ME621 - Advanced Finite Element Methods
Assignment 1

Tommaso Bocchietti

A.Y. 2023/24 - W24

Contents

List of Figures

List of Tables

1 Requests

A system of two aluminum bars of the same material is shown in the following figure. The system is subjected to two external loads, P_x and P_y, at joint B. A and C are connected to pinned supports.

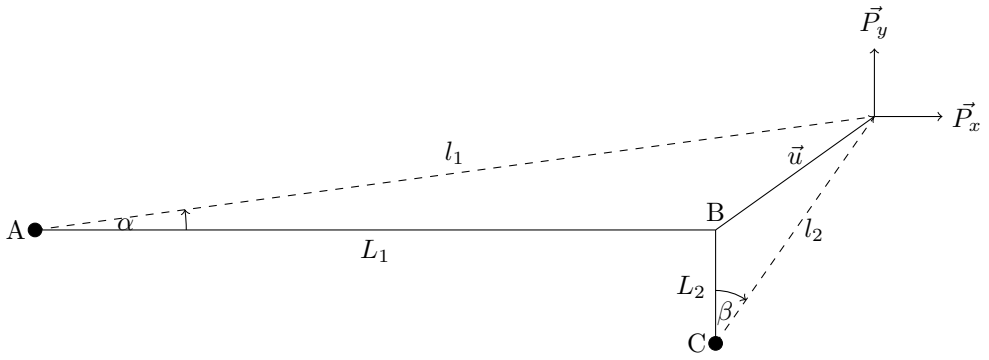


Figure 1: Problem representation

The problem asks to:

- Obtain the external loads P_x and P_y as a function of horizontal and vertical displacements at point B (namely u and v).
- Determine the displacements in both x and y directions for 1000 load increments of +5N for both P_x and P_y (from zero).
- Find the displacement of point B after the final increment.

Write a `MATLAB` code with a convergence error of 10^-5 to numerically solve the problem. Use a combination of (a) Euler and N-R, and (b) Euler and modified N-R. Also plot the resultant force versus the resultant displacement. Use the Green strain measure:

$$E_i = \frac{l_i^2 - L_i^2}{2L_i^2} \tag{1}$$

From now on, we will refer to the Green strain measure as $\epsilon_{1,2}$ to differentiate it from the Young's modulus $E_{1,2}$.

Parameter	**Value**	**Unit**
$E_1 = E_2 = E$	70	GPa
L_1	3	m
L_2	0.5	m
$A_1 = A_2 = A$	0.0001	m^2

Table 1: Parameters of the system

2 Methodology

We will start by writing the equilibrium equations for the system. We will further work on the **equilibrium equations to obtain the force-displacement relationship**. Finally, we will solve the system of equations $\vec{P} = f(\vec{u})$ to obtain the displacement of point B for a given load.
Notice that we will likely obtain a system of non-linear equations, which **we will solve using Euler, Newton-Raphson and modified Newton-Raphson methods**. We will also perform a linearization and an approximation of the system to successively compare the results obtained with the exact solution.
Even if the problem asked to produce a `MATLAB` code, we will first set up the system of equations in `Mathematica`, and then we will translate the code to `MATLAB`. `Mathematica` is a more powerful tool for symbolic computation, and it will allow us to obtain a system of equations that can be easily updated in case of necessity (for example if we are interested in a different order of the Taylor series).

3 Solution

3.1 Equilibrium equations

We can start writing the equilibrium equations for the system. Since we have no data about the mass of the beams, we will assume it to be negligible. This assumption will allow us to neglect the effect of gravity on the system and avoid the introduction of the inertia terms in the equilibrium equations.

$$\vec{F_{ext}} + \vec{F_{int}} = \vec{0} \tag{2}$$

$$\begin{cases} \vec{F_{ext}} = \vec{P_x} + \vec{P_y} \\ \vec{F_{int}} = \vec{F_1} + \vec{F_2} \end{cases} \tag{3}$$

By decomposing the equations in the x and y directions, we obtain:

$$\begin{cases} P_x = F_{int,x} = |F_1|\cos(\alpha) + |F_2|\sin(\beta) \\ P_y = F_{int,y} = |F_1|\sin(\alpha) + |F_2|\cos(\beta) \end{cases} \tag{4}$$

3.2 Force displacement relationship

So far we have obtained the equilibrium equations for the system. We can now proceed to obtain the force-displacement relationship, which will allow us to solve the system of equations $\vec{P} = f(\vec{u})$. To do so, we try to express everything on the right-hand side of the equilibrium equations 4 in terms of the displacements u and v. From simple trigonometrical considerations, we can obtain the following relationships:

$$\cos(\alpha) = \frac{\mathrm{L}_1 + u}{l_1} \tag{5}$$

$$\sin(\alpha) = \frac{v}{l_1} \tag{6}$$

$$\cos(\beta) = \frac{\mathrm{L}_2 + v}{l_2} \tag{7}$$

$$\sin(\beta) = \frac{u}{l_2} \tag{8}$$

We can now proceed working on the forces, knowing that the internal forces are linked to the strains by the following relationship:

$$\begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix} = \begin{Bmatrix} \mathrm{A}_1\mathrm{E}_1\epsilon_1 \\ \mathrm{A}_2\mathrm{E}_2\epsilon_2 \end{Bmatrix} \tag{9}$$

And the strains are linked to the displacements by the following relationship:

$$\begin{Bmatrix} \epsilon_1 \\ \epsilon_2 \end{Bmatrix} = \begin{Bmatrix} \frac{l_1^2 - \mathrm{L}_1^2}{2\mathrm{L}_1^2} \\ \frac{l_2^2 - \mathrm{L}_2^2}{2\mathrm{L}_2^2} \end{Bmatrix} \tag{10}$$

