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

Computational aerospace engineering

Assingment 1

Finite element method

Andreu Craus Vidal

Roger Tatjé i Ramos

Greta- Group 1

Professor

Joaquín A. Hernández Ortega
Raúl Rubio Serrano

Thursday 2nd January, 2025

Contents

List of Figures

Statement of the assignment

Suppose a bar of length L of constant cross-sectional area A and Young's Modulus E. The bar has a prescribed displacement at $x = 0$ equal to $u(0) = -g$, and it is subjected to, on the one hand, an axial force F on the right end ($x = L$), and on the other hand, a distributed axial force (per unit area) given by the expression:

$$q(x) = E\left(\rho u(x) - sr(\bar{x})\right) \tag{1}$$

where $u = u(x)$ is the unknown displacement function, $r = r(\bar{x})$ is an input function, $\bar{x} = x/L$, and

$$\rho = \frac{\pi^3}{L^2}, \quad s = \frac{g\pi^4}{L^2}, \quad \frac{F}{AE} = \frac{g\pi^2}{L} \tag{2}$$

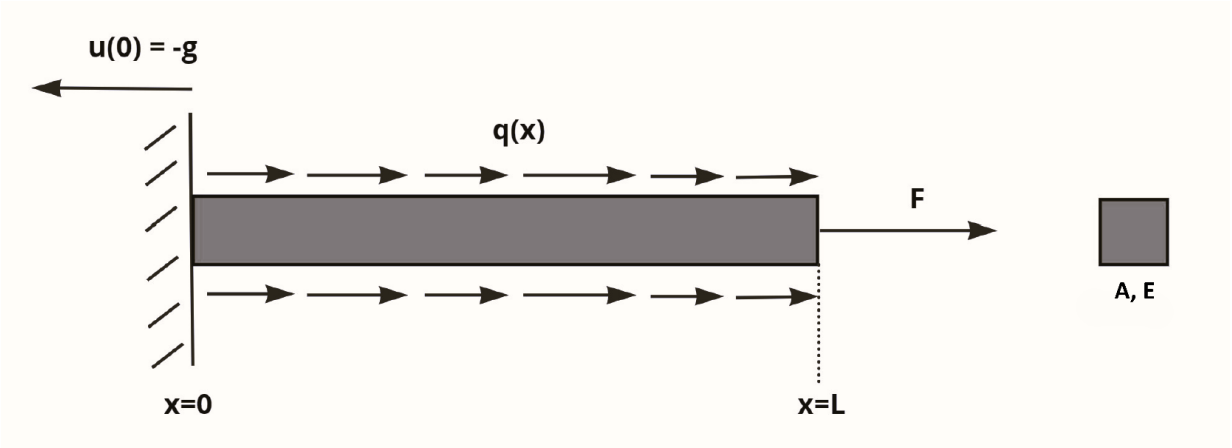


Figure 1: Illustration of the problem statement.

1 Part 1

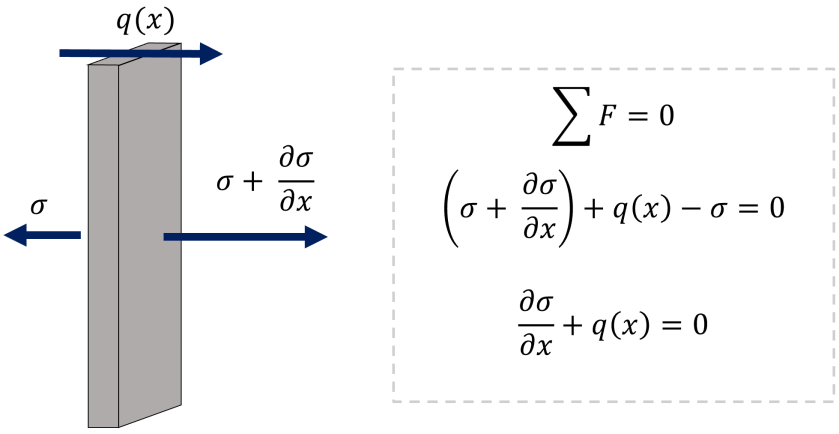


Figure 2: Axial equilibrium in a differential of length.

1.1 Derivation of the corresponding Boundary Value Problem (BVP) for the displacement field $u : [0, L] \to \mathbb{R}$ using the equilibrium equation for 1D problems (strong form)

The strong form of the boundary value problem (BVP) for the displacement field will be determined by $u''(x)$ alongside the boundary conditions of $u(x)$ and $u'(x)$. The first boundary condition is provided by:

$$u(0) = -g \tag{3}$$

To begin, the axial equilibrium equation seen in Figure 2 is applied to the scenario described. Equation 4 represents the balance of forces for a bar under a distributed axial force $q(x)$, which is position-dependent. Here, $\sigma : \bar{\Omega} \to \mathbb{R}$ represents the stress field over the domain $\bar{\Omega} = [0, 1]$.

$$\frac{\partial \sigma}{\partial x} + q(x) = 0 \tag{4}$$

Next, Hooke's law for a linear elastic material gives the relationship between the stress and displacement fields:

$$\sigma = E\,\varepsilon = E\,\frac{\partial u}{\partial x} = E\,u' \tag{5}$$

By substituting Equation 5 into Equation 4, the following expression is derived:

$$E\,\frac{\partial^2 u}{\partial x^2} + q(x) = 0 \tag{6}$$

Next, applying the definition of $q(x)$ given in the problem statement (Equation 1):

$$u'' + \rho\,u - s\,r\left(\frac{x}{L}\right) = 0 \tag{7}$$

At $x = L$ (the Neumann boundary condition), the stress definition $\sigma = F/A$ it is applied, along with the relationship in Equation 5:

$$\varepsilon = \frac{\partial u}{\partial x}, \quad \sigma = E\,u' \implies u'(L) = \frac{F}{A\,E} \tag{8}$$

By substituting the expression from the problem statement (Equation 2), we derive the following boundary condition:

$$u'(L) = \frac{g\,\pi^2}{L} \tag{9}$$

Finally, the strong form of the boundary value problem (BVP) for the displacement field are summarized as follows:

$$\begin{cases} u(0) + g = 0 \\ u'(L) - \frac{F}{EA} = 0 \implies u'(L) - \frac{g\,\pi^2}{L} = 0 \\ u''(x) + \frac{q(x)}{E} = 0 \implies u''(x) + \rho u(x) - sr\left(\frac{x}{L}\right) = 0 \end{cases} \tag{10}$$

1.2 Exact solution of the BVP

This section aims to find and plot the exact solution of the BVP, using the following values for the involved constants:

$$L = 1\ m, \quad g = 0.01\ m, \quad r = (x/L)^2 \tag{11}$$

Consequently, the equation could be rewritten in the following way:

$$u'' + \rho\,u - s\,\frac{x^2}{L^2} = 0 \tag{12}$$

The exact solution to the equilibrium equation, derived from the strong form of the problem, was found by incorporating the boundary conditions. Using the Matlab code provided in Appendix A.1, the following displacement function $u(x)$ was computed:

$$u(x) = 0.0155\sin(5.57\,x) - 0.00797\cos(5.57\,x) + 0.0314\,x^2 - 0.00203 \tag{13}$$

Note that the above expression has been rounded to three significant figures for simplicity. The displacement field $u(x)$, as a function of position x along the bar, can now be plotted based on the result from Equation 13, resulting in the following graph:

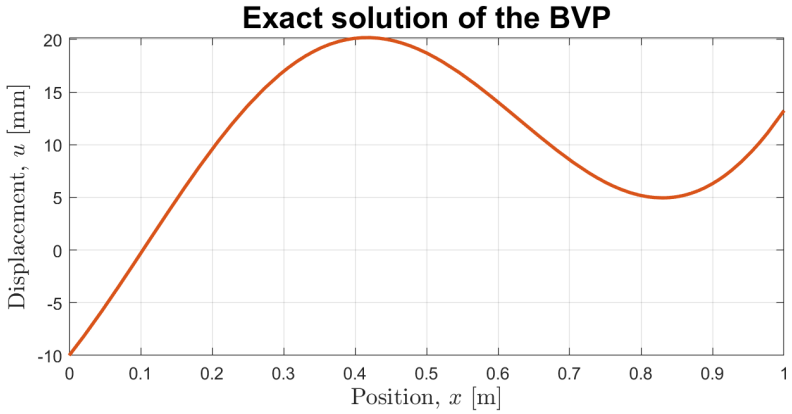


Figure 3: Graph of the exact solution of the displacement field $u(x)$ in the interval $x \in [0, L]$

.

1.3 Formulation of the *variational* (or *weak*) form of the BVP

In order to derive the weak form of the boundary value problem (BVP), we start with the strong form of the differential equation found in Section 1.1:

$$u'' + \rho\,u - s\,\frac{x^2}{L^2} = 0 \tag{14}$$

The boundary conditions are given as:

$$u(0) = -g \quad \text{(Dirichlet boundary condition)} \tag{15}$$

$$u'(L) = \frac{g\pi^2}{L} \quad \text{(Neumann boundary condition)} \tag{16}$$

It is assumed that the unknown solution can be expressed as a linear combination of a set of known basis functions.

$$u(x) \approx \sum_{i=1}^{n} N_i(x) d_i = \underline{\underline{N}}(x) \cdot \underline{d} = N_1 d_1 + N_2 d_2 + N_3 d_3 ... \tag{17}$$

Where $u(x)$ is the unknown function to be determined, $N(x)$ represents the known solution functions, and the vector d contains the unknown coefficients to be determined.

Variational methods do not focus on approximating the differential operator but rather on approximating the functions. The goal is to determine the smallest possible residual (the error with respect to the exact solution).

$$r(x;u) := u'' + \rho\, u - s\, \frac{x^2}{L^2} = 0 \tag{18}$$

The residual is interpreted as residual forces, since the equilibrium between internal and external forces cannot be guaranteed at every point. In this method, instead of minimizing the residual forces, the focus will be on minimizing the work done by these residual forces.

$$W = \int_0^L u(x) r(x;u) dx \tag{19}$$

To facilitate the subsequent computation of the integral, a subtle technique is introduced.

$$u(x) = g + [u(x) - g] \rightarrow W = \int_0^L g r(x;u) dx + \int_0^L [u(x) - g] r(x;u) dx \tag{20}$$

The first term cannot be zero because, in the general case, the residual will be nonzero. However, the focus now shifts to the second term, which will be made zero by finding the vector d hat achieves this. It is also important to note that the residual is determined by $u(x)$, and our goal is to find the solution for $u(x)$. Thus, the terms become:

$$\int_0^L [u(x) - g] r(x;u) dx \rightarrow \int_0^L [(\underline{\underline{N}}\ \underline{d} - g)(\underline{\underline{N''}}\ \underline{d} + \rho \underline{\underline{N}}\ \underline{d} + \underbrace{f(x)}_{-s\frac{x^2}{L^2}}] \tag{21}$$

1.3.1 Equation resolution

One of the requirements of the previous equation is that the displacement at $x(0) = -g$. Consequently, and interpreting the equation as a dot product:

$$u(0) = N(0)d = g \rightarrow d_1 = g \tag{22}$$

Now, the problem will lie in finding the remaining coefficients d_2, d_3, d_4 ...

The equation 21 is rewritten as follows:

$$\int_0^L v(x) r(x;u) = 0 \rightarrow \text{For all } v(x) = \underline{\underline{N}}\ \underline{C}; v(0) = 0 \rightarrow \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \tag{23}$$

Let $v(x)$, the test function, be a translated version of $u(x)$ to the origin. Additionally, $v(x)$ must be orthogonal to the residual. Therefore, to satisfy this orthogonality condition, we will multiply the corresponding functions and compute the integral over the domain, which must equal zero.

$v(x)$ resides in the space generated by the functions $N_1(x), N_2(x), N_3(x)...$ but this concept must be extended to the case where the function space is not limited to a finite number of components.

