Universitat Politècnia de Catalunya

ESEIAAT

DEGREE IN AEROSPACE TECHNOLOGY ENGINEERING

Assignment 1

FINITE ELEMENTS METHOD 1D

Author
Bernardo Márquez López

Professor
Joaquín Hernández Ortega
Department of materials and structural resistance in engineering

September 2022

Contents

1 Part 1 **3**
1.1 Derivation of the correspondent Boundary Value Problem (BVP) for the displacement field $u : [0, L] \longrightarrow \mathbb{R}$ using the equilibrium equation for 1D problems (strong form). 3
1.2 Determination of the exact solution for this BVP. Plot of the solution in a MatLab graph using the following numerical values for constants involved: $\mathbf{g = 0,01}$m, $\mathbf{L = 1}$m. 4
1.3 Formulation of Variational Form of the Boundary Value Problem. 5
1.4 Derivation of the corresponding matrix equation in terms of a generical matrix of basis functions N and their corresponding derivatives $B = \frac{\partial N}{\partial x}$. 6
1.5 Approximation to the solution of this variational form by using polynomial basis functions of increasing order. 7
1.6 Development of a MatLab program able to solve the variational form using finite elements basis functions using number of equally sized finite elements (n) as program input. 9
1.7 Problem analysis and solution for increasing number of finite elements. Plot of approximate solutions with discretizations for n = [5, 10, 15, 20] and comparison with the exact solution. 10

2 Part 2 **12**
2.1 Implementation of a function able to calculate the approximation error for both u and its derivative u'. Plot of the error versus the element size in a logarithmic graphic. Study of the slope of the convergence plot in each case. 12

3 Advanced Assignment 1 : Nonlinear 1D equilibrium **15**
3.1 Formulation of the strong form of the Boundary Value Problem (BVP) for the 1D equilibrium of a bar with varying cross-sectional area. 15
3.2 Determination of the weak form of the BVP. 15
3.3 Formulation of the corresponding matrix equations 16
3.4 Solution of the resulting system of nonlinear equations by means of a Newton-Raphson algorithm. . . . 16
3.5 Verification of solution convergence upon increasing the number of elements and time steps 18

A Appendix **22**
A.1 Code for section 1.2 22
A.2 Code for section 1.5 22
A.2.1 Main program 22
A.2.2 Galerkin Method Function 23
A.3 Code for sections 1.6 and 1.7 23
A.3.1 Main program for section 1.6 23
A.3.2 Main program for section 1.6 24
A.3.3 Main program for section 1.7 24
A.3.4 AssemblyMatrixKFd Function 25
A.4 Code for section 2 26
A.4.1 Main program 26
A.5 Code for Advanced Assignment 28
A.5.1 Main Program 28
A.5.2 Assembly K Function 30
A.5.3 Assembly F_{int} Function 31
A.5.4 Convergence assessment 31

List of Figures

1 Representation of the structural loads and displacements described in the statement. 3
2 Graph of the exact solution of the displacement field $u(x)$ in the interval $x \in [0, L]$ 5
3 Exact solution and approximation of $u(x)$ through polynomial basis functions. 8
4 Exact solution and approximation of $u(x)$ through polynomial basis functions. 8

5 Representation of exact solution and approximate solutions for 5,10,15 and 25 discretization elements. 11
6 Representation of exact solution and approximate solutions for 5,10,15 and 25 discretization elements (expanded view). 11
7 Log-Log plot that shows the evolution of the error in $u(x)$ finite elements approximation as function of the element length in the discretization. 13
8 Log-Log plot that shows the evolution of the error in $u'(x)$ finite elements approximation as function of the element length in the discretization. 13
9 Location of the maximum stress over time for a discretization of the domain in 50 elements and 51 time steps. 17
10 Evolution of the displacement field over time for a discretization of the domain in 50 elements and 51 time steps. 17
11 Evolution of the stress field over time for a discretization of the domain in 50 elements and 51 time steps. 18
12 Evaluation of the convergence in the stress field function over time for a range of different numbers and time steps. 19
13 Evaluation of the convergence in the displacements field function over time for a range of different numbers and time steps. 19
14 Caption 20
15 Detail view of the convergence in the displacements field over time for a increasing number of elements and time steps 20
16 Evaluation of the convergence in the location of the maximum stress over time for a range of different numbers and time steps. 21
17 Detail view of the convergence in the location of the maximum stress over time for a increasing number of elements and time steps 21

General 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) $q(x) = E[\rho u(x) - sx^2]$; $u = u(x)$ is the displacement field, g is a constant. The definition of the parameters ρ, s and F are the following:

$$\rho = \frac{\pi^2}{L^2} \qquad s = g\rho^2 \qquad \frac{F}{AE} = \frac{g\pi^2}{L} \tag{1}$$

1 Part 1

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

The problem presented in the statement is represented in figure 1.

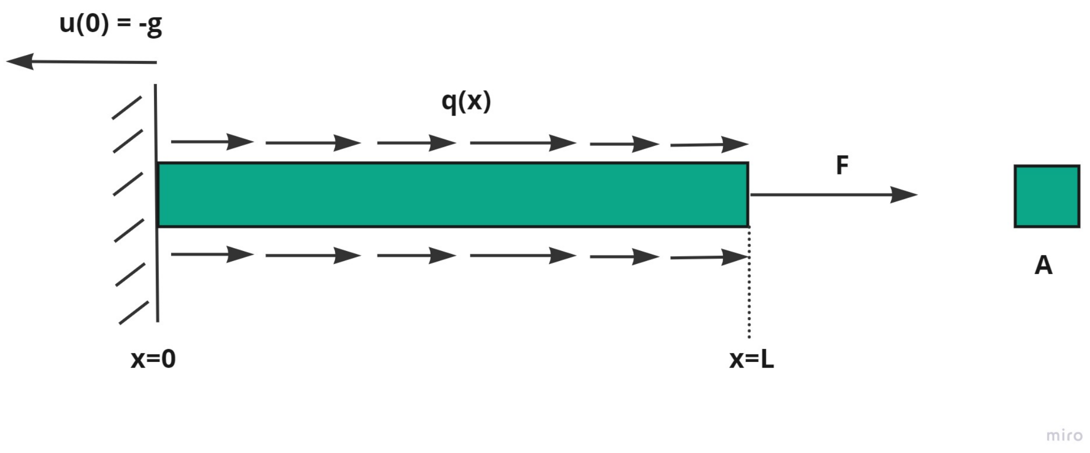

Figure 1: Representation of the structural loads and displacements described in the statement.

Derivatives of displacement function $u(x)$ will be designated by the nomenclature $u'(x) = \frac{\partial u}{\partial x}$ and $u''(x) = \frac{\partial^2 u}{\partial x^2}$ in the development of this report.

The strong form of the BVP for the displacement field will be defined by $u''(x)$ and the boundary conditions of $u(x)$ lower order derivatives. The first boundary condition is given by the statement:

$$\boxed{u(0) = -g} \tag{2}$$

First of all, the axial equilibrium equation will be applied to the case described in the statement. Equation 3 describes the equilibrium of a bar subjected to a distributed axial force dependent to position, $q(x)$. Where $\sigma : \overline{\Omega} \longrightarrow \mathbb{R}$ is the stress field ($\overline{\Omega} = [0, 1]$).

$$\frac{\partial \sigma}{\partial x} + q(x) = 0 \tag{3}$$

Next, the definition given by Hooke's law for a linear elastic material provides a relationship between stress and displacement fields.

$$\sigma = E \cdot \varepsilon = E \cdot \frac{\partial u}{\partial x} = E \cdot u' \tag{4}$$

Substituting equation 4 into the expression in equation 3, the following expression is obtained:

$$E \cdot \frac{\partial^2 u}{\partial x^2} + q(x) = 0 \tag{5}$$

Then, applying the definition of $q(x)$ given in the statement, the second boundary condition of the problem can be inferred:

$$\boxed{u'' + \rho \cdot u - s \cdot x^2 = 0} \tag{6}$$

For the boundary condition at $x = L$ (Neumann condition), the stress definition (σ) as $\sigma = F/A$ will be applied, as well as the relationship given by equation 4.

$$\varepsilon = \frac{\partial u}{\partial x} \tag{7}$$

$$\sigma = \frac{F}{A} = E \cdot u' \Longrightarrow u'(L) = \frac{F}{A \cdot E} \tag{8}$$

Substituting the expression given in the statement for that fraction, the following boundary condition is obtained:

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

Finally, the strong form of the Boundary Vale Problem of the displacement field in the structure is shown in 19.

$$\begin{cases} u(0) + g = 0 \\ u'(L) - \frac{F}{EA} = 0 \\ u''(x) + \frac{q(x)}{E} = 0 \end{cases} \Longrightarrow \begin{cases} u(0) = -g \\ u'(L) - \frac{g \cdot \pi^2}{L} = 0 \\ u''(x) + \rho u(x) - s x^2 = 0 \end{cases} \tag{10}$$

1.2 Determination of the exact solution for this BVP. Plot of the solution in a MatLab graph using the following numerical values for constants involved: $\mathbf{g = 0,01m}$, $\mathbf{L = 1m}$.

From strong form of the problem, the exact solution of the equilibrium equation was determined by applying boundary conditions. Then, from the Matlab code in the appendix A.1, the following expression of u(x) was obtained:

$$\boxed{u(x) = 0.0100 \cdot \cos(\pi \cdot x) + \frac{\pi}{100} \cdot \sin(\pi \cdot x) + 0.0987 \cdot x^2 - 0.0200} \tag{11}$$

It is important to note that this expression has been approximated to three significant figures.
Plotting the displacement field $u(x)$ as a function of the position over the bar x from expression 11, the following graph is obtained:

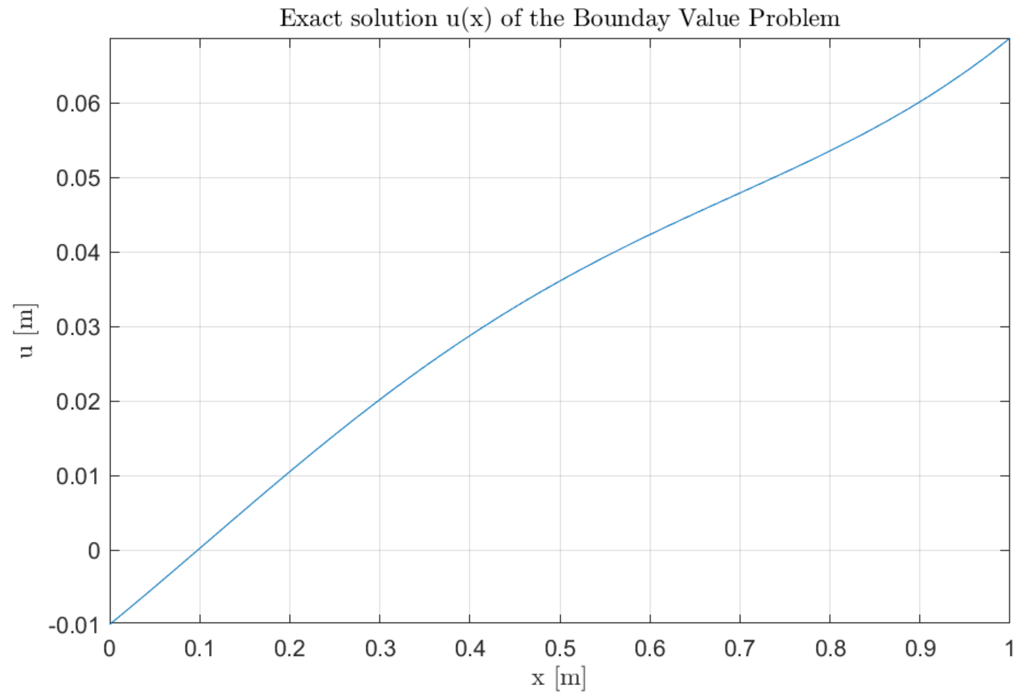

Figure 2: Graph of the exact solution of the displacement field $u(x)$ in the interval $x \in [0, L]$

