber discretization and uniaxial material laws. The framework includes modules for cross-sectional analysis as well as a tool for lamina analysis, specifically useful to get equivalent material properties for homogenized materials.

The framework was validated against established software tools such as FAGUS for cross-sectional properties and force-strain analysis, STATIK for linear structural analysis and OpenSees for nonlinear load-deformation curves. These comparisons confirm the reliability and accuracy of the implemented framework. Beyond this, the educational value lies in the open and extensible structure of the code. Students and researchers can directly interact with the implemented methods and are able to adapt or extend them.

A key advantage is the ease with which any uniaxial material law, whether it is elastic, bilinear or fully nonlinear, can be implemented. This allows for comparative studies of how different material laws affect the structural response.

Overall, the framework not only supports the practical workflow of an engineer but also serves as a didactic platform that builds the bridge between theory and application. While the current scope is limited to nodal loading and an-hysteretic material behavior, the structure of the framework supports future extensions such as distributed loads, hysteretic material models or dynamic analysis. In this way, the project establishes a solid foundation for both educational and engineering tasks.

Bibliography

[1] Marco Terrenzi, Enrico Spacone, and Camata Guido. Comparison between phenomenological and fiber-section non-linear models. *Front. Built Environ*, Volume 6, 2020. URL https://doi.org/10.3389/fbuil.2020.00038.

[2] Fabio Taucer, Enrico Spacone, and Filip Filippou. A fiber beam-column element for seismic response analysis of reinforced concrete structures. 01 1991. URL https://www.researchgate.net/publication/242435677_A_Fiber_Beam-Column_Element_for_Seismic_Response_Analysis_of_Reinforced_Concrete_Structures.

[3] Michael Scott and Gregory Fenves. Plastic hinge integration methods for force-based beam–column elements. *Journal of Structural Engineering-asce - J STRUCT ENG-ASCE*, 132, 02 2006. doi: 10.1061/(ASCE)0733-9445(2006)132:2(244).

[4] I. D. Rodrigues, G. H. F. Cavalvante, E. M. V. Pereira, L. C. M. Vieera Jùnior, A. Liel, and G. H. Siqueira. Seismic fragility assessment of a rc frame considering concentrated and distributed plasticity modelling. *IBRACON*, vol. 17, 2024. URL https://doi.org/10.1590/S1983-41952024000100005.

[5] Michael H. Scott. A tale of two element formulations. 2020. URL https://portwooddigital.com/2020/02/23/a-tale-of-two-element-formulations/.

[6] Mark E. Tuttle. Summary of classical lamination theory (clt) calculations. 2013. URL https://courses.washington.edu/mengr450/CLT_Summary.pdf.

[7] Paul Borke. Surface (polygonal) simplification. 1997. URL https://paulbourke.net/geometry/polygonmesh/.

[8] VIBUMA. Classification of wooden flooring. 2025. URL https://vibuma.com/en/blog/classification-of-wooden-flooring.thread997.html.

[9] Mosallam AS, Bayraktar A, Elmikawi M, Pul S, Adanur S. Polymer composites in construction: An overview. 2013. URL http://dx.doi.org/10.15226/sojmse.2014.00107.

[10] Mahmoud Zidan. Implementation of a fiber beam finite element for the simulation of steel-reinforced concrete beam under dynamic loading. 2019. URL https://zidanmahmoud.github.io/project/beamcolumn/MT.pdf.

[11] Prof. Dr. Eleni Chatzi. Introduction to the MFE. *Lecture FEM I, ETH Zürich*, 2024. URL https://chatzi.ibk.ethz.ch/education/method-of-finite-elements-i.html.

[12] Prof. Dr. Eleni Chatzi and Konstantinos Vlachas. Computer lab 1 – implementation of arc-length solvers with a total lagrangian bar element. *Lecture FEM II, ETH Zürich*, 2024. URL https://chatzi.ibk.ethz.ch/education/method-of-finite-elements-ii.html.

[13] Prof. Dr. Walter Kaufmann. Sbi-2 materialverhalten. *Vorlesung Stahlbeton I, ETH Zürich*, 2024. URL https://concrete.ethz.ch/assets/sbe-i/autographien/stahlbeton-i_2_materialverhalten_hs2024_auto.pdf.

[14] Wisam Hulail and Dina Hamza. Formulation of mathematical model for stress-strain relationship of normal and high strength concrete under compression. *Civil and Environmental Engineering*, 19, 05 2023. doi: 10.2478/cee-2023-0011.

[15] Christophe Geuzaine and Jean-François Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. 2009. URL https://gmsh.info/doc/preprints/gmsh_paper_preprint.pdf.

[16] Anton Schweizer. Festigkeits-, steiffigkeits- und rohdichtekennwerte von holzwerkstoffen. 2022. URL https://www.schweizer-fn.de/festigkeit/festigkeitswerte/holz/holz.php#.

[17] KAST. Sr grid technische daten. 2025. URL https://www.kast.de/wp-content/uploads/2025/02/SR-Grid-technische-Daten_de_en_.pdf.

[18] Dr. Nebojsa Mojsilovic. Ethz lecture: Mauerwerk - steiffigkeit. 2024. URL https://www.kast.de/wp-content/uploads/2025/02/SR-Grid-technische-Daten_de_en_.pdf.

[19] Stahlbau Zentrum Schweiz. Konstruktionstabellen c5/05. 2005.

[20] Prof. Dr. Walter Kaufmann. Stabtragwerke - biegung. *Vorlesung Stahlbeton I, ETH Zürich*, 2024. URL https://concrete.ethz.ch/assets/sbe-i/slides/stahlbeton-i_3_2_stabtragwerke_biegung_hs2024.pdf.

[21] CUBUS AG. Fagus - querschnittsdefinition und querschnittsanalysen. 2025. URL https://www.cubus-software.com/Guests/Produkte/Fagus/d_main.html.

[22] CUBUS AG. Statik - stabtragwerke. 2025. URL https://www.cubus-software.com/Guests/Produkte/Statik/d_main.html.

List of Figures

List of Tables

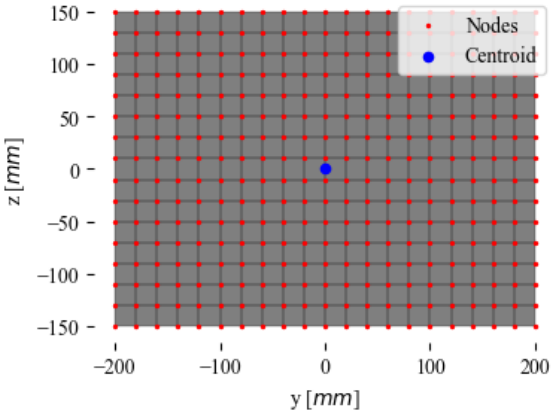
A Examples

A.1 Setting Up Cross Sections

01 Example: Cross-Sectional Properties

In [1]:
```
from Structure import *
```

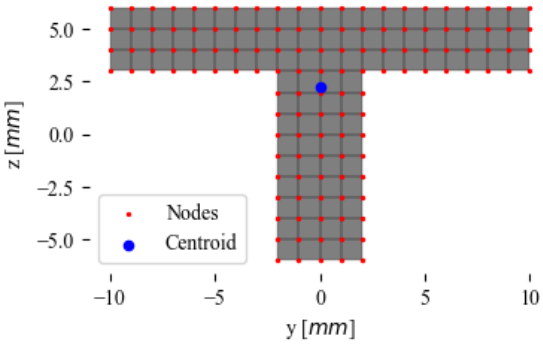
In [2]:
```
rect = Rectangle(width  = 400,
                 height = 300)

mesh = Mesh(rect, "quadrilateral", 20)

mesh.plot()
mesh.print()
```



Mesh Type	quadrilateral
Number of elements	300
Number of nodes	336
Cross Section Area	120000.00

	y	z
Centroid [mm]	0.00	0.00
Moment of inertia [mm^4]	9.00e+08	1.60e+09

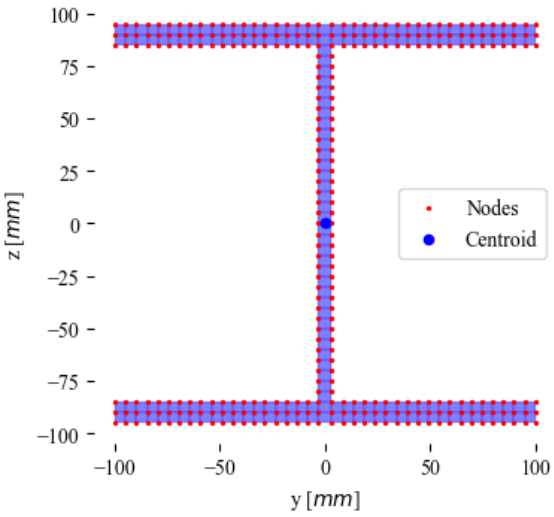
In [3]:
```
T    = T_profile(web_width     =  4,
                 web_height    =  9,
                 flange_width  = 20,
                 flange_height =  3)

mesh = Mesh(T, "quadrilateral", 1)

mesh.plot()
mesh.print()
```