Finally, we can give the following definition to the real length of the beams:

$$\begin{Bmatrix} l_1 \\ l_2 \end{Bmatrix} = \begin{Bmatrix} \sqrt{(\mathrm{L}_1 + u)^2 + v^2} \\ \sqrt{u^2 + (\mathrm{L}_2 + v)^2} \end{Bmatrix} \tag{11}$$

We can now substitute the equations above in the equilibrium equations 4 to obtain the force displacement relationship:

$$\begin{Bmatrix} P_x \\ P_y \end{Bmatrix} = \begin{Bmatrix} \frac{\mathrm{A}_1\mathrm{E}_2(\mathrm{L}_1+u)\left(-\mathrm{L}_1^2+(\mathrm{L}_1+u)^2+v^2\right)}{2\mathrm{L}_1^2\sqrt{(\mathrm{L}_1+u)^2+v^2}} + \frac{\mathrm{A}_2\mathrm{E}_2 u\left(-\mathrm{L}_2^2+(\mathrm{L}_2+v)^2+u^2\right)}{2\mathrm{L}_2^2\sqrt{(\mathrm{L}_2+v)^2+u^2}} \\ \frac{\mathrm{A}_1\mathrm{E}_2 v\left(-\mathrm{L}_1^2+(\mathrm{L}_1+u)^2+v^2\right)}{2\mathrm{L}_1^2\sqrt{(\mathrm{L}_1+u)^2+v^2}} + \frac{\mathrm{A}_2\mathrm{E}_2(\mathrm{L}_2+v)\left(-\mathrm{L}_2^2+(\mathrm{L}_2+v)^2+u^2\right)}{2\mathrm{L}_2^2\sqrt{(\mathrm{L}_2+v)^2+u^2}} \end{Bmatrix} \tag{12}$$

Even if quite long, the equation 12 are nothing more than a function $\vec{P} = f(\vec{u})$ which can be solved numerically for a given value of $\vec{P^*}$ to find the corresponding value of $\vec{u^*}$.

3.3 Linearization of P = f(u)

Before proceeding with the numerical solution of the system, we can try to linearize the system to obtain a linear system of equations. To do so, we can use a Taylor series expansion of the force-displacement relationship 12 around the equilibrium position $\vec{u} = \vec{0}$. A general Taylor series expansion of a function $f(x, y)$ around the point (x_0, y_0) is given by:

$$\begin{aligned} f(x,y) = {} & f(x_0,y_0) + \\ & + \frac{\partial f}{\partial x}(x_0,y_0)\cdot(x-x_0) + \frac{\partial f}{\partial y}(x_0,y_0)\cdot(y-y_0) + \\ & + \frac{1}{2}\frac{\partial^2 f}{\partial x^2}(x_0,y_0)\cdot(x-x_0)^2 + \frac{1}{2}\frac{\partial^2 f}{\partial y^2}(x_0,y_0)\cdot(y-y_0)^2 + \frac{\partial^2 f}{\partial x \partial y}(x_0,y_0)\cdot(x-x_0)\cdot(y-y_0) + \\ & + \dots \end{aligned} \tag{13}$$

3.3.1 Taylor series of order 1

By applying the Taylor series expansion of order 1, we obtain the following system of linear equations:

$$\begin{Bmatrix} \widehat{P_x} \\ \widehat{P_y} \end{Bmatrix} = \begin{Bmatrix} \frac{\mathrm{A_1 E_2} u}{\mathrm{L_1}} \\ \frac{\mathrm{A_2 E_2} v}{\mathrm{L_2}} \end{Bmatrix} \tag{14}$$

3.3.2 Taylor series of order 2

By applying the Taylor series expansion of order 2, we obtain the following system of quadratic equations:

$$\begin{Bmatrix} \widehat{\widehat{P_x}} \\ \widehat{\widehat{P_y}} \end{Bmatrix} = \begin{Bmatrix} \frac{\mathrm{A_1 E_2}\left(u^2+v^2\right)}{2\mathrm{L_1^2}} + \frac{\mathrm{A_1 E_2} u}{\mathrm{L_1}} + \frac{\mathrm{A_2 E_2} uv}{\mathrm{L_2^2}} \\ \frac{\mathrm{A_1 E_2} uv}{\mathrm{L_1^2}} + \frac{\mathrm{A_2 E_2}\left(u^2+v^2\right)}{2\mathrm{L_2^2}} + \frac{\mathrm{A_2 E_2} v}{\mathrm{L_2}} \end{Bmatrix} \tag{15}$$

3.3.3 Taylor series of order 3

By applying the Taylor series expansion of order 3, we obtain the following system of cubic equations:

$$\begin{Bmatrix} \widehat{\widehat{\widehat{P_x}}} \\ \widehat{\widehat{\widehat{P_y}}} \end{Bmatrix} = \begin{Bmatrix} -\frac{\mathrm{A_1 E_2} uv^2}{2\mathrm{L_1^3}} + \frac{\mathrm{A_1 E_2}\left(u^2+v^2\right)}{2\mathrm{L_1^2}} + \frac{\mathrm{A_1 E_2} u}{\mathrm{L_1}} + \frac{\mathrm{A_2 E_2} u\left(u^2-v^2\right)}{2\mathrm{L_2^3}} + \frac{\mathrm{A_2 E_2} uv}{\mathrm{L_2^2}} \\ \frac{\mathrm{A_1 E_2} v\left(v^2-u^2\right)}{2\mathrm{L_1^3}} + \frac{\mathrm{A_1 E_2} uv}{\mathrm{L_1^2}} - \frac{\mathrm{A_2 E_2} u^2 v}{2\mathrm{L_2^3}} + \frac{\mathrm{A_2 E_2}\left(u^2+v^2\right)}{2\mathrm{L_2^2}} + \frac{\mathrm{A_2 E_2} v}{\mathrm{L_2}} \end{Bmatrix} \tag{16}$$

3.4 Approximation of P = f(u)

Under the hypothesis of small displacements, we can approximate the force-displacement relationship 12 to obtain a simpler system of equations.

