TME245 - Finite Element Method - Structures
Department - Industrial and Materials science
Mobility Engineering, M.Sc.

Computer Assignment 2

Pietro Nardon

2024-02-29

Contents

1 Task 1: Linear elastic plate analysis using Matlab

The aim of the first task is to analyse the behaviour of an Epoxy plate subjected to an external even pressure To perform the analysis both the Kirchhoff and Mindlin plate element type were used, highlighting the differences between the two theories. A mesh with 20 elements on the horizontal axis and 30 on the vertical axis was chosen since it was the best compromise between speed of computation and quality of the results, see (Fig. 2a).

1.1 Element load vector

1.1.1 Derivation of the expression

To derive the expression of the element load vector resulting from an evenly distributed vertical pressure load p, the Principle of Virtual Work was used, focusing in particular on the external part. The pressure load is a distributed load per unit area q, resulting into:

$$\delta W^{ext} = \int_V \delta u_i f_i \, dV + \int_S \delta u_i t_i \, dS = \int_A \delta w q \, dA \tag{1}$$

From (Eq. 1) it is possible to extract the vector of the external forces acting on the plate, which in this case will only be along the vertical direction. Substituting q with p:

$$f_w^{ext} = \int_A p \, dA \tag{2}$$

Introducing the Finite Element approximation in the case of a Mindlin plate element, (Eq. 2) becomes:

$$\underline{f}_w^{e,ext} = \int_A \underline{N}_w^{eT} p \, dA \tag{3}$$

Then in order to compute the force applied on every single element a Gauss numerical integration with 4 points is used, resulting into the final formula found also in (Code 3.1):

$$\underline{f}_w^{e,ext} \approx \sum_{i=1}^{4} \underline{N}_w^{eT}(\xi_i) \; p \; det[\underline{F}^{Isop}] \; H_i \tag{4}$$

Where $\underline{F}^{Isop}$ it is the Jacobian of the spatial coordinates over the isoparametric coordinates and $\underline{N}_w^e(\xi_i)$ is the collection of the shape functions evaluated in the different integration points. The latter for a Mindlin plate element is:

$$\underline{N}_w^e(x,y) = [N_w^1(x,y) \; N_w^2(x,y) \; N_w^3(x,y) \; N_w^4(x,y)] \tag{5}$$

1.1.2 Implementation of the expression into a Matlab function

In order to implement the expression in Matlab the function `f_el_4noded` (Code 3.1.1) was created. This function loops over the 4 Gauss integration points of the plate element with the respective weights. For every iterations, it extracts the determinant of the $\underline{F}^{Isop}$ matrix and the shape functions vector $\underline{N}_w^e(\xi_i)$ thanks to the function `Vert_Be_quad_func` (Code 3.1.2) and then, through the numerical integration, it assembles the element load vector. The main script for Task (1.1) (Code 3.1) starts with a part where the necessary data are loaded, subsequently the mesh with Mindlin plate element is created and plotted, lastly it is checked that the sum of all load contributions is equal to the total force generated by the distributed pressure over the element area:

$$\underline{f}_{pressure} = pA = pWL = -60000N \tag{6}$$

1.2 Kirchhoff plate maximum out-of-plane displacement

To compute the maximum out-of-plane displacement of the plate by using Kirchhoff plate elements, the element bending stiffness matrix $K_{ww}^{K,e}$ and the vector of external forces $\underline{f}_w^{ext}$ had to be created. Starting from the function `f_el_4noded` (Code 3.1.1) computed before, and expanding it with the calculations to compute $K_{ww}^{K,e}$ the new function `Kirch_K_f_4noded` (Code 3.2.1) was created. This new function takes as inputs the element coordinates, the pressure load p and the $\underline{\tilde{D}}$ matrix, then again loops along the same integration points and then computes the curvature matrix $\overset{*}{\underline{B}}$ thanks to the function given in the lecture notes. The bending stiffness matrix is then computed as follows:

$$\underline{K}_{ww}^{K,e} \approx \sum_{i=1}^{4} \overset{*}{\underline{B}}^{eT} \underline{\tilde{D}} \overset{*}{\underline{B}}^{e} \, det[\underline{F}^{Isop}] \, H_i \tag{7}$$

1.2.1 Boundary conditions

In order to solve the system of equations, and find the displacements, the necessary boundary conditions must be implemented in the code. Since the plate is simply supported on the edges, for a Kirchhoff plate it is necessary to just constrain the out-of-plane degree of freedom and the rotational degree of freedom along an in-plane axis perpendicular to the edge. In the first place all the nodes that compose the boundary must be extracted by selecting the nodes whose coordinates are equal to either 0, L or W, depending on which edge is being searched for. Then, for all the selected nodes, the out-of-plane degree of freedom w is put equal to 0, in addition to this also the rotational degrees of freedom θ_x for the `Top` and `Bottom` edges (the shorter edges in (Fig.1)) and θ_y for the `Left` and `Right` edges (the longer edges in (Fig.1)) are also put equal to 0. The boundary conditions are shown in (Fig.1), where the red arrows correspond to the constrained degrees of freedom.

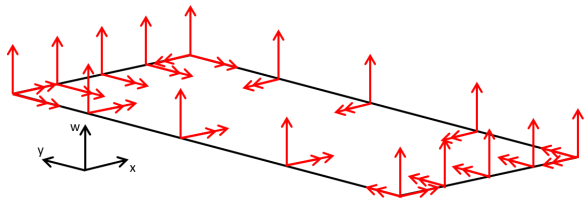


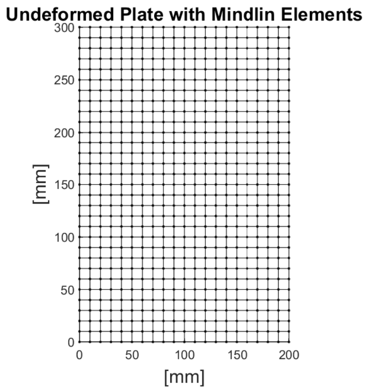
Figure 1: Boundary Conditions of the plate

1.2.2 Maximum out-of-plane displacement

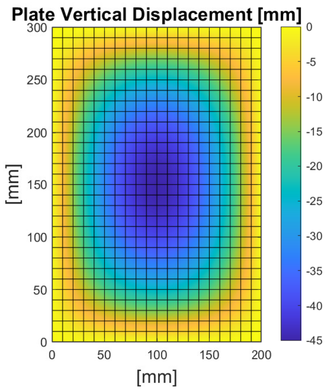
After solving the system of equation shown in (Eq. 8), it is possible to find the displacement vector of the free degrees of freedom $\underline{a}_F$ and assemble it together with the displacement vector of the constrained degrees of freedom $\underline{a}_C$, creating the displacement vector $\underline{a}$.

$$\begin{cases} \underline{a}_F = \underline{K}_{FF}^{-1}(\underline{f}_F - \underline{K}_{FC}\,\underline{a}_C) \\ \underline{f}_C = \underline{K}_{CF}\,\underline{a}_F + \underline{K}_{CC}\,\underline{a}_C \end{cases} \tag{8}$$

Subsequently after selecting only the out-of-plane degrees of freedom the maximum out-of-plane displacement is found and the contour plot that shows the out-of-plane displacement variation along the plate is created, as can be seen in (Code 3.2) and (Fig. 2b).



(a) Initial undeformed mesh



(b) Vertical Displacement for a Kirchhoff plate

Figure 2: Kirchhoff plate

The out-of-plane displacement is negative since p is opposite to the out-of-plane displacement w and the maximum displacement region is located in the center of the plate since is the furthermost place from all the boundary conditions, its value is:

$$\underline{a}_w^{max} = -45.0559\,mm$$

1.3 Mindlin plate element with out-of-plane loading

In order to implement a procedure to compute the element stiffness matrix for a Mindlin plate element with only out-of-plane load a new function, called `Kel_fel_Mindlin_func` (Code 3.3.1) was created, The functions receives as inputs the element coordinates, the constitutive matrices and the pressure load p, then it loops over 4 or 1 Gauss integration points depending on which matrix is being computed. The function was tested against the test cases available on Canvas, as can be seen in (Code 3.3). All the necessary gradient matrices $\underline{B}$ and shape functions matrices $\underline{N}$ were computed through the symbolic functions generated by the code `Creator_N_Mindlin_func` (Code 3.3.2). After the needed computations, the element stiffness matrix and load vector are assembled and their elements reordered.

Assuming the material of the plate to be isotropic the following constitute relations can be computed as shown in (Eq. 9) and are constant along the thickness, in particular, $\underline{G}^0$ is a corrected value of the plate's shear stiffness coming from Mindlin's kinematic assumptions in order to the respect the considerations of the equilibrium along the plate.

$$\underline{D}^0 = h\underline{D} \quad ; \quad \underline{\tilde{D}} = \frac{h^3}{12}\underline{D} \quad ; \quad \underline{G}^0 = \frac{5}{6}h\underline{G} \tag{9}$$

1.3.1 Integration over 4 Gauss points

Thanks to the 4 point integration with the standard weights and coordinates, it was possible to compute the following quantities for each element:

- In-plane stiffness matrix: $\underline{K}_{uu}^e \approx \sum_{i=1}^4 \underline{B}_u^{eT}\, \underline{D}^0\, \underline{B}_u^e\; det[\underline{F}^{Isop}]\, H_i$
- Bending rotational stiffness matrix: $\underline{K}_{\theta\theta}^{B,e} \approx \sum_{i=1}^4 \underline{B}_\theta^{eT}\, \underline{\tilde{D}}\, \underline{B}_\theta^e\; det[\underline{F}^{Isop}]\, H_i$
- External vertical force: $\underline{f}_w^{ext} \approx \sum_{i=1}^4 \underline{N}_w^{eT}\; p\; det[\underline{F}^{Isop}]\, H_i$

1.3.2 Integration over 1 Gauss point

The integration over just 1 Gauss point is used in order to avoid the phenomenon of shear locking. Since the Mindlin plate element is considered a pure bending problem, the shear strain must be $\gamma_{xy} = 0$ all over the plate. The quadrilateral element implemented in the Mindlin theory does a linear approximation of the out-of-plane displacements and of the rotations between all the nodes. Since in a plate subjected to bending, the rotations will not be equal for every node, this means that the plate will not have $\gamma_{xy} = 0$ everywhere: in particular, $\gamma_{xy} \neq 0$ at the nodes where the numerical integration takes place, this leads to numerical errors. In order to avoid these errors the integration is carried only in the centre of the plate, thus making a selectively reduced integration where instead for sure $\gamma_{xy} = 0$. The integration is done for each stiffness matrix that is linked to γ_{xy}, so the following quantities are computed:

– Out-of-plane displacement stiffness matrix: $\underline{K}^e_{ww} \approx \underline{B}^{eT}_w\, \underline{G}^0\, \underline{B}^e_w\, det[\underline{F}^{Isop}]\, H$

– Out-of-plane displacement-rotation stiffness matrix: $\underline{K}^e_{\theta w} \approx -[\underline{B}^{eT}_w\, \tilde{\underline{D}}\, \underline{N}^e_\theta\, det[\underline{F}^{Isop}]\, H]^T$

– Shear rotational stiffness matrix: $\underline{K}^{S,e}_{\theta\theta} \approx \underline{N}^{eT}_\theta\, \underline{G}^0\, \underline{N}^{eT}_\theta\, det[\underline{F}^{Isop}]\, H$

1.4 Comparison between Kirchhoff and Mindlin plate elements

A new scrip was created where: for the part concerning the Kirchhoff plate the same code as in (Code 3.2) was used, together with a code adapted to utilize the Mindlin plate, in order to compare the two theories (Code 3.4). The same nodes were constrained but since this mesh has five degrees of freedom then also the u_x and u_y had to be constrained to a null value. The same code structure of (Code 3.2) was used, with the necessary changes to accommodate the function `Kel_fel_Mindlin_func` (Code 3.3.1), such as constraining the in-plane degrees of freedom and computing the matrices shown in (Eq. 9). A vector `h_vect` sweeping from the minimum to the maximum thickness was created and then it was used in a loop to compute the required $w_{max}h^3$ quantity. The results were plotted in (Fig. 3).