1.3 Formulation of Variational Form of the Boundary Value Problem.

In order to obtain the variational form of the BVP, in this section the following formulation will be applied:

$$u(x) \approx \sum_{i=1}^{n} N_i(x) \cdot d_i = N(x) \cdot d \tag{12}$$

In this assumption, u(x) is defined as a linear combination of a group of known functions N(x). Hence, if we express $u(x) = g + u(x) - g$, and define the function $u(x) - g = v(x) = N(x) \cdot c$, where $v(0) = N(0) \cdot c = 0$ and $u(0) - g = 0$. Then, the following formulation can be made:

$$\int_0^L u(x) \cdot r(x;u)dx = \int_0^L [u(x) - g + g] \cdot r(x;u)dx = \int_0^L v(x) \cdot r(x;u)dx + g\int_0^L r(x;u)dx \tag{13}$$

Where $r(x;u)$ represents the residual in the differential equation of $u''(x)$ associated to functions $N(x)$. The variational form of the problem represents the solution that minimizes the residual forces ($u''(x) + \rho u(x) - sx^2 = r(x;u)$). Therefore, the following condition must be applied:

$$\int_0^L v(x) \cdot r(x;u)dx = 0 \Longrightarrow \int_0^L v(x) \cdot [u''(x) + \rho \cdot u(x) - s \cdot x^2]dx = 0 \tag{14}$$

The next step will be applying integration by parts in order to obtain an equivalent expression which does not include the second order derivative:

$$\int v(x) \cdot u''(x) = v(x) \cdot u'(x) - \int v'(x) \cdot u'(x) \tag{15}$$

Applying this identity into equation 14, the following equation is obtained:

$$[v'(x) \cdot u'(x)]_0^L - \int_0^L u'(x) \cdot v'(x) + \rho \int_0^L v(x) \cdot u(x)dx - s\int_0^L v(x) \cdot x^2 dx = 0 \tag{16}$$

Applying the known conditions of the problem ($u'(L) = g\pi^2/L,\ \ v(0) = 0$), the equation above can be written as:

$$v(L)\cdot\frac{g\cdot\pi^2}{L} - \int_0^L u'(x)\cdot v'(x)dx + \rho\int_0^L u(x)dx - s\int_0^L x^2dx = 0 \tag{17}$$

Finally, the variational form of the problem can be expressed as follows:

$$\boxed{\int_0^L u'(x)\cdot v'(x)dx = \rho\int_0^L v(x)\cdot u(x)dx - s\int_0^L v(x)\cdot x^2dx + v(L)\cdot\frac{g\cdot\pi^2}{L}} \quad \forall v : v(0) = 0 \text{ and } u(0) - g = 0 \tag{18}$$

It is important to remark that $u(x)$ and $v(x)$ must be continuous functions with square-integrable derivative in a domain $\overline{\Omega} \longrightarrow \mathbb{R}$.

1.4 Derivation of the corresponding matrix equation in terms of a generical matrix of basis functions N and their corresponding derivatives $B = \frac{\partial N}{\partial x}$.

From the formulation of $u(x)$ and $v(x)$ stated in the previous section as a linear combination of basis functions $N(x)$ and considering $B(x) = \partial N/\partial x$, then the derivative functions $u'(x)$ and $v'(x)$ can be expressed as follows:

$$\begin{cases} u(x) \approx N(x)\cdot d \\ \\ v(x) \approx N(x)\cdot c \end{cases} \Longrightarrow \begin{cases} u'(x) \approx \frac{\partial N(x)}{\partial x}\cdot d = B(x)\cdot d \\ \\ v'(x) \approx \frac{\partial N(x)}{\partial x}\cdot c = B(x)\cdot c \end{cases} \tag{19}$$

By substituting these definitions into equation 18, the equivalent equation is obtained:

$$c^T\cdot\left[\underbrace{\left(\int_0^L B^T(x)\cdot B(x)dx - \rho\cdot\int_0^L N^T(x)\cdot N(x)dx\right)}_{K}\cdot d + \underbrace{s\int_0^L N^T(x)\cdot x^2dx - \frac{g\cdot\pi^2}{L}\cdot N(L)^T}_{F}\right] = 0 \tag{20}$$

From the definitions of K and F matrix, the equation above can be written as:

$$c^T\cdot(K\cdot d - F) = 0 \tag{21}$$

The equation above is valid for all coefficients in vector c that satisfy the condition $v(0) = N(0)\cdot c = 0$. As for polynomial and finite elements basis functions $v(0) = c_1 = 0$, then the vector c has the following form:

$$c = \begin{bmatrix} c_r \\ c_l \end{bmatrix} = \begin{bmatrix} 0 \\ c_l \end{bmatrix} \quad r = 1 \text{ and } \forall l \in [2,n] \tag{22}$$

Being $n \in \mathbb{N}$ the total number of elements of the vector.
From the conditions given in the problem, d vector can be also determined as the condition $u(0) = N(0)\cdot d = -g; d_r = -g$ is known:

$$d = \begin{bmatrix} d_r \\ d_l \end{bmatrix} = \begin{bmatrix} -g \\ d_l \end{bmatrix} \quad r = 1 \text{ and } \forall l \in [2,n] \tag{23}$$

Finally, the expressions of matrix K and vector F are the following:

$$K = \begin{bmatrix} K_{rr} & K_{rl} \\ K_{lr} & K_{ll} \end{bmatrix} \quad ; \quad F = \begin{bmatrix} F_r \\ F_l \end{bmatrix} \quad r = 1 \text{ and } \forall l \in [2,n] \tag{24}$$

We can define the reaction forces vector $R = K\cdot d - F$ as follows:

$$R = \begin{bmatrix} K_{rr} & K_{rl} \\ K_{lr} & K_{ll} \end{bmatrix}\cdot\begin{bmatrix} d_r \\ d_l \end{bmatrix} - \begin{bmatrix} F_r \\ F_l \end{bmatrix} = \begin{bmatrix} R_r \\ R_l \end{bmatrix} \quad r = 1 \text{ and } \forall l \in [2,n] \tag{25}$$

Substituting into equation 21, the following equation is obtained:

$$c^T \cdot R = \begin{bmatrix} c_r & c_l \end{bmatrix} \cdot \begin{bmatrix} R_r \\ R_l \end{bmatrix} = \begin{bmatrix} 0 & c_l \end{bmatrix} \cdot \begin{bmatrix} R_r \\ R_l \end{bmatrix} = 0 \Longrightarrow c_l \cdot R_l = 0 \Longrightarrow R_l = 0 \tag{26}$$

Then the formulation of equation 18 in terms of a generical matrix of basis functions and their corresponding derivatives is the following:

$$\begin{bmatrix} K_{rr} & K_{rl} \\ K_{lr} & K_{ll} \end{bmatrix} \cdot \begin{bmatrix} -g \\ d_l \end{bmatrix} - \begin{bmatrix} F_r \\ F_l \end{bmatrix} = \begin{bmatrix} R_r \\ 0 \end{bmatrix} \quad r = 1 \text{ and } \forall l \in [2, n] \tag{27}$$

Then, the system of equations obtained is the following:

$$\begin{cases} Rr = K_{rl} \cdot d_l - g \cdot K_{rr} + F_r \\ \\ d_l = K_{ll}^{-1} \cdot (F_l + g \cdot K_{lr}) \end{cases} \tag{28}$$

Where R_r is the value of the reaction forces of the system and the vector d represents the solution of equation $u(x)$.

1.5 Approximation to the solution of this variational form by using polynomial basis functions of increasing order.

In particular, the following polynomial basis functions are used: $\mathbf{N} = [\mathbf{1}, \mathbf{x}]$, $\mathbf{N} = [\mathbf{1}, \mathbf{x}, \mathbf{x^2}]$, $\mathbf{N} = [\mathbf{1}, \mathbf{x}, \mathbf{x^2}, \mathbf{x^3}]$, $\mathbf{N} = [\mathbf{1}, \mathbf{x}, \mathbf{x^2}, \mathbf{x^3}, \mathbf{x^4}]$.

In order to determine the approximation of the solution of the variational form of the problem by using polynomial basis functions, the equations obtained in the previous section have been implemented into the Matlab code shown in Appendix A.2. In this code, the system of equations in 45 has been solved applying Galerkin Method.

The approximations of $u(x)$ obtained from this Matlab code are the following:

- First order polynomial approximation: $y_1(x) = 0.0848x - 0.0100$
- Second order polynomial approximation: $y_2(x) = -0.0153x^2 + 0.0946x - 0.0100$
- Third order polynomial approximation: $y_3(x) = 0.0417x^3 - 0.0819x^2 + 0.1196x - 0.0100$
- Fourth order polynomial approximation: $y_4(x) = 0.1108x^4 - 0.1787x^3 + 0.0472x^2 + 0.0993x - 0.0100$

Figure 3 shows a representation of these polynomial approximation of $u(x)$ together with the exact solution obtained in section 1.2.

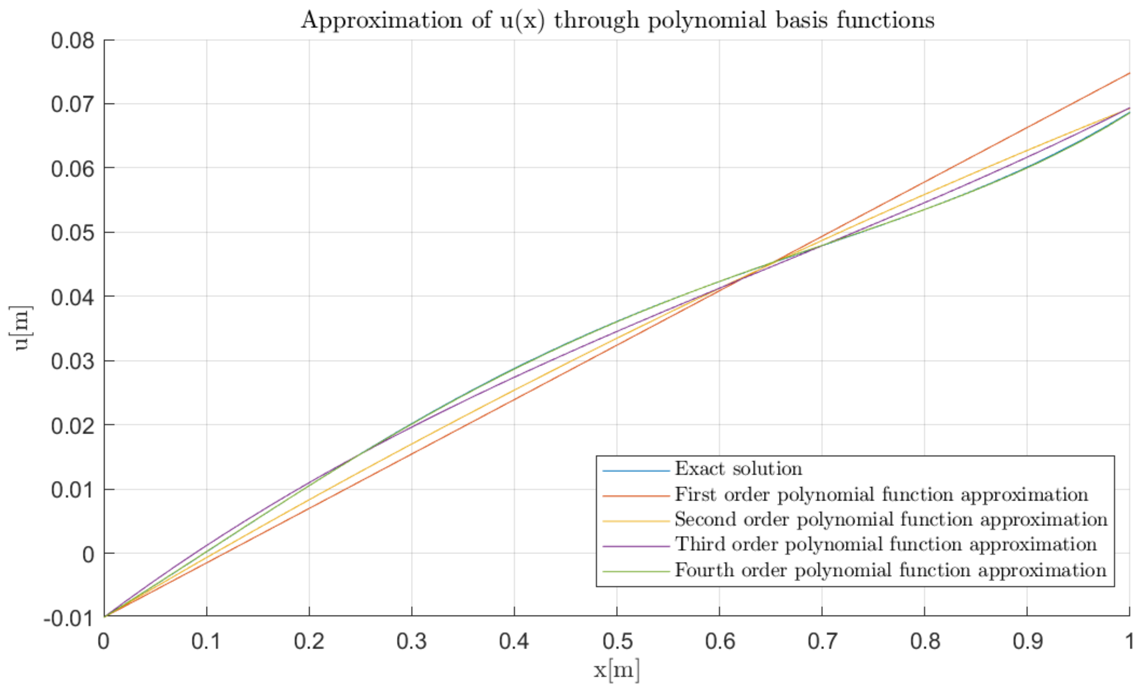

Figure 3: Exact solution and approximation of $u(x)$ through polynomial basis functions.