3.4.1 Approximation (acceptable/soft)

Working on the trigonometric functions, we can approximate the cosine and sine functions as follows:

$$lim_{\alpha \to 0} \cos(\alpha) = 1 \tag{17}$$
$$lim_{\alpha \to 0} \sin(\alpha) = \alpha \tag{18}$$
$$lim_{\beta \to 0} \cos(\beta) = 1 \tag{19}$$
$$lim_{\beta \to 0} \sin(\beta) = \beta \tag{20}$$

By substituting in the force-displacement relationship 12, we obtain the following approximated system of equations:

$$\begin{Bmatrix} P_{x,approx,soft} \\ P_{y,approx,soft} \end{Bmatrix} = \begin{Bmatrix} \frac{\mathrm{A_1 E_1}\left(2\mathrm{L_1} u+u^2+v^2\right)}{\mathrm{L_1^2}} + \frac{\mathrm{A_2 E_2} u\left(v(2\mathrm{L_2}+v)+u^2\right)}{\mathrm{L_2^2}} \\ \frac{\mathrm{A_1 E_1} v\left(2\mathrm{L_1} u+u^2+v^2\right)}{\mathrm{L_1^2}} + \frac{\mathrm{A_2 E_2}\left(v(2\mathrm{L_2}+v)+u^2\right)}{\mathrm{L_2^2}} \end{Bmatrix} \tag{21}$$

3.4.2 Approximation (bad/hard)

To see how a bad approximation could bring to a shifted or wrong solution, we will assume that:

$$\cos(\alpha) = \frac{\mathrm{L}_1 + u}{l_1} \approx 1 \tag{22}$$

$$\sin(\alpha) = \frac{v}{l_1} \approx 0 \tag{23}$$

$$\cos(\beta) = \frac{\mathrm{L}_2 + u}{l_2} \approx 1 \tag{24}$$

$$\sin(\beta) = \frac{v}{l_2} \approx 0 \tag{25}$$

By substituting in the force-displacement relationship 12, we obtain the following approximated system of equations:

$$\begin{Bmatrix} P_{x,approx,hard} \\ P_{y,approx,hard} \end{Bmatrix} = \begin{Bmatrix} \frac{\mathrm{A}_1\mathrm{E}_2\left(-\mathrm{L}_1^2+(\mathrm{L}_1+u)^2+v^2\right)}{2\mathrm{L}_1^2} \\ \frac{\mathrm{A}_2\mathrm{E}_2\left(-\mathrm{L}_2^2+(\mathrm{L}_2+v)^2+u^2\right)}{2\mathrm{L}_2^2} \end{Bmatrix} \tag{26}$$

3.5 Comparison for Px

Now, it can be interesting to compare the results obtained with the different linearization/approximation described in the previous sections. For simplicity, we will perform this type of error analysis only for the force $P_x = f(\vec{u})$, but conceptually it would be the same for the force $P_y = f(\vec{u})$.
Since the following discussion doesn't want to be a formal error analysis, we will not use any formal error measure, but we will simply plot the difference between the approximated / linearized solutions and the exact one, that is:

$$z(u,v) = \mathrm{Error}[P_x(u,v)] = P_{x,approximated/linearized}(u,v) - P_{x,exact}(u,v) \tag{27}$$

In this way, the error analysis can be performed graphically, by plotting a 3D surface of the error in the space (u,v,z).
Moreover, since all our linearized / approximated solutions were obtained in the neighborhood of the point $\vec{u} = \vec{0}$ (because of the hypothesis of small displacements), we will limit our plot to a domain [1] of $(u,v) \in [-0.001, 0.001] \times [-0.001, 0.001]$.

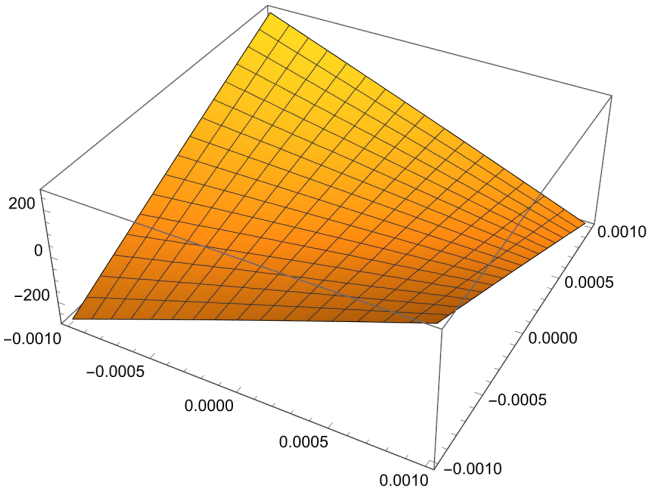


Figure 2: Error analysis for a Taylor series of order 1

[1] At this point, the choice of domain size was made based on a pure intuition of the final value of the displacement