It is clear that the behaviour of $w_{max}h^3$ is different between Kirchhoff and Mindlin plate element: in the first is constant at a value of $-45096.2\,mm^4$, in the second is diminishing. The plots of $w_{max}h^3$ for the two different theories intersect at $h \approx 5.62\,mm$ and from a thickness value of $h \approx 13\,mm$, $w_{max}h^3$ start to have a difference in value greater then $0.06 \cdot 10^4\,mm^4$ or 1.33%. The Mindlin plate theory is more precise in the description of thick plates, since transverse shear deformations are taken into account and also the rotation of the cross-section is described trough the two independent variables θ_x and θ_y, this is why $w_{max}h^3$ for a Mindlin plate varies for an increasing thickness h. The Kirchhoff plate instead in not suited to describe thick plates due to its assumption implying that the end surfaces of the plate will always remain perpendicular to the mid-line, something that for thicker plates under bending is not true. The constant behaviour of $w_{max}h^3$ in the Kirchhoff plate element can be explained making a comparison to the Eulero-Bernoulli beam theory, which keeps the same assumption of the Kirchhoff theory. The maximum displacement for a simply supported beam with rectangular section, subject to a evenly distributed load can be computed and, after rearranging the equation, it is clear to see that the $w_{max}h^3$ quantity is not dependent on the thickness, see (Eq. 10) and (Fig. 4).

$$w_{max} = \frac{5}{384}\frac{pL^4}{E\,\frac{bh^3}{12}} \quad \rightarrow \quad w_{max}h^3 = \frac{5}{384}\frac{pL^4}{E\,\frac{b}{12}} \tag{10}$$

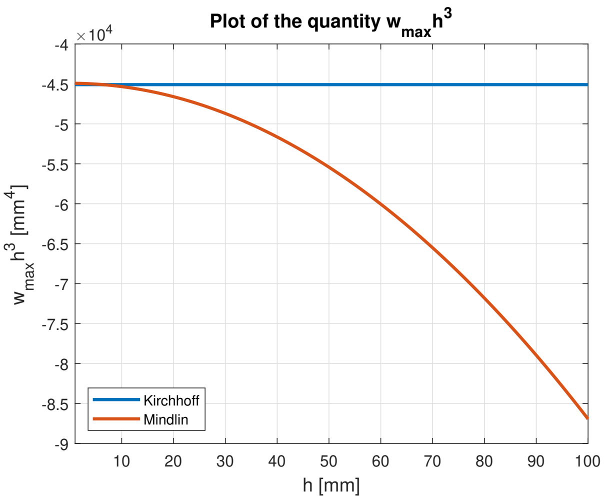


Figure 3: Comparison of the $w_{max}h^3$ quantity for different plate type

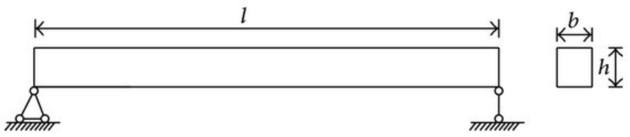


Figure 4: Simply supported (roller+hinge) Eulero-Bernoulli beam

1.5 In-plane and out-of-plane stress components of the Mindlin plate element

1.5.1 Stresses_Mindlin_func

To compute the in-plane and the out-of-plane stresses of a Mindlin plate element, the function `Stresses_-Mindlin_func` (Code 3.5.1) was created. The function receives as input the element coordinates, the element displacements, the thickness coordinate at which the stresses have to be computed and then the material's properties matrices. The element's stress components are computed assuming an elastic, linear and isotropic material with plane stress assumption. After extracting and dividing the displacements $\underline{a}$ into $\underline{a}_u^e$, $\underline{a}_w^e$ and $\underline{a}_\theta^e$, the function computes the $\underline{B}_u^e$, $\underline{B}_\theta^e$ and $\underline{B}_w^e$, together with the shape function matrix $\underline{N}_w^e$ matrices through a 4 point Gauss integration loop over the parent domain. These matrices are used to compute the stresses through (Eq. 11) and (Eq. 12), where σ_{33} is not present since a plane stress assumption is made.

$$\underline{\sigma}_{el} = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} = \underline{D}(\underline{B}_u^e a_u^e - z\underline{B}_\theta^e \underline{a}_\theta^e) \tag{11}$$

$$\underline{\tau}_{el} = \begin{bmatrix} \sigma_{13} & \sigma_{23} \end{bmatrix} = \underline{G}(\underline{B}_w^e \underline{a}_w^e - \underline{N}_\theta^e \underline{a}_\theta^e) \tag{12}$$

1.5.2 Von Mises contour plots

After computing the plate's displacements with Mindlin plate elements (as described in (1.3), (1.4) and (Code 3.4)), the function `Stresses_Mindlin_func` (Code 3.5.1) was used, inside a loop over the required thickness coordinates z, in order to find the stresses in every node of every element, and then average them to have only one stress for each element. The plate's thickness chosen for the computations is $h = 10\,mm$, to have comparable results with the precedent points. After this calculations, the Von Mises stress was computed as described in (Eq. 13), and then stored for each element.

$$\underline{s} = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{23} \end{bmatrix} - \frac{(\sigma_{11} + \sigma_{22} + \sigma_{33})}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \rightarrow \quad \sigma_{VM} = \sqrt{\frac{3}{2}(\underline{s}^T \underline{s} + \sigma_{12}^2 + \sigma_{13}^2 + \sigma_{23}^2)} \tag{13}$$

The contour plots of the Von Mises stresses evaluated at different thickness coordinates are shown in (Fig. 5). The two plots at $z = -\frac{h}{2}$ and $z = \frac{h}{2}$ are identical. This is because the shear stresses (Eq. 12) are constant for different z coordinate, the quantities that appear together with the z coordinate into (Eq. 11) are constant along the thickness and $\underline{a}_u^e$ is always 0 due to our assumptions and boundary conditions, which do not allow any in-plane movement. This means two plots taken at the same distance from the middle line, with these assumptions, will be equal. Moreover in these two plot a concentration of the stresses at the center of the plate is visible, since it is the area that undergoes the maximum out-of-plane displacement due to the applied pressure. Moreover stresses are also concentrated at the cornering nodes, since these are the nodes connecting the two perpendicular edges of the plate, thus with two rotations constrained, resulting into a clamp boundary condition. While when $z = 0$ only the shear components are present since, looking at (Eq. 13), keeping in mind that $\underline{a}_u^e = 0$, then $\underline{\sigma}_{el} = \underline{0}$ for each element. The shear stresses will be very small around the corners due to the clamp boundary condition that impose all displacements $\underline{a}^e$ to be null. Shear stresses are small in the center as well since the $\underline{B}_w^e \underline{a}_w^e$ and $\underline{N}_\theta^e \underline{a}_\theta^e$ will balance each other. At the same time shear stresses will be maximum in the regions along the edges where they are driven by the rotations.

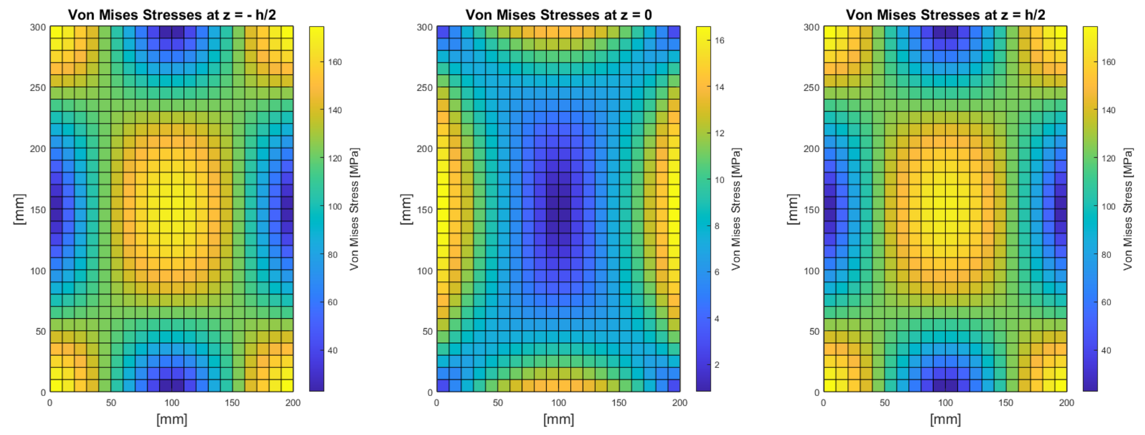


Figure 5: Von Mises contour plots at different z coordinate

2 Task 2: Pre-buckling analysis of a composite specimen

In this task a linerised pre-buckling analysis of a given composite specimen was carried on in order to define its safety factor against buclking. A mesh with 50 elements on the horizontal axis and 5 on the vertical axis was chosen since it was the best compromise between speed of computation and quality of the results.

2.1 Linearised pre-buckling analysis

Expanding the code snippets provided in the lecture notes the function `Kirch_Quad_Routine` (Code 4.1.1) was created, with the aim to use it for a linearised pre-buckling analysis of the given specimen. This function gives as outputs the $\underline{K}^e_{uu}$, $\underline{K}^{K,e}_{ww}$ and $\underline{G}^{e,(R)}$ matrices and the $\underline{f}^{e,ext}_{w}$ vector. The $\underline{K}^e_{uu}$, $\underline{K}^{K,e}_{ww}$ matrices and the $\underline{f}^{e,ext}_{w}$ vector are computed through a 4 points integration cycle after computing, in the same, cycle the necessary quantities to assemble them, such as $\underline{B}^e_u$, $\overset{*}{\underline{B}}$ and $det[\underline{F}^{Isop}]$.

They are computed as follows:

- In-plane stiffness matrix: $\underline{K}^e_{uu} \approx \sum_{i=1}^{4} \underline{B}^{eT}_u\, \underline{D}^0\, \underline{B}^e_u\, det[\underline{F}^{Isop}]\, H_i$
- Out-of-plane displacement stiffness matrix: $\underline{K}^{K,e}_{ww} \approx \sum_{i=1}^{4} \overset{*}{\underline{B}}{}^{eT}\, \underline{\tilde{D}}\, \overset{*}{\underline{B}}{}^{e}\, det[\underline{F}^{Isop}]\, H_i$
- External vertical force: $\underline{f}^{ext,e}_{w} \approx \sum_{i=1}^{4} \underline{N}^{eT}_w\, p\, det[\underline{F}^{Isop}]\, H_i$

The geometric stiffness matrix $\underline{G}^{e,(R)}$, that appears due to the effect of the in-plane section forces, is instead computed through a 9 points integration cycle. This is because the out-of-plane deflection is approximeted as a third grade polynomial, thus the Kirchhoff gradient matrix $\underline{B}^e$ is of second order, as a consequence the 4 point Gauss integration is not enough and 9 points are required. The computation is as follows, the matrix $\underline{\tilde{N}}^{sec(R)}$ is assumed constant over the element.

- Geomtrical stiffness matrix: $\underline{G}^{e,(R)} \approx \sum_{i=1}^{9} \underline{B}^{eT}\, \underline{\tilde{N}}^{sec(R)}\, \underline{B}^{e}\, det[\underline{F}^{Isop}]\, H_i$

2.2 FE program for linearised pre-buckling analysis

The entire FE program is found in (Code 4.2) while the loaded data can be found in (Code 4.4). The geometry, the boundary conditions and the imposed displacements of the specimen can be seen in (Fig. 6).