Thus, the previous expression is integrated over the domain $[0, L]$:

$$\int_0^L \left(u''(x) + \rho u(x) + f(x)\right) v(x)\, dx = 0 \tag{24}$$

1.3.2 Integration by Parts (Elimination of the second derivative)

Next, the second derivative term $\frac{d^2u}{dx^2}$ is addressed by applying integration by parts. This allows us to reduce the second-order derivative in $u(x)$ to a first-order derivative:

$$\int_0^L u''(x)v(x)\, dx = \left[u'(x)v(x)\right]_0^L - \int_0^L u'(x)v'(x)\, dx \tag{25}$$

Substituting this into the equation 24:

$$\left[u'(x)v(x)\right]_0^L - \int_0^L u'(x)v'(x)\, dx + \int_0^L \rho u(x)v(x)\, dx - \int_0^L s\frac{x^2}{L^2}v(x)\, dx = 0 \tag{26}$$

1.3.3 Application of Boundary Conditions

At $x = 0$, the Dirichlet boundary condition $u(0) = -g$ is applied. This imposes that $v(0) = 0$, since the test function must vanish at this boundary.

At $x = L$, the Neumann boundary condition $u'(L) = \frac{g\pi^2}{L}$ is applied. Therefore, the boundary term at $x = L$ becomes:

$$u'(L)v(L) = \frac{g\pi^2}{L}v(L) \tag{27}$$

By substituting this into the weak form, one obtains:

$$\frac{g\pi^2}{L}v(L) - \int_0^L u'(x)v'(x)\, dx + \int_0^L \rho u(x)v(x)\, dx - \int_0^L s\frac{x^2}{L^2}v(x)\, dx = 0 \tag{28}$$

Thus, the final weak form of the BVP can be written as:

$$\boxed{\int_0^L v'(x)\cdot u'(x)\, dx - \rho\int_0^L v(x)\cdot u(x)\, dx = -\frac{s}{L^2}\int_0^L v(x)\cdot x^2\, dx + v(L)\cdot\frac{g\pi^2}{L}} \tag{29}$$

Thus, it has been successfully derived the weak form of the boundary value problem by using integration by parts to eliminate second-order derivatives and applying the appropriate boundary conditions. The result expresses the problem in a form that can be used for further analysis, such as finite element approximation.

1.4 Matrix equation

The Galerkin method will now be applied to obtain a solution for expression X. The idea is to construct a finite-dimensional space that approximates $\mathcal{S}$ and $\mathcal{V}$. This space is spanned by all functions that can be written as a linear combination of shape functions. As the number of shape functions increases, the space $\mathcal{S}^h$ grows accordingly, such that if the number of functions becomes infinite, it approaches the exact solution.

1.4.1 Matrix form

The goal of obtaining a matrix form is to transform Equation 29 into an algebraic system. Before doing so, a series of terms (replacing u and v by linear expansions) will be included in that equation.

$$\underbrace{u}_{1x1} = \underbrace{\underline{\underline{N}}}_{1xn}\ \underbrace{\underline{d}}_{nx1}\ ; \qquad \overbrace{\underbrace{v}_{1x1} = \underbrace{\underline{\underline{N}}}_{1xn}\ \underbrace{\underline{C}}_{nx1} = v^T = \underline{C}^T\ \underline{\underline{N}}^T}^{\text{It's a scalar value and, consequently, can be transposed}}$$
$$v' = \underline{C}^T\ \underline{\underline{B}}^T \text{ where: } \underline{\underline{B}} = \underline{\underline{N}}' = \frac{dN}{dx} \tag{30}$$

It can be observed that both v and u belong to the same family of functions, represented by N. Substituting these terms into the relevant equation yields:[1]

$$\int_0^L C^T B^T B d\,dx - \rho \int_0^L C^T N^T N d\,dx = \int_0^L C^T N^T \underbrace{\left(-s\frac{x^2}{L^2}\right)}_{f(x)} dx + \frac{g\pi^2}{L} C^T N(1)^T \tag{31}$$

The common factor C^T is factored out:

$$C^T \underbrace{\left(\int_0^L B^T B dx - \rho \int_0^L N^T N dx\right)}_{:=K} d = C^T \underbrace{\left(\int_0^L N^T f(x) dx + g\frac{\pi^2}{L} N(1)^T\right)}_{:=F} \rightarrow \boxed{\underline{C}^T \left(\underline{\underline{K}}\ \underline{d} - \underline{F}\right) = 0} \tag{32}$$

The validity of these equations depends on the coefficients c meeting the requirement $v(0) = N(0)$ and $c = 0$. In the case of polynomial and finite element basis functions, this simplifies to $v(0) = c_1 = 0$, thereby defining c as follows where $l = \{2, 3, ...\}$:

$$\begin{aligned} \underline{C} &= \begin{bmatrix} \underline{C_r} \\ \underline{C_l} \end{bmatrix} = \begin{bmatrix} 0 \\ \underline{C_l} \end{bmatrix} \rightarrow \text{Determined unknowns} \\ & \qquad\qquad\quad \rightarrow \text{Unknown} \\ \underline{\underline{K}} &= \begin{bmatrix} \underline{\underline{K_r}} \\ \underline{\underline{K_l}} \end{bmatrix} = \begin{bmatrix} 0 \\ \underline{\underline{K_l}} \end{bmatrix} \rightarrow \text{1st row} \\ & \qquad\qquad\quad \rightarrow \text{It is decomposed into rows} \\ \underline{d} &= \begin{bmatrix} \underline{d_r} \\ \underline{d_l} \end{bmatrix} = \begin{bmatrix} -g \\ \underline{d_l} \end{bmatrix} \rightarrow \text{Known displacement in x=0} \\ & \qquad\qquad\quad \rightarrow \text{Unknown coefficients} \end{aligned} \tag{33}$$

Where r is the list of known DOFs and l is the list of unknown DOFs.

Therefore, beginning from equation 32, the following result is obtained:

$$\begin{bmatrix} 0 & C_l^T \end{bmatrix} \left[\begin{pmatrix} K_r \\ K_l \end{pmatrix} d - \begin{pmatrix} F_r \\ F_l \end{pmatrix} \right] = 0 \rightarrow \boxed{C_l^T \left(K_l d - F_l\right) = 0} \tag{34}$$

The previous result must hold for any component of C. Consequently, the expression within the parentheses must evaluate to zero.

[1] From this point onward, matrices and vectors of the same type are assumed, but the corresponding horizontal bar notation will not be explicitly indicated.

$$Cl^T(K(l,:)d - F(l)) = 0; \forall C_l \rightarrow K(l,:)d = F(l) \rightarrow K(l,l)d(l) + K(l,r)\underbrace{d_r}_{-g} - F(l) = 0 \rightarrow$$
$$\rightarrow \boxed{K(l,l)d_l = F(l) + K(l,r)g} \tag{35}$$

Where the unknown to be determined is the vector of coefficients d_l.

$$dl = K_{l,l}^{-1} \cdot (F(l) + K(l,r)g) \tag{36}$$

1.4.2 Final result

Finally, the expression for the approximate function can be written as:

$$u(x) = \underline{\underline{N}}\ \underline{d} = N(1)d_1 + N(2)d_2 + N(3)d_3... \rightarrow u = g + N(l)d(l) \tag{37}$$

1.5 Approximation using basis polynomial basis functions (up to 6)

Next, sections 1.3 and 1.4 will be solved computationally for the relevant equation in this part. The main routine can be found in Appendix A.2, along with the auxiliary function, which is thoroughly commented.

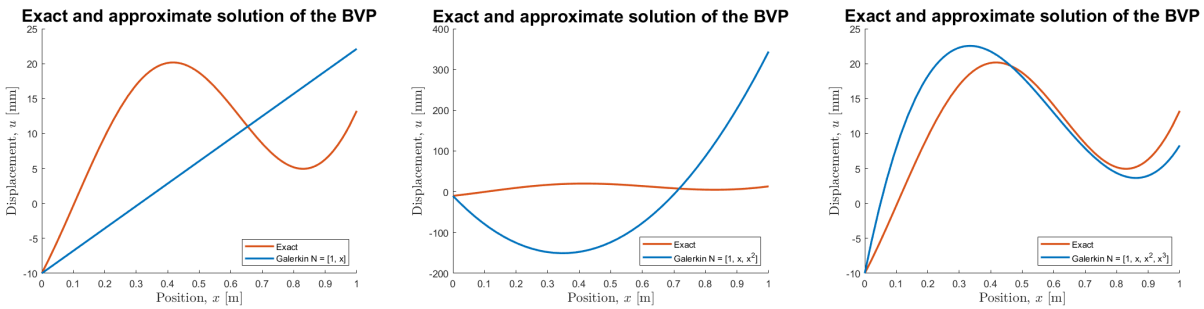
In this section, the following shape functions are presented, which will be used to solve the developed problem applying galerkin method.

$$N = [1, x],\ N = [1, x, x^2],\ \ldots\ N = [1, x, \ldots x^i],\ \ldots\ N = [1, x, \ldots, x^6] \tag{38}$$

The following approximation of $u(x)$ were obtained using the Matlab code:

- $x^1 \rightarrow u_1(x) = -0.01 + 0.03212x$
- $x^2 \rightarrow u_2(x) = -0.01 - 0.809x + 1,163x^2$
- $x^3 \rightarrow u_3(x) = -0.01 + 0.2236x - 0.4643x^2 + 0.259x^3$
- $x^4 \rightarrow u_4(x) = -0.01 + 0.1101x - 0.07615x^2 - 0.4948x^3 + 0.3396x^4$
- $x^5 \rightarrow u_5(x) = -0.01 + 0.07668x - 0.3174x^2 - 1.239x^3 + 1.231x^5 - 0.3625x^5$
- $x^6 \rightarrow u_6(x) = -0.01 + 0.08453x + 0.1832x^2 - 0.5916x^3 - 0.07676x^4 + 0.8161x^5 - 0.3922x^6$

Below, the various plots along with the exact solution can be seen.



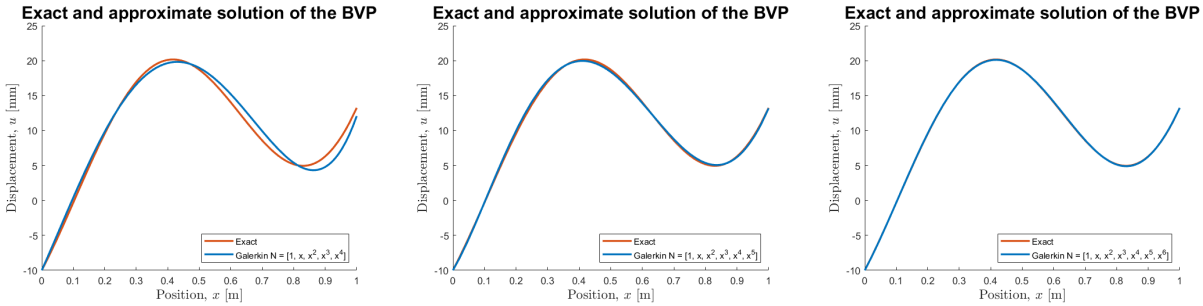


Figure 4: Exact and approximate solution for the different shape functions.

1.6 Linear finite element basis functions

Up to this point, the work has focused on a globally defined function over the entire domain. From now on, the approach will shift to a global description that pertains to each individual element.

1.6.1 Description of an element

For each element, a set of local components will be introduced, as we transition from the domain of x to the parent domain $\Omega = [\xi_1, \xi_2] = [-1, 1]$. This parent domain will have a different shape function for each node, expressed in terms of the variable ξ as follows.

$$N^e(\xi) := \begin{bmatrix} N_1^e(\xi) \\ N_2^e(\xi) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1-\xi \\ 1+\xi \end{bmatrix} \tag{39}$$

In the same way as for the global approach, the displacement of each element can also be calculated where in this case, the coefficients d^e refers to those specific to the element.

$$u^e(\xi) = \begin{bmatrix} N_1^e(\xi) & N2^e(\xi) \end{bmatrix} \begin{bmatrix} d_1^e \\ d_2^e \end{bmatrix} = N^e(\xi) d^e \tag{40}$$

1.6.2 Relationship of domains, isoparametric elements

It is important to reference the global domain from the element domain and vice versa. Therefore, the following relationship is established where we can calculate the global parameter x from local ones ξ.

$$x^e = \begin{bmatrix} N_1^e(\xi) & N_2^e(\xi) \end{bmatrix} \overbrace{\begin{bmatrix} x_1^e \\ x_2^e \end{bmatrix}}^{\mathbf{x^e}} = \mathbf{N^e}(\xi)\mathbf{x^e} \tag{41}$$

1.6.3 Derivation of the stiffness matrix K

The derivation of the stiffness matrix K will be demonstrated below. However, this can also be solved using computational tools, as shown in Appendix A.3.