Mesh Type	quadrilateral
Number of elements	96
Number of nodes	129
Cross Section Area	96.00

	y	z
Centroid [mm]	0.00	2.25
Moment of inertia [mm^4]	1098.00	2048.00

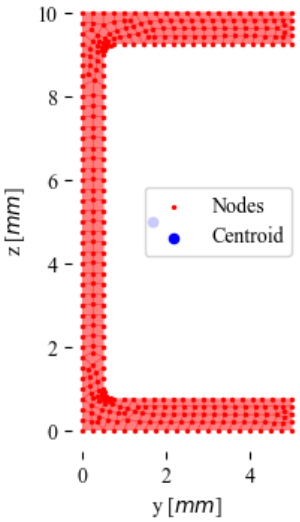
In [4]:
```
H    = H_beam(web_width     =   6.5,
              web_height    = 170.0,
              flange_width  = 200.0,
              flange_height =  10.0)

mesh = Mesh(geometry  = H,
            mesh_type = "quadrilateral",
            mesh_size = 5)

mesh.plot()
mesh.print()
```



Mesh Type	quadrilateral
Number of elements	190
Number of nodes	306
Cross Section Area	5105.00

	y	z
Centroid [mm]	0.00	0.00
Moment of inertia [mm^4]	3.51e+07	1.33e+07

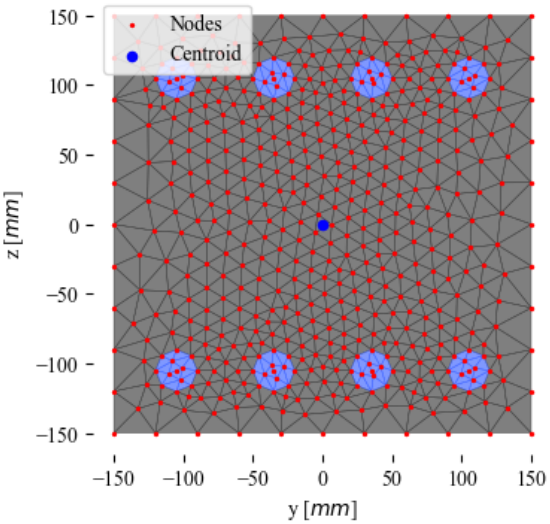
In [5]:
```
points = [( 0.0, 0.0),
          ( 5.0, 0.0),
          ( 5.0, 0.75),
          ( 0.75, 0.75),
          ( 0.575, 0.825),
          ( 0.5, 1.0),
          ( 0.5, 9.0),
          ( 0.575, 9.175),
          ( 0.75, 9.25),
          ( 5.0, 9.25),
          ( 5.0,10.0),
          ( 0.0,10.0)]

poly = Polygon(points)
mesh = Mesh(poly, "quadrilateral", 0.25)

mesh.plot()
mesh.print()
```



Mesh Type	quadrilateral
Number of elements	265
Number of nodes	348
Cross Section Area	11.79

	y	z
Centroid [mm]	1.68	5.00
Moment of inertia [mm^4]	187.03	29.49

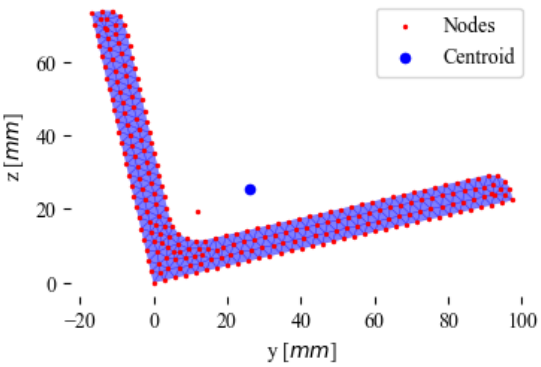
In [6]:
```
RC = ReinforcedConcreteColumn(width  = 300,
                              height = 300,
                              concrete_cover = 30,
                              rebar_diameter = 30,
                              rebar_spacing  = 50)
mesh = Mesh(geometry  = RC,
            mesh_type = "triangle",
            mesh_size = 30)
mesh.plot()
mesh.print()
```



Mesh Type	triangle
Number of elements	1038
Number of nodes	548
Cross Section Area	90000.00

	y	z
Centroid [mm]	0.00	0.00
Moment of inertia [mm^4]	6.75e+08	6.75e+08

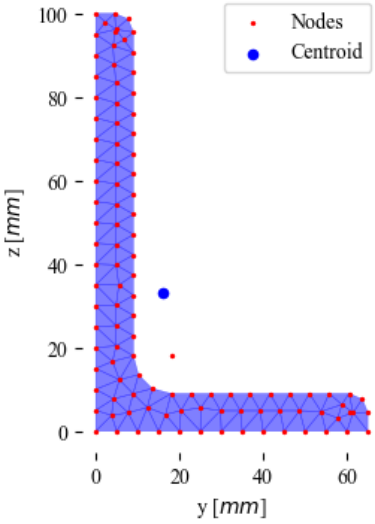
In [7]:
```
L = L_beam(100, 75, 8, 8, 5, 12.95)
mesh = Mesh(L, "triangle", 3)

mesh.plot()
mesh.print()
```



Mesh Type	triangle
Number of elements	363
Number of nodes	244
Cross Section Area	1338.06

	y	z
Centroid [mm]	26.12	25.24
Moment of inertia [mm^4]	4.37e+05	1.52e+06

In [8]:
```
L = L_beam(65, 100, 9, 9, 4.5, 0)
mesh = Mesh(L, "triangle", 5)

mesh.plot()
mesh.print()
```



Mesh Type	triangle
Number of elements	141
Number of nodes	108
Cross Section Area	1412.39

	y	z
Centroid [mm]	15.93	33.18
Moment of inertia [mm^4]	1.40e+06	4.66e+05

A.2 Lamina Analysis

02 Example: Lamina Analysis

Setup

import all the needed libraries and files.

In [1]:
```
from Material import Laminate, LaminateLoadAnalysis
```

Build the Lamina

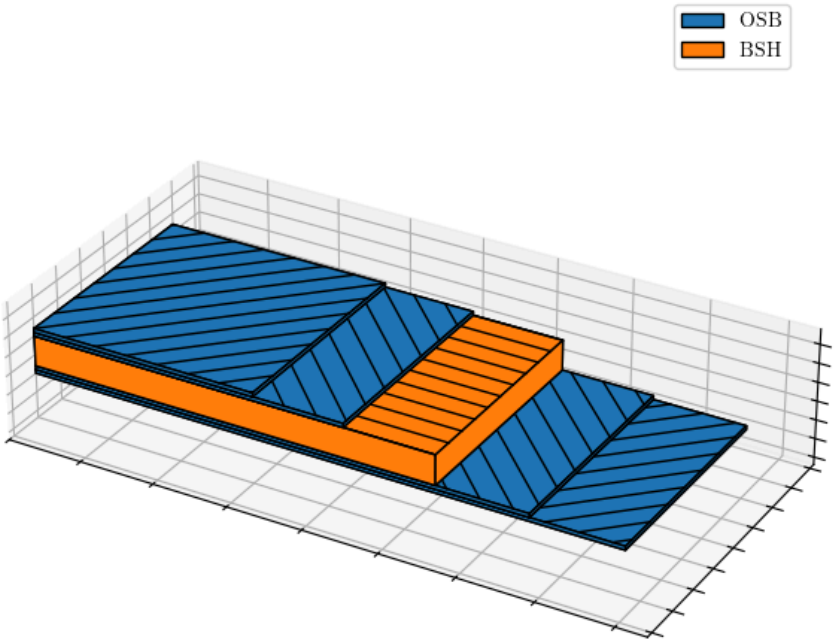
A lamina can have as many layers as you would like it to have. Each layer consist of a name and the following parameters:

- E_1 Young's modulus in the main direction
- E_2 Young's modulus in the lateral direction
- G_{12} Shear Modulus
- v_{12} Poisson's number
- θ orientation of the main direction
- h thickness of the specific ply

In [ ]:
```
layers = [
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta':  45, 'thickness': 10},
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta': -45, 'thickness': 10},
    {'name': "BSH", 'E1': 9100, 'E2':  250, 'G12': 540, 'v12': 0.3,
     'theta':   0, 'thickness': 80},
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta': -45, 'thickness': 10},
    {'name': "OSB", 'E1': 3230, 'E2': 2550, 'G12': 920, 'v12': 0.3,
     'theta':  45, 'thickness': 10}
]

Lam = Laminate(layers)
```

A Laminate can be plotted to check the applied materials as well as their orientation

In [3]:
```
Lam.plot()
```



The ABD-Matrix as well as the properties of a corresponding material then can be printed

In [4]:
```
Lam.print_results()
```

A Matrix	B Matrix	D Matrix
[[8.27e+05 6.63e+04 0.00e+00] [6.63e+04 1.17e+05 0.00e+00] [0.00e+00 0.00e+00 1.35e+05]]	[[0.00e+00 -1.16e-10 0.00e+00] [-1.16e-10 0.00e+00 0.00e+00] [0.00e+00 0.00e+00 0.00e+00]]	[[6.35e+08 1.56e+08 -7.32e+06] [1.56e+08 2.57e+08 -7.32e+06] [-3.66e+06 -3.66e+06 2.55e+08]]