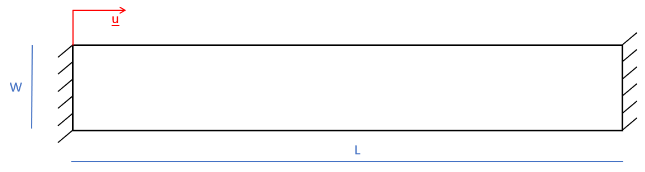


Figure 6: Initial geometry, boundary conditions and prescribed displacement

2.2.1 Solution of the in-plane deformation problem with the in-plane load

To solve the in-plane deformation problem a mesh with in-plane degrees of freedom had to be generated. After extracting the nodes at the left and right edge of the specimen both the u_x and u_y degrees of freedom of these nodes were imposed null in order to implement the clamp boundary condition. Subsequently the imposed compressive horizontal displacement $\overline{u}$ was applied to the left nodes, where $\overline{u}$ is the displacement corresponding to the compressive strain of $\varepsilon = 1,13\%$ as shown in (Eq. 14).

$$\varepsilon = \frac{\Delta L}{L} \quad \rightarrow \quad \overline{u} = \Delta L = \varepsilon L = 0.565\, mm \tag{14}$$

The in-plane stiffness matrix $\underline{K}_{uu}$ was assembled with the in-plane element stiffness matrices $\underline{K}^{e}_{uu}$ computed with the function `Kirch_Quad_Routine` (Code 4.1.1) (which as $\underline{\tilde{N}}^{esec(R)}$ input received a null matrix) together with the element force vector $\underline{f}^{ext,e}_{uu}$ (which is a vector full of zeros since there is no force applied). After assembling $\underline{K}_{uu}$ and $\underline{f}^{ext}_{uu}$ the system of equation (Eq. 15) is solved for the in-plane displacements $\underline{a}_{uu}$ that are then plotted over the undeformed mesh in (Fig. 7) and also after averaging per each element, as a contour plot in (Fig. 8).

$$\begin{cases} \underline{a}_{uuF} = \underline{K}^{-1}_{uuFF}(\underline{f}_{uuF} - \underline{K}_{uuFC}\,\underline{a}_{uuC}) \\ \underline{f}_{uuC} = \underline{K}_{uuCF}\,\underline{a}_{uuF} + \underline{K}_{uuCC}\,\underline{a}_{uuC} \end{cases} \tag{15}$$

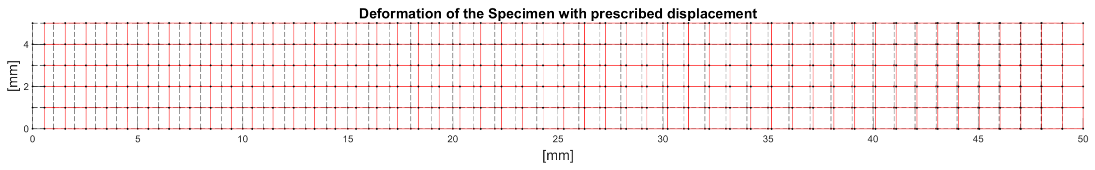


Figure 7: Deformed plate (in red) due to the imposed displacement

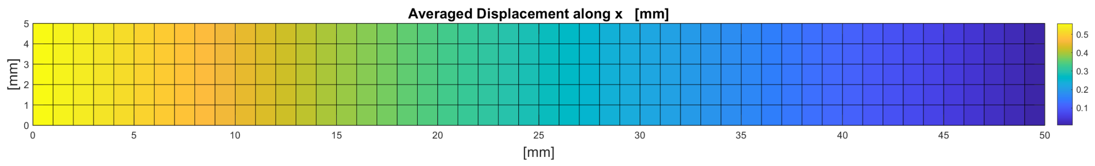


Figure 8: Averaged displacement of the loaded plate

2.2.2 Spatial distribution of the stress

In order to compute the in-plane stress components in the element midpoint, the function `stress_-inplane` was created (Code 4.2.1). This function uses as inputs: the element coordinates, the element in-plane displacements and the constitutive matrix. Since the stress is computed in the element mid-point at $z = 0$ the parent domain nodal coordinates will be $\xi = [0,\ 0]^T$ and the stress will be computed as in (Eq. 16).

$$\underline{\sigma}_{el} = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} = \underline{D}\,\underline{B}^{e}_{u}\,\underline{a}^{e}_{u} \tag{16}$$

Then, the in-plane loading that may cause buckling is approximated as in (Eq. 17)

$$\underline{\tilde{N}}^{e,sec(R)} \approx h \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} \tag{17}$$

After storing the values of $\underline{\tilde{N}}^{e,sec(R)}$ for each element, the stress component in the direction of the compressive load $\underline{\sigma}_{xx}$ was plotted in (Fig. 9). As can be seen in the plot, the higher compressive loads are reached in correspondence of the cornering nodes: this is where the stress will be concentrated, since they are where the clamps are located. In (Fig. 10) the elements that exceed the compressive strength of the specimen $\sigma_{compr} = 1695MPa$ are plotted in yellow. The specimen, under the imposed horizontal displacement $\bar{u}$, will crack and fail first in the regions highlighted by the elements.

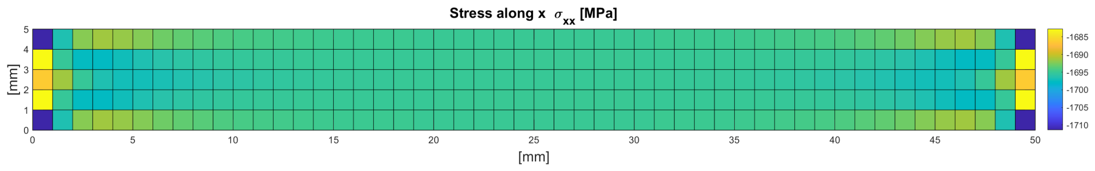


Figure 9: Plot of the stress component in the direction of the compressive load

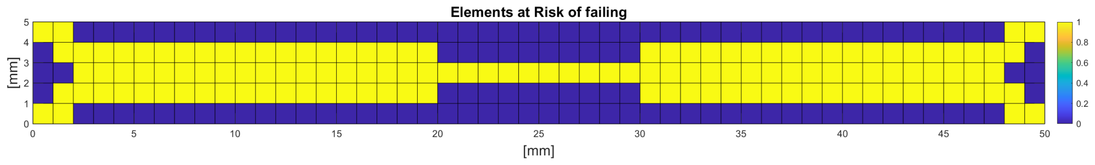


Figure 10: Plot of the areas at that of risk buckling (in yellow)

2.2.3 Vertical and geometrical stiffness matrix assembling

To assemble the vertical and geometrical stiffness matrix $\underline{K}_{ww}^{e,K}$ and $\underline{G}^{e(R)}$ into $\underline{K}_{ww}^{K}$ and $\underline{G}^{(R)}$ the already defined function `Kirch_Quad_Routine` (Code 4.1.1) was used together with the `assem` CALFEM function. For these computations the mesh had to be recreated,in order to have as degrees of freedom for each node w, θ_x and θ_y, and the boundary conditions were re-formulated in order to implement the clamp constrain. Among the inputs for the function `Kirch_Quad_Routine` (Code 4.1.1) the $\underline{\tilde{N}}^{sec(R)}$ matrix, extracted per each element from the one computed in (2.2.2), was used.

2.2.4 Safety factor and buckling mode shape

The analysis to check if buckling instability appears is done through the resolution of the generalised eigenvalue problem (Eq. 18), where the eigenvectors $\underline{z}_i$ indicate the shapes of the buckling modes (meaning how the plate will deform while experiencing buckling), and the eigenvalues λ_i indicate the critical load levels at which buckling will start appearing. In particular, the most important λ_i is the lowest positive one, "λ_1", since it indicates at which load the first buckling modes appears, thus when buckling starts. Since the in-plane equilibrium was solved applying the design loads, then λ_1 corresponds to the safety factor against buckling.

$$\left[\underline{K}_{wwFF}^{K} + \lambda_i \underline{G}_{FF}^{(R)}\right] \underline{z}_i = 0 \quad \rightarrow \quad \underline{K}_{wwFF}^{K} \underline{z}_i = -\lambda_i \underline{G}_{FF}^{(R)} \underline{z}_i \tag{18}$$

In particular, if λ_1 is multiplied by the applied load used to compute the eigenvalues themselves, then the result is the load for which buckling would appear. Then, only if $\lambda_1 > 1$ buckling does not happen.

In the studied case with the current boundary conditions:

$$\lambda_1 = 0.729 \, ; \, \lambda_2 = 1.4922 \, ; \, \lambda_3 = 2.9213 \, ; \, \lambda_4 = 3.6790 \, ; \, \lambda_5 = 4.42 \, ; \, \lambda_6 = 4.4417 \tag{19}$$

As said before, since $\lambda_1 \leq 1$, buckling will occur before the compressive strength is reached, thus actions should be taken in order to avoid it (these manouvers will be described in (2.3)). In (Fig. 11) are shown the first six buckling modes: these were plotted by extracting only the displacement of the out-of-plane degrees of freedom from the displacement vector.

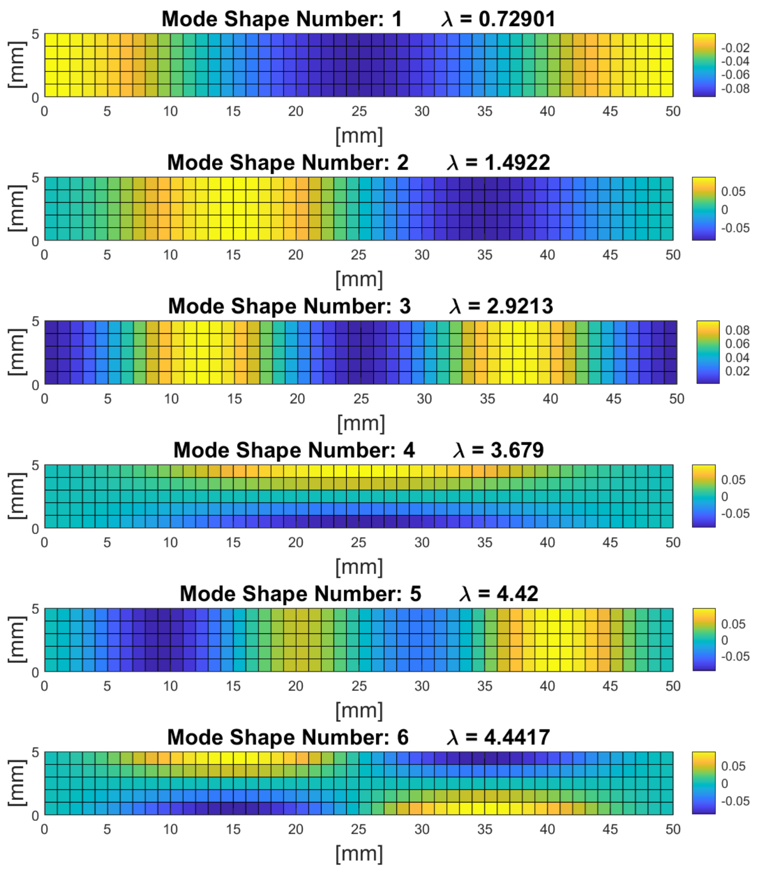


Figure 11: First six buckling mode shapes

2.3 Additional support

A possible solution to avoid buckling in the specimen is to add an additional support in the midspan of the plate, in order to prevent out-of-plane displacement w along the midspan itself. The new boundary conditions are displayed in (Fig. 12), the new additional support is highlighted by a red dashed line.