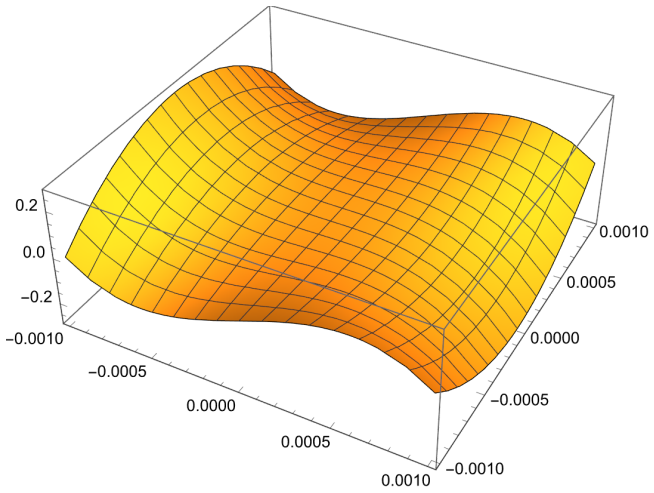


Figure 3: Error analysis for a Taylor series of order 2

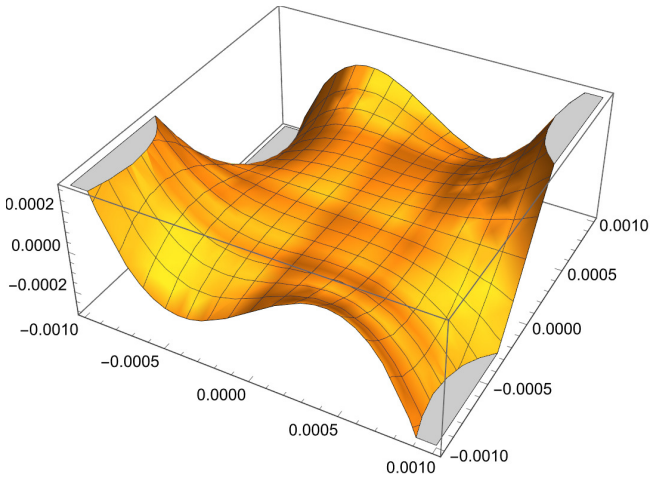


Figure 4: Error analysis for a Taylor series of order 3

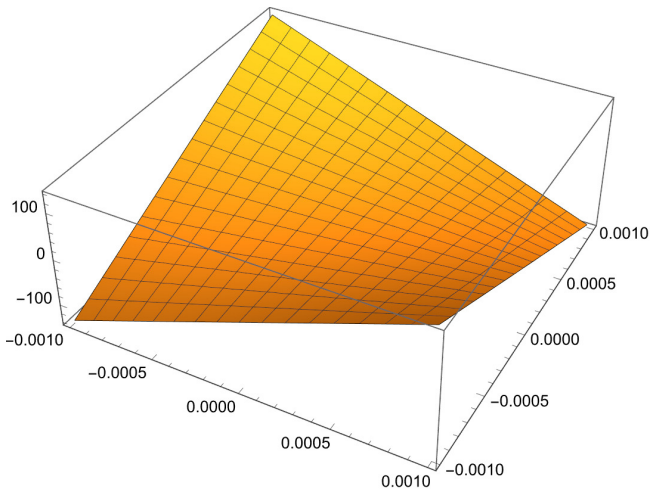


Figure 5: Error analysis for approximated (soft) solution

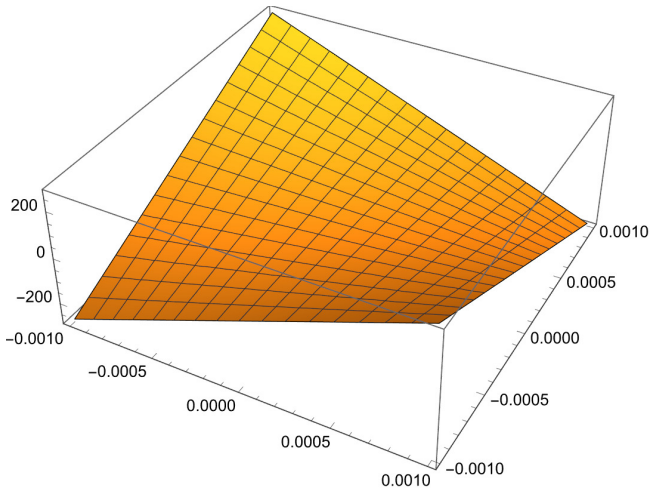


Figure 6: Error analysis for approximated (hard) solution

As we can see, in the neighborhood of point $\vec{u} = \vec{0}$, the error is negligible for every linearization and/or approximations.
However, as we move away from the point $\vec{u} = \vec{0}$, the error increases. In particular, we can see that the error introduced by the linearization decreases as we increase the order of the Taylor series expansion.

4 Numerical solution

We can now proceed to solve the system of equations $\vec{P} = f(\vec{u})$ numerically. To do so, we will use two different combined methods, in the form of Predictor-Corrector, that are:

- Euler and Newton-Raphson
- Euler and modified Newton-Raphson

Just as a review of the methods, we report in the following the algorithms for the tree methods applied to a 2D problem (easily extendable to 3D).
Notice that with the symbol $[K_t]$ we are referring to the tangent stiffness matrix, that by definition is:

$$[K_t] = \frac{\partial \vec{P_{ext}}}{\partial \vec{u}} = \begin{bmatrix} \frac{\partial P_{ext,x}}{\partial u} & \frac{\partial P_{ext,x}}{\partial v} \\ \frac{\partial P_{ext,y}}{\partial u} & \frac{\partial P_{ext,y}}{\partial v} \end{bmatrix} \tag{28}$$

Instead, with the symbol $\vec{R}$ we are referring to the residual vector, that by definition is:

$$\vec{R} = \vec{F_{int}} - \vec{P_{ext}} \tag{29}$$

Since we will work in a discretized domain, we will use the subscript n to refer to the previous iteration, and the subscript $n+1$ to refer to the current iteration. So $[K_{t,n}]$ and $\vec{R_n}$ will simply be the tangent stiffness matrix and the residual vector evaluated at the previous iteration referring to the displacement $\vec{u_n}$.