The derivation starts from the expression of Equation 29. The objective is to transform this equation into one that can be calculated element by element, and subsequently assembled with the matrices of other elements. For the stiffness matrix term, those terms involving the displacement $u(x)$ will be considered.

$$\underbrace{\int_0^1 (v'u')dx}_{1} - \underbrace{\int \rho vudx}_{2} \tag{42}$$

Term 1

Transforming to the element domain

$$\int_0^1 (v'u')dx = \int_0^1 \frac{v^h}{dx}\frac{du^h}{dh}dx = \sum_{e=1}^{n_{elem}} \int_{\Omega^e} \frac{dv^e}{dx}\frac{du^e}{dx}dx \tag{43}$$

Before proceeding, a set of relationships must be defined.

$$\frac{du^e}{dx} = \frac{dN^e}{dx}d^e = B^e d^e \;\; ; \frac{dv^e}{dx} = \frac{dN^e}{dx}C^e = \overbrace{B^e C^e}^{Scalar} = (B^e C^e)^T = C^{e^T} B^{e^T} \tag{44}$$

Thus:

$$C^{e^T} \underbrace{\left(\int_{\Omega^e} B^{e^T} B^e dx\right)}_{K^e} d^e = C^{e^T} K^e d^e \tag{45}$$

The expression for the elemental matrix K has now been obtained. However, both the variables x and ξ still appear in the expression, which is not acceptable. Therefore, additional relationships must be defined.

$$B^e := \frac{dN^e}{dx} = \frac{dN^e}{d\xi}\frac{d\xi}{dx} = \frac{1}{2}\begin{bmatrix}-1 & 1\end{bmatrix}\frac{d\xi}{dx}$$

$$\Downarrow \quad \frac{d\xi}{dx} = \left(\frac{dx}{d\xi}\right)^{-1} = \left(\frac{dN^e}{d\xi}\overbrace{x^e}^{[x_1^e x_2^e]}\right)^{-1} = 2\underbrace{(-x_1^e + x_2^e)^{-1}}_{-h^e} = \frac{2}{h^e} \tag{46}$$

$$B^e = \frac{1}{2}\begin{bmatrix}-1 & 1\end{bmatrix}\frac{2}{h^e} \rightarrow \boxed{B^e = \frac{1}{h^e}\begin{bmatrix}-1 & 1\end{bmatrix}}$$

Now, this term can be introduced into the elemental expression of K^e.

$$K^e = \int_{\Omega^e} \overbrace{\left(\frac{1}{h^e}\begin{bmatrix}-1 & 1\end{bmatrix}\right)^T}^{B^{e^T}} \overbrace{\frac{1}{h^e}\begin{bmatrix}-1 & 1\end{bmatrix}}^{B^e} dx = \frac{1}{(h^e)^2}\begin{bmatrix}-1\\1\end{bmatrix}\begin{bmatrix}-1 & 1\end{bmatrix}\int_{\Omega^e} dx \tag{47}$$

$$\boxed{K^e = \frac{1}{h^e}\begin{bmatrix}1 & -1\\-1 & 1\end{bmatrix}}$$

Term 2:

For this term it will be proceed in a similar way.

$$\int_0^1 \rho vudx = \int_{\Omega^e} \rho v^e u^e dx = \rho \int_{\Omega} N^e C^e N^e d^e dx = \rho \int_{\Omega^e} C^{e^T} N^{e^T} N^e d^e dx = C^{e^T} \overbrace{\rho \int_{\Omega^e} N^{e^T} N^e dx}^{K^e} d^e \tag{48}$$

Introducing the same relations than in 44 and 46 it can be obtained the following expression.

$$K^e = \rho \int_{\Omega^e} \frac{1}{2}\begin{bmatrix}1-\xi\\1+\xi\end{bmatrix}\frac{1}{2}\begin{bmatrix}1-\xi & 1+\xi\end{bmatrix}\frac{h^e}{2}d\xi = \frac{\rho h^e}{4}\begin{bmatrix}\xi-\xi^2+\frac{\xi^3}{3} & \xi-\frac{\xi^3}{3}\\ \xi-\frac{\xi}{3} & \xi+\xi`2+\frac{\xi^3}{3}\end{bmatrix}\Bigg|_{-1}^{1} \qquad \boxed{k^e = \frac{\rho h^e}{6}\begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}} \tag{49}$$

Therefore, by combining both expressions, the final elemental stiffness matrix K is obtained.

$$\boxed{\boxed{K^e(\xi) = \frac{1}{h^e}\begin{bmatrix}1 & -1\\-1 & 1\end{bmatrix} - \frac{\rho h^e}{6}\begin{bmatrix}2 & 1\\1 & 2\end{bmatrix}}} \tag{50}$$

For the force vector, the same procedure will be followed; however, in this case, the expression will not be developed explicitly. The resulting expression is obtained using MATLAB, as shown in Appendix A.3.

$$F = \int_0^1 v \overbrace{f}^{-s\frac{x^2}{L^2}} dx + g\frac{\pi^2}{L}N^T(1) = \sum_{e=1}^{n_{elem}} \int_{\Omega^e} v^e f dx = C^T \sum_{e=1}^{n_{elem}} L^{e^T} \overbrace{\int_{\Omega^e} N^{e^T} f dx}^{\overline{F}^e} \tag{51}$$

Adding the corresponding terms:

$$\overline{F}^e = \int_{\Omega^e} N(\xi)^{e^T}\left(-s\frac{x^2}{L^2}\right)\frac{h^e}{2}d\xi \rightarrow \boxed{\boxed{\overline{F}^e = \frac{sh^e}{12L^2}\begin{bmatrix}3x_1^2+2x_1x_2+x_2^2\\x_1^2+2x_1x_2+3x_2^2\end{bmatrix}}} \tag{52}$$

This expression is for the distributed force. Eventually, at the nodes where there is a Force applied this must be included in the

1.6.4 Assembly of matrices

Finally, the matrices are assembled according to the corresponding nodes. Additionally, it is important to note that in Equation 51, there was a term corresponding to a point force, which must also be included in the force vector.

1.6.5 Solution of the algebraic system

Once the global matrices K and F have been obtained, the system of equations can be solved to determine the nodal displacements along the x-axis.

$$\begin{bmatrix}K_{rr} & K_{rl}\\K_{lr} & K_{ll}\end{bmatrix}\begin{bmatrix}d_r & d_l\end{bmatrix} = \begin{bmatrix}F_r\\F_l\end{bmatrix} \tag{53}$$

Where d_l are the unknown displacements and d_r the known ones[2], thus:

$$K_{lr}d_r + K_{ll}d_l = F_l \rightarrow \boxed{d_l = (K_{ll})^{-1}(F_l - K_{lr}d_r)} \tag{54}$$

[2]Dirichlet Condition of $u(0) = -g$

1.7 Solution for an increasing number of finite elements

In this section, the results from the previous part are presented for an increasing number of elements (5, 10, 20, 40), showing how the solution progressively converges towards the exact solution.

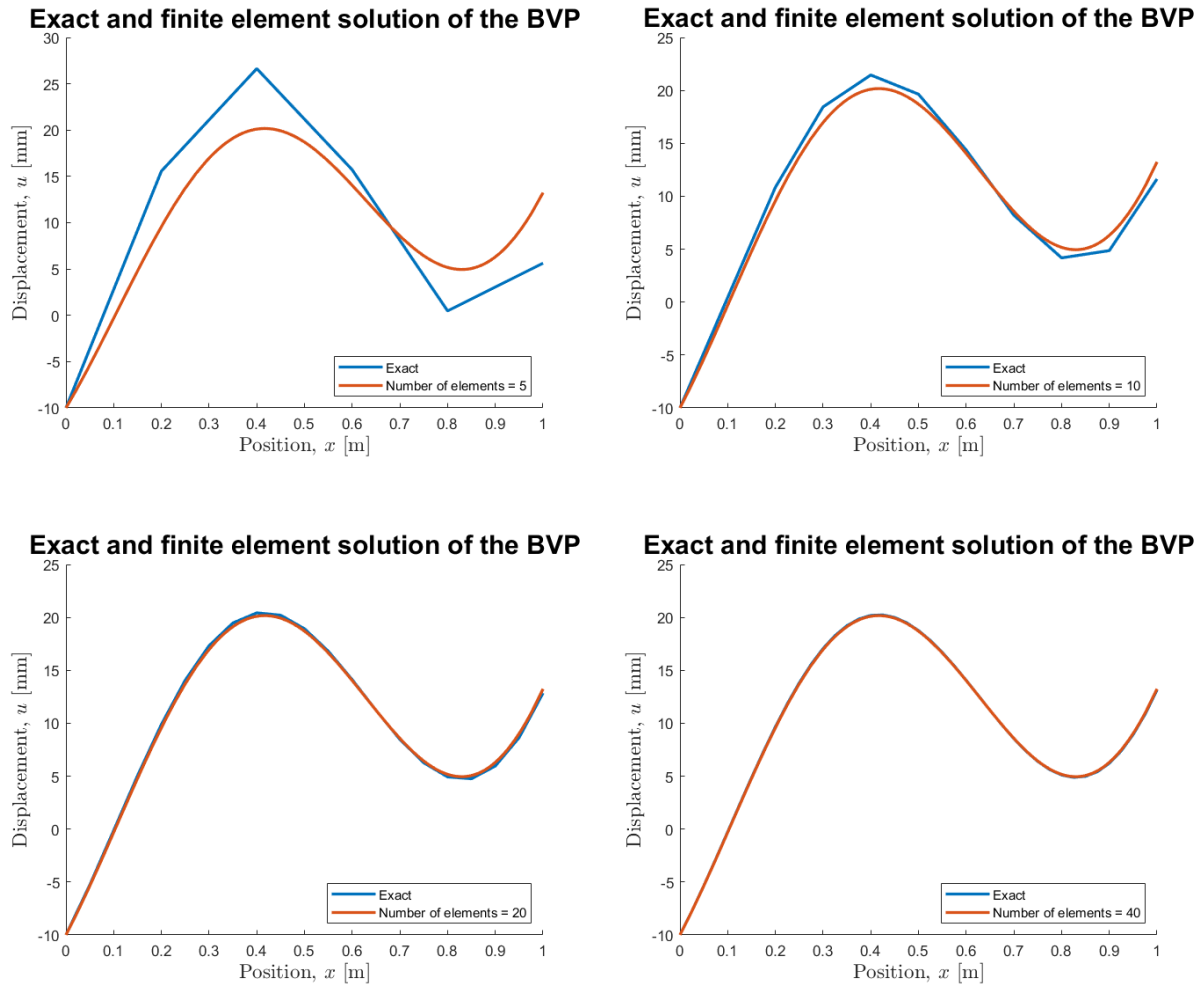


Figure 5: Exact and finite element solution for different number of elements.

2 Part 2

The second part of this assignment aims to find the error between the exact solution and FEM approximation of the $u(x)$ and $u'(x)$ functions. In physical terms, this would correspond to the error in displacements and stresses, respectively.

The error is defined and computed as follows:

$$\|e\| = \|u - u^h\| = \left(\int_\Omega (u - u^h)^2 \, dx \right)^{\frac{1}{2}} \tag{55}$$

$$\|e'\| = \|u' - u'^h\| = \left(\int_\Omega (u' - u'^h)^2 \, dx \right)^{\frac{1}{2}} \tag{56}$$

where u^h refers to the finite element approximation and u refers to the exact solution found in Section 1.2 (expression 13).