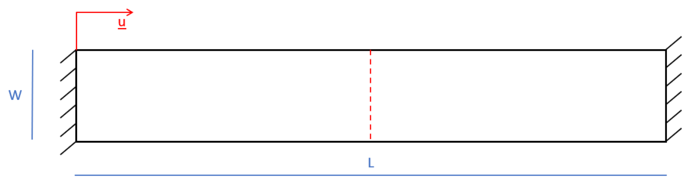


Figure 12: New boundary conditions to prevent buckling

This new support was implemented in (Code 4.3) by just adding a few lines of code to (Code 4.2): also the nodes in the midspan were considered as constrained nodes. In particular, the nodes `Middle` were only constrained on the out-of-plane direction, $w = 0$. As a result of this new boundary condition the λ_i increase their values, as can be seen in (Eq. 20) and, as a consequence of that, also the new safety factor is $\lambda_1 > 1$ meaning that the specimen is now safe from buckling at this load case.

$$\lambda_1 = 1.4922\ ;\ \lambda_2 = 2.9229\ ;\ \lambda_3 = 4.42\ ;\ \lambda_4 = 4.4417\ ;\ \lambda_5 = 5.8691\ ;\ \lambda_6 = 5.9867 \tag{20}$$

It is worth noticing that, in this new configuration, some particular λ_i and buckling modes are the same as the ones in the original boundary conditions configuration. In particular, the buckling modes for the case with the additional support are the ones that in the double-clamped configuration will have a value of 0 at the midspan. This is due to the fact that the material, load case, assumptions, ecc... are all the same between the two different cases, while the only difference is in the addition of this support preventing the out-of-plane displacement of the midspan. This means that all the modes that had a non-null out-of-plane displacement in the midspan will disappear while the ones that already had a null out-of-plane displacement in the midspan will remain. This can be seen in (Fig. 13), where it is possible to see that the buckling modes 2, 3, 5 and 6 of the original case (Fig. 11) are identical to the buckling modes 1, 2, 3, 4 of the plate with the additional support. Moreover the last two modes are also equal to higher modes of the original case, they were not shown for brevity. The color differences are only due to a different normalisation of the modes that have different signs, but the buckling mode are the same.

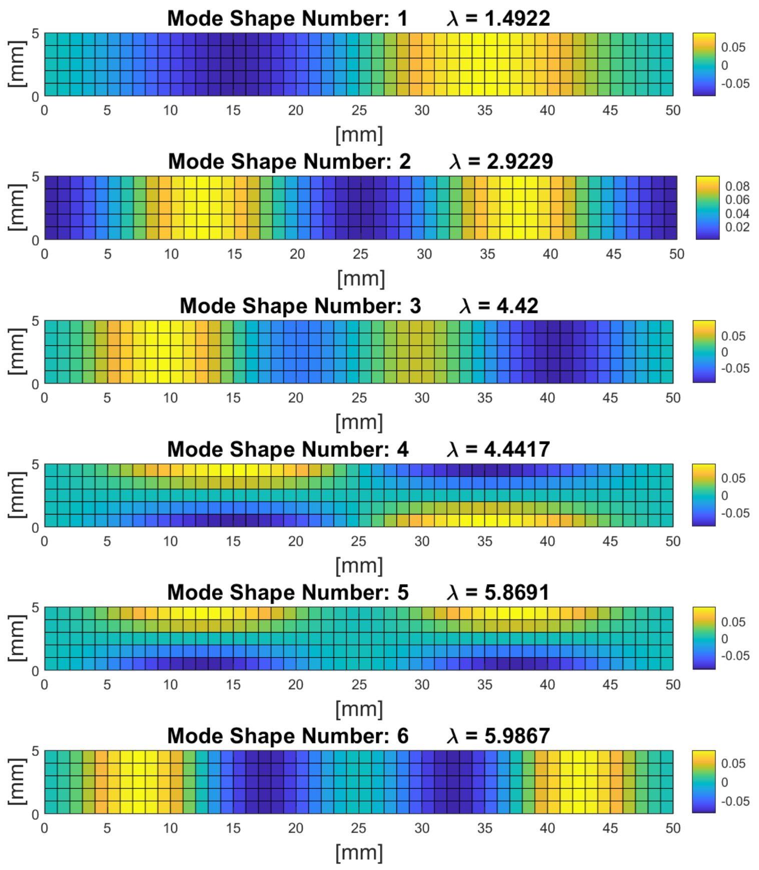


Figure 13: First six buckling mode shapes with additional support

3 Matlab Code Task 1

At the beginning of every script the data useful to solve the problem are always imported through the function DataGen_CA2, that can be found in (Code 3.6).

3.1 Element load vector

```
clc
clear
close all

%% %%%%%%%%%%% Computer Assignment 2 - Task_1 a) %%%%%%%%%%%%%%%

%% Loading Datas
data = DataGen_CA2;

L = data.L;
W = data.W;
p = data.p;


%% Task_1 - a) Element Load Contribution

% Creating the mesh
n_el_x = 20;
n_el_y = 30;
n_type = 5;                % 5 Mindlin
[mesh, Coord, Edof] = rectMesh(0, W, 0, L, n_el_x, n_el_y,n_type);

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

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

% Plot mesh
figure(1)
plotpar = [1 1 0];
eldraw2(Ex,Ey,plotpar); grid on
title('Undeformed Plate', 'FontSize',15); axis tight
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)

% Numbers
n_el = size(mesh, 2);
```

```
n_nodes = size(Coord, 1);

% Loop over the elements to compute the element force due to the
   pressure
f_el_ext_w = zeros(4, n_el);
for el = 1: n_el
    f_el_ext_w(:, el) = f_el_4noded(Ex(el,:), Ey(el, :), p);
end

% Checking the total pressure
F_Load_sum = sum(sum(f_el_ext_w));
F_Load = p*L*W;
if (abs(F_Load - F_Load_sum) < 1e-6)
    fprintf(['\nThe sum of all load contributions is equal to the
       total force and it is: ', num2str(F_Load_sum), ' N\n'])
end
```

3.1.1 f_el_4noded

```
function [f_el_ext] = f_el_4noded(Ex, Ey, p)

% xi-coordinates Vector and Weights
H_vect = ones(1,4);
xi_vect = [-1/sqrt(3) -1/sqrt(3)  1/sqrt(3)  1/sqrt(3);
           -1/sqrt(3)  1/sqrt(3) -1/sqrt(3)  1/sqrt(3)];

% Initializing
f_el_ext = zeros(4,1);

% Loop over integration points
for ii = 1:4
    H = H_vect(ii);
    xi = xi_vect(:, ii);
    [~, det_Fisop, N_w] = Vert_Be_quad_func(xi, [Ex(1);Ey(1)], [Ex
       (2);Ey(2)], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);
    f_el_ext = f_el_ext + N_w'*p*det_Fisop*H;
end
end
```

3.1.2 Vert_Be_quad_func

```
xi = sym('xi',[2,1],'real');
N1 = 0.25*(xi(1) - 1)*(xi(2) - 1); N2 = -0.25*(xi(1) + 1)*(xi(2) -
    1);
N3 = 0.25*(xi(1) +1)*(xi(2) + 1); N4 = -0.25*(xi(1) - 1)*(xi(2) +
   1);
```

```
%define N-matrix for the element
N_w = [N1 N2 N3 N4];
%differentiate shape functions wrt isoparam. coordinates
dN1_dxi=gradient(N1,xi);
dN2_dxi=gradient(N2,xi);
dN3_dxi=gradient(N3,xi);
dN4_dxi=gradient(N4,xi);
%introduce node positions
xe1 = sym('xe1',[2,1],'real');
xe2 = sym('xe2',[2,1],'real');
xe3 = sym('xe3',[2,1],'real');
xe4 = sym('xe4',[2,1],'real');
%introduce spatial coordinate as fcn of isoparam. coord.
x=N1*xe1+N2*xe2+N3*xe3+N4*xe4;
%compute Jacobian
Fisop=jacobian(x,xi);
detFisop = det(Fisop);
%use chain rule to compute spatial derivatives
dN1_dx=simplify( inv(Fisop)'*dN1_dxi );
dN2_dx=simplify( inv(Fisop)'*dN2_dxi );
dN3_dx=simplify( inv(Fisop)'*dN3_dxi );
dN4_dx=simplify( inv(Fisop)'*dN4_dxi );
%define B-matrix of element
Be=[dN1_dx(1), 0, dN2_dx(1), 0, dN3_dx(1), 0, dN4_dx(1), 0;
0, dN1_dx(2), 0, dN2_dx(2), 0, dN3_dx(2), 0, dN4_dx(2);
dN1_dx(2),dN1_dx(1),dN2_dx(2),dN2_dx(1),dN3_dx(2),dN3_dx(1),...
dN4_dx(2),dN4_dx(1)];
matlabFunction(Be,detFisop,N_w, 'File','Vert_Be_quad_func','Vars'
   ,{xi,xe1,xe2,xe3,xe4});
```

3.2 Kirchhoff plate maximum out-of-plane displacement

```
clc
clear
close all

%% %%%%%%%%%%%%%% Computer Assignment 2 - Task_1 b)
   %%%%%%%%%%%%%%%%%%%%%%


%% Loading Datas
data = DataGen_CA2;

L = data.L;
W = data.W;
h_min = data.h_min;
```

```
h_max = data.h_max;
E_Mat = data.E_Mat;
nu = data.nu;
p = data.p;

% Costitutive Matrix
ptype = 1;              % Plane Stress
D_Mat = hooke(ptype, E_Mat, nu);

% Creating the mesh
n_el_x = 20;
n_el_y = 30;
n_type = 3;             % 3 Kirch        % 5 Mindlin
[mesh, Coord, Edof]=rectMesh(0, W, 0, L, n_el_x, n_el_y, n_type);

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

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

% Plot mesh
figure(1)
subplot(121)
plotpar = [1 1 0];
eldraw2(Ex,Ey,plotpar); grid on
title('Undeformed Plate', 'FontSize',15); axis tight
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;

% Picking dofs of the edge elements
Nodes_Bottom = find(Coord(:, 2) == 0);
Nodes_Top = find(Coord(:, 2) == L);
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == W);

Dof = reshape(1:n_dof, n_type, [])';

% Applying Contrains
```

```
Dof_Free = 1:n_dof;
Dof_Constr_w = [Dof(Nodes_Bottom, 1); Dof(Nodes_Right,1); Dof(
   Nodes_Left,1); Dof(Nodes_Top,1)];
Dof_Constr_Rot = [Dof(Nodes_Bottom, 2); Dof(Nodes_Right,3); Dof(
   Nodes_Left,3); Dof(Nodes_Top,2)];
Dof_Constr = unique([Dof_Constr_w; Dof_Constr_Rot]);
n_constr = length(Dof_Constr);
Dof_Free(Dof_Constr) = [];

% D constant
h = 10;          % [mm]
D_bar = h^3/12*D_Mat;

% Initialization
K = spalloc(n_dof,n_dof,20*n_dof);
f_ext = zeros(n_dof,1);
a = zeros(n_dof, 1);
a_constr = zeros(n_constr, 1);

% Assemble element stiffness matrix and load vector
for el = 1:n_el
    [K_el, f_el_ext] = Kirch_K_f_4noded(Ex(el,:), Ey(el, :), p,
       D_bar);
    [K, f_ext] = assem(Edof(el, :), K, K_el, f_ext, f_el_ext);
end

% Solving system of equations
a_free = K(Dof_Free, Dof_Free)\(f_ext(Dof_Free) - K(Dof_Free,
   Dof_Constr)*a_constr);
a(Dof_Free) = a_free;
a(Dof_Constr) = a_constr;

% Find maximum vertical displacement
Max_Vert_Disp = min(a(1:3:end));
fprintf(['The Maximum Vertical Displacement in the y direction is:
    ' num2str(Max_Vert_Disp), ' mm'])

% Plot Deformations
figure(1)
subplot(122)
Ed = extract_dofs(Edof,a); % extract element displacements for
   plotting
fill(Ex',Ey',Ed(:,1:3:end)')
grid on; colorbar; axis equal; axis tight
title('Plate Vertical Displacement [mm]', 'FontSize', 15)
xlabel('[mm]', 'FontSize', 15); ylabel('[mm]', 'FontSize', 15)
```