4.1 Euler

Used as predictor, the Euler method is defined as:

$$\vec{u}_{n+1} = \vec{u}_n + [K_{t,n}]^{-1} * \vec{P_{ext,n}} \tag{30}$$

4.2 Newton-Raphson

Used as corrector, the Newton-Raphson method is defined as:

$$\vec{u}_{n+1} = \vec{u}_n - [K_{t,n}]^{-1} * \vec{R_n} \tag{31}$$

4.3 Modified Newton-Raphson

Similarly to the Newton-Raphson method, the modified Newton-Raphson method is defined as:

$$\vec{u}_{n+1} = \vec{u}_n - [K_{t,0}]^{-1} * \vec{R_n} \tag{32}$$

Where $[K_{t,0}]$ is the tangent stiffness matrix evaluated at the first iteration.

5 Implementation

For each of the previous models (12,14,15,16,21,26) we can then compute the tangent stiffness matrix and the internal force vector as function of $\vec{u}$. By applying the definition 28 and 29 we are able to basically compute for each model two functions $f_{K_t}(\vec{u})$ and $f_R(\vec{u})$ that will return the tangent stiffness matrix and the residual vector evaluated at the current iteration referring to the displacement $\vec{u_n}$.
As an example, we report the functions $f_{K_t}(\vec{u})$ and $f_R(\vec{u})$ for the model 14:

$$f_{K_t}(\vec{u}) = \begin{bmatrix} \frac{\partial P_{ext,x}}{\partial u} & \frac{\partial P_{ext,x}}{\partial v} \\ \frac{\partial P_{ext,y}}{\partial u} & \frac{\partial P_{ext,y}}{\partial v} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial u}\left(\frac{\mathrm{A_1E_1}}{L_1}(u-u_0)\right) & \frac{\partial}{\partial v}\left(\frac{\mathrm{A_1E_1}}{L_1}(u-u_0)\right) \\ \frac{\partial}{\partial u}\left(\frac{\mathrm{A_2E_2}}{L_2}(v-v_0)\right) & \frac{\partial}{\partial v}\left(\frac{\mathrm{A_2E_2}}{L_2}(v-v_0)\right) \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{A_1E_1}}{\mathrm{L_1}} & 0 \\ 0 & \frac{\mathrm{A_2E_2}}{\mathrm{L_2}} \end{bmatrix} \tag{33}$$

$$f_R(\vec{u}) = \vec{F_{int}} - \vec{P_{ext}} \tag{34}$$

$$\vec{F_{int}} = \begin{bmatrix} \frac{\mathrm{A_1E_1}}{\mathrm{L_1}}(u-u_0) \\ \frac{\mathrm{A_2E_2}}{\mathrm{L_2}}(v-v_0) \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{A_1E_1}}{\mathrm{L_1}}u \\ \frac{\mathrm{A_2E_2}}{\mathrm{L_2}}v \end{bmatrix} \tag{35}$$

When it comes to `MATLAB` code, the functions $f_{K_{t,n}}(\vec{u_n})$ and $f_{F_{int,n}}(\vec{u_n})$ are implemented as follows:

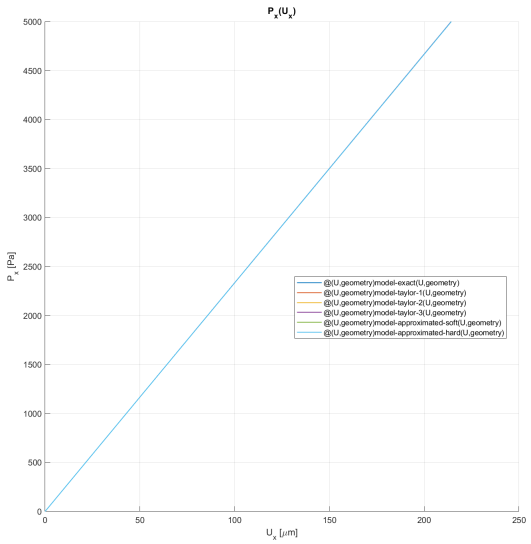
```
function [Kt, Fint] = model_taylor_1(U, geometry)

    [L1, A1, E1, L2, A2, E2] = decompose_geometry(geometry); % From a struct to variables
    [u, v] = decompose_u(U); % From a vector to variables

    Kt = [
        [A1.*E1.*L1.^(-1) 0];
        [0 A2.*E2.*L2.^(-1)]
        ];

    Fint = [
        A1.*E1.*L1.^(-1).*u;
        A2.*E2.*L2.^(-1).*v
        ];

end
```

6 Results

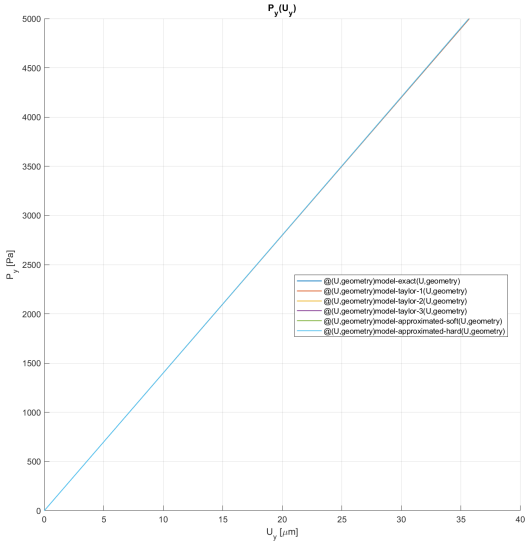
The numerical results for the problem described in Section 1 are reported in the following.

Model name	**Px**	**Py**	**Ux**	**Uy**
model-exact	5000	5000	2.14e-04	3.57e-05
model-taylor-1	5000	5000	2.14e-04	3.57e-05
model-taylor-2	5000	5000	2.14e-04	3.57e-05
model-taylor-3	5000	5000	2.14e-04	3.57e-05
model-approximated-soft	5000	5000	2.14e-04	3.57e-05
model-approximated-hard	5000	5000	2.14e-04	3.57e-05

Table 2: Numerical results for the different models used.