For a better visualization of the convergence in solution for increasing order polynomial functions, figure 14 shows the individual representation of each polynomial basis function solution with the exact solution.

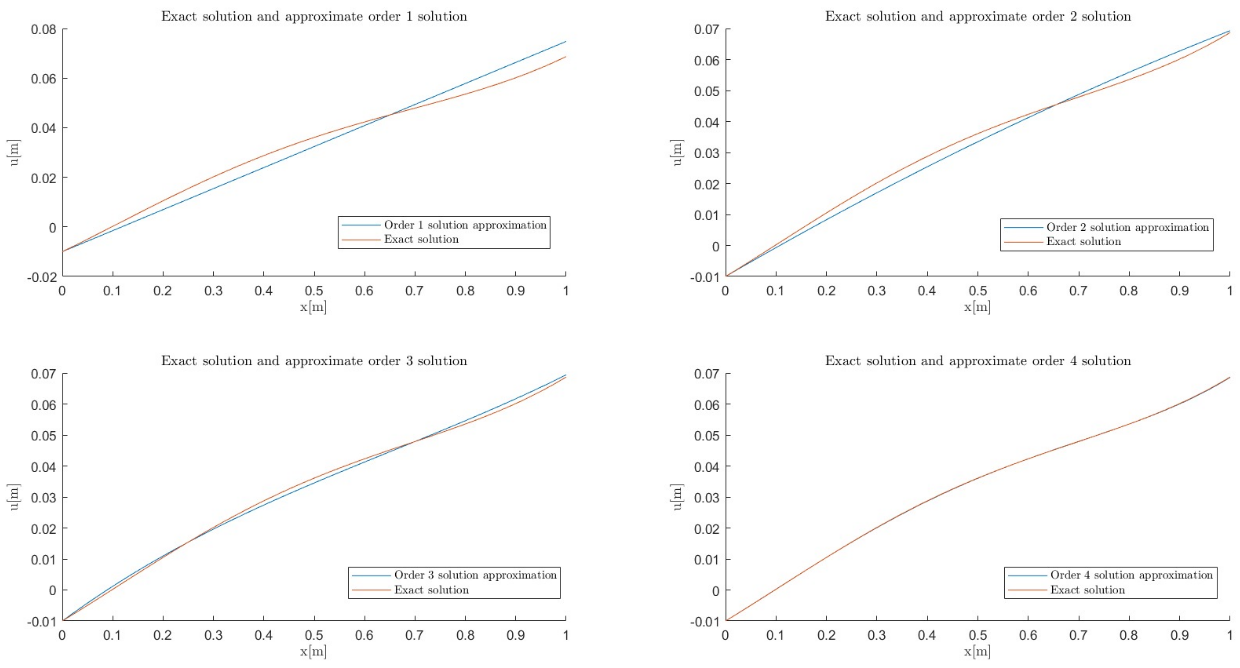

Figure 4: Exact solution and approximation of $u(x)$ through polynomial basis functions.

The convergence of these approximate solutions can be computed in terms of the absolute error in the approximation relative to the exact solution. The expression of this error is the following:

$$\text{error} = \left(\int_0^L (u(x) - p(x))^2 dx\right)^{1/2} \tag{29}$$

The error values obtained are the following ones:

- Error in first order polynomial approximation: $3.9 \cdot 10^{-3}$
- Error in second order polynomial approximation: $2.2 \cdot 10^{-3}$
- Error in third order polynomial approximation: 10^{-3}
- Error in fourth order polynomial approximation: $6.3424 \cdot 10^{-5}$

As can be seen, approximation error is reduced significatively as the polynomial basis order is increased.

1.6 Development of a MatLab program able to solve the variational form using finite elements basis functions using number of equally sized finite elements (n) as program input.

In order to develop the Matlab code in this section, K matrix and F vector defined in section 1.4 will need to be solved from the elemental point of view. First of all, the matrix of elemental shape functions $N^e(x)$ is defined in the parent domain $\Omega^e = [\xi_1, \xi_2] = [-1, 1]$ as follows:

$$N^e(\xi) := \frac{1}{2} \cdot \begin{bmatrix}(1-\xi) & (1+\xi)\end{bmatrix} \tag{30}$$

Then, the matrix of derivative shape functions in the parent domain can be expressed as:

$$B^e(x) := \frac{\partial N^e}{\partial x} = \frac{\partial N^e}{\partial \xi} \cdot \frac{\partial \xi}{\partial x} = \frac{1}{2} \cdot \begin{bmatrix}-1 & 1\end{bmatrix} \cdot \frac{\partial \xi}{\partial x} \tag{31}$$

Defining $x = x^e \cdot N^e$, and h^e as the length of the local element in the parent domain, then:

$$x = N^e(\xi) \cdot \begin{bmatrix} x_1^e \\ x_2^e \end{bmatrix}; \tag{32}$$

$$\int_{\Omega_e} dx = \int_{-1}^{1} h^e \cdot \frac{1}{2} d\xi = \frac{h^e}{2} \cdot 2 = h^e \cdot \begin{bmatrix}-1 & 1\end{bmatrix} \tag{33}$$

Then the derivatives shape function $B^e(x)$ can be expressed as:

$$\frac{\partial \xi}{\partial x} = \left(\frac{\partial x}{\partial \xi}\right)^{-1} = \left(\frac{\partial N^e}{\partial \xi} \cdot x^e\right)^{-1} = 2 \cdot (-x_1^e + x_2^e)^{-1} = \frac{2}{h^e} \Longrightarrow B^e(\xi) = \frac{1}{h^e}\begin{bmatrix}-1 & 1\end{bmatrix} \tag{34}$$

Once the expressions of $N^e(\xi)$ and $B^e(\xi)$ have been obtained, K^e and F^e will be determined from the expressions in section 1.4.

$$K^e = \int_{x_1^e}^{x_2^e} B^{e^T}(x) \cdot B^e(\xi) \frac{h^e}{2} d\xi - \rho \int_{x_1^e}^{x_2^e} N^{e^T}(\xi) \cdot N^e(\xi) \frac{h^e}{2} d\xi \tag{35}$$

In order to see more properly the steps followed, the expressions of both terms in the RHS will be determined separately:

$$\int_{x_1^e}^{x_2^e} B^{e^T}(\xi) \cdot B^e(\xi) \frac{h^e}{2} d\xi = \int_{-1}^{1} (B^e(\xi))^T \cdot B^e(\xi) \cdot \frac{h^e}{2} \frac{h^e}{2} d\xi = \frac{1}{h^e} \cdot \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{36}$$

$$\rho \cdot \int_{x_1^e}^{x_2^e} (N^e(\xi))^T \cdot N^e(\xi) \cdot \frac{h^e}{2} dx = \rho \cdot \left(\frac{1}{2}\right)^2 \int_{-1}^{1} \cdot \begin{bmatrix}(1-\xi) & (1+\xi)\end{bmatrix}^T \cdot \begin{bmatrix}(1-\xi) & (1+\xi)\end{bmatrix} \cdot \frac{h^e}{2} \cdot d\xi = \\ = \frac{\rho \cdot h^e}{8} \cdot \begin{bmatrix}(1-\xi)^2 & (1-\xi^2) \\ (1-\xi^2) & (1-\xi)^2\end{bmatrix} = \frac{\rho \cdot h^e}{6} \cdot \begin{bmatrix}2 & 1 \\ 1 & 2\end{bmatrix} \tag{37}$$

Finally, K^e can be expressed as follows:

$$K^e(\xi) = \frac{1}{h^e} \cdot \begin{bmatrix}1 & -1 \\ -1 & 1\end{bmatrix} + \frac{\rho \cdot h^e}{6} \cdot \begin{bmatrix}2 & 1 \\ 1 & 2\end{bmatrix} \tag{38}$$

Next, the expression of vector $F^e(\xi)$ will be determined:

$$F^e(\xi) = -s \int_{x_1^e}^{x_2^e} N^{e^T}(x) \cdot x^2 \cdot \frac{h^e}{2} d\xi + \frac{g\pi^2}{L} \cdot N^{e^T}(L) \tag{39}$$

This integral has been solved by implementing a Matlab code shown in Appendix A.3.2.
The result obtained after computing the integral is the following:

$$s \int_{x_1^e}^{x_2^e} N^{e^T}(x) \cdot x^{e^2} \frac{h^e}{2} \cdot d\xi = \frac{s \cdot h^e}{12} \begin{bmatrix}3 \cdot x_1^2 + 2 \cdot x_1 \cdot x_2 + x_2^2 \\ x_1^2 + 2 \cdot x_1 \cdot x_2 + 3 \cdot x_2^2\end{bmatrix} \tag{40}$$

Finally, once these expressions have been obtained, a Matlab program for solving the variational form using finite element functions can be developed. The code for this program is shown in Appendix A.3.1 This program uses a function called *AssemblyMatrixKFd* (appendix A.3.4) which computes K and F matrix and the solves for displacements vector d, applying the conditions determined in 45 for solving the approximate displacement of each element.

1.7 Problem analysis and solution for increasing number of finite elements. Plot of approximate solutions with discretizations for n = [5, 10, 15, 20] and comparison with the exact solution.

For this section, the results of the code developed from results in section 1.6 are shown for different number of discretization elements. This code is shown in Appendix A.3.3
The results after executing the program for a discretization of 5, 10, 15 and 20 elements are shown in the following plots.

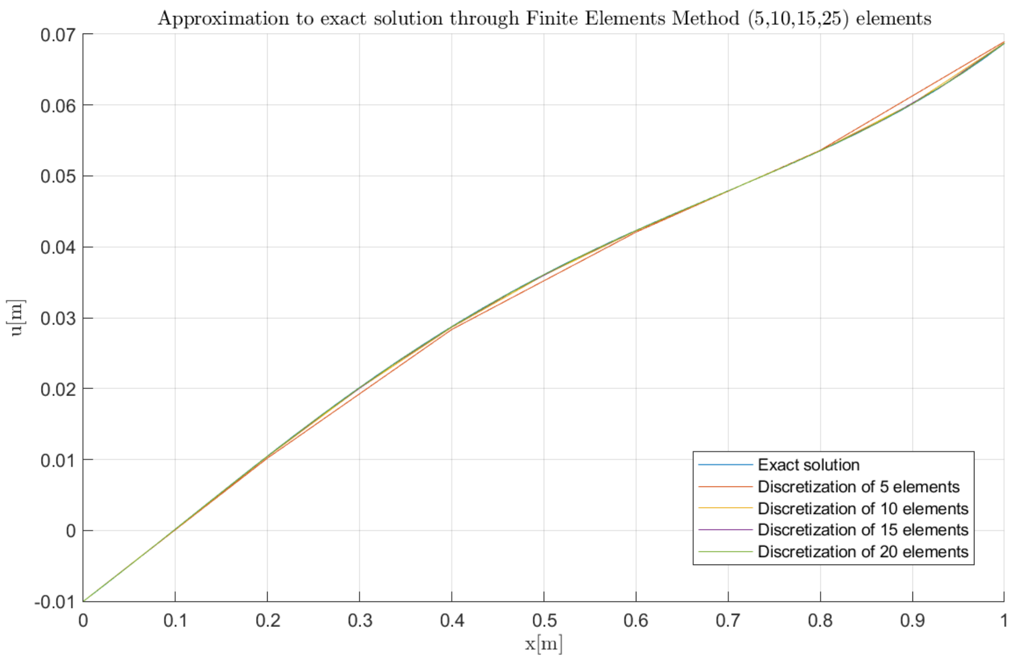

Figure 5: Representation of exact solution and approximate solutions for 5,10,15 and 25 discretization elements.