3.2.1 Kirch_K_f_4noded

```
function [K_el, f_el_ext] = Kirch_K_f_4noded(Ex, Ey, p, D_bar)

% xi-coordinates Vector
xi_vect = [-1/sqrt(3) -1/sqrt(3)  1/sqrt(3)  1/sqrt(3);
           -1/sqrt(3)  1/sqrt(3) -1/sqrt(3)  1/sqrt(3)];

% Integration weights
H_vect = ones(1,4);

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

% Loop over integration points
for ii = 1:4
    xi = xi_vect(:, ii);
    H = H_vect(ii);

    % Compute Determinant
    det_Fisop = detFisop_4node_func(xi, [Ex(1);Ey(1)], [Ex(2);Ey
       (2)], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);

    % Compute Shape Functions
    N = N_kirchoff_func(xi, [Ex(1);Ey(1)], [Ex(2);Ey(2)], [Ex(3);
       Ey(3)], [Ex(4);Ey(4)]);

    % Compute element external force
    f_el_ext = f_el_ext + N'*p*det_Fisop*H;

    % Compute Bstar
    B_Star = Bast_kirchoff_func(xi, [Ex(1);Ey(1)], [Ex(2);Ey(2)],
       [Ex(3);Ey(3)], [Ex(4);Ey(4)]);

    % Compute Stiffnes matrix
    K_el = K_el + B_Star'*D_bar*B_Star*det_Fisop*H;
end
end
```

3.3 Mindlin element stiffness matrix

```
clc
clear
close all

%% %%%%%%%%%%%%%%% Computer Assignment 2 - Task_1 b)
   %%%%%%%%%%%%%%%%%%%%%%
% Geometry
Ex_Test = [2 11 12 3]*1e-3;          % [m]
Ey_Test = [4 5 21 22]*1e-3;          % [m]
h_test = 50e-3;                      % [m]


% Material
E_Test = 80e9;                       % [Pa]
nu_Test = 0.2;
G_modulus_Test = E_Test/(2*(1+nu_Test));
G_Test = G_modulus_Test*eye(2,2);
G_0_Test = G_Test*h_test*5/6;
D_Test = hooke(1,E_Test,nu_Test);  % Note: Thin plate = plane
   stress 2D assumption
D_bar_Test = h_test^3/12*D_Test;
D0_Test = D_Test*h_test;

% Displacements
ae_Test = (1:20)'*1e-6;
z_Test = 5e-3;

% Out-of-plane load qz
q_Test = 0.5*1e3;

% Plot Mesh
figure
plotpar = [1 1 0];
eldraw2(Ex_Test,Ey_Test,plotpar); grid on
title('Undeformed Plate Test', 'FontSize',15);
xlabel('[m]', 'FontSize',15); ylabel('[m]', 'FontSize',15)

% Stiffness Matrix and Load Vector
[K_e_Test, K_uu_Test, K_ww_Test, K_wtheta_Test,
   K_thetatheta_B_Test, K_thetatheta_S_Test, fe_ext_Test] =
   Kel_fel_Mindlin_func(Ex_Test, Ey_Test, D_bar_Test, D0_Test,
   G_0_Test, q_Test);

% Stresses
```

```
[sigma_el_Test, tau_el_Test] = Stresses_Mindlin_func(Ex_Test,
   Ey_Test, ae_Test, z_Test, D_Test, G_Test);
```

3.3.1 Kel_fel_Mindlin_func

```
function [K_e, K_uu, K_ww, K_wtheta, K_thetatheta_B,
   K_thetatheta_S, fe_ext] = Kel_fel_Mindlin_func(Ex, Ey, D_bar,
   D_0, G_0, p)

%% Components computable through a 4 points Gauss integration

% xi-coordinates Vector
xi_vect_4 = [-1/sqrt(3) -1/sqrt(3)  1/sqrt(3)  1/sqrt(3);
             -1/sqrt(3)  1/sqrt(3) -1/sqrt(3)  1/sqrt(3)];

% Integration weights
H_vect_4 = ones(1,4);

% Initialitations
K_uu = zeros(8);
K_thetatheta_B = zeros(8);
fe_ext_w = zeros(4,1);

% Cycle over the 4 integration points
for ii = 1:4
    xi = xi_vect_4(:, ii);
    H = H_vect_4(ii);

    % Compute Determinant
    det_Fisop = detFisop_4node_func(xi, [Ex(1);Ey(1)], [Ex(2);Ey
       (2)], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);

    % Compute Be and Ne matrices
    [Be_u, Be_theta] = Be_Mindlin_func(xi, [Ex(1);Ey(1)], [Ex(2);
       Ey(2)], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);
    [~, ~, Ne_w] = N_Mindlin_func(xi, [Ex(1);Ey(1)], [Ex(2);Ey(2)
       ], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);

    % Compute K matrices
    K_uu = K_uu + Be_u'*D_0*Be_u*det_Fisop*H;
    K_thetatheta_B = K_thetatheta_B + Be_theta'*D_bar*Be_theta*
       det_Fisop*H;

    % Compute Force
    fe_ext_w = fe_ext_w + Ne_w'*p*det_Fisop*H;
end
```

```
%% Components computable only through a 1 points Gauss integration
    (Due to shear locking)
% (Only 1 integration point for 4 nodes)

% xi-coordinates Vector
xi_1 = [0; 0];

% Integration Weight
H_1 = 4;

% Compute Determinant
det_Fisop = detFisop_4node_func(xi_1, [Ex(1);Ey(1)], [Ex(2);Ey(2)
   ], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);

% Compute Be and Ne matrices
[~, ~, Be_w] = Be_Mindlin_func(xi_1, [Ex(1);Ey(1)], [Ex(2);Ey(2)],
    [Ex(3);Ey(3)], [Ex(4);Ey(4)]);
[~, Ne_theta] = N_Mindlin_func(xi_1, [Ex(1);Ey(1)], [Ex(2);Ey(2)],
    [Ex(3);Ey(3)], [Ex(4);Ey(4)]);

% Compute K matrices
K_ww = Be_w'*G_0*Be_w*det_Fisop*H_1;
K_wtheta = - Be_w'*G_0*Ne_theta*det_Fisop*H_1;
K_thetaw = K_wtheta';
K_thetatheta_S = Ne_theta'*G_0*Ne_theta*det_Fisop*H_1;

%% Assemblying Ke
K_e = zeros(20);

K_e(1:8, 1:8) = K_uu;
K_e(9:12, 9:12) = K_ww;
K_e(9:12, 13:20) = K_wtheta;
K_e(13:20, 9:12) = K_thetaw;
K_e(13:20, 13:20) = K_thetatheta_S + K_thetatheta_B;

% Reordering
K_e = K_e([1 2 9 13 14 3 4 10 15 16 5 6 11 17 18 7 8 12 19 20], [1
    2 9 13 14 3 4 10 15 16 5 6 11 17 18 7 8 12 19 20]);

%% Assemblying fe_ext
fe_ext = zeros(20, 1);

% Inserting Forces (all the other kind of forces are 0 here)
fe_ext(9:12) = fe_ext_w;
```

```
% Reordering
fe_ext = fe_ext([1 2 9 13 14 3 4 10 15 16 5 6 11 17 18 7 8 12 19
   20]');


end
```

3.3.2 Be_Mindlin_func / N_Mindlin_func

```
%% Code to Generate the functions to compute the N and B matrices
   for Mindlin
clc
clear
close all

%% Mindlin plate with isoparametrical quadrilateral element
xi = sym('xi',[2,1],'real');

% Define base functions for isoparam quadrilateral element
N1 = 0.25*(xi(1) - 1)*(xi(2) - 1); N2 = -0.25*(xi(1) + 1)*(xi(2) -
    1);
N3 = 0.25*(xi(1) + 1)*(xi(2) + 1); N4 = -0.25*(xi(1) - 1)*(xi(2) +
    1);

% Introduce node positions
xe1 = sym('xe1',[2,1],'real');
xe2 = sym('xe2',[2,1],'real');
xe3 = sym('xe3',[2,1],'real');
xe4 = sym('xe4',[2,1],'real');

% Introduce spatial coordinate as fcn of isoparam. coord.
x=N1*xe1+N2*xe2+N3*xe3+N4*xe4;

% Compute Jacobian
F_Isop = jacobian(x,xi);
invFisop = simplify(inv(F_Isop));
detFisop = simplify(det(F_Isop));


%% Shape Function Matrices
Ne_u = [N1 0 N2 0 N3 0 N4 0;
        0 N1 0 N2 0 N3 0 N4];

Ne_theta = Ne_u;
```

```
Ne_w =[N1 N2 N3 N4];

matlabFunction(Ne_u,Ne_theta,Ne_w, 'File','N_Mindlin_func','Vars'
    ,{xi,xe1,xe2,xe3,xe4});


%% B Matrices

% Differentiate shape functions
dN1_dxi=gradient(N1,xi);
dN2_dxi=gradient(N2,xi);
dN3_dxi=gradient(N3,xi);
dN4_dxi=gradient(N4,xi);

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

% B-Matrix of the element
Be_u = [dN1_dx(1),     0,     dN2_dx(1),     0,     dN3_dx(1),
       0,     dN4_dx(1),     0,   ;
            0,     dN1_dx(2),     0,     dN2_dx(2),     0,
               dN3_dx(2),     0,     dN4_dx(2);
        dN1_dx(2), dN1_dx(1), dN2_dx(2), dN2_dx(1), dN3_dx(2),
           dN3_dx(1), dN4_dx(2), dN4_dx(1)];

Be_theta = Be_u;

Be_w = [dN1_dx(1), dN2_dx(1), dN3_dx(1), dN4_dx(1);
        dN1_dx(2), dN2_dx(2), dN3_dx(2), dN4_dx(2)];


matlabFunction(Be_u, Be_theta, Be_w, 'File','Be_Mindlin_func','
    Vars',{xi,xe1,xe2,xe3,xe4});
```