Property	Value
Ex	6890
Ey	976
Gxy	1120
vxy	0.566

Stress-Strain Analysis

Once the lamina is build toghether correctly, one can also do a stress-strain analysis on the material for some applied axial forces N_x, N_y, shear forces N_{xy} or bending moments M_x, M_y or M_{xy}

In [5]:
```
laminate_analysis = LaminateLoadAnalysis(Lam)
```

At first, the midplane stress and curvature is calculated depending on the applied loads using the ABD matrix

In [6]:
```
midplane_strains, midplane_curvatures = laminate_analysis.apply_load(
                                                        Nx=100,
                                                        Ny=0,
                                                        Nxy=0)
```

After, the stresses and strains at each ply can be backcalculated

In [7]:
```
ply_strains, ply_stresses = laminate_analysis.compute_ply_stresses_strains(
    midplane_strains,
    midplane_curvatures
)
```

Lastly, one can print and plot the laminas reaction to the applied forces

In [8]:
```
laminate_analysis.print_ply_results(ply_strains, ply_stresses)
laminate_analysis.plot_stress_strain_variation()
```

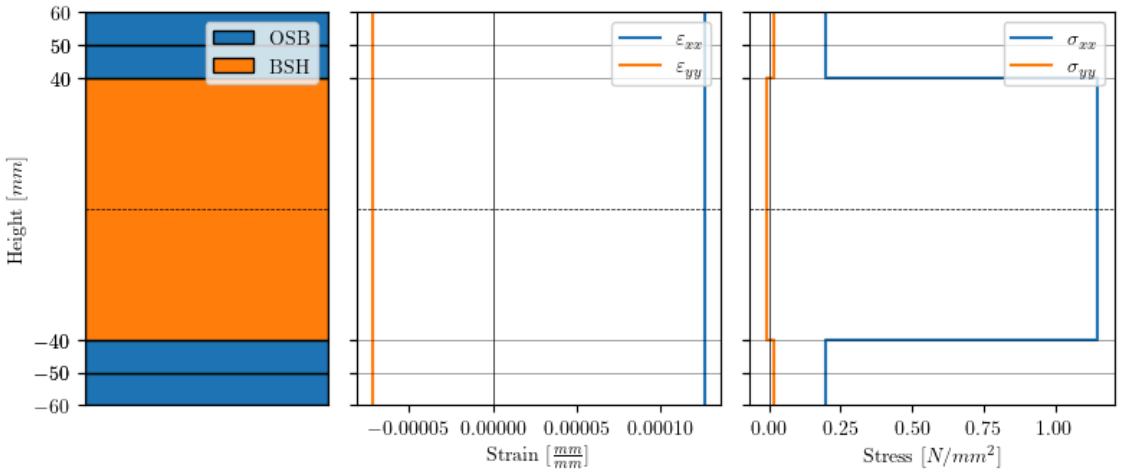
strains per layer:

layer	e_xx	e_yy	e_xy
1	0.000127	-7.17e-05	-3.55e-23
2	0.000127	-7.17e-05	-2.9e-23
3	0.000127	-7.17e-05	0
4	0.000127	-7.17e-05	2.9e-23
5	0.000127	-7.17e-05	3.55e-23

stresses per layer:

layer	s_xx	s_yy	t_xy
1	0.199	0.0169	-0.0101
2	0.199	0.0169	0.0101
3	1.15	-0.00845	0
4	0.199	0.0169	0.0101
5	0.199	0.0169	-0.0101

$$N_x = 100,\ N_y = 0,\ N_{xy} = 0$$
$$M_x = 0,\ M_y = 0,\ M_{xy} = 0$$



Another example

In [9]:
```
midplane_strains, midplane_curvatures = laminate_analysis.apply_load(
    Nx=10 , Ny=0, Nxy=0,
    Mx=100, My=0, Mxy=0
)

ply_strains, ply_stresses = laminate_analysis.compute_ply_stresses_strains(
```

```
    midplane_strains,
    midplane_curvatures
)

laminate_analysis.print_ply_results(ply_strains, ply_stresses)
laminate_analysis.plot_stress_strain_variation()
```

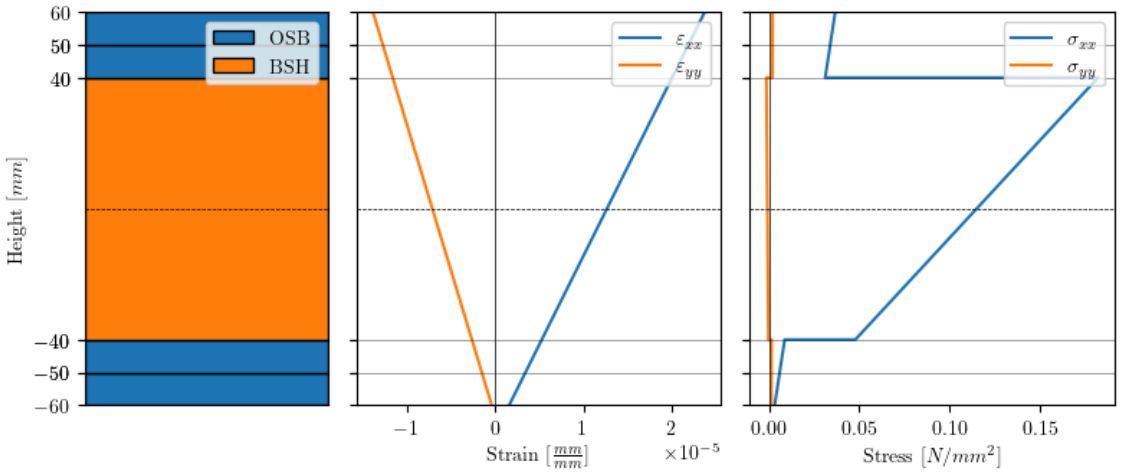
strains per layer:

layer	e_xx	e_yy	e_xy
1	2.49e-06	-9.89e-07	-5.74e-08
2	4.34e-06	-2.11e-06	-4.69e-08
3	1.27e-05	-7.17e-06	0
4	2.1e-05	-1.22e-05	4.69e-08
5	2.28e-05	-1.34e-05	5.74e-08

stresses per layer:

layer	s_xx	s_yy	t_xy
1	0.00458	0.00138	-0.000406
2	0.00734	0.0014	0.0003
3	0.115	-0.000845	0
4	0.0325	0.00198	0.00171
5	0.0353	0.002	-0.00161

$$N_x = 10,\ N_y = 0,\ N_{xy} = 0$$
$$M_x = 100,\ M_y = 0,\ M_{xy} = 0$$



A.3 Linear Solver

03 Example: Linear Frame Analysis

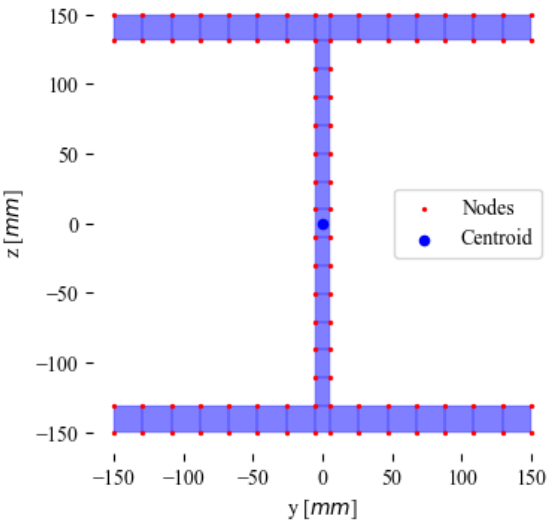
Setup

import all the needed libraries and files.