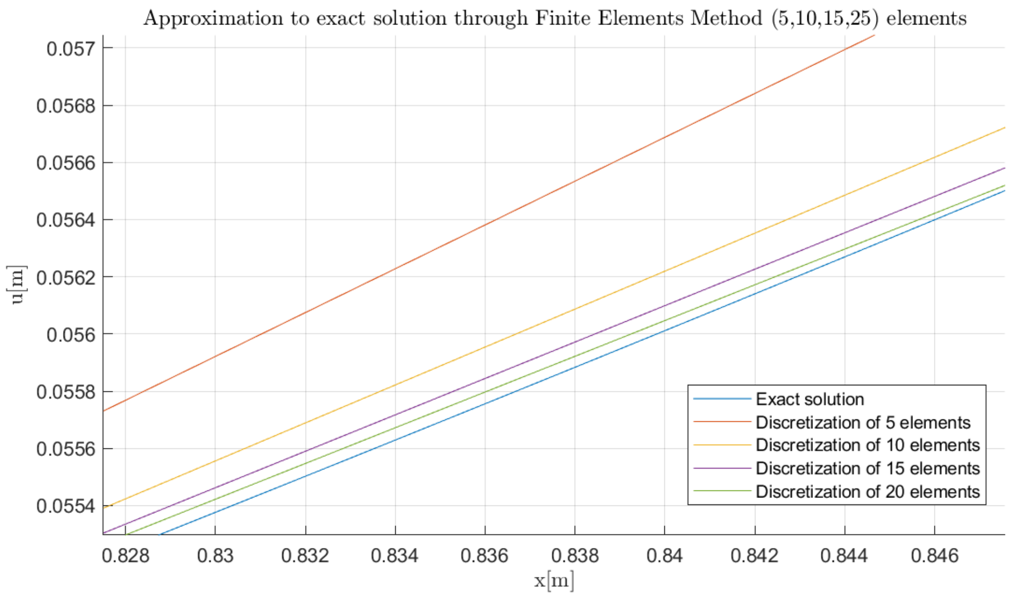

Figure 6: Representation of exact solution and approximate solutions for 5,10,15 and 25 discretization elements (expanded view).

From these plots, it is concluded that as the number of discretization elements is increased, the solution given by this finite elements methods is closer to the exact solution determined.

2 Part 2

2.1 Implementation of a function able to calculate the approximation error for both u and its derivative u'. Plot of the error versus the element size in a logarithmic graphic. Study of the slope of the convergence plot in each case.

$$\|e^h\|_{L_2} = \left(\int_0^1 \left(u(x) - u^h(x)\right)^2\right)^{\frac{1}{2}} \tag{41}$$

$$\|e^{h'}\|_{L_2} = \left(\int_0^1 \left(u'(x) - u^{h'(x)}\right)^2\right)^{\frac{1}{2}} \tag{42}$$

The integrals will be computed by subdividing the domain into elements, and the applying Gauss quadrature into each element. A Gauss integration rule with 3 gauss points ($m = 3$), which is appropiate for approximating polynomials or orders lower than 5, according to the expression $p \leq 2 \cdot m - 1; \quad (m = 3); \quad p \leq 5$. Gauss quadrature integration rule allows to get a numerical approximation of an integral over the parent domain $\Omega^e = [\xi_1, \xi_2] = [-1, 1]$.

$$I = \int_{-1}^{1} q(\xi)d\xi \approx \sum_{g=1}^{m} w_g q(\xi_g) \tag{43}$$

In the case $m = 3$, the tabulated values of gauss points weights (w) and positions (ξ) are the following:

$$\begin{cases} \xi_1 = \sqrt{\frac{3}{5}} \quad ; \quad w_1 = \frac{5}{9} \\ \xi_2 = 0 \quad ; \quad w_2 = \frac{8}{9} \\ \xi_3 = -\sqrt{\frac{3}{5}} \quad ; \quad w_3 = \frac{5}{9} \end{cases} \tag{44}$$

This integration rule has been implemented in a Matlab code routine, which can be consulted in Appendix A.4.1. The computational application of Gauss quadrature to the integrals in equations 41 and 42 has allowed to study the relationship between the evolution in the approximation error and the element length in the FEM discretization. These results are shown in the graphs in figures 7 and 8.

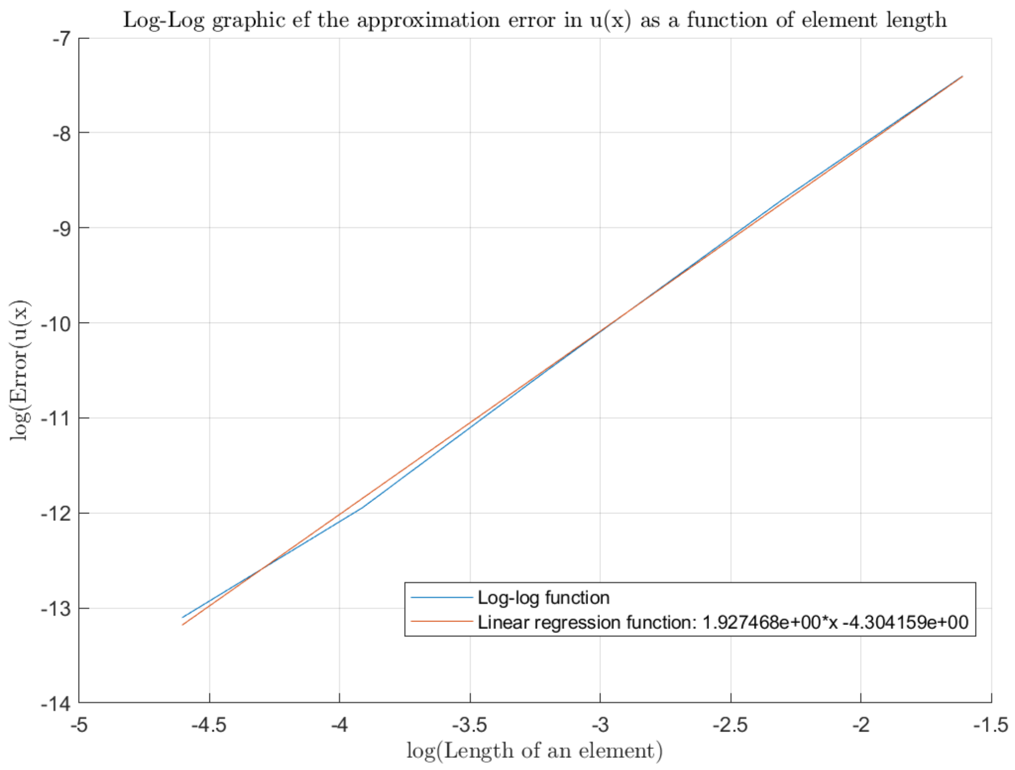

Figure 7: Log-Log plot that shows the evolution of the error in $u(x)$ finite elements approximation as function of the element length in the discretization.

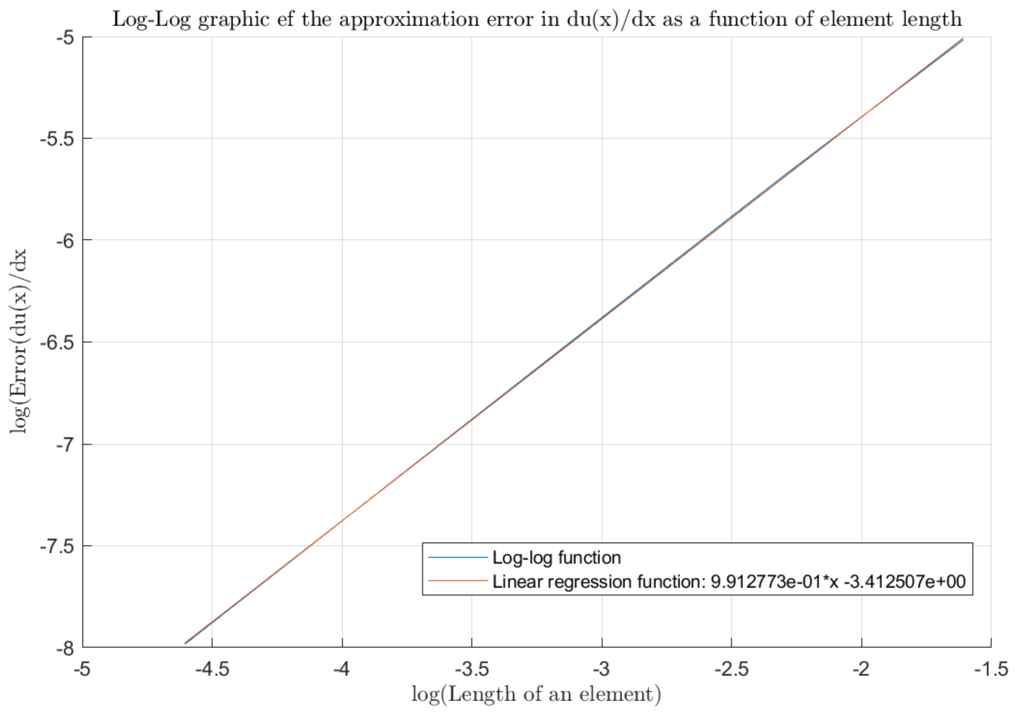

Figure 8: Log-Log plot that shows the evolution of the error in $u'(x)$ finite elements approximation as function of the element length in the discretization.

As shown in the graphs, a linear regression function has been determined in each case in order to analyse this relationship in the case explained in the statement. The following linear regression functions have been obtained:

$$\begin{cases} \frac{\log(u(x))}{log(h^e)} \approx 1.9275 \cdot x - 4.3042 \\ \\ \frac{\log(u'(x))}{log(h^e)} \approx 0.99128 \cdot x - 3.4125 \\ \end{cases} \tag{45}$$

As it could be expected, the error in $u(x)$ has a higher growth with the length of the elements in discretization than the relative to the function $u(x)$.