(a) Force versus displacement $P_x(u_x)@P_y = 0$



(b) Force versus displacement $P_y(u_y)@P_x = 0$

Notice how the final displacement are basically the same for all the models. This is due to the fact that the displacement is in the order of 10^{-4} and the error introduced by the linearization or the approximation models is negligible.

6.1 Comparison between different correctors used (CPU time and iterations)

Here follows a graphical comparison of the CPU time and the number of iterations for the different corrector methods used.
Notice that in the legend of the graphs, the following abbreviations are used:

- **NR**: Newton-Raphson method
- **mNR**: modified Newton-Raphson method

Parameter	Value	Unit
Number of steps	1	
ΔP_x	$5 * 10^5$	N
ΔP_y	$5 * 10^5$	N
Tolerance	10^{-7}	N

Table 3: Parameters used for the comparison

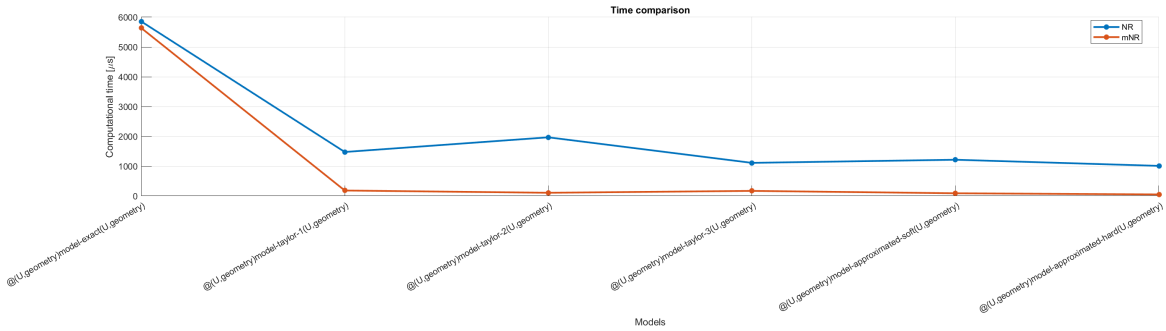


Figure 8: Comparison of CPU time for different correctors used.

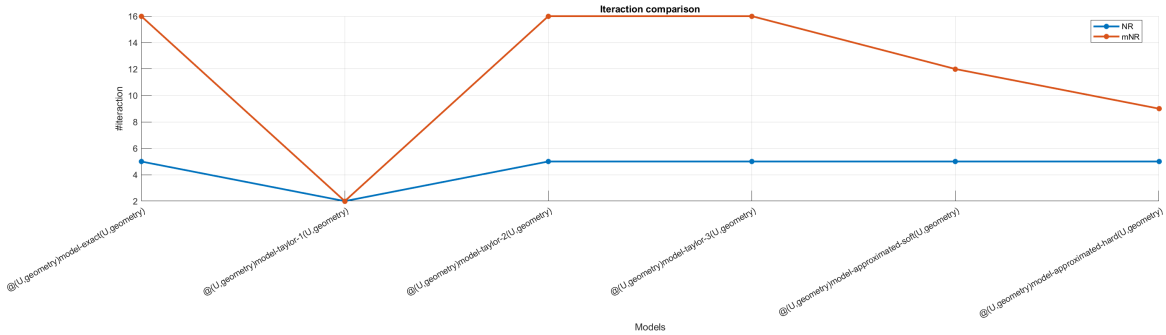


Figure 9: Comparison of number of iterations for different correctors used.

From the graph in Figure 8 we can see that the CPU time for the modified Newton-Raphson method is the lower then the one for the Newton-Raphson method. Instead, in Figure 9 we can see that the number of iterations for the modified Newton-Raphson method is higher than the one for the Newton-Raphson method.
These results align with the theoretical expectations. In the Newton-Raphson method, the computation of the inverse of $[K_{t,n}]$ at each iteration contributes to increased CPU time and enhanced precision in each step, corresponding to a lower iteration number. On the other hand, in the modified Newton-Raphson method, the inverse of $[K_{t,n}]$ is computed only in the initial iteration, leading to reduced CPU time and precision per step, causing a higher iteration number.

6.2 Comparison between different models used (final displacement)

Here follows a graphical comparison of the final displacement for the different models used.

Parameter	Value	Unit
Number of steps	10	
ΔP_x	$5 * 10^5$	N
ΔP_y	$5 * 10^5$	N
Tolerance	10^{-5}	N