In [1]:
```
import numpy as np

from Structure import *
from Solver import *

from Plotting_Functions import *
```

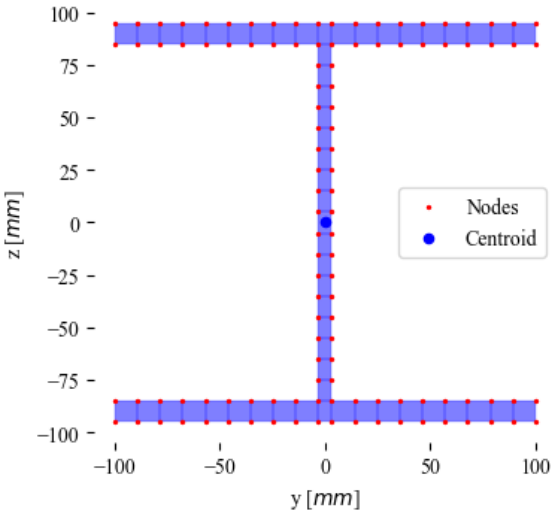
Generate the Geometry and the corresponding mesh

In [2]:
```
# Initalize the geometry of the columns
HEB_300 = H_beam(web_width     =  11.0,
                 web_height    = 262.0,
                 flange_width  = 300.0,
                 flange_height =  19.0)

# Create the mesh of the column
column = Mesh(HEB_300, mesh_type="quadrilateral", mesh_size=20)

# Print and plot the mesh of the column
column.plot()
column.print()

# Initalize the geometry of the horizontal beam
HEA_200 = H_beam(web_width     =   6.5,
                 web_height    = 170.0,
                 flange_width  = 200.0,
                 flange_height =  10.0)

beam = Mesh(HEA_200, mesh_type="quadrilateral", mesh_size=10)
beam.plot()
```



Mesh Type	quadrilateral
Number of elements	43
Number of nodes	88
Cross Section Area	14282.00

	y	z
Centroid [mm]	0.00	0.00
Moment of inertia [mm^4]	2.42e+08	8.55e+07

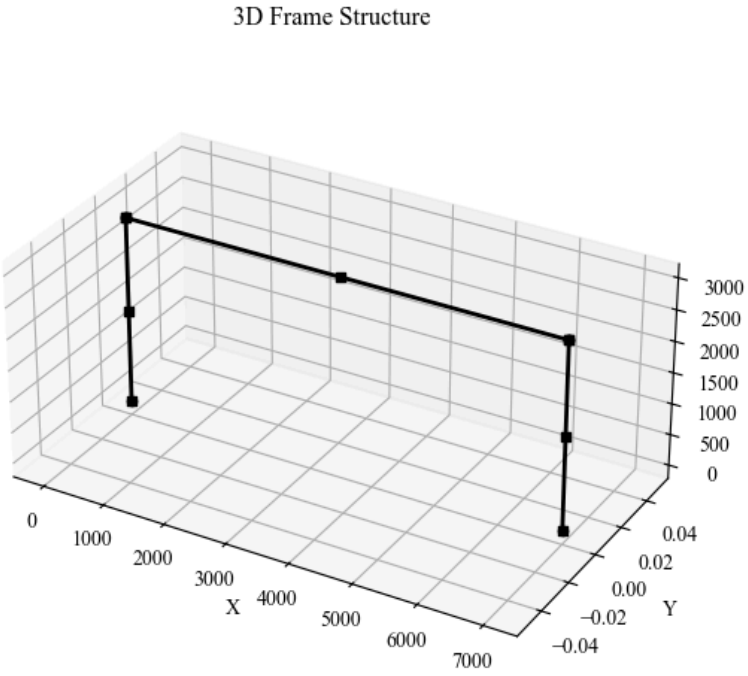


Structure

Initialize the structure. For a frame, one has to define the columns and the beams geometry, the number of storeys, their width and height as well as the DOF where nodal loads will be applied and their value.

Each node has 6 DOFs (counting starts from 0). The nodes start counting on the bottom left (0), then bottom right (1), then one storey up on the left side (2) and so on. To apply a lateral load we need the first DOF (global x-direction) of the node number 2, so therefore DOF n*6* = *2*6 = 12.

In [3]:
```
frame = Frame(column            = column,
              beam              = beam,
              number_of_stories = 1,
              story_height      = 3000,
              story_width       = 7000,
              load_DOFs         = [12],
              nodal_loads       = [100000],
              number_of_sections_per_element = 3)

plot_initial_structure(frame)
```

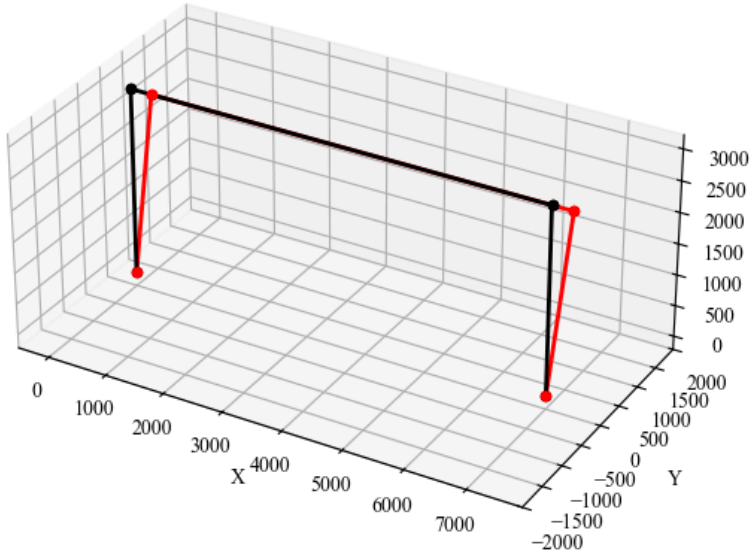


Solver

In [4]:
```
lin_solver = Linear(frame)
```

In [5]:
```
lin_solver.solve()
plot_displaced_structure(lin_solver, scale=50.0)
```

```
-------------------------------------
Node 0
          Displacement
u [mm]        0.000000
v [mm]        0.000000
w [mm]        0.000000
θx [rad]      0.000000
θy [rad]      0.000000
θz [rad]      0.000000
          Internal Force
Fx [kN]      -50.983094
Fy [kN]        0.000000
Fz [kN]       -5.823099
Mx [kNm]      -0.000000
My [kNm]    -132.405234
Mz [kNm]       0.000000
-------------------------------------
Node 1
          Displacement
u [mm]        0.000000
v [mm]        0.000000
w [mm]        0.000000
θx [rad]      0.000000
θy [rad]      0.000000
θz [rad]      0.000000
          Internal Force
Fx [kN]      -49.016906
Fy [kN]       -0.000000
Fz [kN]        5.823099
Mx [kNm]      -0.000000
My [kNm]    -126.833077
Mz [kNm]       0.000000
-------------------------------------
Node 2
          Displacement
u [mm]        7.226855
v [mm]        0.000000
w [mm]        0.005825
θx [rad]     -0.000000
θy [rad]      0.003310
θz [rad]     -0.000000
          Internal Force
Fx [kN]      100.000000
Fy [kN]       -0.000000
Fz [kN]       -0.000000
Mx [kNm]       0.000000
My [kNm]       0.000000
Mz [kNm]       0.000000
-------------------------------------
Node 3
          Displacement
u [mm]        6.906797
v [mm]        0.000000
w [mm]       -0.005825
θx [rad]     -0.000000
θy [rad]      0.003154
θz [rad]     -0.000000
          Internal Force
Fx [kN]       -0.000000
Fy [kN]       -0.000000
Fz [kN]       -0.000000
Mx [kNm]      -0.000000
My [kNm]      -0.000000
Mz [kNm]      -0.000000
```


3000
2500
2000
1500
1000
500
0
0
1000
2000
3000
X
4000
5000
6000
7000
2000
1500
1000
500
0
−500
Y
−1000
−1500
−2000

A.4 Nonlinear Solver

04 Example: Nonlinear Frame Analysis and Cross-Sectional Analysis

---

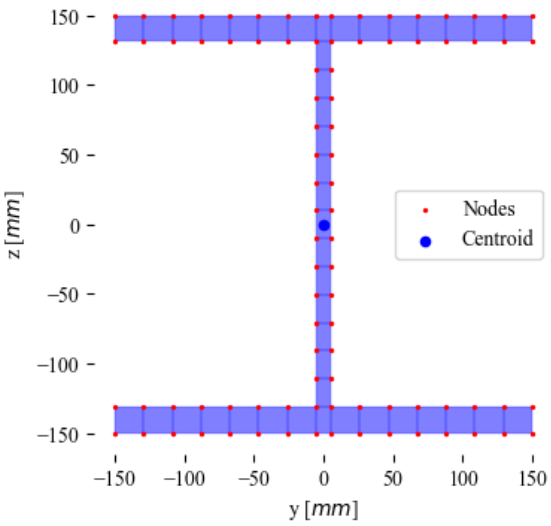
Setup

import all the needed libraries and files.

In [1]:
```
import numpy as np

from Structure import *
from Material import *
from Solver import *

from Plotting_Functions import *
```

Generate the Geometry and the corresponding mesh

In [2]:
```
# Initalize the geometry of the column
HEB_300 = H_beam(web_width     =  11.0,
                 web_height    = 262.0,
                 flange_width  = 300.0,
                 flange_height =  19.0)

# Create the mesh of the column
column = Mesh(HEB_300, mesh_type="quadrilateral", mesh_size=20)

# Print and plot the mesh of the column
column.plot()
column.print()
```



Mesh Type	quadrilateral
Number of elements	43
Number of nodes	88
Cross Section Area	14282.00

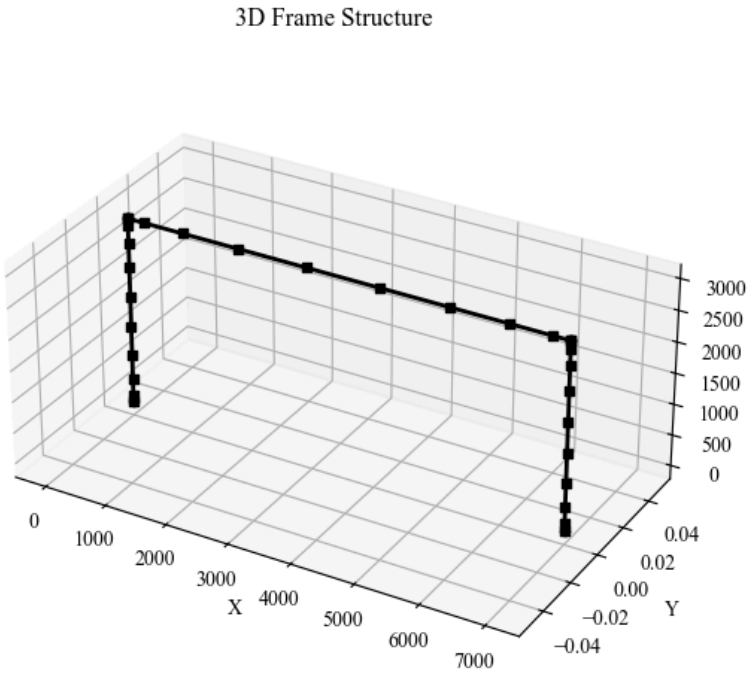
	y	z
Centroid [mm]	0.00	0.00
Moment of inertia [mm^4]	2.42e+08	8.55e+07

Structure

Initialize the structure. For a frame, one has to define the columns and the beams geometry, the number of storeys, their width and height as well as the DOF where nodal loads will be applied and their value.