3 Advanced Assignment 1 : Nonlinear 1D equilibrium

General statement of the assignment

A bar of length L and varying cross-sectional area

$$A(x) = A_0 \cdot \left[1 + \frac{2x}{L} \cdot \left(\frac{x}{L} - 1\right)\right] \tag{46}$$

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).

$$\sigma = \sigma_0 \cdot \left(1 - e^{-\frac{E}{\sigma_0} \cdot \varepsilon}\right) \tag{47}$$

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

3.1 Formulation of the strong form of the Boundary Value Problem (BVP) for the 1D equilibrium of a bar with varying cross-sectional area.

The formulation of the strong form in the case of a bar with varying cross-sectional area is similar to the process developed in the first section. The relationship between the stress in the bar and its cross sectional area is given by the force applied

$$A(x) \cdot \sigma(\varepsilon) = F \tag{48}$$

Then, considering $\varepsilon = u'(x)$ and taking derivatives in both sides of the equation, the following expression is obtained:

$$\frac{\partial}{\partial x}[A(x) \cdot \sigma(u'(x))] = \frac{\partial F}{\partial x} \tag{49}$$

As the force is constant along the beam, the RHS term is neglected in the equation above. Finally, the boundary conditions will be set in this case. From the information in the statement, these boundary conditions are $u_0 = 0$ and $u_m = \frac{T}{t}$. Therefore, the strong form of the BVP in this case is the following:

$$\begin{cases} \frac{\partial}{\partial x}[A(x) \cdot \sigma(u'(x))] = 0 \\ u_0 = u(0) = 0 \\ u_L(t) = u_m \cdot \frac{T}{t} \end{cases} \tag{50}$$

3.2 Determination of the weak form of the BVP.

In order to obtain the weak form of the BVP in this case, the strong form obtained will be integrated and multiplied by an arbitrary function $v(x) : v(0) = 0$ and integrated over the domain $x \in [0, L]$.

$$\int_0^L v \cdot \frac{\partial}{\partial x} \cdot [A(x) \cdot \sigma(u'(x))] dx = 0 \tag{51}$$

Applying integration by parts, the following relation is obtained:

$$\frac{\partial v(x) \cdot [A(x) \cdot \sigma(u'(x))]}{\partial x} = [A(x) \cdot \sigma(u'(x))] \cdot \frac{\partial v(x)}{\partial x} + v(x) \cdot \frac{\partial A(x) \cdot \sigma(u'(x))}{\partial x} \tag{52}$$

Substituting this relation into equation 54, the following expression is obtained:

$$v \cdot A(x) \cdot \sigma(u'(x))|_L = \int_0^L \frac{\partial v}{\partial x} \cdot A(x) \cdot \sigma(u'(x)) \cdot dx \tag{53}$$

$$\int_0^L v \frac{\partial}{\partial x} \cdot [A(x) \cdot \sigma(u'(x))] dx = \int_0^L \cdot \frac{\partial v(x) \cdot [A(x) \cdot \sigma(u'(x))]}{\partial x} dx - \int_0^L \cdot [A(x) \cdot \sigma(u'(x))] \cdot \frac{\partial v(x)}{\partial x} dx = 0 \tag{54}$$

Therefore, the Variotional form of the problem in this case has the following formulation:

$$\boxed{\int_0^L \frac{\partial v(x)}{\partial x} \cdot [A(x) \cdot \sigma(u'(x))] dx = v(L) \cdot A(L) \cdot \sigma(u'(L))} \quad \forall v \in v^h \text{ with } v(0) = 0 \tag{55}$$

3.3 Formulation of the corresponding matrix equations

As it was explained in section 1.4, in order to obtain the corresponding matrix equations of the Variational form of the problem, the first step is defining $u(x)$ and $v(x)$ as linear combination of known functions $N(x)$, $B(x)$:

$$\begin{cases} u(x) \approx N(x) \cdot d \\ \\ v(x) \approx N(x) \cdot c \end{cases} \Longrightarrow \begin{cases} u'(x) \approx \frac{\partial N(x)}{\partial x} \cdot d = B(x) \cdot d \\ \\ v'(x) \approx \frac{\partial N(x)}{\partial x} \cdot c = B(x) \cdot c \end{cases} \tag{56}$$

From the expression of the Variational form of the problem obtained in the previous section, and taking into account the definitions in 56, the following formulation can be made:

$$c^T \cdot \int_0^L B^T(x) \cdot A(x) \cdot \sigma(B(x) \cdot d) dx = c^T \cdot N^T(L) \cdot A(L \cdot \sigma(B(L) \cdot d)) \tag{57}$$

By excluding the factor c^T in the previous result, the internal forces vector can be defined:

$$F = \int_0^L B^T(x) \cdot A(x) \cdot \sigma(B(x) \cdot d) dx = N^T(L) \cdot A(L \cdot \sigma(B(L) \cdot d)) \tag{58}$$

Finally, from this formulation a system of nonlinear equations in the displacement terms d_l is obtained:

$$F_l(d_l) = F_l \tag{59}$$

3.4 Solution of the resulting system of nonlinear equations by means of a Newton-Raphson algorithm.

In order to solve the system of nonlinear equations using Newton-Raphson algorithm, the internal forces vector in the elementwise form must be computed. From the definition of the vector F obtained in the previous section, F^e can be expressed as follows:

$$F^e = \int_{x_1^e}^{x_2^e} B^{e^T}(\xi) \cdot A(\xi) \cdot \sigma(B^e(\xi) \cdot d) \frac{h^e}{2} d\xi \tag{60}$$

In order to get the solution of the resulting system, the Matlab code shown in Appendix **??** computes the internal forces vector residual, the stress and the strain field using an iterative routine that calculates the displacement and compares the corresponding value of the residual with a certain numerical tolerance stablished (10^{-10} was used in this case). Then, if the residual has a higher value than the tolerance, the program proposes a new displacement and repeats the calculations in a new iteration until achieving convergence in the displacement values.

In order to compute that process, the assembly of K and F_{int} matrix has been necessary for the new conditions in this case. The codes implemented for that process are shown in appendix A.5.2 and A.5.3. These functions support the main routine that solves this system of nonlinear equation for displacements.

Then, the strain and displacement fields as a function of the position over the bar have been determined by applying this algorithm for 50 elements and 51 time steps. These functions are shown in figures 9, 10 and 11.

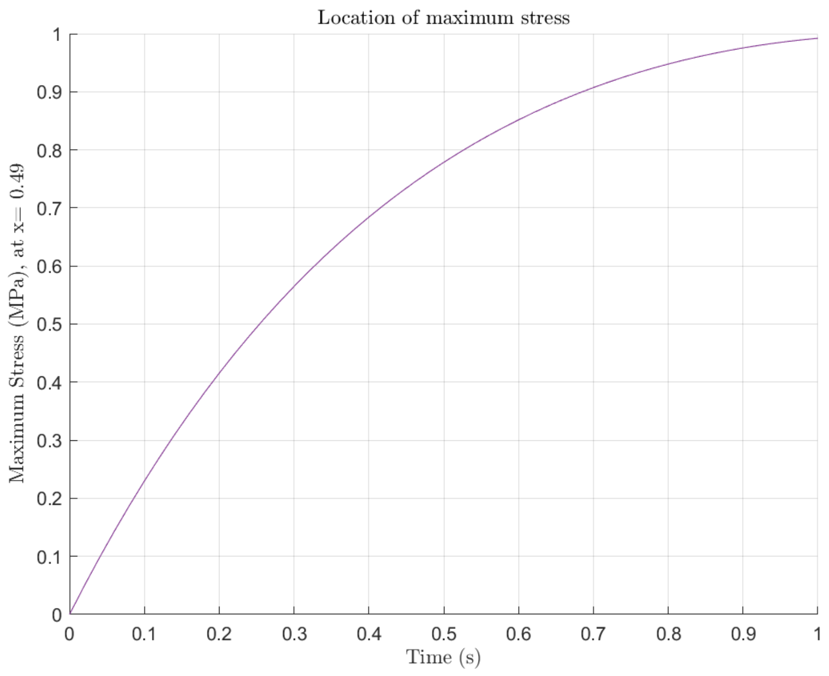

Figure 9: Location of the maximum stress over time for a discretization of the domain in 50 elements and 51 time steps.

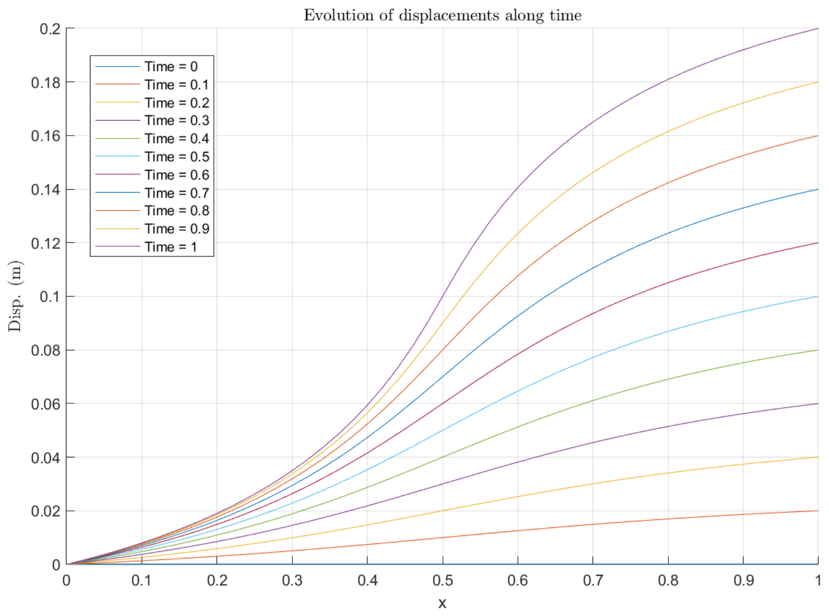

Figure 10: Evolution of the displacement field over time for a discretization of the domain in 50 elements and 51 time steps.

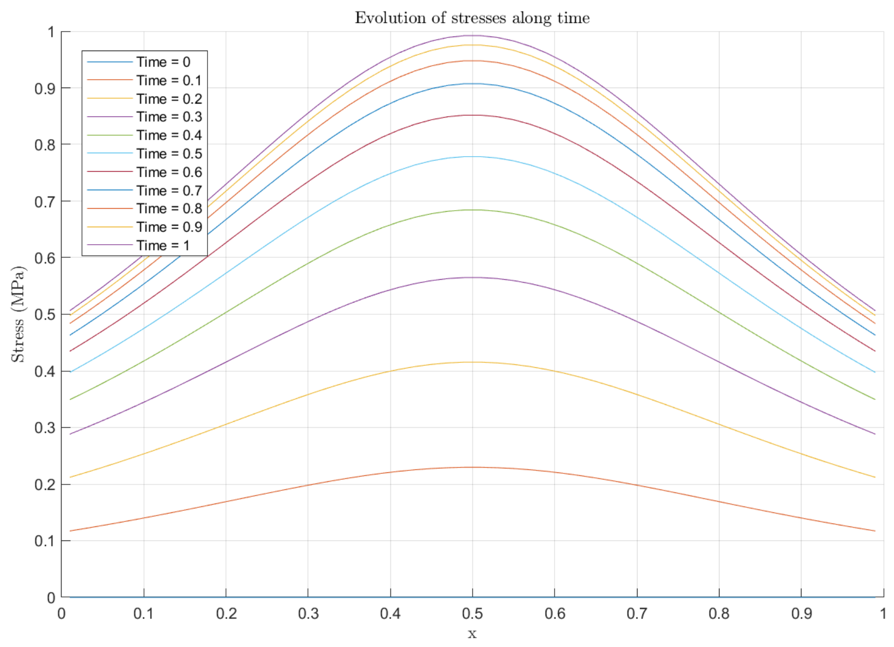

Figure 11: Evolution of the stress field over time for a discretization of the domain in 50 elements and 51 time steps.