Table 4: Parameters used for the comparison

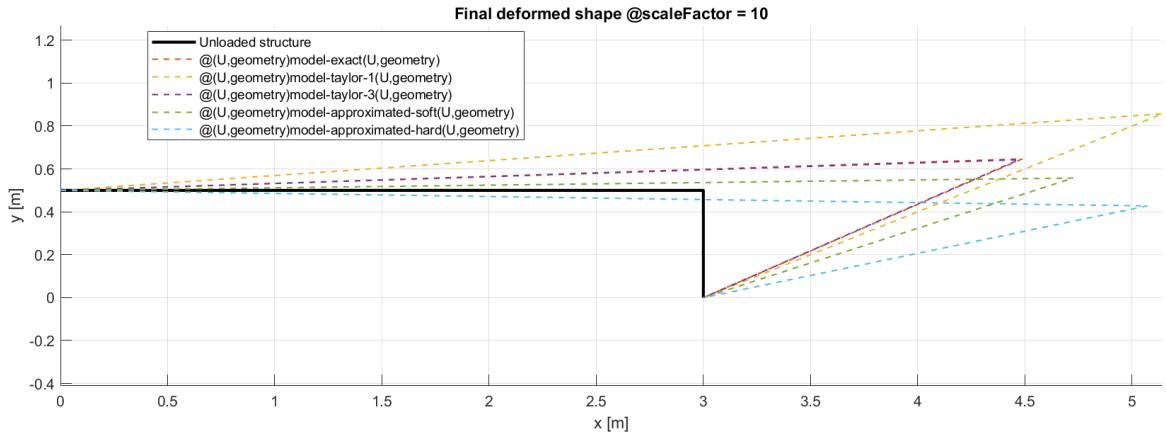


Figure 10: Comparison of final displacement for different models used

Model name	Px	Py	Ux	Uy
model-exact	5000000	5000000	1.49e-01	1.47e-02
model-taylor-1	5000000	5000000	2.14e-01	3.57e-02
model-taylor-3	5000000	5000000	1.47e-01	1.43e-02
model-approximated-soft	5000000	5000000	1.73e-01	5.75e-03
model-approximated-hard	5000000	5000000	2.07e-01	-7.24e-03

Table 5: Numerical results for the different models used.

As we can see, from the graph in Figure 10 and from the table in Table 5, the final displacement for the different models used varies a lot. This behavior is due to the fact that the different models were obtained under the assumption of small displacements. When the displacements becomes bigger (due to high loads), the different models started to diverge from the exact one.
It's also interesting to understand how the different models behave differently for a set of different loads. This means to analyze the trajectory of point B predicted by the different models.

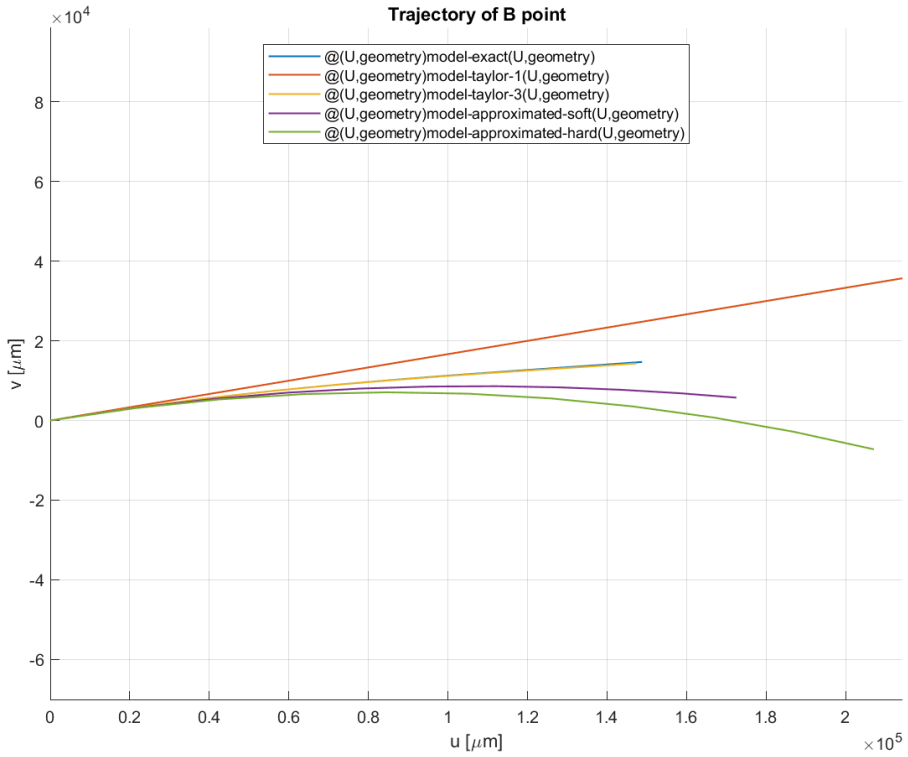
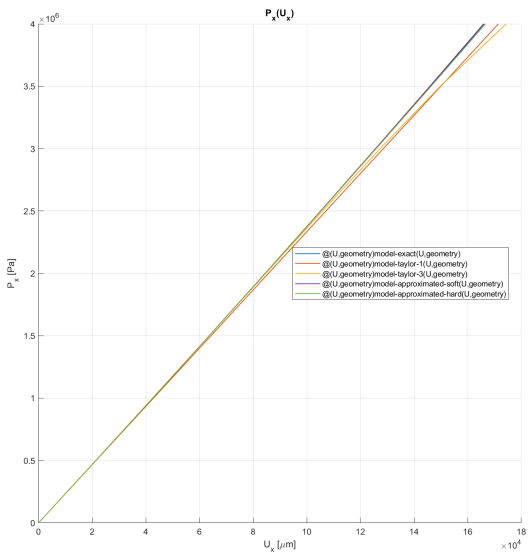
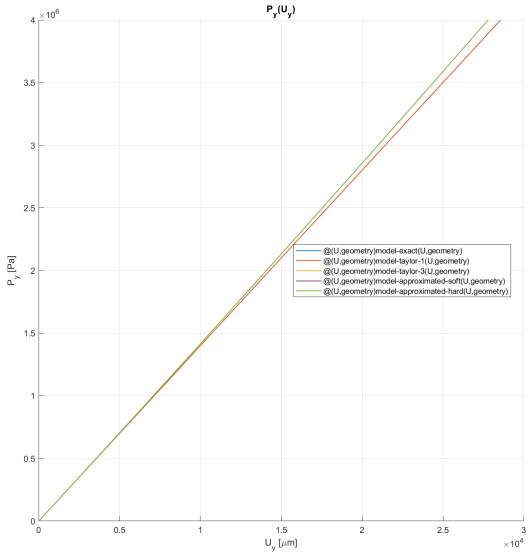


Figure 11: Comparison of trajectory for the different models used

From the graph in Figure 11 we can appreciate the behavior of the linearized models based on the Taylor's order used.
Also, we can appreciate how the approximated models start to diverge and assume a completely wrong trajectory that is of course not feasible in the real world.