In a discretized setting, this can be expressed as:

$$\|e\| = \left(\sum_{e=1}^{n_{el}} \int_0^1 (u - N\,d)^2 dx \right)^{\frac{1}{2}} \tag{57}$$

Next, a change of variable can be applied using relationship 46:

$$\int_0^1 (u - N\,d\hat{e})^2 dx \xrightarrow{dx = \frac{h^e}{2} d\xi} \frac{h^e}{2} \int_{-1}^1 \underbrace{\left(u(x) - N(\xi)\,d^2\right)^2}_{q(\xi, x(\xi))} d\xi \tag{58}$$

obtaining the following final expression:

$$\|e\| = \left(\sum_{e=1}^{n_{el}} \frac{h^e}{2} \int_{-1}^1 q(\xi)\, d\xi \right)^{\frac{1}{2}} \tag{59}$$

It is important to note that for$\|e'\|$, the only modification is that $q(\xi) = (u'(x) - B\,d)^2$.

2.1 Gauss Quadrature

The integral in each finite element will be evaluated using the Gauss Quadrature, which offers an accurate approximation of an integral of the type:

$$I = \int_{-1}^1 q(\xi)\, d\xi \approx \sum_{g=1}^{m} w_g q(\xi_g) \tag{60}$$

Three Gauss points ($m = 3$) will be used, which is suitable for approximating polynomials of order $p \leq 5$. The positions and weights for $m = 3$ have been retrieved from Appendix B, and are as follows:

$$\begin{cases} \xi_1 = \sqrt{\frac{3}{5}} & \rightarrow \quad w_1 = \frac{5}{9} \\ \xi_2 = 0 & \rightarrow \quad w_2 = \frac{8}{9} \\ \xi_3 = -\sqrt{\frac{3}{5}} & \rightarrow \quad w_3 = \frac{5}{9} \end{cases} \tag{61}$$

The Gauss Quadrature rule has been incorporated into a Matlab script, which is detailed in Appendix A.5. By using this numerical method on the integrals from equation 59, an analysis can be conducted on how the approximation error evolves in relation to the element size in the finite element method discretization. The outcomes of this analysis are presented in Figures 6 and 7.

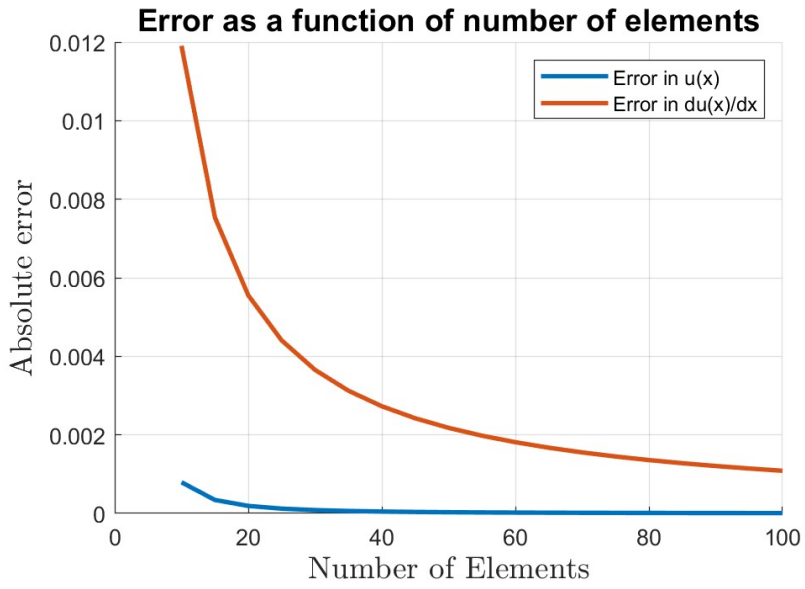


Figure 6: Graph of the dependency between absolute error and the number of finite elements.

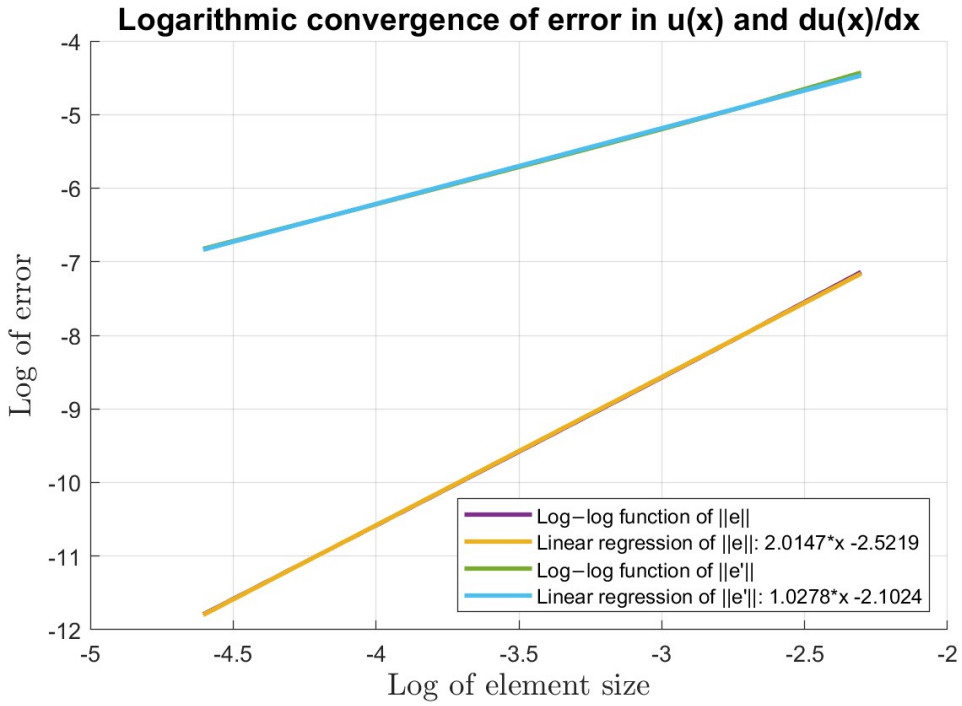


Figure 7: A log-log plot illustrating how the error in the finite element approximation of $u(x)$ and $u'(x)$ varies with respect to the element length in the discretization.

The expected linear relationship between the logarithms of the error and the size of two-node elements can be observed in Figure 7, with the error growing at a higher rate in the case of the

derivative $u'(x)$.

$$\log \|e\| = \alpha \log h^e + c \tag{62}$$

$$\begin{cases} \text{For } u(x) \rightarrow \alpha = 2.015 \approx 2 \\ \text{For } u'(x) \rightarrow \alpha = 1.028 \approx 1 \end{cases}$$

2.2 Conclusions

In conclusion, the convergence study demonstrates that the finite element method has been effectively applied. A clear relationship between element size and absolute error was observed, with the error decreasing significantly as the number of elements increases. Furthermore, the analysis revealed the expected rate of convergence for both displacements and stresses. Stress errors exhibited a slower rate of convergence, reflecting the sensitivity of stress calculations to mesh refinement. These findings highlight the importance of element size selection and mesh density in ensuring the accuracy of FEM, particularly for stress evaluation, where more refinement may be necessary to achieve desired precision.

3 "Advanced" assignment: Nonlinear 1D equilibrium

3.1 Statement of the assignment

A bar of length L and varying cross-sectional area

$$A(x) = A_0 \left(1 + 2\frac{x}{L}\left(\frac{x}{L} - 1\right)\right) \tag{63}$$

where $A_0 > 0$, is fixed at one end while the other end is subjected to a linearly increasing displacement $u_L(t) = u_m \frac{t}{T}$, where $u_m > 0$ and $t \in [0, T]$, $T > 0$ being the interval of time to analyze —this interval is considered sufficiently large so as to ignore inertial effects. On the other hand, the material of the bar obeys the following (exponential) constitutive equation (relation between stress σ and strain ϵ).

$$\sigma = \sigma_0 \left(1 - e^{-\frac{E}{\sigma_0}\varepsilon}\right). \tag{64}$$

Using the Finite Element method, find the displacement solution $u = u(x, t)$ as a function of $t \in [0, T]$ and $x \in [0, X]$.

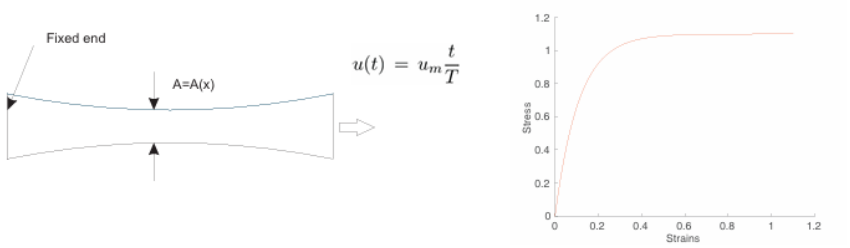


Figure 8: Geometry and constitutive equation

3.2 Strong form of the boundary value problem (BVP) for the 1D equilibrium of a bar with varying cross-sectional area

To formulate the equilibrium equation, a similar approach to that shown in Figure 2 will be followed. However, in this case, no distributed force is applied along the bar, and the cross-sectional area varies along its length. Thus, the equation results in:

$$\begin{aligned}
&A(x)\sigma(x) = A(x + dx)\left[\sigma(x) + \frac{\partial \sigma(x)}{\partial x} dx\right] \rightarrow \\
&A(x)\sigma(x) = \left(A(x) + \frac{\partial A(x)}{\partial x} dx\right)\left(\sigma(x) + \frac{\partial \sigma(x)}{\partial x} dx\right) \rightarrow \\
&A(x)\sigma(x) = A(x)\sigma(x) + \frac{\partial A(x)}{\partial x} dx \sigma(x) + A(x)\frac{\partial \sigma(x)}{\partial x} dx + \frac{\partial A(x)}{\partial x}\frac{\partial \sigma(x)}{\partial x} dx
\end{aligned} \tag{65}$$

Simplifying and neglecting second-order terms, the strong form of the equation is obtained.

$$\frac{\partial}{\partial x}\left(A(x)\sigma(x)\right) = 0 \tag{66}$$

Similarly, following Equation 64, σ depends on ε, which in turn, considering that this is a one-dimensional problem, is known to be related to the strain by $\sigma = E\varepsilon$. Here, $\varepsilon = u' = \frac{\partial u}{\partial x}$, and since this is a nonlinear problem, the displacement varies both in space and time. Thus, σ can be expressed as a function of the displacement gradient, $\sigma = \sigma(u'(x, t))$.

$$\boxed{\frac{\partial}{\partial x}\left(A(x)\sigma(u'(x,t))\right) = 0} \tag{67}$$

Along with the boundary value problems (BVPs)

$$\begin{cases} u_L(L,t) = u_m \frac{t}{T} \\ u(0,t) = 0 \end{cases} \tag{68}$$

Given $f : \Omega \times [0,T] \to \mathbb{R}$, $u_0 : [0,T] \to \mathbb{R}$, and $u_L : [0,T] \to \mathbb{R}$, find $u \in \mathbb{S}$ such that for all $V \in \mathbb{V}$, where σ is nonlinear.