3.5 Verification of solution convergence upon increasing the number of elements and time steps

The verification of the convergence in the solution will be undertaken by checking that the result is stable for increasing number of elements and time steps. In order to undertake the convergence study, a vector that includes a different number of elements and time steps (5,10,25,50,100 and 250 in this case) has been defined, and the algoritm explained in the previous section has been computed in different iterations for these elements. The code implemented for convergence study is shown in Appendix A.5.4.

Then, the results obtained for this study are presented in the graphs in figures 12, 14 and 16. In order to see the convergence to the solution in the different cases, a detail view for each case has been included in each case (figures 13, 15 and 17).

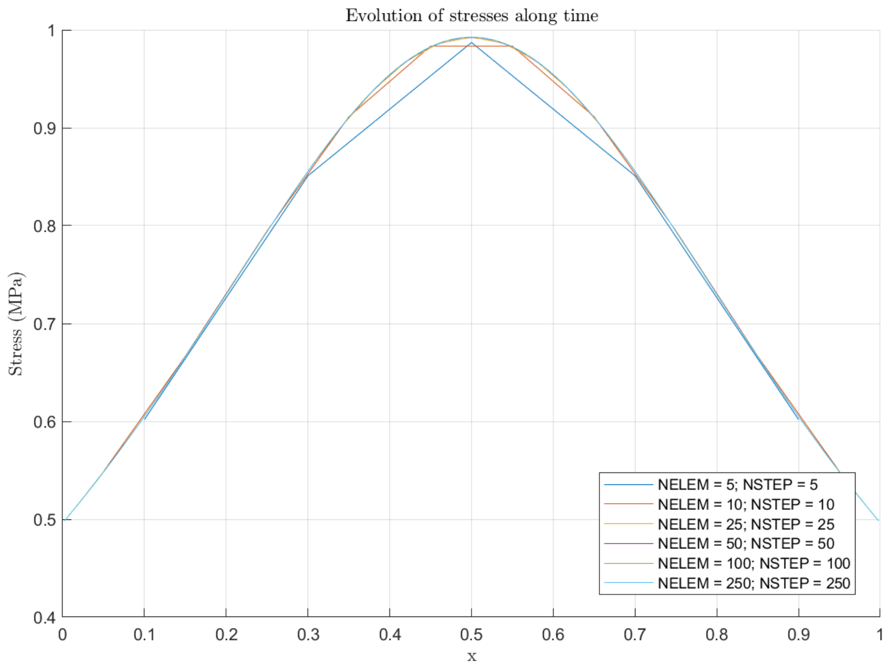

Figure 12: Evaluation of the convergence in the stress field function over time for a range of different numbers and time steps.

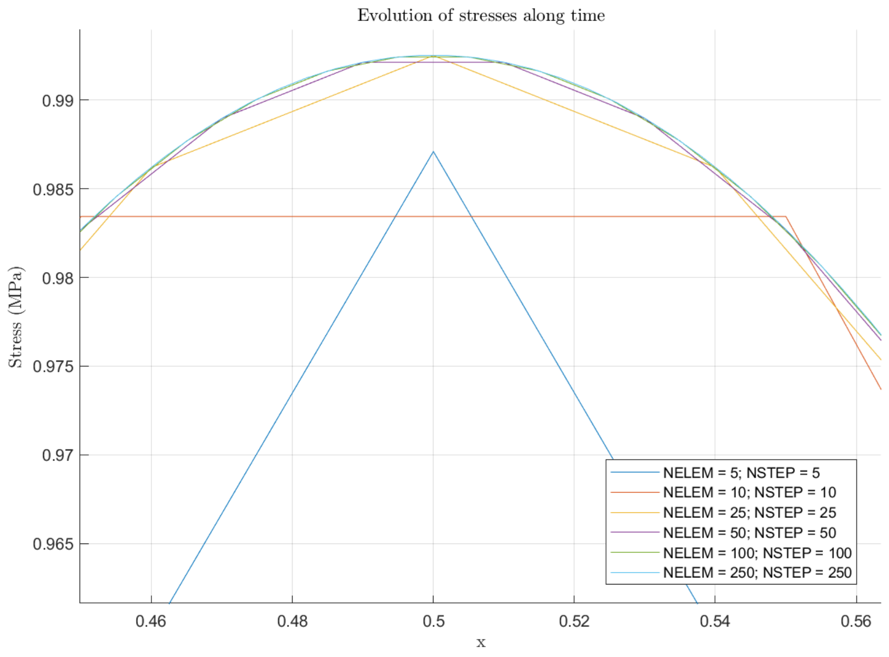

Figure 13: Evaluation of the convergence in the displacements field function over time for a range of different numbers and time steps.

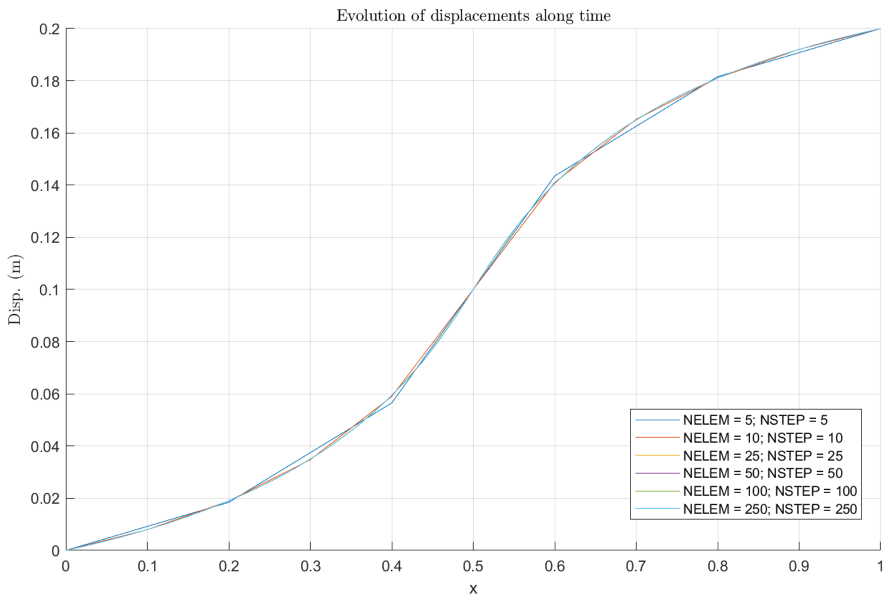

Figure 14: Caption

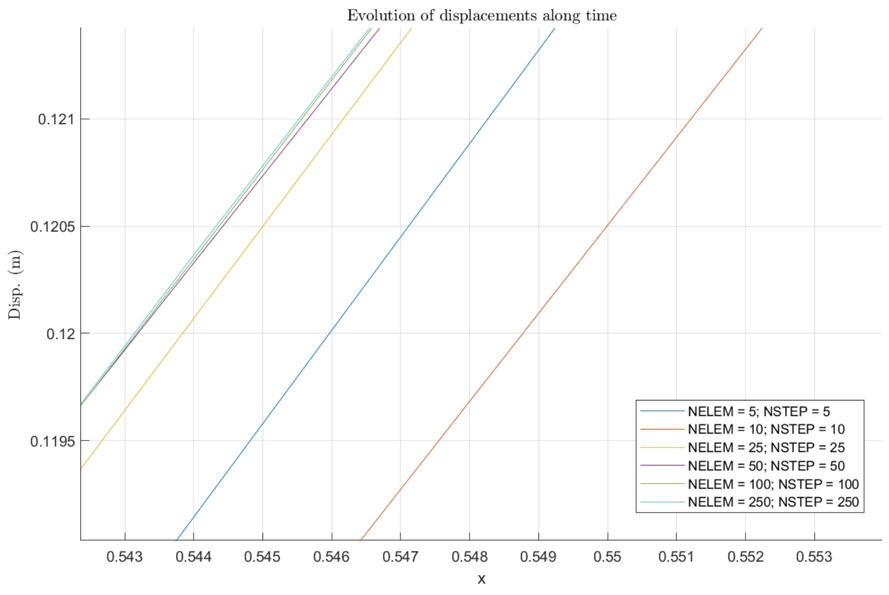

Figure 15: Detail view of the convergence in the displacements field over time for a increasing number of elements and time steps

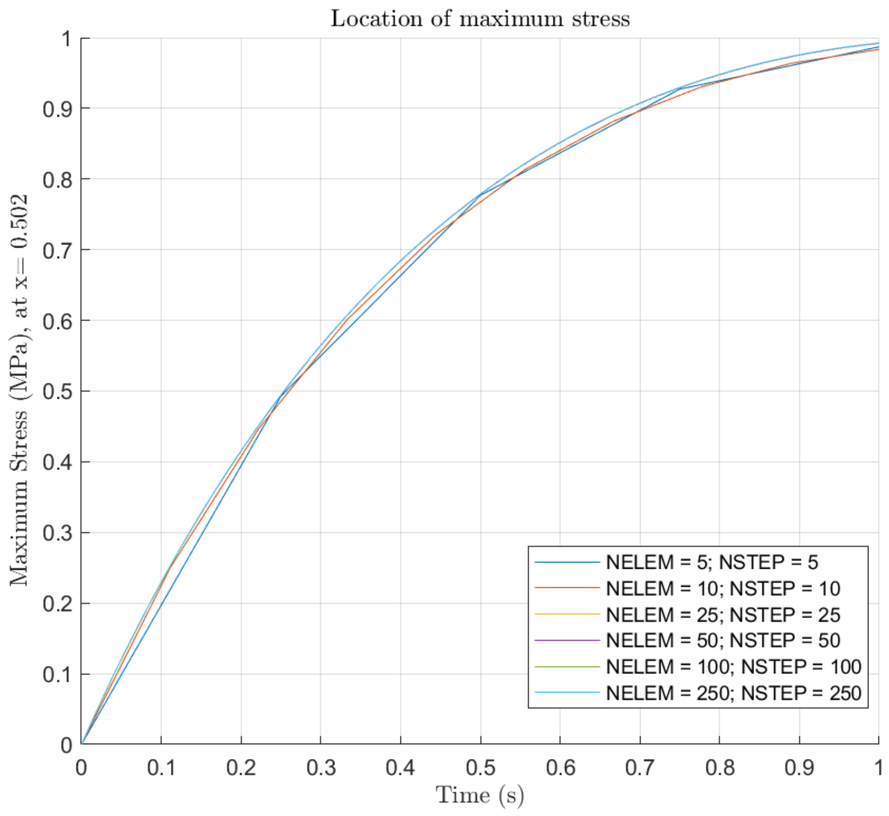

Figure 16: Evaluation of the convergence in the location of the maximum stress over time for a range of different numbers and time steps.