Each node has 6 DOFs (counting starts from 0). The nodes start counting on the bottom left (0), then bottom right (1), then one storey up on the left side (2) and so on. To apply a lateral load we need the first DOF (global x-direction) of the node number 2, so therefore DOF n*6* = *2*6 = 12.

In [3]:
```
frame = Frame(column            = column,
              beam              = column,
              number_of_stories = 1,
              story_height      = 3000,
              story_width       = 7000,
              load_DOFs         = [12],
              nodal_loads       = [1],
              number_of_sections_per_element = 10)

plot_initial_structure(frame)
```



Solver

In [4]:
```
non_linear_solver = Nonlinear(frame, constraint="Load")
```

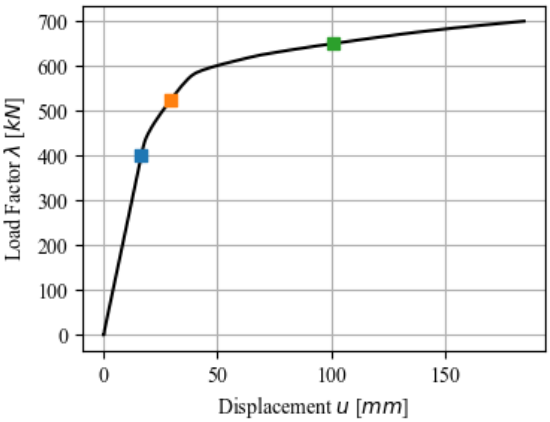
In [5]:
```
increments = np.zeros(140)
increments.fill(5000)

u_history, lambda_history, section_forces, section_strains = non_linear_solver.solve(increments)
```

```
  0%|          | 0/140 [00:00<?, ?it/s]
------------------------------------------------
Load step 1 of 140
   Attempt  1
      NR Iteration  0
         Beam Element  1
            Element iteration  0
         Beam Element  2
            Element iteration  0
         Beam Element  3
            Element iteration  0
      Residuals Norm  2.3428007491026083e-09
NR Converged!
------------------------------------------------
Load step 2 of 140
   Attempt  1
      NR Iteration  0
         Beam Element  1
            Element iteration  0
         Beam Element  2
            Element iteration  0
         Beam Element  3
            Element iteration  0
      Residuals Norm  2.8062904113499834e-09
NR Converged!
------------------------------------------------
.
.
.
------------------------------------------------
Load step 140 of 140
   Attempt  1
      NR Iteration  0
         Beam Element  1
            Element iteration  0
            Element iteration  1
         Beam Element  2
            Element iteration  0
         Beam Element  3
            Element iteration  0
            Element iteration  1
      Residuals Norm  41004.4817883947
      NR Iteration  1
         Beam Element  1
            Element iteration  0
         Beam Element  2
            Element iteration  0
         Beam Element  3
            Element iteration  0
      Residuals Norm  5.820766091445162e-11
NR Converged!
```

Load Displacement Curve

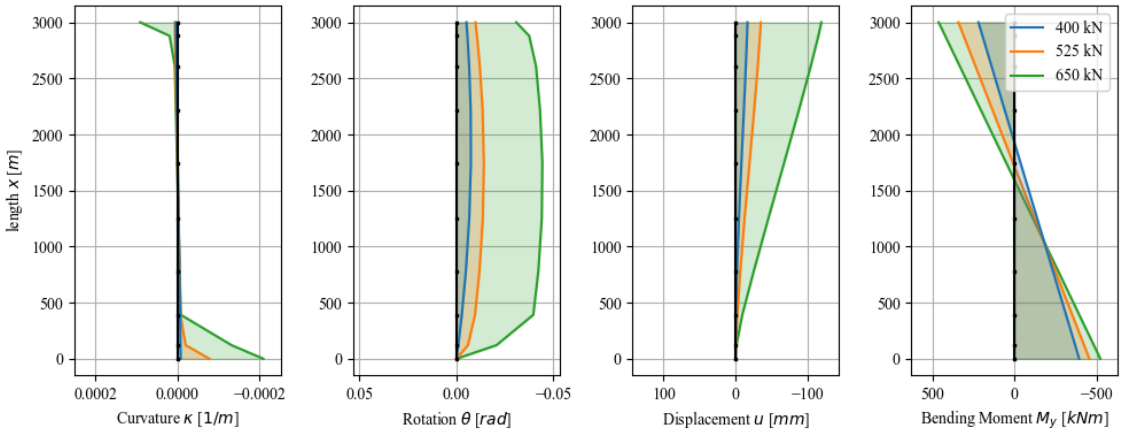
In [6]:
```
import matplotlib.pyplot as plt

# Main figure and axes
fig, ax = plt.subplots(figsize=(4, 3))
plt.rcParams["font.family"] = "Times New Roman"


ax.plot(u_history[:,12], lambda_history/1000,   label="Frame", color='k')

# plot the corresponding points to the figure of the below
ax.scatter(u_history[:,12][80], lambda_history[80]/1000, color='C0', s=30, zorder=5, marker='s')
ax.scatter(u_history[:,12][105], lambda_history[105]/1000, color='C1', s=30, zorder=5, marker='s')
ax.scatter(u_history[:,12][130], lambda_history[130]/1000, color='C2', s=30, zorder=5, marker='s')
```

```
ax.set_xlabel("Displacement $u$ [$mm$]")
ax.set_ylabel("Load Factor $\\lambda$ [$kN$]")
ax.grid()
plt.show()
```



Section Forces and Curvature

In [7]:
```
steps = [80, 105, 130]
plot_moments(steps, section_forces, section_strains, non_linear_solver, length=3000)
```



Cross Section Analysis

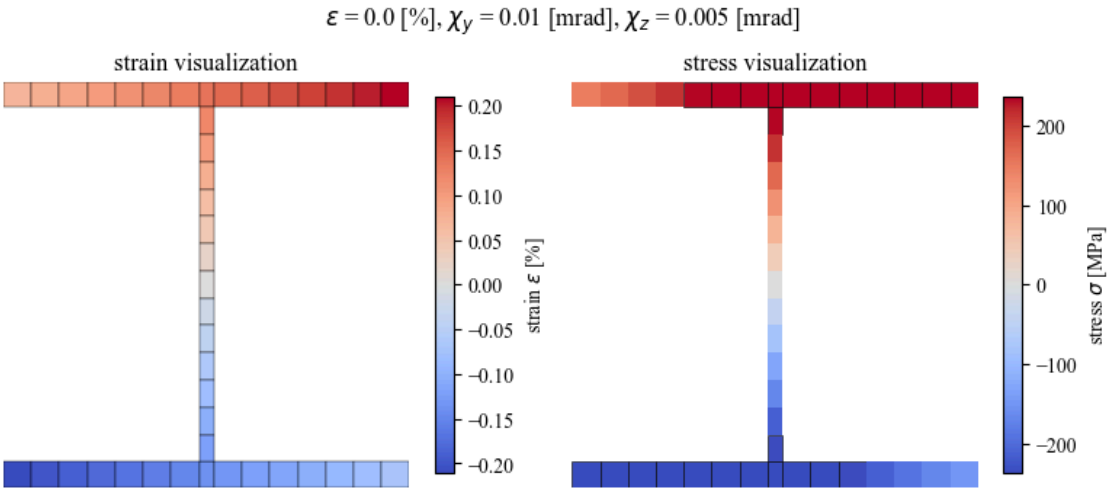
Setup Analysis

In [8]:
```
Analysis = stress_strain_analysis(column)
```

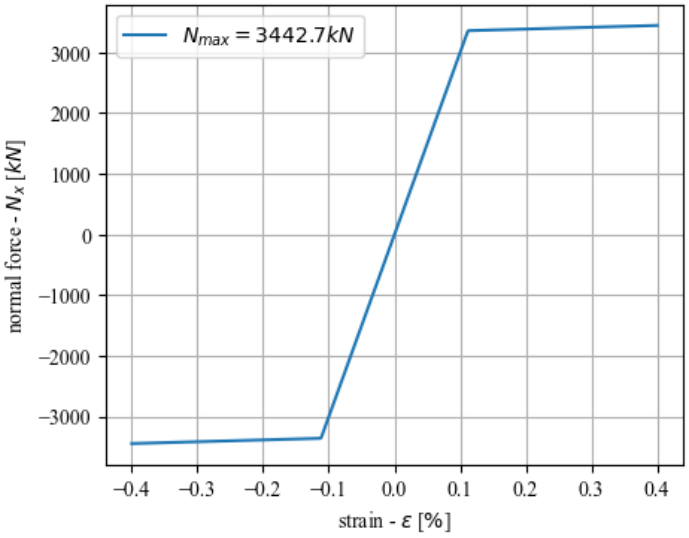
Apply some given strains

In [9]:
```
eps_x, chi_y, chi_z = 0.00, 0.00001, 0.000005

N, My, Mz = Analysis.get_section_forces(eps_x, chi_y, chi_z)

print("N  = {:.2f} kN".format(N))
print("My = {:.2f} kNm".format(My))
print("Mz = {:.2f} kNm".format(Mz))

plot_stress_strain_steel(Analysis)
```