3.4 Comparison between Kirchhoff and Mindlin plate elements

```
clc
clear
close all

%% %%%%%%%% Computer Assignment 2 - Task_1 d) %%%%%%%%%%%%

%% Loading Datas
```

```
data = DataGen_CA2;

L = data.L;
W = data.W;
h_min = data.h_min;
h_max = data.h_max;
E_Mat = data.E_Mat;
nu = data.nu;
p = data.p;

% Costitutive Matrix
ptype = 1;                % Plane Stress
D_Mat = hooke(ptype, E_Mat, nu);

%% ------------------ Kirchhoff Plate ----------------------------

% Creating the mesh
n_el_x = 20;
n_el_y = 30;
n_type = 3;               % 3 Kirch       % 5 Mindlin
[mesh, Coord, Edof]=rectMesh(0, W, 0, L, n_el_x, n_el_y, n_type);

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

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

% Plot mesh
figure(1)
subplot(121)
plotpar = [1 1 0];
eldraw2(Ex,Ey,plotpar); grid on
title('Undeformed Plate with Kirchhoff Elements', 'FontSize',15);
   axis tight
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;
```

```
% Picking dofs of the edge elements
Nodes_Bottom = find(Coord(:, 2) == 0);
Nodes_Top = find(Coord(:, 2) == L);
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == W);

Dof = reshape(1:n_dof, n_type, [])';

% Applying Contrains
Dof_Free = 1:n_dof;
Dof_Constr_w = [Dof(Nodes_Bottom, 1); Dof(Nodes_Right,1); Dof(
   Nodes_Left,1); Dof(Nodes_Top,1)];
Dof_Constr_Rot = [Dof(Nodes_Bottom, 2); Dof(Nodes_Right,3); Dof(
   Nodes_Left,3); Dof(Nodes_Top,2)];
Dof_Constr = unique([Dof_Constr_w; Dof_Constr_Rot]);
n_constr = length(Dof_Constr);
Dof_Free(Dof_Constr) = [];



%% Displacements

% Thickness Vector
h_vect = h_min:1:h_max;

% Initializations
Max_Vert_Disp_kirch_quantity = zeros(1, length(h_vect));

% Cycle to pass by every thickness
for h = h_vect

    % D_bar computation
    D_bar = h^3/12*D_Mat;

    % Resetting the variables
    clear K_el f_el_ext
    K = spalloc(n_dof,n_dof,20*n_dof);
    f_ext = zeros(n_dof,1);
    a = zeros(n_dof, 1);
    a_constr = zeros(n_constr, 1);

    % Cycle over the elements
    for el = 1:n_el
        [K_el, f_el_ext] = Kirch_K_f_4noded(Ex(el,:), Ey(el, :), p
           , D_bar);
        [K, f_ext] = assem(Edof(el, :), K, K_el, f_ext, f_el_ext);
    end

    % Solving system of equations
    a_free = K(Dof_Free, Dof_Free)\(f_ext(Dof_Free) - K(Dof_Free,
       Dof_Constr)*a_constr);
    a(Dof_Free) = a_free;
    a(Dof_Constr) = a_constr;

    % Compute and store maximum displacement
    Max_Vert_Disp_kirch_quantity(h) = min(a(1:3:end))*h^3;
end

% Plotting Kirchhoff displacements
figure(2)
subplot(223)
plot(h_vect, Max_Vert_Disp_kirch_quantity); grid on
ylim([-9 -4]*1e4); xlim tight
title('Plot of the quantity wh^3 for a Kirchhoff element plate', '
   FontSize', 12);
ylabel('wh^3 [mm^4]', 'FontSize', 12); xlabel('h [mm]' , 'FontSize
   ', 12);
subplot(221)
Ed = extract_dofs(Edof,a); % extract el displacements for plotting
fill(Ex',Ey',Ed(:,1:3:end)')
grid on; colorbar; axis equal; axis tight
title('Kirchhoff Plate Vertical Displacement [m]', 'FontSize', 15)
xlabel('[mm]', 'FontSize', 15); ylabel('[mm]', 'FontSize', 15)

figure(3)
plot(h_vect, Max_Vert_Disp_kirch_quantity, 'LineWidth',2); grid on
   ; hold on
ylim([-9 -4]*1e4); xlim tight


%% ---------------- Mindlin ---------------------------------

% Creating the mesh
n_el_x = 20;
n_el_y = 30;
n_type = 5;             % 3 Kirch       % 5 Mindlin
[mesh, Coord, Edof]=rectMesh(0, W, 0, L, n_el_x, n_el_y, n_type);

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

```
Coord_y = Coord(:, 2);
Edof_y = Edof(:, 3:n_type:end);
Ey = Coord_y(ceil(Edof_y/n_type));

% Plot mesh
figure(1)
subplot(122)
plotpar = [1 1 0];
eldraw2(Ex,Ey,plotpar); grid on
title('Undeformed Plate with Mindlin Elements', 'FontSize',15);
   axis tight
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;


% Picking dofs of the edge elements
Nodes_Bottom = find(Coord(:, 2) == 0);
Nodes_Top = find(Coord(:, 2) == L);
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == W);

Dof = reshape(1:n_dof, n_type, [])';

% Applying Contrains
Dof_Free = 1:n_dof;
Dof_Constr_x = [Dof(Nodes_Bottom, 1); Dof(Nodes_Right,1); Dof(
   Nodes_Left,1); Dof(Nodes_Top,1)];
Dof_Constr_y = [Dof(Nodes_Bottom, 2); Dof(Nodes_Right,2); Dof(
   Nodes_Left,2); Dof(Nodes_Top,2)];
Dof_Constr_w = [Dof(Nodes_Bottom, 3); Dof(Nodes_Right,3); Dof(
   Nodes_Left,3); Dof(Nodes_Top,3)];
Dof_Constr_Rot = [Dof(Nodes_Bottom, 4); Dof(Nodes_Right,5); Dof(
   Nodes_Left,5); Dof(Nodes_Top,4)];
Dof_Constr = unique([Dof_Constr_x; Dof_Constr_y; Dof_Constr_w;
   Dof_Constr_Rot]);
n_constr = length(Dof_Constr);
Dof_Free(Dof_Constr) = [];



%% Displacements
```

```
% Thickness Vector
h_vect = h_min:1:h_max;

% Initializations
Max_Vert_Disp_mindlin_quantity = zeros(1, length(h_vect));

% Cycle to pass by every thickness
for h = h_vect

    % Material Matrices computations
    D_bar = h^3/12*D_Mat;
    D_0 = h*D_Mat;
    G_Mat = E_Mat/(2*(1+nu));
    G_0 = G_Mat*5/6*h;

    % Resetting the variables
    clear K_el f_el_ext
    K = spalloc(n_dof,n_dof,20*n_dof);
    f_ext = zeros(n_dof,1);
    a = zeros(n_dof, 1);
    a_constr = zeros(n_constr, 1);

    % Cycle over the elements
    for el = 1:n_el
        [K_el, ~, ~, ~, ~, ~, f_el_ext] = Kel_fel_Mindlin_func(Ex(
           el,:), Ey(el, :), D_bar, D_0, G_0, p);
        [K, f_ext] = assem(Edof(el, :), K, K_el, f_ext, f_el_ext);
    end

    % Solving system of equations
    a_free = K(Dof_Free, Dof_Free)\(f_ext(Dof_Free) - K(Dof_Free,
       Dof_Constr)*a_constr);
    a(Dof_Free) = a_free;
    a(Dof_Constr) = a_constr;

    % Compute and store maximum displacement
    Max_Vert_Disp_mindlin_quantity(h) = min(a(3:5:end))*h^3;
end

% Plotting Mindlin displacements
figure(2)
subplot(224)
plot(h_vect, Max_Vert_Disp_mindlin_quantity); grid on; xlim tight
title('Plot of the quantity wh^3 for a Mindlin element plate', '
   FontSize', 12);
```

```
ylabel('wh^3 [mm^4]', 'FontSize', 12); xlabel('h [mm]' , 'FontSize
   ', 12);
subplot(222)
Ed = extract_dofs(Edof,a);
fill(Ex',Ey',Ed(:,3:5:end)')
grid on; colorbar; axis equal; axis tight
title('Mindlin Plate Vertical Displacement [m]', 'FontSize', 15)
xlabel('[mm]', 'FontSize', 15); ylabel('[mm]', 'FontSize', 15)

figure(3)
plot(h_vect, Max_Vert_Disp_mindlin_quantity, 'LineWidth',2); grid
   on; xlim tight
title('Plot of the quantity w_{max}h^3', 'FontSize', 12);
ylabel('w_{max}h^3 [mm^4]', 'FontSize', 12); xlabel('h [mm]' , '
   FontSize', 12);
legend('Kirchhoff', 'Mindlin', 'Location','southwest');
```

3.5 Stresses computation in a Mindlin plate element

```
clc
clear
close all

%% %%%%%%%%% Computer Assignment 2 - Task_1 e) %%%%%%%%%%%%%%%


%% Loading Datas
data = DataGen_CA2;

L = data.L;
W = data.W;
h_min = data.h_min;
h_max = data.h_max;
E_Mat = data.E_Mat;
nu = data.nu;
p = data.p;
h = 10;             %[mm]

% Material Matrix
ptype = 1;                 % Plane Stress
D_Mat = hooke(ptype, E_Mat, nu);
D_bar = h^3/12*D_Mat;
D_0 = h*D_Mat;
G_Mat = E_Mat/(2*(1+nu));
G_0 = G_Mat*5/6*h;
```

```
% Creating the mesh
n_el_x = 20;
n_el_y = 30;
n_type = 5;               % 3 Kirch        % 5 Mindlin
[mesh, Coord, Edof] = rectMesh(0, W, 0, L, n_el_x, n_el_y,n_type);

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

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

% Plot mesh
figure(1)
plotpar = [1 1 0];
eldraw2(Ex,Ey,plotpar); grid on
title('Undeformed Plate with Mindlin Elements', 'FontSize',15);
   axis tight
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;

% Picking dofs of the edge elements
Nodes_Bottom = find(Coord(:, 2) == 0);
Nodes_Top = find(Coord(:, 2) == L);
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == W);

% Applying Contrains
Dof = reshape(1:n_dof, n_type, [])';
Dof_Constr_x = [Dof(Nodes_Bottom, 1); Dof(Nodes_Right,1); Dof(
   Nodes_Left,1); Dof(Nodes_Top,1)];
Dof_Constr_y = [Dof(Nodes_Bottom, 2); Dof(Nodes_Right,2); Dof(
   Nodes_Left,2); Dof(Nodes_Top,2)];
Dof_Constr_w = [Dof(Nodes_Bottom, 3); Dof(Nodes_Right,3); Dof(
   Nodes_Left,3); Dof(Nodes_Top,3)];
Dof_Constr_Rot = [Dof(Nodes_Bottom, 4); Dof(Nodes_Right,5); Dof(
   Nodes_Left,5); Dof(Nodes_Top,4)];
Dof_Constr = unique([Dof_Constr_x; Dof_Constr_y; Dof_Constr_w;
   Dof_Constr_Rot]);
```

```
n_constr = length(Dof_Constr);

Dof_Free = 1:n_dof;
Dof_Free(Dof_Constr) = [];



%% Computation of the displacements

% Initializations
K = spalloc(n_dof,n_dof,20*n_dof);
f_ext = zeros(n_dof,1);
a = zeros(n_dof, 1);
a_constr = zeros(n_constr, 1);


% Cycle looping over all the elements to compute K and f
for el = 1:n_el
    [K_el, ~, ~, ~, ~, ~, f_el_ext] = Kel_fel_Mindlin_func(Ex(el
       ,:), Ey(el, :), D_bar, D_0, G_0, p);
    [K, f_ext] = assem(Edof(el, :), K, K_el, f_ext, f_el_ext);
end

% Solving system of equations
a_free = K(Dof_Free, Dof_Free)\(f_ext(Dof_Free) - K(Dof_Free,
   Dof_Constr)*a_constr);
a(Dof_Free) = a_free;
a(Dof_Constr) = a_constr;


%% Von Mises Stresses


% Thickness Coordinate Vector
z_coord = [-h/2 0 h/2];
name = {'-h/2', '0', 'h/2'};

% Initializations
sigma_VM = zeros(n_el, length(z_coord));
figure(2)

% Cycle to loop over the different coordinates and then elements
for ii = 1:3

    % Picking z
    z = z_coord(ii);
    for el = 1:n_el

        % Displacement of the single element
        a_el = a(Edof(el, 2:end));

        % Compute Stresses
        [sigma_el, tau_el] = Stresses_Mindlin_func(Ex(el,:), Ey(el
           , :), a_el, z, D_Mat, G_Mat);

        % Averaging
        sigma_avg = mean(sigma_el, 2);       % Sigma 11, 22, 12
        tau_avg = mean(tau_el, 2);           % Sigma 23, 13

        % Computation of s variable to use in VM
        s = [sigma_avg(1); sigma_avg(2); 0; sigma_avg(3); tau_avg
           (1); tau_avg(2)] - (sigma_avg(1) + sigma_avg(2) + 0)
           /3*[1;1;1;0;0;0];

        % Von Mises
        sigma_VM(el, ii) = sqrt(3/2*(s'*s + sigma_avg(3)^2 +
           tau_avg(1)^2 + tau_avg(2)^2));
    end

    % Plotting
    subplot(1, 3, ii)
    fill(Ex', Ey', sigma_VM(:, ii)');
    title(['Von Mises Stresses at ', name{ii}], 'FontSize',15);
    axis equal; axis tight
    xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)
    c = colorbar; c.Label.String = 'Von Mises Stress [MPa]'; c.
       Label.FontSize = 12;
end
```

3.5.1 Stresses_Mindlin_func