3.3 Weak form

To formulate the variational form for this nonlinear case, the procedure will follow exactly the same steps as in Section 1.3. Thus, the test function is introduced, and integration by parts is applied in the same manner:

$$\int_0^L \left(v\frac{\partial(A\sigma)}{\partial x}\right)dx = 0 \to \int_0^L v'A\sigma, dx = \int_0^L \frac{\partial(v(A\sigma))}{\partial x}dx \tag{69}$$

Considering that by definition $v(0) = 0$, the weak or variational form of the problem is obtained as follows:

$$\boxed{\int_0^L v'A\sigma, dx = v(L)A(L)\sigma(L)} \quad \forall v \in \mathbb{V}_h \text{ with } v(0) = 0 \text{ and } \sigma = \sigma(u'(x,t)) \tag{70}$$

3.4 Formulation of the corresponding matrix equations

Following the same steps as in section 1.4, it is now time to define the solution as a linear combination of the shape functions.

$$\begin{cases} u^h = N(x) \cdot d(t) \\ v^h = N(x) \cdot c \end{cases} \implies \begin{cases} u'^h = \dfrac{\partial N(x)}{\partial x} \cdot d(t) = B(x) \cdot d(t) \\ v'^h = \dfrac{\partial N(x)}{\partial x} \cdot c = B(x) \cdot c \end{cases} \tag{71}$$

Introducing this changes into the equation 70 results in:

$$\mathbf{c}^T \underbrace{\int_0^L \mathbf{B}^T A\sigma \, dx}_{\hat{\mathbf{F}}} = \mathbf{c}^T \underbrace{\left(\mathbf{N}^T(L)A(L)\sigma(L)\right)}_{\mathbf{F}} \quad ; \quad \sigma = \sigma(Bd) \tag{72}$$

Where:

- $F \to$ External forces, which may vary over time.
- $\hat{F} \to$ Internal forces, which may also vary over time.

Thus, finally, the simplified equilibrium equation would result in:

$$\boxed{\hat{F}_l(d_l) = F_l} \quad ; \quad l_{2,3\ldots n+1} \to \text{Unknown values of d} \tag{73}$$

3.5 Solution of the nonlinear equations system by means of a *Newton-Raphson algorithm*

To solve the system of equations using the iterative *Newton-Raphson* method, the internal forces must first be discretized into elements and can be expressed as follows using the same procedure as in the linear case:

$$\hat{F} = \sum_{e=1}^{n_{el}} L^{e^T} \hat{F}^e = A_{e=1}^{n_{el}} \quad \text{where} \quad \hat{F}^e = \int_{\Omega^e} B^{e^T} A^e \sigma^e \, dx \quad \text{where} \quad \sigma^e = \sigma(B^e d^e) \tag{74}$$

The terms of $\hat{F}^e$ are known:

- $B^e = \frac{1}{h^e}[-1\ ,\ 1]$
- $A^e \rightarrow$ Area of the center of the element.
- $\sigma^e \rightarrow$ Given the displacement of the element, ε can be calculated and then σ using the expression 64.

Thus, for each element, this can be solved and expressed as:

$$\hat{F}^e = B^{e^T} A^e \sigma^e \int_{\Omega^e} dx \rightarrow \hat{F}^e = B^{e^T} A^e \sigma^e h^e \tag{75}$$

3.5.1 Time discretization

In this case, the result being sought is not a continuous solution (a single given value). Instead, solutions are desired at specific time instances:

$$[0, T] = [0, t_1] \cup [t_1, t_2] \cup \cdots \cup [t_n, t_{n+1}] \cdots [t_{M-1}, T] \tag{76}$$

3.5.2 Residual

From the displacement value at $t_n \rightarrow d(t_n)$ (initial condition), we aim to find the solution at the next time instant t_{n+1}.

$$\boxed{\hat{F}_e\left(d_l(t_{n+1})\right) = F_l(t_{n+1})} \tag{77}$$

For notational simplicity, the residual is defined as:

$$R := \underbrace{\hat{F}_e\left(d_l(t_{n+1})\right)}_{\text{Internal forces}} - \underbrace{F_l(t_{n+1})}_{\text{External forces}} \tag{78}$$

Now, the *Newton-Raphson* method will be applied as outlined in Appendix C.

3.5.3 Jacobian Matrix

Following Appendix C, in this case using the Residual, the equation can be written as:

$$R\left(y^{(k)}\right) + \left[\frac{\partial R}{\partial y}\right]_{y^{(k)}} \left(y^{(k+1)} - y^{(k)}\right) = 0 \tag{79}$$

Where $R(y^{(k)})$ is the residual vector and $\frac{\partial R}{\partial y}$ is the Jacobian matrix evaluated at $y^{(k)}$.

Accordingly, the solution at the $k+1$-th iteration is given by:

$$y^{(k+1)} = y^{(k)} - \left(J^{(k)}\right)^{-1} R\left(y^{(k)}\right) \tag{80}$$

Where:

$$J^{(k)} = \left[\frac{\partial R}{\partial y}\right]_{y^{(k)}} = \begin{bmatrix} \frac{\partial R_1}{\partial y_1} & \frac{\partial R_1}{\partial y_2} & \cdots & \frac{\partial R_1}{\partial y_n} \\ \frac{\partial R_2}{\partial y_1} & \frac{\partial R_2}{\partial y_2} & \cdots & \frac{\partial R_2}{\partial y_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial R_n}{\partial y_1} & \frac{\partial R_n}{\partial y_2} & \cdots & \frac{\partial R_n}{\partial y_n} \end{bmatrix} \tag{81}$$

This is the Jacobian matrix of the system of equations, which needs to be inverted or solved iteratively to obtain the solution. In this context, the value of $y^{(k+1)}$ represents the displacement to be obtained at the next time step, t_{n+1}, based on the current displacement $y^{(k)}$ at time step t_n.

The Jacobian matrix J in the Newton-Raphson algorithm is defined as:

$$J = \frac{\partial R}{\partial y} = \frac{\partial \hat{F}}{\partial d_l} = K_{II} \tag{82}$$

Where:

$$K := \frac{\partial \hat{F}}{\partial d} \tag{83}$$

This expression for the Jacobian matrix is essential for determining the direction in which the iterative process should proceed.

Using the equation provided earlier, the matrix K can be expressed as:

$$K = \frac{\partial}{\partial d}\left(\sum_{e=1}^{n_{el}} {L^e}^T \hat{F}^e\right) = \sum_{e=1}^{n_{el}} {L^e}^T \frac{\partial \hat{F}^e}{\partial d} \tag{84}$$

Since $d^e = L^e d$, applying the chain rule gives:

$$K = \sum_{e=1}^{n_{el}} {L^e}^T \frac{\partial \hat{F}^e}{\partial d^e}\frac{\partial d^e}{\partial d} = \sum_{e=1}^{n_{el}} {L^e}^T \frac{\partial \hat{F}^e}{\partial d^e} L^e \tag{85}$$

Thus, the matrix K can be computed by assembling the element contributions K^e:

$$K = A\sum_{e=1}^{n_{el}} K^e \tag{86}$$

To further elaborate, the term K^e is defined as:

$$K^e := \frac{\partial \hat{F}^e}{\partial d^e} \tag{87}$$

According to the previous equation, this term can be computed as:

$$K^e = \frac{\partial}{\partial d^e} \int_{\Omega^e} B^{e^T} A^e \sigma^e dx = \int_{\Omega^e} B^{e^T} A^e \frac{\partial \sigma^e}{\partial d^e} dx \tag{88}$$

Since $\sigma^e = \sigma(u^e)$ is the only parameter that relies on displacement, applying the chain rule results in:

$$\frac{\partial \sigma^e}{\partial d^e} = \overbrace{\frac{\partial \sigma^e}{\partial u'^e}}^{E^e} \frac{\partial u'^e}{\partial d^e} = E^e B^e \tag{89}$$

where $u^e = B^e d^e$. Finally, the elemental coefficient matrix K^e is given by:

$$K^e = \int_{\Omega^e} B^{e^T} (A^e E^e) B^e dx \tag{90}$$

3.5.4 Small Elements

If small elements are considered, the Jacobian matrix can be rewritten as:

$$J = \underline{\underline{K^e}} = \frac{AE}{h^e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{91}$$

Where AE is known as the axial stiffness, and E varies along the bar. The value of E can be calculated as $E = \frac{\partial \sigma}{\partial \varepsilon}$, where σ is the stress and ε is the strain. Finally, it can also be observed that the linear case is a special case where the residual is a straight line. Consequently, if the iterative method is applied, only one iteration would be required.

3.5.5 Plots

All of these processes have been implemented and solved using MATLAB to generate the corresponding plots. The complete script can be found in Appendix A.6, along with the necessary support functions used in the calculations.

Before conducting a convergence study, the preliminary results will be obtained using a discretization of **60 elements and 61 time steps**.

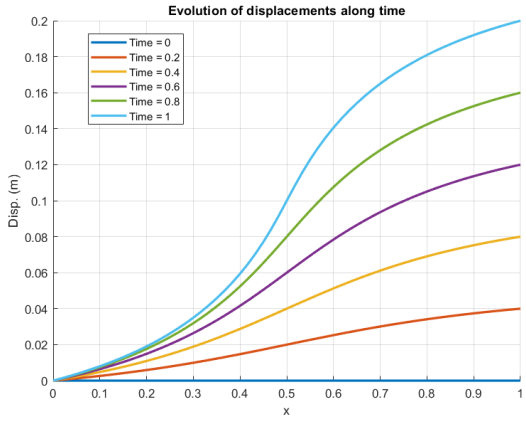


Figure 9: Evolution of stress along time

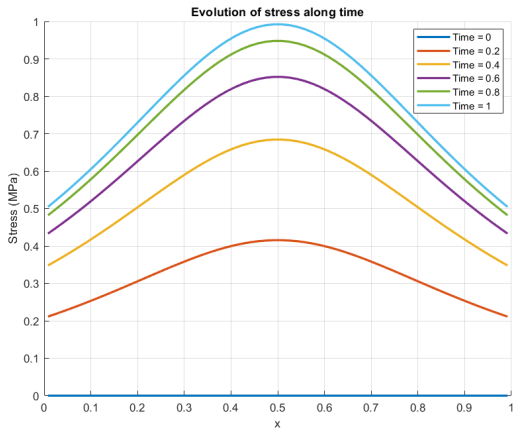


Figure 10: Evolution of displacement along time

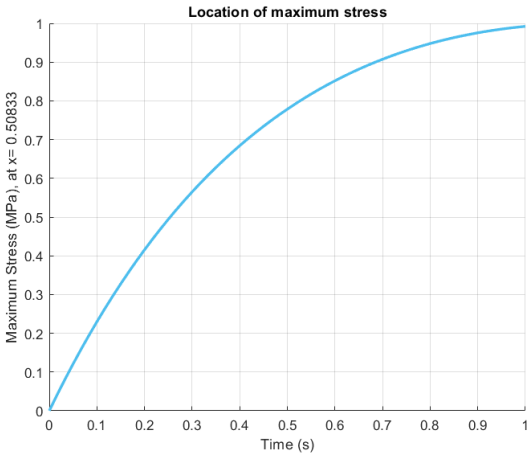


Figure 11: Location of maximum stress

3.6 Convergence upon increasing the number of elements and time steps

Lastly, the convergence of the solution will be verified by ensuring that the results remain stable as the number of elements and time steps increases. To perform this convergence study, a vector with varying numbers of elements and time steps (5, 10, 20, 50, 100, and 200) has been established. The algorithm described in the previous section has been executed iteratively for these different cases. The code used for this convergence study is provided in Appendix A.6.4.

The resulting data are illustrated in the graphs of Figures 12, 13, and 14.

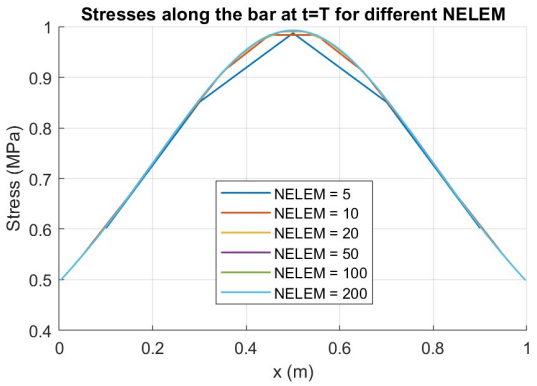


Figure 12: Evolution of stress for different number of elements.