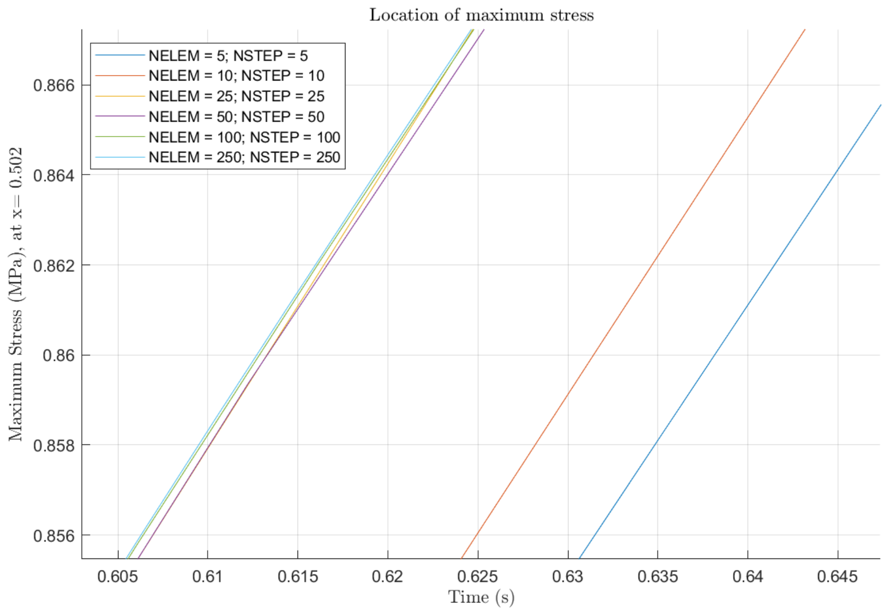

Figure 17: Detail view of the convergence in the location of the maximum stress over time for a increasing number of elements and time steps

As it can be deducted from these plots, the results are coherent with the ones in the previous section, hence the solution is stable and the code has converged.

A Appendix

A.1 Code for section 1.2

```matlab
clc; % Clean the command window
clear all; % Clean the workspace
syms x u(x); % Define symbolic variables/functions

L = 1;                    %[m]
g = 0.01;                 %[m]
rho = pi^2/L^2;           %[1/m^2]
s = g*rho^2;              %[1/m^3]
b = g*pi^2/L;             %[] (Point load at x=L)
fe = -s*x^2;              %[m]
q_E = rho*u+fe;           % Source function (Force over the bar)

% Exact solution
 Du = diff(u,x);
 uexact = dsolve(diff(Du,x)+q_E==0,u(0)==-g,Du(L)==b) ;
 uexact_simplified=simplify(uexact);
 uexact_display=['Exact solution: ',char(vpa(uexact_simplified))];
 disp(uexact_display); % Output exact solution
 figure("Name",'Exact solution of the Boundary Value Problem');
 fplot(uexact, [0 L]);
 title('Exact solution u(x) of the Bounday Value Problem','Interpreter','latex');
 xlabel('x [m]','Interpreter','latex');
 ylabel('u [m]','Interpreter','latex');
 grid on;
```

A.2 Code for section 1.5

A.2.1 Main program

```matlab

clc; % Clean the command window
clear all; % Clean the workspace
syms x u(x); % Define symbolic variables/functions

L = 1;                    %[m]
g = 0.01;                 %[m]
rho = pi^2/L^2;           %[1/m^2]
s = g*rho^2;              %[1/m^3]
b = g*pi^2/L;             %[] (Point load at x=L)
fe = -s*x^2;              %[m]
q_E = rho*u+fe;           % Source function (Force over the bar)
N = [1 x x^2 x^3 x^4]; % Basis functions (polynomials)

%%  Plotting the functions
figure('Name','Approximation to exact solution through elemental polynomial bases of ...
    increasing order') ;
hold on;
title('Approximation of u(x) through polynomial basis functions','Interpreter','latex');
xlabel('x[m]','Interpreter','latex') ;
ylabel('u[m]','Interpreter','latex');
fplot(uexact,[0 L]);
grid on;

for a = 2:length(N)
    uapprox(a-1) = GalerkinMethod(N(1:a),fe,b,g,rho,L);
    fplot(uapprox(a-1),[0 L]);
    uapprox_display=['Approximate solution order ...
        ',num2str(a-1),':',char(vpa(uapprox(a-1),a))];
    disp(uapprox_display);
end
legend('Exact solution','First order polynomial function approximation','Second order ...
    polynomial function approximation','Third order polynomial function ...
    approximation','Fourth order polynomial function ...
    approximation','Interpreter','latex','Location','best');
hold off;

figure('Name','Approximation to exact solution through elemental polynomial bases of ...
    increasing -order');
for a=1:length(N)-1
subplot(2,2,a)
hold on;
fplot(uapprox(a),[0 L]);
title_uapprox_display=['Exact solution and approximate order ',num2str(a),' solution'];
title(title_uapprox_display,'Interpreter','latex');
xlabel('x[m]','Interpreter','latex');
ylabel('u[m]','Interpreter','latex');
fplot(uexact,[0 L]);
legend_uapprox_display=['Order ',num2str(a),' solution approximation'];
legend(legend_uapprox_display,'Exact solution','Interpreter','latex',Location='best');
hold off;
grid on;
end

%% Determination of error associated to each polynomial basis function

 for a=2:length(N)
     error_polynomial=sqrt(int((uexact-GalerkinMethod(N(1:a),fe,b,g,rho,L))^2,x,[0 L]));
     error_display=['Error in order ',num2str(a-1),' polynomial ...
         approximation:',char(vpa(error_polynomial))];
     disp(error_display);
 end
```

A.2.2 Galerkin Method Function

```

function [u]= GalerkinMethod(N,f,b,g,rho,L)
x=sym('x','real');
N_L = subs(N,L);
B = diff(N,x);
NtN = N.'*N;
BtB = B.'*B;
K = int(BtB,x,[0 L])-rho*int(NtN,x,[0 L]);
F = int(N.'*f,x,[0 L])+N_L.'*b;
d=zeros(length(N),1);
r = 1;
l = 2:length(N);
d(r)=-g;
d(l)=K(l,l)\(F(l)-K(l,r)*d(r));

u=N*d;
```

A.3 Code for sections 1.6 and 1.7

A.3.1 Main program for section 1.6

```
    clc; % Clean the command window
clear all; % Clean the workspace
syms x u(x); % Define symbolic variables/functions

```

```
L = 1;                  %[m]
g = 0.01;               %[m]
rho = pi^2/L^2;         %[1/m^2]
s = g*rho^2;            %[1/m^3]
b = g*pi^2/L;           %[] (Point load at x=L)
fe = -s*x^2;            %[m]
q_E = rho*u+fe;         % Source function (Force over the bar)

%% (SECTION 1.6)

% Exact solution
 Du = diff(u,x);
 uexact = dsolve(diff(Du,x)+q_E==0,u(0)==-g,Du(L)==b) ;

%% Application of the Finite Elements Method (FEM)

% Discretization definition
x0=0; %Position of the first node
xF=L; %Position of the last node
n_elements=input('Number of discretization elements: '); % Number of discretization elements

% Coordinates and Connectivity matrix assembly
COOR=linspace(x0,xF,n_elements+1)'; %Coordinates matrix
CN=zeros(n_elements,2);             %Connectivity matrix
for a=1:n_elements
       CN(a,1)=a;
       CN(a,2)=a+1;
end

% Determination of u(x) applying FEM
[K,F,d]=AssemblyMatrixKFd(COOR,CN,L,g);

% Plotting of exact solution and solution obtained from FEM (input number of elements)
figure('Name','Approximation to exact solution through Finite Elements Method')
hold on;
fplot(uexact,[0 L]);
plot(COOR,d);
hold off;
legend('Exact solution',sprintf('Discretization of %d elements',n_elements));
```

A.3.2 Main program for section 1.6

```
clc; clear all; close all;

syms xi xe1 xe2;
Ne=(1/2)*[1-xi, 1+xi];
xe=[xe1;xe2];
x=Ne*xe;
I = int(Ne.'*x^2,xi,-1,+1);
I_simp=simplify(I)
```

A.3.3 Main program for section 1.7

```
    clc; % Clean the command window
clear all; % Clean the workspace
syms x u(x); % Define symbolic variables/functions

L = 1;                  %[m]
g = 0.01;               %[m]
rho = pi^2/L^2;         %[1/m^2]
s = g*rho^2;            %[1/m^3]
```

```
b = g*pi^2/L;            %[] (Point load at x=L)
fe = -s*x^2;             %[m]
q_E = rho*u+fe;          % Source function (Force over the bar)
n_elements = [5 10 15 20]; % Basis functions (polynomials)

%% (SECTION 1.7)

% Exact solution
 Du = diff(u,x);
 uexact = dsolve(diff(Du,x)+q_E==0,u(0)==-g,Du(L)==b) ;

%% Application of the Finite Elements Method (FEM)

% Discretization definition
x0=0; %Position of the first node
xF=L; %Position of the last node

% Plotting of exact solution and solution obtained from FEM (input number of elements)
figure('Name','Approximation to exact solution through Finite Elements Method (5,10,15,25) ...
    elements')
title('Approximation to exact solution through Finite Elements Method (5,10,15,25) ...
    elements','Interpreter','latex');
xlabel('x[m]',Interpreter='latex');
ylabel('u[m]',Interpreter='latex');
hold on;
fplot(uexact,[0 L]);
for i=1:length(n_elements)
     % Coordinates and Connectivity matrix assembly
    COOR=linspace(x0,xF,n_elements(i)+1)'; %Coordinates matrix
    CN=zeros(n_elements(i),2);             %Connectivity matrix
    for a=1:n_elements(i)
           CN(a,1)=a;
           CN(a,2)=a+1;
    end
% Determination of u(x) applying FEM
[K,F,d]=AssemblyMatrixKFd(COOR,CN,L,g);
plot(COOR,d);
end
hold off;
grid on;
legend('Exact solution','Discretization of 5 elements','Discretization of 10 ...
    elements','Discretization of 15 elements','Discretization of 20 ...
    elements',Location='best');
```

A.3.4 AssemblyMatrixKFd Function

```
    function [K,F,d]=AssemblyMatrixKFd(COOR,CN,L,g,b)

%% Definition of parameters

nelem = size(CN,1);              % Number of elements
nnode = nelem+1;                 % Number of nodes
nnodeE = size(CN,2);             % Number of nodes per element
rho = (pi/L)^2;
s = g*rho^2;


% Initialization of K,F,d matrix
K=sparse(nnode,nnode);
F = sparse(nnode,1);
d=sparse(nnode,1);

%% K and F matrix assembly
for e=1:nelem
NODOSe=CN(e,:);
```

```
COORe=COOR(NODOSe);
x1=COORe(1);
x2=COORe(2);
he=x2-x1;

Ke=(1/he)*[1 -1; -1 1]-rho*(he/6)*[2 1; 1 2];
Fe=-s*(he/12)*[3*x1^2+2*x1*x2+x2^2 ; 3*x2^2+2*x1*x2+x1^2];

    for a = 1:nnodeE
           for b = 1:nnodeE
                A = CN(e,a);
                B = CN(e,b);
                K(A,B) = K(A,B) + Ke(a,b);
           end
    end

    F(e,1) = F(e,1)-s*(he/12)*(3*x1^2+2*x1*x2+x2^2);
    F(e+1,1) = F(e+1,1)-s*(he/12)*(3*x1^2+2*x1*x2+x2^2);
end

F(e+1)=F(e+1)+g*pi^2/L;

%% Solver for displacements
d(DOFr) = K(DOFr,DOFr)\(F(DOFr)-K(DOFr,DOFl)*d(DOFl));

end
```