```
function [sigma_el, tau_el] = Stresses_Mindlin_func(Ex, Ey, ae, z,
    D_Mat, G_Mat)
% This function computes the stresses in a mindlin plate element
%assuming isotropic linear elasticity with plane stress
% Output:    sigma_el = [11 12; 21 22] tau_el = [13 23]

% xi-coordinates Vector
xi_vect = [-1/sqrt(3) -1/sqrt(3)  1/sqrt(3)  1/sqrt(3);
           -1/sqrt(3)  1/sqrt(3) -1/sqrt(3)  1/sqrt(3)];
```

```
% Displacements Extracions
ae_u = ae([1 2 6 7 11 12 16 17]);
ae_w = ae([3 8 13 18]);
ae_theta = ae([4 5 9 10 14 15 19 20]);

% Initializations
sigma_el = zeros(3, 4);
tau_el = zeros(2, 4);

% Cycle over the 4 quadrature points
for ii = 1:4
    xi = xi_vect(:, ii);

    % Compute Be and Ne matrices
    [Be_u, Be_theta, Be_w] = Be_Mindlin_func(xi, [Ex(1);Ey(1)], [
       Ex(2);Ey(2)], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);
    [~, Ne_theta] = N_Mindlin_func(xi, [Ex(1);Ey(1)], [Ex(2);Ey(2)
       ], [Ex(3);Ey(3)], [Ex(4);Ey(4)]);

    % Compute stresses
    sigma_el(:, ii) = D_Mat*(Be_u*ae_u - z*Be_theta*ae_theta);
    tau_el(:, ii) = G_Mat*(Be_w*ae_w - Ne_theta*ae_theta);
end
end
```

3.6 DataGen_CA2

```
function data_out = DataGen_CA2()

% Geometrical
L = 30e1;                % [mm]
W = 20e1;                % [mm]
h_min = 0.1e1;           % [mm]
h_max = 10e1;            % [mm]

% Material (Linear Elastic)
E_Mat = 3e3;             % [MPa]
nu = 0.3;                % [1]

% Load
p = -10e-1;              % [MPa]

% Output
data_out.L = L;
data_out.W = W;
data_out.h_min = h_min;
data_out.h_max = h_max;
data_out.E_Mat = E_Mat;
data_out.nu = nu;
data_out.p = p;

end
```

4 Matlab Code Task 2

At the beginning of every script the data useful to solve the problem are always imported through the function `DataGen_CA2_Task2`, that can be found in (Code 4.4). All the computations in (Code 4.1) were checked against the existing test cases found on Canvas.

4.1 Linearised pre-buckling analysis

```
clc
clear
close all

%% %%%%%%%%%% Computer Assignment 2 - Task_2 a) %%%%%%%%%%%%%%%

% Geometry
Ex_Test = [2 11 12 3]*1e-3;          % [m]
Ey_Test = [4 5 21 22]*1e-3;          % [m]
h_test = 50e-3;                      % [m]

% Material
E_Test = 80e9;                       % [Pa]
nu_Test = 0.2;
D_Test = hooke(1,E_Test,nu_Test);    % Note: Thin plate = plane
   stress 2D assumption

% Stresses
sigma2D = [1 2; 2 3]*1e6;
N_sec = h_test*sigma2D;

% Matrices
[K_uu_el, K_ww_K_el, G_el] = Kirch_Quad_Routine(Ex_Test, Ey_Test,
   D_Test, N_sec, h_test, -1);

% Display
disp(K_ww_K_el); disp(G_el); disp(K_uu_el)
```

4.1.1 Kirch_Quad_Routine

```
function [K_uu_el, K_ww_K_el, G_el, fe_u] = Kirch_Quad_Routine(Ex,
    Ey, D_Mat, N_sec, h, p)
% Output:     K_uu_el =  [8x8]
%             K_ww_el = [12x12]
%                G_el = [12x12]
%                fe_u = [12x1]

% Material Matrices
D_bar = h^3/12*D_Mat;
D_0 = h*D_Mat;

%% Cycle over 4 Integration Points

% Gauss Integration points and weihts
H_vect_4 = ones(1,4);
xi_vect_4 = [-1/sqrt(3) -1/sqrt(3)  1/sqrt(3)  1/sqrt(3);
             -1/sqrt(3)  1/sqrt(3) -1/sqrt(3)  1/sqrt(3)];

% Initializations
K_uu_el = zeros(8);
K_ww_K_el = zeros(12);
fe_u=zeros(8,1);
P = [0; p];

for ii = 1:4
    H = H_vect_4(ii);
    xi = xi_vect_4(:,ii);

    % Computation of determinant and B matrices
    det_Fisop = detFisop_4node_func(xi,[ Ex(1) Ey(1) ]',[ Ex(2) Ey
       (2) ]',[ Ex(3) Ey(3) ]',[ Ex(4) Ey(4) ]');
    Bastn = Bast_kirchoff_func(xi,[ Ex(1) Ey(1) ]',[ Ex(2) Ey(2)
       ]',[ Ex(3) Ey(3) ]',[ Ex(4) Ey(4) ]');
    Be_u = Be_Mindlin_func(xi,[ Ex(1) Ey(1) ]',[ Ex(2) Ey(2) ]',[
       Ex(3) Ey(3) ]',[ Ex(4) Ey(4) ]');
    Ne_u = N_Mindlin_func(xi,[ Ex(1) Ey(1) ]',[ Ex(2) Ey(2) ]',[
       Ex(3) Ey(3) ]',[ Ex(4) Ey(4) ]');

    % Computation of Matrices
    K_uu_el = K_uu_el + Be_u'*D_0*Be_u*det_Fisop*H;
    K_ww_K_el = K_ww_K_el + Bastn'*D_bar*Bastn*det_Fisop*H;

    % Computation of the force
    fe_u = fe_u + Ne_u'*P*det_Fisop*H;
```

```
end


%% Cycle over 9 integration points

% Gauss Integration points and weights
H_vect_9 = [0.309 0.494 0.309 0.494 0.790 0.494 0.309 0.494
   0.309];
xi_vect_9 = [-sqrt(3/5)      0      sqrt(3/5) -sqrt(3/5)    0
   sqrt(3/5) -sqrt(3/5)     0     sqrt(3/5);
             -sqrt(3/5) -sqrt(3/5) -sqrt(3/5)      0        0
                      0      sqrt(3/5) sqrt(3/5) sqrt(3/5)];

% Initialization
G_el = zeros(12);

for ii = 1:9
    H = H_vect_9(ii);
    xi = xi_vect_9(:,ii);

    % Compute determinant B_el Matrix
    det_Fisop = detFisop_4node_func(xi,[ Ex(1) Ey(1) ]',[ Ex(2) Ey
       (2) ]',[ Ex(3) Ey(3) ]',[ Ex(4) Ey(4) ]');
    [~, B_el_T] = Bast_kirchoff_func(xi,[ Ex(1) Ey(1) ]',[ Ex(2)
       Ey(2) ]',[ Ex(3) Ey(3) ]',[ Ex(4) Ey(4) ]');

    % Compute G matrix
    G_el = G_el + B_el_T*N_sec*B_el_T'*det_Fisop*H;
end
end
```

4.2 FE programme for linearised pre-buckling analysis

```
clc
clear
close all

%% %%%%%%%%%%%%%% Computer Assignment 2 - Task_2 b)
   %%%%%%%%%%%%%%%%%%%%

% Loading Data
data_out = DataGen_CA2_Task2;

L = data_out.L;
W = data_out.W;
h = data_out.h;
```

```
D_Mat = data_out.D_Mat;
Compr_Strain = data_out.Compr_Strain;
Compr_Strenght = data_out.Compr_Strenght;

% Material Matrices computations
D_bar = h^3/12*D_Mat;
D_0 = h*D_Mat;



%% Point 1) Solution of the in-plane deformation

% Creating the mesh
n_el_x = 50;
n_el_y = 5;
n_type = 2;              % 2 In-Plane solution
[mesh, Coord, Edof] = rectMesh(0, L, 0, W, n_el_x, n_el_y, n_type)
   ;

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

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

% Plot mesh
figure(1)
plotpar = [2 1 0];
eldraw2(Ex,Ey,plotpar); grid on; hold on;
title('Deformation of the Specimen with prescribed displacement',
   'FontSize',15);
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;

% Picking the contrained nodes
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == L);

% Applying Contrains
```

```
Dof = reshape(1:n_dof, n_type, [])';
Dof_Constr_x = [Dof(Nodes_Right,1); Dof(Nodes_Left,1)];
Dof_Constr_y = [Dof(Nodes_Right,2); Dof(Nodes_Left,2)];

Dof_Constr = unique([Dof_Constr_x; Dof_Constr_y]);
n_constr = length(Dof_Constr);

Dof_Free = 1:n_dof;
Dof_Free(Dof_Constr) = [];

% Displacements
u_bar = Compr_Strain*L;
a_uu = zeros(n_dof, 1);
a_constr = zeros(n_dof, 1);
a_constr(Dof(Nodes_Left,1)) = u_bar;
a_constr = a_constr(Dof_Constr);

% Initializations
K_uu = spalloc(n_dof,n_dof,20*n_dof);
f_ext_uu = zeros(n_dof,1);

% Cycle over the elements
for el = 1:n_el
    [K_uu_el, ~, ~, f_el_ext] = Kirch_Quad_Routine(Ex(el,:), Ey(el
        , :), D_Mat, zeros(2), h, 0);
    [K_uu, f_ext_uu] = assem(Edof(el, :), K_uu, K_uu_el, f_ext_uu,
        f_el_ext);
end

% Solving system of equations
a_free_uu = K_uu(Dof_Free, Dof_Free)\(f_ext_uu(Dof_Free) - K_uu(
   Dof_Free, Dof_Constr)*a_constr);
a_uu(Dof_Free) = a_free_uu;
a_uu(Dof_Constr) = a_constr;

% Plotting Defromation
Ed = extract_dofs(Edof, a_uu);
eldisp2(Ex, Ey, Ed, [1 4 0], 1); axis tight; axis equal

figure('WindowState', 'maximized')
subplot(311)
Ed_x_avg = sum(Ed(:,1:2:end), 2)/4;
fill(Ex', Ey', Ed_x_avg); colorbar; axis equal; axis tight
title('Averaged Displacement along x   [mm]', 'FontSize',15)
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)
```

```
%% Point 2) In-plane Stresses

sigma_xx = zeros(1, n_el);
N_sec = zeros(2,2, n_el);

for el = 1:n_el
    % Extraction of the element displacement
    ae_u = a_uu(Edof(el, 2:end));

    % Computing the stress
    [sigma] = stress_inplane(Ex(el, :), Ey(el, :), ae_u, D_Mat);

    % Allocating stresses
    sigma_xx(el) = sigma(1);
    N_sec(:, :, el) = h*[sigma(1) sigma(3);
                         sigma(3) sigma(2)];

end

% Plotting
figure(2)
subplot(312)
fill(Ex', Ey', sigma_xx); colorbar; axis equal; axis tight
title('Stress along x  \sigma_{xx} [MPa]', 'FontSize',15)
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)
subplot(313)
buckl = zeros(1, n_el);
buckl(:, sigma_xx < Compr_Strenght) = ones(1, size(nonzeros(
   sigma_xx < Compr_Strenght), 2));
fill(Ex', Ey', buckl); axis equal; axis tight
title('Elements at Risk of buckling', 'FontSize',15)
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)