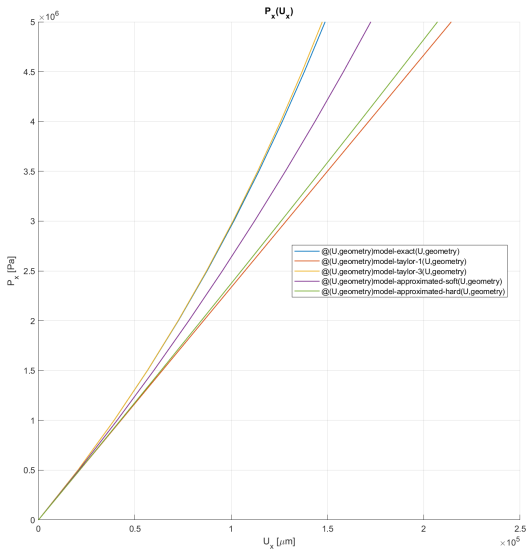


(a) Force versus displacement $P_x(u_x)@P_y = 0$

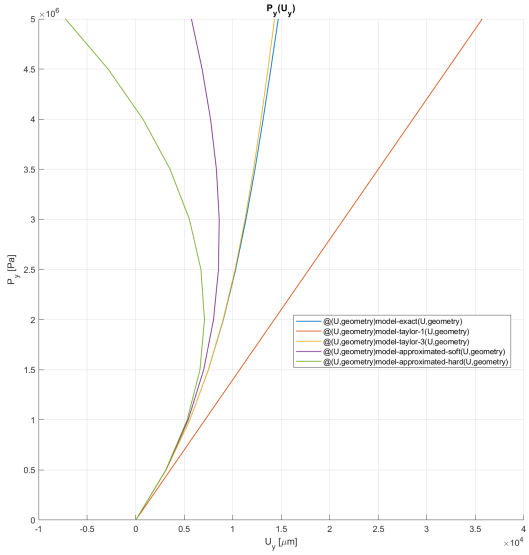


(b) Force versus displacement $P_y(u_y)@P_x = 0$

As we can see from the graph in Figure 13a and in Figure 13b, the error / divergence between the different models increases with the load (as expected).



(a) Force versus displacement $P_x(u_x)$



(b) Force versus displacement $P_y(u_y)$

An interesting behavior of the model is observed under very high loads. In fact, if we plot the same graph as in Figure 12a and in Figure 12b but without keeping zeroing the other load, we can see that the model starts to behave differently. In particular, all the curves force versus displacement get more inclined and eventually start to have a negative slope. This trend is clearly visible for the approximated models.
Another way to visualize this behavior is in Figure 11 and in Figure 10, where in both cases we can clearly see the approximated models diverging to a negative v displacement. Remember that the applied loads P_x and P_y are both positive, so the negative displacement is not a feasible solution. What causes this behavior is still unknown and will probably be investigated in the future.

A Mathematica code

Here follows the `Mathematica` notebook used for symbolic analysis of the equations.

```
Clear["Global`*"];
plots := True;
$Assumptions=Element[{L1,L2,A1,A2,E1,E2,u,v},Reals]&&L1>0&&L2>0&&A1>0&&A2>0&&E1>0&&E2>0;

(*Setting values for the problem datas*)
problemDatas = {
E1-> 70*10^9,A1->10^-3,L1->3,
E2->70*10^9,A2->10^-3,L2->0.5
};

domainPlotLimit:=0.001;

(*Beam lenght*)
l[u_,v_] ={
{Sqrt[(L1+u)^2+v^2]},
{Sqrt[u^2+(L2+v)^2]}
};

(*Green strain*)
eps[u_,v_] = {
{(l[u,v][[1, 1]]^2-L1^2)/(2*L1^2)},
{(l[u,v][[2, 1]]^2-L2^2)/(2*L2^2)}
};

(*F_{int}*)
Fint[u_,v_]={
{E1*A1*eps[u,v][[1,1]]},
{E2*A2*eps[u,v][[2,1]]}
};

(*Trigonometry relations*)
cosAlpha[u_,v_]=(L1+u)/l[u,v][[1, 1]];
sinAlpha[u_,v_]=v/l[u,v][[1, 1]];
cosBeta[u_,v_]=(L2+v)/l[u,v][[2, 1]];
sinBeta[u_,v_]=u/l[u,v][[2, 1]];

(*Equilibrium of the system*)
Pexact[u_,v_]={
{Fint[u,v][[1, 1]]*cosAlpha[u,v]+Fint[u,v][[2, 1]]*sinBeta[u,v]},
{Fint[u,v][[1, 1]]*sinAlpha[u,v]+Fint[u,v][[2, 1]]*cosBeta[u,v]}
};

(*Defining the taylor expansion series*)
Ptaylor[x_, y_, taylorOrder_] := Normal @ Series[Pexact[u, v] /. Thread[{u, v} -> {0, 0} + t
    {x, y}], {t, 0, taylorOrder}] /. t -> 1;

(*Approximated models*)
PapproximatedSoft[u_,v_]={
{Fint[u,v][[1, 1]]*1+Fint[u,v][[2, 1]]*u},
{Fint[u,v][[1, 1]]*v+Fint[u,v][[2, 1]]*1}
};

PapproximatedHard[u_,v_]={
{Fint[u,v][[1, 1]]*1+Fint[u,v][[2, 1]]*0},
{Fint[u,v][[1, 1]]*0+Fint[u,v][[2, 1]]*1}
};
```

Listing 1: Mathematica notebook used for symbolic analysis of the equations.