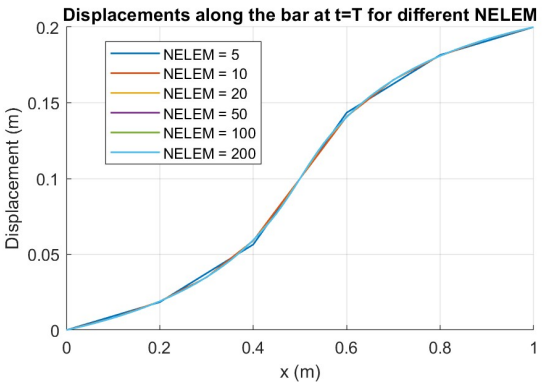


Figure 13: Evolution of strain for different number of elements.

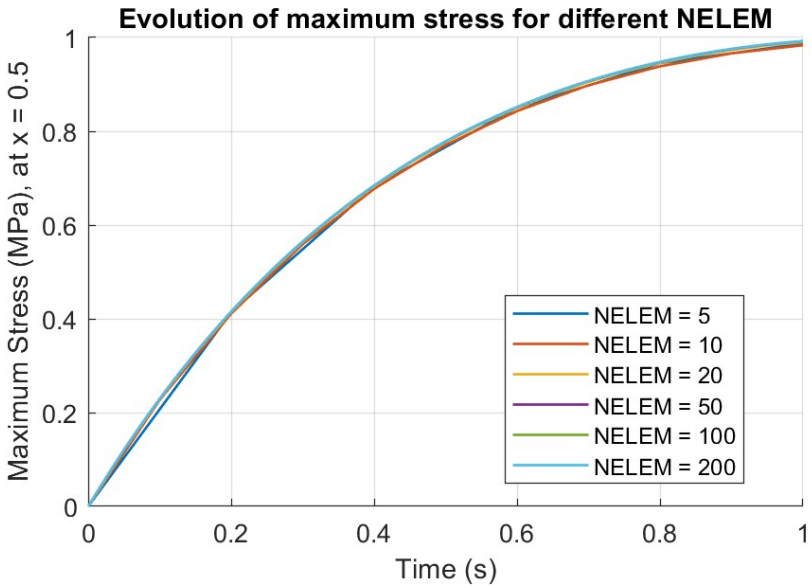


Figure 14: Maximum stress over time for a different number of elements.

The differences in number of elements can be mostly appreciated in Figure 12, where as the NELEM increases, the graph starts adjusting to the curve for $t = T$ seen in Figure 10.

In conclusion, the analysis of the plots shows that the results match well with those from the previous section, indicating that the solution is stable and that the code has reached convergence. This means that as the number of elements and time steps increased, the results remained consistent and reliable. Therefore, the numerical methods used in this study effectively captured the problem's behavior, ensuring accuracy in the solution. The convergence observed here also builds confidence in the reliability of the code for future use in more complex situations or larger-scale problems.

Appendix A Code

A.1 Code for section 1.2

```
%% BOUNDARY VALUE PROBLEM

% Symbolic variables
syms E A L g real % Constants
syms x real % Position variable
syms u(x) % Displacement function

% Constants definitions
rho = pi^3/L^2;
s = g*pi^4/L^2;
F = A*E*g*pi^2/L;
r = (x/L)^2;

% Distributed force
q = E*(rho*u-s*r);

% Hooke's law (We don't need to define them as symbolic)
epsilon = diff(u,x)
sigma = E*epsilon;

% Equilibrium
eq1 = diff(sigma,x)+q==0

% Boundary conditions
cond1 = u(0) == -g; % Displacement
du_dx_L = diff(u,x);
cond2 = A*E*subs(du_dx_L,x,L) == F; % Force

% General solution
u_sol = dsolve(eq1, cond1, cond2);
u_sol_simp = vpa(u_sol,4)

%% EXACT SOLUTION

% Apply numeric values
L_val=1;
g_val=0.01;

% Substitute
u_sol_val=(subs(u_sol,[g L], [g_val, L_val]));
u_sol_val_simp=vpa(u_sol_val,4)

% Plot of the exact solution
fplot(u_sol_val_simp*1000, [0 L_val], 'LineWidth',2) % Converted to millimetres:
    *1000
xlabel('Position, $x$ [m]', 'Interpreter', 'latex', 'FontSize', 12, 'FontWeight',
     'bold')
ylabel('Displacement, $u$ [mm]', 'Interpreter', 'latex', 'FontSize', 12, '
    FontWeight', 'bold')
grid on
title('Exact Solution Boundary Value Problem', 'FontSize', 16, 'FontWeight', '
    bold');
```

A.2 Code for section 1.4

```

b=g_val*pi^2/L_val; % Boundary conditions
s_val = subs(s,[g L],[g_val L_val]); % Constant
rho_val=subs(rho,L,L_val); % Constant
```

```
f = -s_val*x^2/L_val^2; % Source function --> Function in the EDO

% N = [1, x, x^2, x^3,x^4, x^5, x^6]; % Basis functions. Resolution functions (3rd order), we have to obtain the amplitudes (constants d)

n_end = 6; % Last exponent (x^{n_end})
N = cell(1, n_end); % Inicializar un arreglo de celdas para almacenar cada vector

for i = 1:n_end  % Loop from 1 until n_end
    N{i} = x.^(0:i);  % Guardar cada vector en una celda de N
end
for i=1:length(N)
    u_approx{i} = galerkin_solve_Prac1(N{i},f,b,g_val,L_val,rho_val);
    vpa(u_approx{i},4)
    % Plot for every N{i}
    figure(i+1);
    hold on;
    xlabel('Position, $x$ [m]', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', 'bold')
    ylabel('Displacement, $u$ [mm]', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', 'bold')
    title('Exact and approximate solution of the BVP', 'FontSize', 18, 'FontWeight', 'bold');
    fplot(u_exact*1000, [0 L_val], 'Color', colors(2,:), 'LineWidth',2) % Converted to millimetres: *1000
    fplot(u_approx{i}*1000, [0 L_val], 'Color', colors(1,:), 'LineWidth',2) % Converted to millimetres: *1000
    legend('Exact', ['Galerkin N = ', char(N{i})],'Location','southeast');
    hold off;
end


function [u] = galerkin_solve_Prac1(N,f,b,g,L,rho);
    syms x
    N_1 = subs(N,1); B = diff(N,x); % Matrix expressions defined in the resolution
    BtB = B.'*B; K = int(BtB,0,L)-rho*int(N.'*N,0,L);
    F = int(N.'*f,0,1)+N_1.'*b;
    r = 1; l = 2:length(N); % Freedom degrees. known/unkown coeficients
    dl = K(l,l)\ (F(l)+K(l,r)*g); % Solving equations system
    u = -g + N(l)*dl; %Problem solution --> u = N(1)*d1 +
end
```

A.3 Code for section 1.6

```
%General relations
syms xi x1 x2 he rho
Ne_xi = 1/2*[1-xi, 1+xi];
Be_xi = diff(Ne_xi,xi)
x = Ne_xi*[x1 ,x2]';

%Matrix K expression
ke=1/he*int(Be_xi'*Be_xi,xi,-1,1)+rho*he/2*int(Ne_xi'*Ne_xi,xi,-1,1);
disp(ke)

%Force element expression
fe=int(Ne_xi.'*x^2,xi,-1,1);
disp(fe)

%Discretitzation
x0 = 0; % First node
xF = L_val; % Last node
n_elem_vect = [5 10 20 40]; % Number of elements
```

A.4 Code for section 1.7

```
%Discretitzation
x0 = 0; % First node
xF = L_val; % Last node
n_elem_vect = [5 10 20 40]; % Number of elements


for n=1:length(n_elem_vect)
    n_elem=n_elem_vect(n)
    COOR = linspace(x0,xF,n_elem+1)'; % Equispaced vector of nodes
    CN = zeros(n_elem,2); % Connectivty matrix
    for i=1:n_elem
        CN(i,1) = i;
        CN(i,2) = i+1;
    end

    K = AssemblyK(COOR, CN, rho_val);
    F = AssemblyF(COOR, CN, s_val, L_val,g_val);
    r = 1; l = 2:(n_elem+1); % Freedom degrees. known/unkown coeficients
    dl = K(l,l)\ (F(l)+K(l,r)*g_val); % Solving equations system
    d=[-g_val, dl'];

    figure
    hold on
    plot(COOR,(d*1000),'Color', colors(1,:), 'LineWidth',2)
    fplot(u_exact*1000, [0 L_val], 'Color', colors(2,:), 'LineWidth',2) %
    Converted to millimetres: *1000

    xlabel('Position, x [m]', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight
    ', 'bold')
    ylabel('Displacement, u [mm]', 'Interpreter', 'latex', 'FontSize', 14, '
    FontWeight', 'bold')
    title('Exact and finite element solution of the BVP', 'FontSize', 18, '
    FontWeight', 'bold');
    legend('Exact', ['Number of elements = ', num2str(n_elem)],'Location','
    southeast');
  end


function K = AssemblyK(COOR,CN, rho_val)
    % COOR: Coordinates matrix, CN: Element connectivity matrix
    nelem = size(CN,1); % Number of elements as the number of rows of the
    connectivty matrix
    nnode = size(COOR,1); % Number of umber of nodes as it is the number of rows
    of points in the coordinates
    nnodeE = size(CN,2); % Number of nodes per element as it is the number of
    columns
    K = sparse(nnode,nnode); % To not store the zeros in the matrix
    for e=1:nelem
        % Element matrix
        NODES_e = CN(e,:); % Obtain initial and final nodes of the element e
        COOR_e = COOR(NODES_e); % Corresponding coordinates to the element
        he = COOR_e(2) - COOR_e(1); % Element longitude
        % Elemental matrix
        Ke = 1/he*[1 -1; -1 1] - rho_val*he/6*[2 1; 1 2];
        % It has the form Ke = 1/he*[1 -1; -1 1] only in the case of u''+f=0 (and
     thus has been modified for the case u''+u+f=0).

        % Assembly
        % Extract global indices for the current element
        A = CN(e, :); % Global indices for 'a'
        B = CN(e, :); % Global indices for 'b'
```

```
53
54          % Add the local stiffness matrix components to the global stiffness
        matrix
55          K(A, B) = K(A, B) + Ke;
56      end
57  end
58
59  function F = AssemblyF(COOR, CN,s,L,g)
60      % COOR: Coordinates matrix, CN: Element connectivity matrix
61      nnode = size(COOR,1); % Number of umber of nodes as it is the number of rows
        of points in the coordinates
62      n_elem = size(CN,1);
63      %Element force
64      F = sparse (nnode,1);
65      for n=1:n_elem
66          NODES_e = CN(n,:); % Obtain initial and final nodes of the element e
67          COOR_e = COOR(NODES_e); % Corresponding coordinates to the element
68          he = COOR_e(2) - COOR_e(1); % Element longitude
69          x1=COOR_e(1);
70          x2=COOR_e(2);
71          Fe=-(s*he/(12*L^2))*[3*x1^2+2*x1*x2+x2^2 ; x1^2+2*x1*x2+3*x2^2];
72          F(NODES_e)=F(NODES_e)+Fe;
73      end
74      F(n+1)=F(n+1)+g*pi^2/L;
75  end
```