%% Point 3) Assemblying Matrices

% Creating the mesh
n_el_x = 50;
n_el_y = 5;
n_type = 3;             % 3 To compute the buckling
[mesh, Coord, Edof] = rectMesh(0, L, 0, W, n_el_x, n_el_y, n_type)
   ;
```

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

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

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;

% Picking the contrained nodes
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == L);

% Applying Contrains
Dof = reshape(1:n_dof, n_type, [])';
Dof_Constr_w = [Dof(Nodes_Right,1); Dof(Nodes_Left,1)];
Dof_Constr_omega_x = [Dof(Nodes_Right,2); Dof(Nodes_Left,2)];
Dof_Constr_omega_y = [Dof(Nodes_Right,3); Dof(Nodes_Left,3)];
Dof_Constr = unique([Dof_Constr_w; Dof_Constr_omega_x;
   Dof_Constr_omega_y]);
n_constr = length(Dof_Constr);

% Extraction of the free dof
Dof_Free = 1:n_dof;
Dof_Free(Dof_Constr) = [];

% Initializations
K_ww_K = spalloc(n_dof,n_dof,20*n_dof);
G_R = spalloc(n_dof,n_dof,20*n_dof);

% Cycle over the elements
for el = 1:n_el
    [~, K_ww_K_el, G_R_el] = Kirch_Quad_Routine(Ex(el,:), Ey(el,
       :), D_Mat, N_sec(:,:, el), h, 0);
    K_ww_K = assem(Edof(el, :), K_ww_K, K_ww_K_el);
    G_R = assem(Edof(el, :), G_R, G_R_el);
end



%% Point 4) Eigenvalues problem
```

```
% Extracting Free part of the matrices
K_ww_K_FF = K_ww_K(Dof_Free, Dof_Free);
G_R_FF = G_R(Dof_Free, Dof_Free);

% Solving Eigenvalues problems
n_lambda = 8;
[Eig_Vect, Eig_Val_Mat] = eigs(K_ww_K_FF, -G_R_FF, n_lambda, '
    smallestabs');
lambda = diag(Eig_Val_Mat);

% Displaying Safety Factor
fprintf(['The Safety Factor against buckling is: ', num2str(lambda
    (1))]);

figure('WindowState', 'maximized')
sgtitle('Mode Shapes', 'FontSize', 18, 'FontWeight', 'bold')
for ii = 1:n_lambda
    % Find the modeshape
    Mode_Shape = zeros(n_dof, 1);
    Mode_Shape(Dof_Free, :) = Eig_Vect(:, ii);

    % Extraction to plot
    E_Mode_Shape = extract_dofs(Edof, Mode_Shape);
    E_Mode_Shape_avg = sum(E_Mode_Shape(:,1:3:end), 2)/4;

    % Plotting
    subplot(ceil(n_lambda/2), 2, ii)
    fill(Ex', Ey', E_Mode_Shape_avg); colorbar; axis equal; axis
       tight
    title(['Mode Shape Number: ', num2str(ii), '       \lambda = ',
        num2str(lambda(ii))], 'FontSize',15);
    xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)
end
```

4.2.1 stress_inplane

```
function [sigma] = stress_inplane (Ex, Ey, ae_u, D_Mat)

% xi-coordinate of the midpoint
xi = [0; 0];

% Computing B_u matrix
Be_u = Be_Mindlin_func(xi, [Ex(1);Ey(1)], [Ex(2);Ey(2)], [Ex(3);Ey
    (3)], [Ex(4);Ey(4)]);
```

```
% Computing the stresses
sigma = D_Mat*Be_u*ae_u;

end
```

4.3 Additional support

```
clc
clear
close all

%% %%%%%%%%%%%%%% Computer Assignment 2 - Task_2 c)
   %%%%%%%%%%%%%%%%%%%%

% Loading Data
data_out = DataGen_CA2_Task2;

L = data_out.L;
W = data_out.W;
h = data_out.h;
D_Mat = data_out.D_Mat;
Compr_Strain = data_out.Compr_Strain;
Compr_Strenght = data_out.Compr_Strenght;

% Material Matrices computations
D_bar = h^3/12*D_Mat;
D_0 = h*D_Mat;


%% Solution of the in-plane deformation

% Creating the mesh
n_el_x = 50;
n_el_y = 5;
n_type = 2;               % 2 In-Plane solution
[mesh, Coord, Edof] = rectMesh(0, L, 0, W, n_el_x, n_el_y, n_type)
   ;

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

Coord_y = Coord(:, 2);
Edof_y = Edof(:, 3:n_type:end);
Ey = Coord_y(ceil(Edof_y/n_type));
```

```
% Plot mesh
figure(1)
plotpar = [1 1 0];
eldraw2(Ex,Ey,plotpar); grid on; hold on;
title('Deformation of the Specimen with prescribed displacement',
   'FontSize',15);
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;

% Picking the contrained nodes
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == L);

% Applying Contrains
Dof = reshape(1:n_dof, n_type, [])';
Dof_Constr_x = [Dof(Nodes_Right,1); Dof(Nodes_Left,1)];
Dof_Constr_y = [Dof(Nodes_Right,2); Dof(Nodes_Left,2)];

Dof_Constr = unique([Dof_Constr_x; Dof_Constr_y]);
n_constr = length(Dof_Constr);

Dof_Free = 1:n_dof;
Dof_Free(Dof_Constr) = [];

% Displacements
u = Compr_Strain*L;
a_uu = zeros(n_dof, 1);
a_constr = zeros(n_dof, 1);
a_constr(Dof(Nodes_Left,1)) = u;
a_constr = a_constr(Dof_Constr);

% Initializations
K_uu = spalloc(n_dof,n_dof,20*n_dof);
f_ext_uu = zeros(n_dof,1);

% Cycle over the elements
for el = 1:n_el
    [K_uu_el, ~, ~, f_el_ext] = Kirch_Quad_Routine(Ex(el,:), Ey(el
       , :), D_Mat, zeros(2), h, 0);
    [K_uu, f_ext_uu] = assem(Edof(el, :), K_uu, K_uu_el, f_ext_uu,
        f_el_ext);
```

```
end

% Solving system of equations
a_free_uu = K_uu(Dof_Free, Dof_Free)\(f_ext_uu(Dof_Free) - K_uu(
   Dof_Free, Dof_Constr)*a_constr);
a_uu(Dof_Free) = a_free_uu;
a_uu(Dof_Constr) = a_constr;

% Plotting Defromation
Ed = extract_dofs(Edof, a_uu);
eldisp2(Ex, Ey, Ed, [2 4 0], 1); axis tight; axis equal

figure('WindowState', 'maximized')
subplot(311)
Ed_x_avg = sum(Ed(:,1:2:end), 2)/4;
fill(Ex', Ey', Ed_x_avg); colorbar; axis equal; axis tight
title('Averaged Displacement along x   [mm]', 'FontSize',15)
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)



%% In-plane Stresses

sigma_xx = zeros(1, n_el);
N_sec = zeros(2,2, n_el);

for el = 1:n_el
    % Extraction of the element displacement
    ae_u = a_uu(Edof(el, 2:end));

    % Computing the stress
    [sigma] = stress_inplane(Ex(el, :), Ey(el, :), ae_u, D_Mat);

    % Allocating stresses
    sigma_xx(el) = sigma(1);
    N_sec(:, :, el) = h*[sigma(1) sigma(3);
                         sigma(3) sigma(2)];

end

% Plotting
figure(2)
subplot(312)
fill(Ex', Ey', sigma_xx); colorbar; axis equal; axis tight
title('Stress along x  \sigma_{xx} [MPa]', 'FontSize',15)
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)
```

```
subplot(313)
buckl = zeros(1, n_el);
buckl(:, sigma_xx < Compr_Strenght) = ones(1, size(nonzeros(
   sigma_xx < Compr_Strenght), 2));
fill(Ex', Ey', buckl); colorbar; axis equal; axis tight
title('Elements at Risk of failing', 'FontSize',15)
xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)


%% New Boundary Conditions

% Creating the mesh
n_el_x = 50;
n_el_y = 5;
n_type = 3;             % 3 To compute the buckling
[mesh, Coord, Edof] = rectMesh(0, L, 0, W, n_el_x, n_el_y, n_type)
   ;

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

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

% Numbers
n_el = size(mesh, 2);
n_nodes = size(Coord, 1);
n_dof = n_nodes*n_type;

% Picking the contrained nodes
Nodes_Left = find(Coord(:, 1) == 0);
Nodes_Right = find(Coord(:, 1) == L);
Nodes_Middle = find(Coord(:, 1) == L/2);

% Applying Contrains
Dof = reshape(1:n_dof, n_type, [])';
Dof_Constr_w = [Dof(Nodes_Right,1); Dof(Nodes_Left,1); Dof(
   Nodes_Middle,1)];
Dof_Constr_omega_x = [Dof(Nodes_Right,2); Dof(Nodes_Left,2)];
Dof_Constr_omega_y = [Dof(Nodes_Right,3); Dof(Nodes_Left,3)];
Dof_Constr = unique([Dof_Constr_w; Dof_Constr_omega_x;
   Dof_Constr_omega_y]);
n_constr = length(Dof_Constr);
```

```
% Extraction of the free dof
Dof_Free = 1:n_dof;
Dof_Free(Dof_Constr) = [];

% Initializations
K_ww_K = spalloc(n_dof,n_dof,20*n_dof);
G_R = spalloc(n_dof,n_dof,20*n_dof);
f_ext_ww = zeros(n_dof,1);

% Cycle over the elements
for el = 1:n_el
    [~, K_ww_K_el, G_R_el] = Kirch_Quad_Routine(Ex(el,:), Ey(el,
       :), D_Mat, N_sec(:,:, el), h, 0);
    K_ww_K = assem(Edof(el, :), K_ww_K, K_ww_K_el);
    G_R = assem(Edof(el, :), G_R, G_R_el);
end



%% Eigenvalues problem

% Extracting Free part of the matrices
K_ww_K_FF = K_ww_K(Dof_Free, Dof_Free);
G_R_FF = G_R(Dof_Free, Dof_Free);

% Solving Eigenvalues problems
n_lambda = 8;
[Eig_Vect, Eig_Val_Mat] = eigs(K_ww_K_FF, -G_R_FF, n_lambda, '
   smallestabs');
lambda = diag(Eig_Val_Mat);

% Displaying Safety Factor
fprintf(['The Safety Factor against buckling is: ', num2str(lambda
   (1))]);

figure('WindowState', 'maximized')
sgtitle('Mode Shapes', 'FontSize', 18, 'FontWeight', 'bold')
for ii = 1:n_lambda
    % Find the modeshape
    Mode_Shape = zeros(n_dof, 1);
    Mode_Shape(Dof_Free, :) = Eig_Vect(:, ii);

    % Extraction to plot
    E_Mode_Shape = extract_dofs(Edof, Mode_Shape);
    E_Mode_Shape_avg = sum(E_Mode_Shape(:,1:3:end), 2)/4;

    % Plotting
    subplot(ceil(n_lambda/2), 2, ii)
    fill(Ex', Ey', E_Mode_Shape_avg); colorbar; axis equal; axis
       tight
    title(['Mode Shape Number: ', num2str(ii), '      \lambda = ',
        num2str(lambda(ii))], 'FontSize',15);
    xlabel('[mm]', 'FontSize',15); ylabel('[mm]', 'FontSize',15)
end
```

4.4 DataGen_CA2_Task2

```
function data_out = DataGen_CA2_Task2()

% Geometrical
L = 50;                          % [mm]
W = 5;                           % [mm]
h= 2.5;                          % [mm]

% Material (Linear Elastic)
E_L = 150e3;                     % [MPa]
E_T = 11e3;                      % [MPa]
D_Mat = 1e3*[151 3.4  0;         % [MPa]
             3.4 11.1 0;
             0    0   5];

% Load
Compr_Strenght = -1.695e3;       % [MPa]
Compr_Strain = 1.13e-2;          % [1]

% Output
data_out.L = L;
data_out.W = W;
data_out.h = h;
data_out.E_T = E_T;
data_out.E_L = E_L;
data_out.D_Mat = D_Mat;
data_out.Compr_Strenght = Compr_Strenght;
data_out.Compr_Strain = Compr_Strain;

end
```