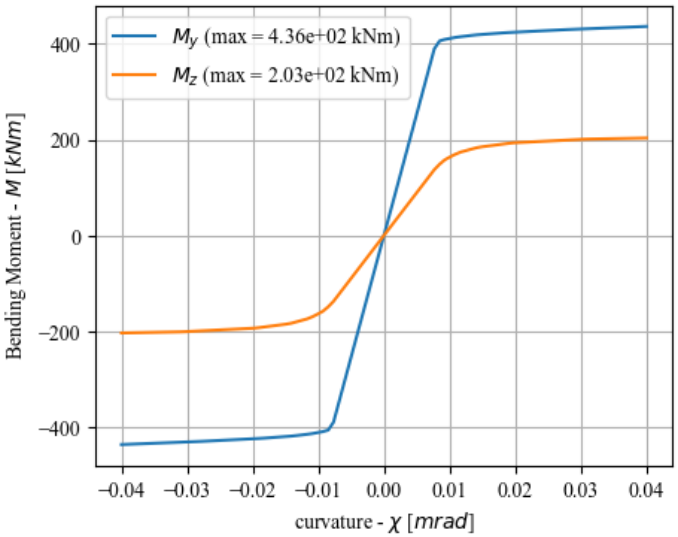
```
N  = 0.00 kN
My = 387.16 kNm
Mz = 20.90 kNm
```



In [10]:
```
strains = np.linspace(-0.004, 0.004, 500)
plot_linear_variation_eps(Analysis, strains)
```



In [11]:
```
curvs = np.linspace(-0.00004, 0.00004, 100)
plot_linear_variation_curv(Analysis, curvs)
```

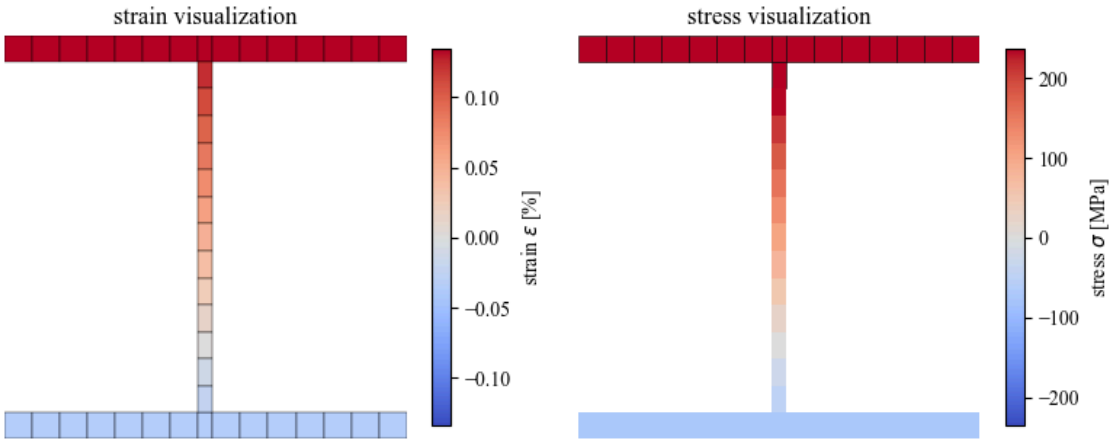


Apply some given section forces

In [12]:
```
from math import log10, floor
def round_to_significant(x):
        return round(x, -int(floor(log10(abs(x/100)))))

N  = 1200 # kN
My = 270 # kNm
Mz = 0   # kNm

eps_x, chi_y, chi_z = Analysis.get_strain_and_curvatures(N, My, Mz)

eps_x, chi_y, chi_z = round_to_significant(eps_x), round_to_significant(chi_y), round_to_significant(chi

print("eps_x = {:.2e}".format(eps_x))
print("chi_y = {:.2e}".format(chi_y))
print("chi_z = {:.2e}".format(chi_z))


Analysis.set_strain_and_curvature(eps_x, chi_y, chi_z)
plot_stress_strain_steel(Analysis)
```

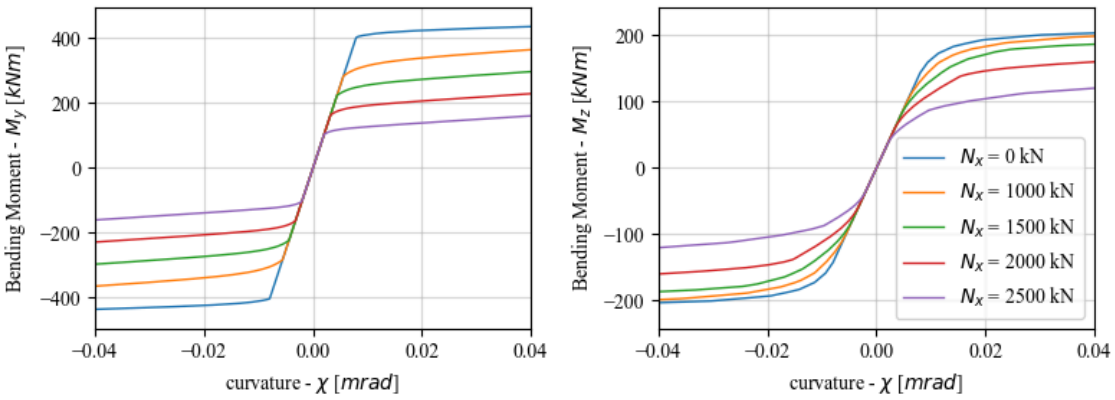
```
eps_x = 4.92e-04
chi_y = 6.08e-06
chi_z = -2.31e-20
```

$\varepsilon = 0.0492$ [%], $\chi_y = 0.00608$ [mrad], $\chi_z = -2.31e\text{-}17$ [mrad]

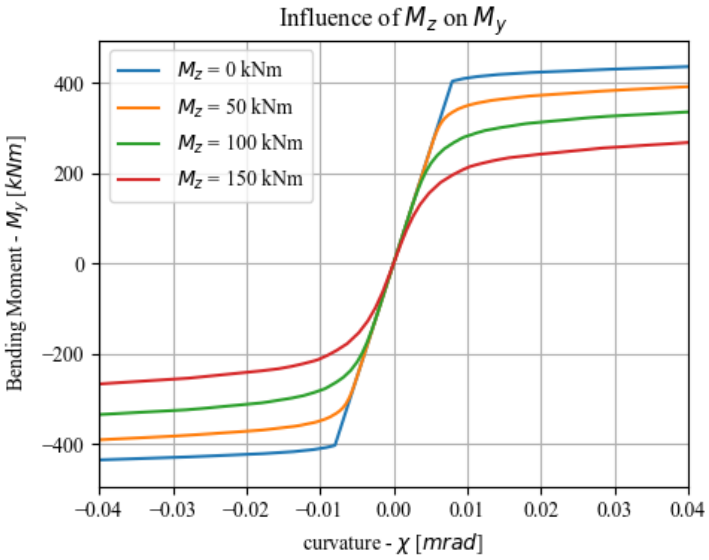


$N - M$ and $M_y - M_z$ influence plots

In [13]:
```
N = [0, 1000, 1500, 2000, 2500]
My_lim = [-450, 450]
Mz_lim = [-220, 220]
plot_influence_of_N_on_M(Analysis, N, My_lim, Mz_lim)
```



In [14]:
```
Mz = [0, 50, 100, 150]
My_lim = [-450, 450]
plot_influence_of_Mz_on_My(Analysis, My_lim, Mz)
```


Influence of M_z on M_y
M_z = 0 kNm
M_z = 50 kNm
M_z = 100 kNm
M_z = 150 kNm
Bending Moment - M_y [kNm]
curvature - χ [$mrad$]
400
200
0
−200
−400
−0.04
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03
0.04