A.5 Code for Part 2

```
1       syms x real
2
3   n_elem_vect = 5:5:200; % Number of elements
4
5   % Gauss quadrature for m = 3
6   xi_g = [sqrt(3/5) -sqrt(3/5) 0]; % Positions
7   w = [5/9 5/9 8/9]; % Weights
8
9   u_exact_diff = diff(u_exact); % Calculation of u' (exact)
10
11  Ne_xi=@(xi) (1/2)*[1-xi 1+xi]; % Redefinition of Ne as a function handle
12
13  % Inicialization of error convergence vectors
14  Error_u = zeros(length(n_elem_vect),1);
15  Error_Du = zeros(length(n_elem_vect),1);
16  elem_size = zeros(length(n_elem_vect),1);
17
18  % Error calculation loop
19  for n = 1:length(n_elem_vect)
20      n_elem = n_elem_vect(n);
21      COOR = linspace(x0,L_val,n_elem+1)'; % Equispaced vector of nodes
22      CN = zeros(n_elem,2); % Connectivty matrix
23      for i = 1:n_elem
24          CN(i,1) = i;
25          CN(i,2) = i+1;
26      end
27
28      K = AssemblyK(COOR, CN, rho_val);
29      F = AssemblyF(COOR, CN, s_val, L_val, g_val);
30      r = 1; l = 2:(n_elem+1); % Freedom degrees. known/unkown coeficients
31      dl = K(l,l)\(F(l)+K(l,r)*g_val); % Solving equations system
32      d = [-g_val, dl'];
33
34      error_u = 0;
35      error_du = 0;
```

```
36
37      for e = 1:n_elem
38          COORe = [COOR(CN(e,1)); COOR(CN(e,2))]; % Physical coordinates of u
39          de = [d(CN(e,1));d(CN(e,2))]; % Nodal values of u
40          he = COOR(CN(e,2))-COOR(CN(e,1)); % Element size
41          B = 1/he*[-1 1]; % Derivative of the shape function
42
43          % Integration by Gauss quadrature
44          for m = 1:3
45              u_h = Ne_xi(xi_g(m))*de; % Value of the FEM function
46              u_h_diff = B*de; %  dem  for the derivative
47              xF = Ne_xi(xi_g(m))*COORe; % Physical coordinate of xi_g
48              u_Exact = subs(u_exact, x, xF); % Evaluation of the exact function at
      the point
49              u_Exact = double(u_Exact);
50              u_Exact_diff = subs(u_exact_diff, x, xF); %  dem  for the derivative
51              u_Exact_diff = double(u_Exact_diff);
52              % Error computation
53              error_u = error_u+he/2*(w(m)*(u_Exact-u_h)^2);
54              error_du = error_du+he/2*(w(m)*(u_Exact_diff-u_h_diff)^2);
55          end
56
57      end
58
59      Error_u(n) = sqrt(error_u);
60      Error_Du(n) = sqrt(error_du);
61      elem_size(n) = he;
62
63  end
64
65  % Linear regression functions
66
67  log_elem_size = log(elem_size);
68  log_Error_u = log(Error_u);
69  log_Error_Du = log(Error_Du);
70
71  p1 = polyfit(log_elem_size,log_Error_u,1);
72  p2 = polyfit(log_elem_size,log_Error_Du,1);
73
74  p_u = polyval(p1, log_elem_size);
75  p_Du = polyval(p2, log_elem_size);
76
77  % Error plots
78  figure
79  hold on
80  plot(elem_size*100, Error_u,'Color', colors(1,:), 'LineWidth',2)
81  plot(elem_size*100, Error_Du,'Color', colors(2,:), 'LineWidth',2)
82  xlabel('Element size, [cm]', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight'
      , 'bold')
83  ylabel('Absolute error', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', '
      bold')
84  title('Error as a function of element size', 'FontSize', 18, 'FontWeight', 'bold'
      );
85  grid on
86  legend('Error in u(x)','Error in du(x)/dx', 'Location','northwest');
87  hold off
88
89  figure
90  hold on
91  plot(n_elem_vect, Error_u,'Color', colors(1,:), 'LineWidth',2)
92  plot(n_elem_vect, Error_Du,'Color', colors(2,:), 'LineWidth',2)
93  xlabel('Number of Elements', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight'
      , 'bold')
```

```
ylabel('Absolute error', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', 'bold')
title('Error as a function of number of elements', 'FontSize', 18, 'FontWeight', 'bold');
grid on
legend('Error in u(x)','Error in du(x)/dx', 'Location','northwest');
hold off

figure
hold on
plot(log_elem_size, log_Error_u,'Color', colors(4,:), 'LineWidth',2)
plot(log_elem_size, p_u,'Color', colors(3,:), 'LineWidth',2)
xlabel('Log of element size', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', 'bold')
ylabel('Log of error', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', 'bold')
title('Logarithmic convergence of error in u(x)', 'FontSize', 18, 'FontWeight', 'bold');
grid on
legend(' L o g log  function',sprintf('Linear regression function: %s*x %s', num2str(p1(1)), num2str(p1(2))), 'Location','southeast');
hold off

figure
hold on
plot(log_elem_size, log_Error_Du,'Color', colors(4,:), 'LineWidth',2)
plot(log_elem_size, p_Du,'Color', colors(3,:), 'LineWidth',2)
xlabel('Log of element size', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', 'bold')
ylabel('Log of error', 'Interpreter', 'latex', 'FontSize', 14, 'FontWeight', 'bold')
title('Logarithmic convergence of error in du(x)/dx', 'FontSize', 18, 'FontWeight', 'bold');
grid on
legend(' L o g log  function',sprintf('Linear regression function: %s*x %s', num2str(p2(1)), num2str(p2(2))), 'Location','southeast');
hold off
```

A.6 "Advanced" Assignment: Nonlinear 1D equilibrium

A.6.1 MAIN Routine

```
    clc
clear all
format long g
% Template for assignment 1, advanced part
% ----------------------------------------------
% INPUTS
% ------

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%55
NELEM = 60 ;  % Number of elements
nsteps = 61 ;  % Number of steps
stepsPOST = [1:12:nsteps]; % Steps to post-process


A0 = 1e-2; %m  % Reference AREA
E0 = 10 ; % MPa  % Reference Young's Modulus
sigma_0 = 1; %MPa  % Saturation stress
L = 1 ;%m    % Length of the beam
T = 1; % s Final time
um = 0.2*L; %m   % Maximum displacement
AreaFUN = @(x) (A0*(1+2*(x/L).*(x/L-1)))  ;  % Area as a function of the distance
```

```
    from the left end
stressFUN = @(strain) (sigma_0*(1-exp(-E0/sigma_0*strain))) ;  % Constitutive
    equation (stress versus strain)
DerStressFUN = @(strain) (E0*exp(-E0/sigma_0*strain)) ;  % Tangent modulus (
    Derivative of the stress with respect to the strain)
r = [1,NELEM+1]  ; % Indexes Prescribed DOF  %
uEND = @(t) (t/T*um);  %  Non-zero prescribed displacement, as a function of time
TOL_residual = 1e-6;  % Convergence tolerance for the Newton-Raphson
MAXITER = 50 ;  % Maximum number of iterations  for the Newton-Raphson
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% END INPUTS % ----------------------------------------
% MESH INFORMATION
nnode = NELEM+1 ;
COOR = linspace(0,L,nnode)' ;
CN = [(1:(nnode-1))',(2:nnode)'] ;
l = setdiff([1:nnode],r) ;

d = zeros(nnode,1) ;  % Displacement at time tn

TIMES = linspace(0,T,nsteps) ;

% STORE INFORMATION for post-processing purpose
% ------------------------
STRESS_GLO = zeros(NELEM,nsteps) ;
STRAIN_GLO  = zeros(NELEM,nsteps) ;
d_GLO  = zeros(nnode,length(stepsPOST)) ;

iplot = 1;



for istep=1:nsteps
    t = TIMES(istep) ;  % Time
    d(nnode) = uEND(t) ;

    disp(['Number of  step = ',num2str(istep), ', Time = ',num2str(t),', uEND =
    ',num2str( uEND(t)) ])
    disp('**************************************************************')
    % Boundary conditions
    %-------------------
    % Compute the vector of internal forces = residual
    % as a function of the nodal displacement n

    % Solution of the equations Residual(d) = 0     via Newton-Raphson
    % algoritm
    d_k = d; %   nodal displacements at iteration k
    k = 0 ; % Number of iteration
    while  k <= MAXITER
        % ASsembly the residual vector (= internal forces)
        [Residual,STRAIN,STRESS] = AssemblyFint(COOR,CN,d_k,stressFUN,AreaFUN) ;
        normRESIDUAL = norm(Residual(l)) ; % Euclidean norm residual
        disp(['k = ',num2str(k),' res=',num2str(normRESIDUAL),', MAX strain = ',
    num2str(max(STRAIN)),', MAX stress = ',num2str(max(STRESS))])
        if normRESIDUAL < TOL_residual
            % Convergence
            d = d_k ;
            break
        end
        % The norm of the residual is greater than the prescribed
        % tolerance.  This means that equilibrium is not met
        % Let us calculate the new displacement vector

```

```
        % Compute the Jacobian of the system of equations
        K = AssemblyKnon(COOR,CN,d_k,AreaFUN,DerStressFUN);
            Delta_d_l = -K(1,1)\Residual(1) ;
        d_k(1) = d_k(1) + Delta_d_l;
        k = k +1 ;
    end

    if  k >  MAXITER
        error('The Newtown-R. algorithm has failed to converge')
    else
        d=d_k ;
        d_GLO(:,iplot) = d ;
        STRAIN_GLO(:,iplot) = STRAIN ;
        STRESS_GLO(:,iplot) = STRESS ;
        iplot = iplot +1 ;
    end
end


PLOTS
% Evolution of stresses along time
% ----------------------------------------
figure
hold on
xlabel('x')
ylabel('Stress (MPa)')
title('Evolution of stress along time')
COOR_gauss = (COOR(CN(:,2))+  COOR(CN(:,1)))/2 ;
nsteps_plot = length(stepsPOST) ;
hplot = zeros(nsteps_plot,1) ;
LegendPlot = cell(nsteps_plot,1) ;
for i = 1:nsteps_plot
    hplot(i) =  plot(COOR_gauss,STRESS_GLO(:,stepsPOST(i)),'LineWidth',2) ;
    %LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i))),'; NELEM =',num2str(
    NELEM),'; NSTEP = ',num2str(nsteps)] ;
    LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i)))] ;end
grid on
legend(hplot,LegendPlot)

legend('off')
legend

% Evolution of displacements along time
% ----------------------------------------
figure
hold on
xlabel('x')
ylabel('Disp. (m)')
title('Evolution of displacements along time')
nsteps_plot = length(stepsPOST) ;
hplot = zeros(nsteps_plot,1) ;
LegendPlot = cell(nsteps_plot,1) ;
for i = 1:nsteps_plot
    hplot(i) =  plot(COOR,d_GLO(:,stepsPOST(i)),'LineWidth',2) ;
   %LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i))),'; NELEM =',num2str(
    NELEM),'; NSTEP = ',num2str(nsteps)] ;
    LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i)))] ;

end
grid on
legend(hplot,LegendPlot)

legend('off')
```

```
legend



% Location of maximum stress

[stressEND,indELEM ]= max(STRESS_GLO(:,end)) ;
COOR_max_stress= COOR_gauss(indELEM) ;

figure
hold on
title('Location of maximum stress')
xlabel('Time (s)')
ylabel([['Maximum Stress (MPa), at x= ',num2str(COOR_max_stress)]]) ;
for i = 1:nsteps_plot
    h =   plot(TIMES,STRESS_GLO(indELEM,:),'LineWidth',2) ;
end
grid on
%legend(h,['NELEM =',num2str(NELEM),'; NSTEP = ',num2str(nsteps)])

%legend('off')
%legend
```

A.6.2 AssemblyFint

```
    function [Residual,STRAIN,STRESS] = AssemblyFint(COOR,CN,d_k,stressFUN,
    AreaFUN)

%%% INPUTS %%%
% COOR --> Coordinates vector
% CN --> Connectivity matrix
% d_k --> Previous displacements
% stressFUN & AreaFUN --> Functions of area and stress (sigma)


    nnode = size(COOR,1); % Number of nodes as it is the number of rows of points
     in the coordinates
    n_elem = size(CN,1);

    Residual = zeros(nnode,1); % One residual equation for every element
    STRAIN = zeros(n_elem, 1); % One strain for every element
    STRESS = zeros(n_elem, 1); % One stress for every element

    for e=1:n_elem
        NODES_e = CN(e,:); % Obtain initial and final nodes of the element e
        COOR_e = COOR(NODES_e); % Corresponding coordinates to the element e
        x1=COOR_e(1); % Real position node 1
        x2=COOR_e(2); % Real position node 2
        he = x2-x1; % Element longitude
        d_k_elem = d_k(NODES_e,1); % Previous displacement node solution of the
    element
        Be = 1/he*[-1, 1]; % Matrix
        A_e = AreaFUN((x2+x1)/2); % Area of the element (Average of nodes)
        Strain_e = Be*d_k_elem; % Strain of the element
        Stress_e = stressFUN(Strain_e); % Stress of the element
        F_e = he*(Be'*A_e*Stress_e); % Elemental internal force

        Residual(NODES_e) = Residual(NODES_e) + F_e; % No external forces.
    Contribution of each element

        STRAIN(e) = Strain_e; % Strain of the element
        STRESS(e) = Stress_e; % Stress of the element
    end
```

```
end
```