A.4 Code for section 2

A.4.1 Main program

```
clc; % Clean the command window
clear all; % Clean the workspace
syms x u(x); % Define symbolic variables/functions

L = 1;                  %[m]
g = 0.01;               %[m]
rho = pi^2/L^2;         %[1/m^2]
s = g*rho^2;            %[1/m^3]
b = g*pi^2/L;           %[] (Point load at x=L)
fe = -s*x^2;            %[m]
q_E = rho*u+fe;         % Source function (Force over the bar)
n = [5 10 25 50 100]; % Number of elements


%% (PART 2)

% Exact solution
u_exact=@(x) 0.0987*x^2+0.01*cos(pi*x)+pi/100*sin(pi*x)-0.02;
du_exact=@(x) (987*x)/5000 - (pi*sin(pi*x))/100 + (pi^2*cos(pi*x))/100;
%% Application of the Finite Elements Method (FEM)

Error_U=zeros(length(n),1);
Error_Du=zeros(length(n),1);
size_elem=zeros(length(n),1);
for i=1:length(n)
    x0=0; %Position of the first node
    xF=L; %Position of the last node
    n_elements=n(i);
     % Coordinates and Connectivity matrix assembly
    COOR=linspace(x0,xF,n_elements+1)'; %Coordinates matrix
    CN=zeros(n_elements,2);              %Connectivity matrix
    for a=1:n(i)
           CN(a,1)=a;
           CN(a,2)=a+1;
    end

% Determination of u(x) applying FEM
[K,F,d]=AssemblyMatrixKFd(COOR,CN,L,g);

%% Error computation
error_u=0; error_du=0;
%% Weights and positions of Gauss Points

xi_g=[sqrt(3/5) -sqrt(3/5) 0];
w=[5/9 5/9 8/9];
N=@(xi) (1/2)*[1-xi 1+xi];

    for e=1:n_elements
    COORe=[COOR(CN(e,1)); COOR(CN(e,2))]; % Physical coordinates of u
    de=[d(CN(e,1));d(CN(e,2))]; % Nodal values of u
    he=COOR(CN(e,2))-COOR(CN(e,1)); % Element size
    B=1/he*[-1 1]; % Derivative of Shape function

        for m=1:3
        u_FEM=N(xi_g(m))*de;
        du_FEM=B*de;
        xF=N(xi_g(m))*COORe;
        u_Exact=u_exact(xF);
        du_Exact=du_exact(xF);

        error_u=error_u+he/2*(w(m)*(u_Exact-u_FEM)^2);
        error_du=error_du+he/2*(w(m)*(du_Exact-du_FEM)^2);
        end
    end
Error_U(i)=sqrt(error_u);
Error_Du(i)=sqrt(error_du);
size_elem(i)=he;

end


% Linear regression functions
log_size=log(size_elem);
log_eu=log(Error_U);
log_edu=log(Error_Du);
p1=polyfit(log_size,log_eu,1);
p2=polyfit(log_size,log_edu,1);
p_u=polyval(p1,log_size);
p_du=polyval(p2,log_size);
reg1=p_u(1)+p_u(2)*x;
sprintf('Linear regression of error in u(x): %s*x %d',p1(1),p1(2))
sprintf('Linear regression of error in du(x)/dx: %s*x%d',p2(1),p2(2))

%% Plotting the graphs

figure('Name','Log-Log graphic ef the approximation error in u(x) as a function of element ...
    length')
hold on;
plot(log_size,log_eu);
plot(log_size,p_u);
title('Log-Log graphic ef the approximation error in u(x) as a function of element ...
    length',Interpreter='latex');
xlabel('log(Length of an element)',Interpreter='latex');
ylabel('log(Error(u(x)',Interpreter='latex');
legend('Log-log function',sprintf('Linear regression function: %s*x %d',p1(1),p1(2)));
grid on;
hold off;

figure('Name',"Log-Log graphic ef the approximation error in u'(x) as a function of ...
    element length")
hold on;
plot(log_size,log_edu);
plot(log_size,p_du);
title('Log-Log graphic ef the approximation error in du(x)/dx as a function of element ...
     length',Interpreter='latex');
xlabel('log(Length of an element)',Interpreter='latex');
ylabel('log(Error(du(x)/dx',Interpreter='latex');
legend('Log-log function',sprintf('Linear regression function: %s*x %d',p2(1),p2(2)));
grid on;
hold off;
```

A.5 Code for Advanced Assignment

A.5.1 Main Program

```
    clc
clear all
format long g

% INPUTS

E0=10;         % Reference Young's Modulus [MPa]
sigma_0=1;   % Saturation stress [MPa]
L=1;           % Length of the beam [m]
A0=1e-2;      % Reference Area [m^2]
um=0.2*L;     % Maximum displacement [m]
u0=0;         % Initial displacement [m]
T=1;           % Final time [s]


NELEM = 50;  % Number of elements
nsteps = 51 ;  % Number of steps
stepsPOST = [1:5:nsteps]; % Steps to post-process

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)

% Internal variables

B=@(he) 1/he*[-1 1];

T_step=0:nsteps:T;

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(l,l)\Residual(l) ;
        d_k(l) = d_k(l) + 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



% Evolution of stresses along time
% --------------------------------------------
figure(1)
title('Evolution of stresses along time',Interpreter='latex');
hold on
xlabel('x',Interpreter='latex')
ylabel('Stress (MPa)',Interpreter='latex')
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))) ;
    LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i)))] ;
    %LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i))),'; NELEM =',num2str(NELEM),'; ...
        NSTEP = ',num2str(nsteps)] ;
end
grid on;
legend(hplot,LegendPlot)

legend('off')
legend

% Evolution of displacements along time
% --------------------------------------------
figure(2)
title('Evolution of displacements along time',Interpreter='latex');
hold on
xlabel('x')
ylabel('Disp. (m)',Interpreter='latex')
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))) ;
    LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i)))] ;
%     LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i))),'; NELEM ...
    =',num2str(NELEM),'; NSTEP = ',num2str(nsteps)] ;
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(3)
title('Location of maximum stress',Interpreter='latex');
hold on
xlabel('Time (s)',Interpreter='latex')
ylabel([['Maximum Stress (MPa), at x= ',num2str(COOR_max_stress)]],Interpreter="latex") ;
for i = 1:nsteps_plot
    h =   plot(TIMES,STRESS_GLO(indELEM,:)) ;
end
grid on;
```

A.5.2 Assembly K Function

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

nelem=size(CN,1);
nnode = size(COOR,1);
nnodeE = size(CN,2);
K=zeros(nnode,nnode);
for e=1:nelem
    NODOSe = CN(e,:);
    COOR_e = COOR(NODOSe);
    he = COOR_e(2)-COOR_e(1);
    Be = 1/he*[-1 +1];
    de = d_k(NODOSe);
    strain = Be*de;
    E = DerStressFUN(strain);
    x_average = (COOR_e(2)+COOR_e(1))/2;
    Area = AreaFUN(x_average);
    Ke = Area*E/he*[1 -1; -1 1];
    for a=1:nnodeE
        for b =1:nnodeE
            A = CN(e,a);
            B = CN(e,b);
            K(A,B) = K(A,B) + Ke(a,b);
        end
    end
end
```

A.5.3 Assembly F_{int} Function

```
 function [Residual,STRAIN,STRESS] = AssemblyFint(COOR,CN,d,DerstressFUN,AreaFUN) ;

nelem=size(CN,1);
nnode = size(COOR,1);

Residual=zeros(nnode,1);
STRAIN=zeros(nelem,1);
STRESS=zeros(nelem,1);

for e=1:nelem
    NODOSe = CN(e,:);
    COOR_e = COOR(NODOSe);
    he = COOR_e(2)-COOR_e(1);
    de = d(NODOSe);
    Be = 1/he*[-1 +1];
    Strain = Be*de;
    E = DerstressFUN(Strain);
    x_average = (COOR_e(2)+COOR_e(1))/2;
    Area = AreaFUN(x_average);

    Residual(CN(e,1))=Residual(CN(e,1))-E*Area;
    Residual(CN(e,2))=Residual(CN(e,2))+E*Area;
    STRAIN(CN(e,1))=Strain;
    STRESS(CN(e,1))=E;
end
end
```

A.5.4 Convergence assessment

```
clc
clear all
format long g

% INPUTS

E0=10;          % Reference Young's Modulus [MPa]
sigma_0=1;   % Saturation stress [MPa]
L=1;            % Length of the beam [m]
A0=1e-2;      % Reference Area [m^2]
um=0.2*L;     % Maximum displacement [m]
u0=0;          % Initial displacement [m]
T=1;            % Final time [s]

for n=[5 10 25 50 100 250]
```

```
NELEM = n;  % Number of elements
nsteps = n; ;  % Number of steps
stepsPOST = [n]; % Steps to post-process

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)

% Internal variables

B=@(he) 1/he*[-1 1];

T_step=0:nsteps:T;

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(l,l)\Residual(l) ;
        d_k(l) = d_k(l) + 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



% Evolution of stresses along time
% --------------------------------------------
figure(1)
title('Evolution of stresses along time',Interpreter='latex');
hold on
xlabel('x',Interpreter='latex')
ylabel('Stress (MPa)',Interpreter='latex')
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))) ;
    LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i)))] ;
    %LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i))),'; NELEM =',num2str(NELEM),'; ...
        NSTEP = ',num2str(nsteps)] ;
end
grid on;
legend(['NELEM = 5','; NSTEP = 5'],['NELEM = 10','; NSTEP = 10'], ['NELEM = 25','; NSTEP = ...
    25'], ['NELEM = 50','; NSTEP = 50'],['NELEM = 100','; NSTEP = 100'],['NELEM = 250','; ...
    NSTEP = 250']);

% Evolution of displacements along time
% --------------------------------------------
figure(2)
title('Evolution of displacements along time',Interpreter='latex');
hold on
xlabel('x')
ylabel('Disp. (m)',Interpreter='latex')
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))) ;
    LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i)))] ;
%     LegendPlot{i} = ['Time = ',num2str(TIMES(stepsPOST(i))),'; NELEM ...
    =',num2str(NELEM),'; NSTEP = ',num2str(nsteps)] ;
end
grid on;
legend(['NELEM = 5','; NSTEP = 5'],['NELEM = 10','; NSTEP = 10'], ['NELEM = 25','; NSTEP = ...
    25'], ['NELEM = 50','; NSTEP = 50'],['NELEM = 100','; NSTEP = 100'],['NELEM = 250','; ...
    NSTEP = 250']);




% Location of maximum stress

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

figure(3)
title('Location of maximum stress',Interpreter='latex');
hold on
xlabel('Time (s)',Interpreter='latex')
ylabel([['Maximum Stress (MPa), at x= ',num2str(COOR_max_stress)]],Interpreter="latex") ;
for i = 1:nsteps_plot
    h =   plot(TIMES,STRESS_GLO(indELEM,:)) ;
end
grid on;
end
legend(['NELEM = 5','; NSTEP = 5'],['NELEM = 10','; NSTEP = 10'], ['NELEM = 25','; NSTEP = ...
    25'], ['NELEM = 50','; NSTEP = 50'],['NELEM = 100','; NSTEP = 100'],['NELEM = 250','; ...
    NSTEP = 250']);
```