A.5 Cross Sectional Analysis

05 Example: Cross-Sectional Analysis

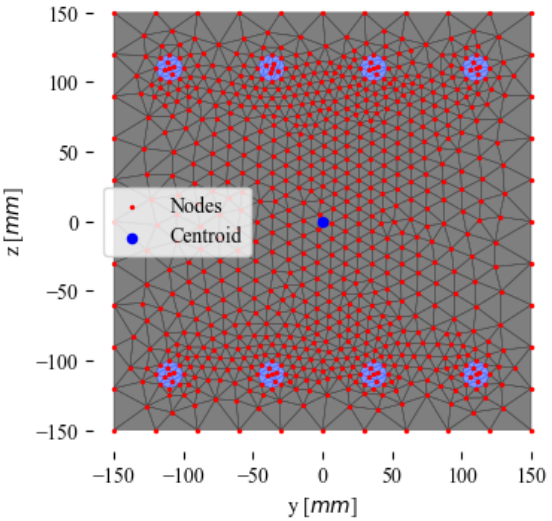
Setup

import all the nedded libraries and files

In [1]:
```
import numpy as np

from Structure import *
from Material import *
from Plotting_Functions import *
```

Generate the Geometry and the corresponding mesh

In [2]:
```
ReinforcedConcrete = ReinforcedConcreteColumn(width  = 300,
                                              height = 300,
                                              concrete_cover = 30,
                                              rebar_diameter = 20,
                                              rebar_spacing  = 50)
mesh = Mesh(ReinforcedConcrete, "triangle", 30)

mesh.plot()
mesh.print()
```



Mesh Type	triangle
Number of elements	1480
Number of nodes	769
Cross Section Area	90000.00

	y	z
Centroid [mm]	0.00	0.00
Moment of inertia [mm^4]	6.75e+08	6.75e+08

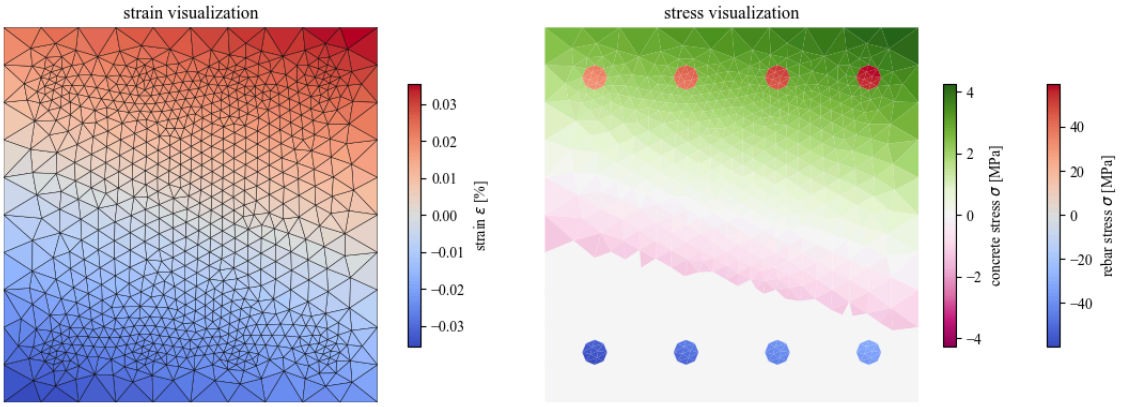
Setup Analysis

In [3]:
```
Analysis = stress_strain_analysis(mesh)
```

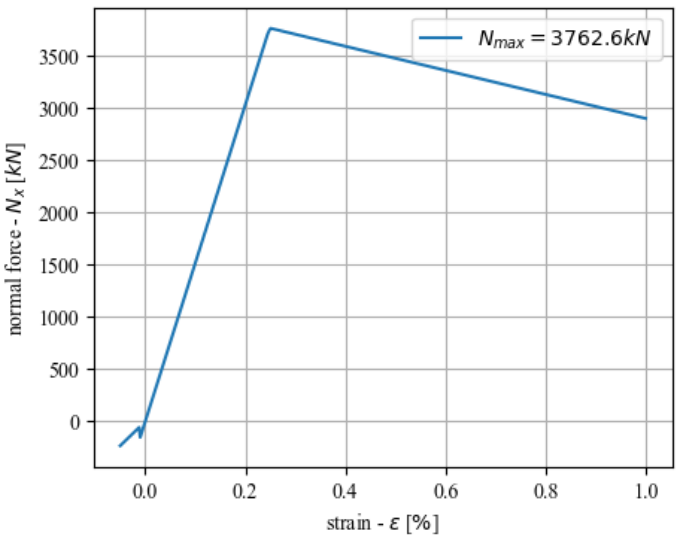
Apply some given strains

In [4]:
```
eps_x, chi_y, chi_z = 0, 0.000002, 0.0000005
# eps_x, chi_y, chi_z = 0, 0.00002, 0.000005

N, My, Mz = Analysis.get_section_forces(eps_x, chi_y, chi_z)

print("N  = {:.2f} kN".format(N))
print("My = {:.2f} kNm".format(My))
print("Mz = {:.2f} kNm".format(Mz))

plot_stress_strain_RC(Analysis)
```

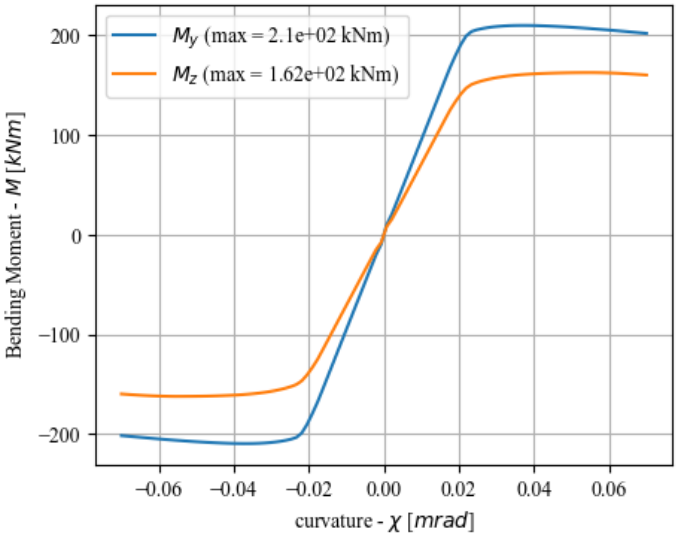
```
N  = 68.78 kN
My = 19.40 kNm
Mz = 3.56 kNm
```



In [5]:
```
strains = np.linspace(-0.0005, 0.01, 500)
plot_linear_variation_eps(Analysis, strains)
```



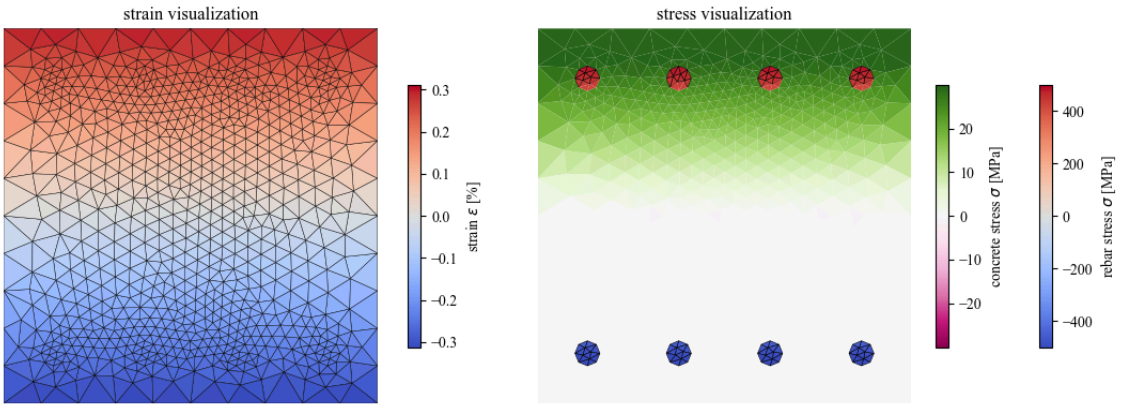
In [6]:
```
curvs = np.linspace(-0.00007, 0.00007, 100)
plot_linear_variation_curv(Analysis, curvs)
```



Apply some given section forces

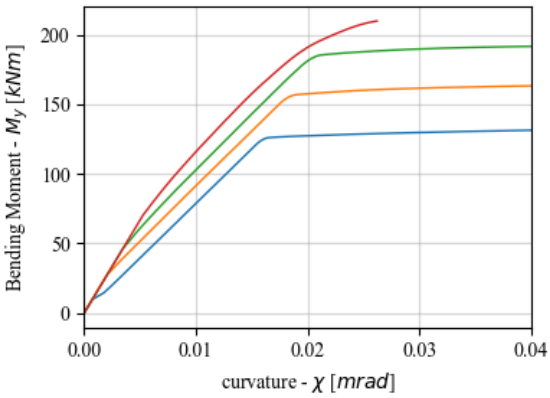
In [7]:
```
from math import log10, floor
def round_to_sig(x):
        return round(x, -int(floor(log10(abs(x/100)))))

N  =  700  # kN
My =  190  # kNm
Mz =  0    # kNm

eps_x, chi_y, chi_z = Analysis.get_strain_and_curvatures(N, My, Mz)

eps_x, chi_y, chi_z = round_to_sig(eps_x), round_to_sig(chi_y), round_to_sig(chi_z)

print("eps_x = {:.2e}".format(eps_x))
print("chi_y = {:.2e}".format(chi_y))
print("chi_z = {:.2e}".format(chi_z))


Analysis.set_strain_and_curvature(eps_x, chi_y, chi_z)
plot_stress_strain_RC(Analysis)
```