A.6.3 AssemblyKnon

```
    function K = AssemblyKnon(COOR,CN,d_k,AreaFUN,DerStressFUN);

    nelem = size(CN,1); % Number of elements as the number of rows of the
    connectivty matrix
    nnode = size(COOR,1); % Number of umber of nodes as it is the number of rows
    of points in the coordinates
    nnodeE = size(CN,2); % Number of nodes per element as it is the number of
    columns
    K = zeros(nnode,nnode); % To not store the zeros in the matrix

    for e=1:nelem
        % Element matrix
        NODES_e = CN(e,:); % Obtain initial and final nodes of the element e
        COOR_e = COOR(NODES_e); % Corresponding coordinates to the element
        x1=COOR_e(1); % Node 1
        x2=COOR_e(2); % Node 2
        he = x2-x1; % Element longitude
        Be = 1/he*[-1, 1]; % Matrix
        d_k_elem = d_k(NODES_e,1);
        Strain = Be * d_k_elem;
        A = AreaFUN((x2+x1)/2);
        E = DerStressFUN(Strain);
        % Elemental matrix
        Ke = (A*E)/he*[1 -1; -1 1];

        % Assembly
        % Extract global indices for the current element
        A = CN(e, :); % Global indices for 'a'
        B = CN(e, :); % Global indices for 'b'

        % Add the local stiffness matrix components to the global stiffness
    matrix
        K(A, B) = K(A, B) + Ke;
    end
end
```

A.6.4 Convergence study

```
clc
clear all
format long g

% INPUTS
NELEM_vec = [5 10 20 50 100 200];  % Number of elements vector

% Define cell arrays to store varying sizes of matrices
STRESS_GLO = cell(length(NELEM_vec), 1);
STRAIN_GLO = cell(length(NELEM_vec), 1);
d_GLO = cell(length(NELEM_vec), 1);
COOR_gauss = cell(length(NELEM_vec), 1);  % Cell array to store COOR_gauss for
    each NELEM
COOR = cell(length(NELEM_vec), 1);        % Cell array to store COOR for each
    NELEM
stepsPOST = cell(length(NELEM_vec), 1);
TIMES = cell(length(NELEM_vec), 1);

for e = 1:length(NELEM_vec)
    NELEM = NELEM_vec(e);  % Number of elements
    nsteps = NELEM + 1;  % Number of steps
```

```
    stepsPOST{e} = 1:round(nsteps/3):nsteps;  % Steps to post-process

    % Material and geometric properties
    A0 = 1e-2; %m  % Reference AREA
    E0 = 10; % MPa  % Reference Young's Modulus
    sigma_0 = 1; %MPa  % Saturation stress
    L = 1; %m  % Length of the beam
    T = 1; % s Final time
    um = 0.2 * L; %m   % Maximum displacement
    AreaFUN = @(x) (A0 * (1 + 2 * (x/L) .* (x/L - 1)));  % Area function
    stressFUN = @(strain) (sigma_0 * (1 - exp(-E0/sigma_0 * strain)));  % Stress
    function
    DerStressFUN = @(strain) (E0 * exp(-E0/sigma_0 * strain));  % Derivative
    function
    r = [1, NELEM + 1];  % Boundary condition indices
    uEND = @(t) (t/T * um);  % Prescribed displacement as a function of time
    TOL_residual = 1e-6;  % Convergence tolerance
    MAXITER = 50;  % Maximum number of Newton-Raphson iterations

    % MESH INFORMATION
    nnode = NELEM + 1;
    COOR{e} = linspace(0, L, nnode)';  % Store COOR for current NELEM in cell
    array
    CN = [(1:(nnode - 1))', (2:nnode)'];
    l = setdiff(1:nnode, r);
    d = zeros(nnode, 1);  % Displacement at time tn

    TIMES{e} = linspace(0, T, nsteps);

    % Initialize post-processing arrays for current NELEM
    STRESS_GLO{e} = zeros(NELEM, nsteps);
    STRAIN_GLO{e} = zeros(NELEM, nsteps);
    d_GLO{e} = zeros(nnode, length(stepsPOST));

    iplot = 1;

    % Time-stepping loop
    for istep = 1:nsteps
        t = TIMES{e}(istep);
        d(nnode) = uEND(t);

        d_k = d;
        k = 0;
        while k <= MAXITER
            [Residual, STRAIN, STRESS] = AssemblyFint(COOR{e}, CN, d_k, stressFUN
    , AreaFUN);
            normRESIDUAL = norm(Residual(l));
            if normRESIDUAL < TOL_residual
                d = d_k;
                break;
            end

            K = AssemblyKnon(COOR{e}, CN, d_k, AreaFUN, DerStressFUN);
            Delta_d_l = -K(l, l) \ Residual(l);
            d_k(l) = d_k(l) + Delta_d_l;
            k = k + 1;
        end

        if k > MAXITER
            error('The Newton-R. algorithm failed to converge');
        else
            d = d_k;
            d_GLO{e}(:, iplot) = d;
```

```
            STRAIN_GLO{e}(:, iplot) = STRAIN;
            STRESS_GLO{e}(:, iplot) = STRESS;
            iplot = 1 + iplot;
        end
    end

    % Calculate and store Gauss coordinates for current NELEM
    COOR_gauss{e} = (COOR{e}(CN(:, 2)) + COOR{e}(CN(:, 1))) / 2;
end


%% PLOTS

% Plot stresses along the bar at the last timestep for each NELEM value
figure;
hold on;
xlabel('x (m)');
ylabel('Stress (MPa)');
title('Stresses along the bar at t=T for different NELEM');
legendEntries = cell(length(NELEM_vec), 1);

% Loop through each NELEM value to plot the stresses at the last timestep
for e = 1:length(NELEM_vec)
    plot(COOR_gauss{e}, STRESS_GLO{e}(:, end), 'LineWidth', 1);
    legendEntries{e} = ['NELEM = ', num2str(NELEM_vec(e))];
end

grid on;
legend(legendEntries, 'Location', 'Best');


% Plot displacements along the bar at the last timestep for each NELEM value
figure;
hold on;
xlabel('x (m)');
ylabel('Displacement (m)');
title('Displacements along the bar at t=T for different NELEM');
legendEntries = cell(length(NELEM_vec), 1);

% Loop through each NELEM value to plot the displacements at the last timestep
for e = 1:length(NELEM_vec)
    plot(COOR{e}, d_GLO{e}(:, end), 'LineWidth', 1);
    legendEntries{e} = ['NELEM = ', num2str(NELEM_vec(e))];
end

grid on;
legend(legendEntries, 'Location', 'Best');


% Location of maximum stress for different NELEM values

figure;
hold on;
title('Evolution of maximum stress for different NELEM');
xlabel('Time (s)');
ylabel('Maximum Stress (MPa), at x = 0.5');
legendEntries = cell(length(NELEM_vec), 1);

% Loop through each NELEM value to plot the maximum stress at x = 0.5 over time
for e = 1:length(NELEM_vec)
    % Find the index of the Gauss coordinate closest to x = 0.5
    [~, indELEM] = min(abs(COOR_gauss{e} - 0.5));

```

```
    % Plot the maximum stress over time for this NELEM
    plot(TIMES{e}, STRESS_GLO{e}(indELEM, :), 'LineWidth', 1);
    legendEntries{e} = ['NELEM = ', num2str(NELEM_vec(e))];
end

grid on;
legend(legendEntries, 'Location', 'Best');
```

Appendix B Gaussian quadrature

Number of points, m	Points, ξ_g	Weights, w_g
1	0	2
2	$\pm\sqrt{\frac{1}{3}}$	1
3	0	$\frac{8}{9}$
	$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{9}$
4	$\pm\sqrt{\frac{3}{7}-\frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18+\sqrt{30}}{36}$
	$\pm\sqrt{\frac{3}{7}+\frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{18-\sqrt{30}}{36}$
5	0	$\frac{128}{225}$
	$\pm\frac{1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	$\frac{322+13\sqrt{70}}{900}$
	$\pm\frac{1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$	$\frac{322-13\sqrt{70}}{900}$

Table 1: Gauss points and weights for different values of m. [2]

Appendix C Explanation of the Newton-Raphson Method

The **Newton-Raphson method** is an iterative procedure used to find approximate solutions to nonlinear equations of the form $f(x) = 0$. Below is a detailed explanation of the method.

C.1 Purpose of the Method

The goal of the Newton-Raphson method is to find the root of a nonlinear function $f(x)$, that is, the value of x for which $f(x) = 0$. This method is particularly useful when the equation cannot be solved analytically, but we need a numerical approximation.

C.2 Mathematical Basis

The method is based on approximating the nonlinear function $f(x)$ by a tangent line at a point near the root. Starting with an initial guess x_0, the method uses the equation of the tangent line to update x_0 and obtain a better approximation of the root. Mathematically, this is derived from a first-order Taylor expansion.

C.2.1 Derivation of the Formula

The key idea is to use the Taylor series expansion to approximate $f(x)$ around a given point x_0. The first-order Taylor expansion is:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

By setting $f(x) = 0$ (since we are looking for the root), the equation becomes:

$$0 \approx f(x_0) + f'(x_0)(x - x_0)$$

Solving for x, we obtain:

$$x \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

This gives the iterative formula of the Newton-Raphson method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where x_n is the current approximation, and x_{n+1} is the next (and hopefully better) approximation.

C.3 Iterative Process

1. Start with an initial guess x_0. 2. Compute $f(x_0)$ and $f'(x_0)$, where $f'(x)$ is the derivative of $f(x)$. 3. Update the guess using the formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

4. Repeat the process until the change in x_n between iterations is smaller than a chosen tolerance or until $f(x_n)$ is close enough to zero.

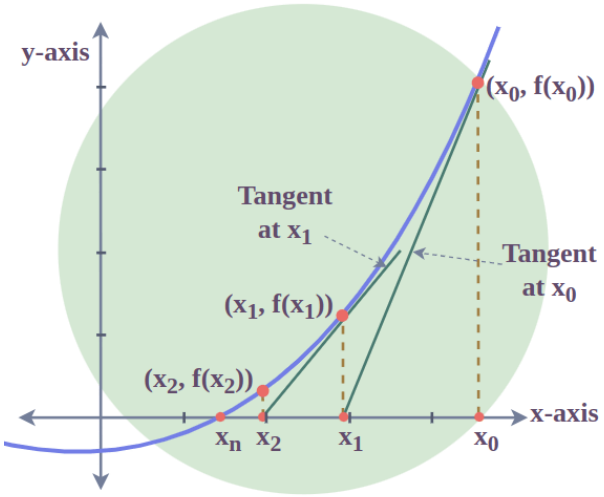


Figure 15: Graphical visualitzation of Newton-Raphson method[1].

C.4 Convergence

The Newton-Raphson method converges quadratically if the initial guess x_0 is sufficiently close to the true root and if $f'(x)$ is not zero at the root. This means that the error in the approximation decreases very quickly in each iteration. However, if the initial guess is far from the root or if the function behaves poorly (e.g., has inflection points or discontinuities), the method may fail to converge or may converge slowly.

Thus, the Newton-Raphson method is an efficient tool for finding the roots of nonlinear equations, provided that a good initial guess and the derivative of the function are available.

References

[1] GeeksForGeeks. Newton raphson method: Definition, formula, examples calculation, 2024. URL https://www.geeksforgeeks.org/newton-raphson-method/. Accessed: 2024-10-20.

[2] N. F. UPC. Gauss quadrature in 1d, 2023. URL https://numfactory.upc.edu/web/Calculo2/P2_Integracio/html/Gauss1D.html. Accessed: 2024-10-13.