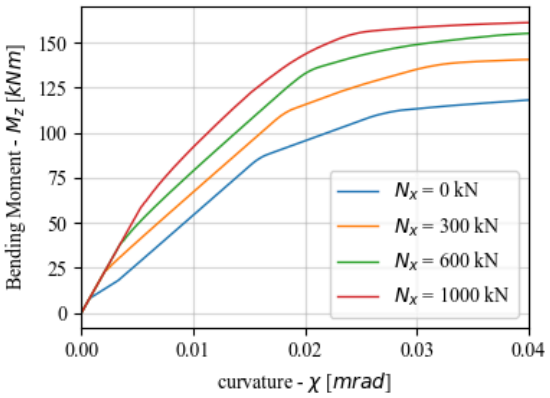
```
eps_x = -8.87e-05
chi_y = 2.08e-05
chi_z = -4.62e-10
```



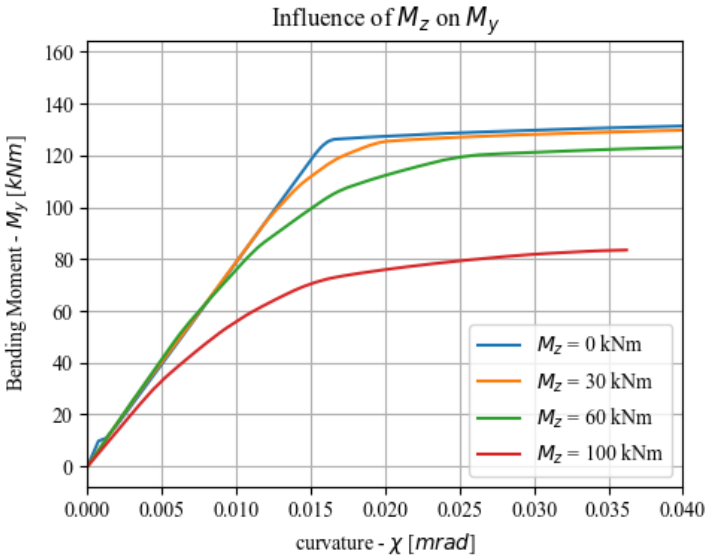
$N - M$ and $M_y - M_z$ influence plots

In [8]:
```
N = [0, 300, 600, 1000]
My_lim = [0, 250]
Mz_lim = [0, 250]
plot_influence_of_N_on_M(Analysis, N, My_lim, Mz_lim, False)
```





In [9]:
```
Mz = [0, 30, 60, 100]
My_lim = [0, 250]
plot_influence_of_Mz_on_My(Analysis, My_lim, Mz, False)
```



B OpenSeesPy Code

B.1 Steel Beam

```python
# Import libraries
from openseespy.opensees import *

# Clean up, initialize 2D structure, 3 DOFs per node
wipe()
model('basic', '-ndm', 2, '-ndf', 3)

# Analysis parameters
numIncrements = 100
dt = 1.0 / numIncrements

# Steel01 material
E  = 210000.0         # Modulus of elasticity
fy = 235.0            # Yield stress
b  = 0.01             # Second-slope stiffness is Ep=E*b
uniaxialMaterial('Steel01', 1, fy, E, b)

# Wide-flange cross-section
h  = 300.0            # Web height
bf = 300.0            # Flange width
tf = 19.0             # Flange thickness
tw = 11.0             # Web thickness
nf = 4                # Number of fibers in the flange
nw = 8                # Number of fibers in the web
section('WFSection2d', 1, 1, h, tw, bf, tf, nw, nf)

# Nodes
nel = 5               # Number of elements along cantilever
L   = 1000.0          # Total length of cantilever
for i in range(nel+1):
    node(i+1, 0, i*L/nel)

# Fixed base
fix(1, 1, 1, 1)
```

```
35
36 # Integration rule
37 beamIntegration('Lobatto', 1, 1, 10)
38
39 # Dealing with the axial force
40 geomTransf('PDelta', 1)
41
42 # Create elements
43 for i in range(nel):
44     element('dispBeamColumn', i+1, i+1, i+2, 1, 1)
45
46 # Lateral load
47 F = 500e3
48 timeSeries("Linear", 2)
49 pattern("Plain", 2, 2)
50 load(nel+1, F, 0.0, 0.0)
51
52 # Analysis setup
53 system("ProfileSPD")
54 numberer("Plain")
55 constraints("Plain")
56 integrator('LoadControl', dt)
57 algorithm("Newton")
58 test('NormDispIncr', 1e-8, 100, 0)
59 analysis("Static")
60
61 # Perform analysis
62 timeArray = [0.0]
63 dispArray = [0.0]
64 for i in range(numIncrements):
65     # Try one increment
66     ok = analyze(1)
67     # if the increment didn't converge
68     if ok != 0:
69         print('\n'"Failed to converge at time %.2f." % getTime())
70     # Store response for plotting
71     timeArray.append(getTime())
72     dispArray.append(nodeDisp(nel+1, 1))
73
74 # Print top displacement response
75 print('\n'"OpenSees top displacement:   %.1fmm" % nodeDisp(nel+1, 1))
```

B.2 Reinforced Concrete Beam

```python
# Import libraries
from openseespy.opensees import *

# Clean up, initialize 2D structure, 3 DOFs per node
wipe()
model('basic', '-ndm', 2, '-ndf', 3)

# Analysis parameters
numIncrements = 100
dt = 1.0 / numIncrements

# Rebar material
E  = 205000.0          # Modulus of elasticity
fy = 500.0             # Yield stress
b  = 0.004             # Second-slope stiffness is Ep=E*b
uniaxialMaterial('Steel01', 2, fy, E, b)

# Concrete material
fpc   = -30.0
epsc0 = - 0.0025
fpcu  =   0.0
epsU  = - 0.025
uniaxialMaterial('Concrete01', 1, fpc, epsc0, fpcu, epsU)

# RC cross-section
d       = 300.0        # height
b       = 300.0        # width
cover   = 46.9         # concrete cover
Atop    = 30.0*30.0    # reinforcement area top
Abot    = 30.0*30.0    # reinforcement area bottom
Aside   = 0            # reinforcement area side
Nfcore  = 30           # Number of fibers in the core
Nfcover = 30           # Number of fibres in the cover
Nfs     = 2            # Number of reinforcing bars in the top layer
section('RCSection2d', 1, 1, 1, 2,
        d,      b,       cover,
        Atop,   Abot,    Aside,
        Nfcore, Nfcover, Nfs  )

# Nodes
nel = 10               # Number of elements along cantilever
L   = 2000.0           # Total length of cantilever
for i in range(nel+1):
```

```
    node(i+1, 0, i*L/nel)

# Fixed base
fix(1, 1, 1, 1)

# Integration rule
beamIntegration('Lobatto', 1, 1, 5)

# Dealing with the axial force
geomTransf('Linear', 1)

# Create elements
for i in range(nel):
    element('dispBeamColumn', i+1, i+1, i+2, 1, 1)

# Lateral load
F = 110e3
timeSeries("Linear", 2)
pattern("Plain", 2, 2)
load(nel+1, F, 0.0, 0.0)

# Analysis setup
system("ProfileSPD")
numberer("Plain")
constraints("Plain")
integrator('LoadControl', dt)
algorithm("Newton")
test('NormDispIncr', 1e-8, 100, 0)
analysis("Static")

# Perform analysis
timeArray = [0.0]
dispArray = [0.0]
for i in range(numIncrements):
    # Try one increment
    ok = analyze(1)
    # if the increment didn't converge
    if ok != 0:
        print('\n'"Failed to converge at time %.2f." % getTime())
    # Store response
    timeArray.append(getTime())
    dispArray.append(nodeDisp(nel+1, 1))

# Print top displacement response
print('\n'"OpenSees top displacement:   %.1fmm" % nodeDisp(nel+1, 1))
```

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. **In consultation with the supervisor**, one of the following two options must be selected:

☒ I hereby declare that I authored the work in question independently, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

☐ I hereby declare that I authored the work in question independently. In doing so I only used the authorised aids, which included suggestions from the supervisor regarding language and content and generative artificial intelligence technologies. The use of the latter and the respective source declarations proceeded in consultation with the supervisor.

Title of paper or thesis:

Towards Implementing and Dimensioning of Fiber Elements in Educational Software

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Berner	Fabio

With my signature I confirm the following:
- I have adhered to the rules set out in the Citation Guidelines.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**
Oensingen, 30.06.2016	

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] For further information please consult the ETH Zurich websites, e.g. https://ethz.ch/en/the-eth-zurich/education/ai-in-education.html and https://library.ethz.ch/en/researching-and-publishing/scientific-writing-at-eth-zurich.html (subject to change).