



## Solid mechanics and linear Finite Element Analysis

Foundations of Materials Simulation

**Author:** Atharva Sinnarkar

Matriculation Number : 23353068

MSc. Computational Engineering

December 18, 2024

# Contents

<b>1 Task 1.1</b>	<b>5</b>
<b>2 Task 2.1</b>	<b>6</b>
<b>3 Task 2.2</b>	<b>6</b>
<b>4 Task 2.3</b>	<b>7</b>
<b>5 Task 3.1</b>	<b>8</b>
<b>6 Task 4.1</b>	<b>9</b>
6.1 Introduction . . . . .	9
6.2 Sketch of 6 Elements with Shape Functions . . . . .	9
6.3 Shape Functions for One Element . . . . .	9
6.4 Numerical Approximation with Different Functions . . . . .	10
6.5 Exact Representation of Functions . . . . .	11
6.6 Effect on Displacement and Stress Fields . . . . .	11
6.6.1 Displacement Fields . . . . .	11
6.6.2 Stress Fields . . . . .	11
<b>7 Task 4.2</b>	<b>11</b>
<b>8 Task 5.1</b>	<b>13</b>
8.1 Implementation Details . . . . .	13
8.2 Results . . . . .	13
<b>9 Task 5.2</b>	<b>14</b>
9.1 Output Verification . . . . .	14
9.2 Results Table . . . . .	15
9.3 Summary of Exactness . . . . .	15
9.4 Integration Points Needed . . . . .	15
<b>10 Task 5.3</b>	<b>16</b>
10.1 System of Linear Equations . . . . .	16
10.2 Comparison with MATLAB Implementation . . . . .	16
10.3 Handwritten Solution . . . . .	17
10.4 Choice of Solver . . . . .	18
10.5 Solution Comparison . . . . .	18
<b>11 Task 6.1</b>	<b>18</b>
<b>12 Task 6.2</b>	<b>19</b>
<b>13 Task 6.3</b>	<b>19</b>
13.1 Global Matrix Output . . . . .	19
13.2 Properties of the Global Stiffness Matrix . . . . .	19
13.3 Why the Global System Cannot Be Solved Yet . . . . .	19
13.4 Physical Explanation/Interpretation . . . . .	20
13.5 MATLAB Output . . . . .	20

<b>14 Task 7.1</b>	<b>20</b>
<b>15 Task 7.2</b>	<b>21</b>
<b>16 Task 8.1</b>	<b>22</b>

## List of Figures

1	Sketch of 6 elements and the corresponding shape functions $N_1$ and $N_2$ . . .	9
2	Shape functions $N_1$ and $N_2$ for a single element. . . . .	10
3	Approximation of linear, quadratic, and cubic functions using linear shape functions. . . . .	10
4	Comparison of Exact Derivatives and Approximate Derivatives for Linear, Quadratic, and Cubic Functions. . . . .	12
5	Derivatives of Shape Functions $N_1(x)$ and $N_2(x)$ for a single element. . . .	13
6	MATLAB Command Window Output . . . . .	17
7	Handwritten Solution of the System of Linear Equations . . . . .	17
8	Handwritten Calculations for Task 6.1 . . . . .	18
9	Global Stiffness Matrix Output for Task 6.3 . . . . .	20
10	Permutation matrix for a 4×4 system with Dirichlet boundary condition at the first node. . . . .	21
11	MATLAB output . . . . .	21
12	Displacement and stress distribution along the bar. . . . .	22

List of Tables

1    Integration Results for Different Polynomials . . . . . 15

# 1 Task 1.1

*Simplify the local form equation (14) in the case of statics (no acceleration) and give an example of body forces other than the gravitational force. Describe a case where body forces can be neglected.*

Given Equation (14): The local form of the governing differential equation of the deformation  $\mathbf{u}$  of a solid is given by:

$$\rho \ddot{\mathbf{u}} = \text{div}(\boldsymbol{\sigma}) + \rho \mathbf{b}$$

where:

- $\rho$  is the mass density,
- $\ddot{\mathbf{u}}$  is the acceleration,
- $\boldsymbol{\sigma}$  is the Cauchy stress tensor,
- $\mathbf{b}$  is the body force per unit mass.

In the static case, there is no motion or change in velocity, so the acceleration term  $\ddot{\mathbf{u}}$  becomes zero. This simplifies the governing equation as follows:

$$0 = \text{div}(\boldsymbol{\sigma}) + \rho \mathbf{b}$$

This is the simplified form of the governing equation in the static case.

Body forces are forces that act throughout the volume of a body. Apart from gravitational force, other examples of body forces include:

- **Electromagnetic Forces:** These occur in materials that are subject to electric or magnetic fields, such as in the case of charged particles in a magnetic field.
- **Centrifugal Forces:** These arise in rotating bodies due to the inertia of the mass elements that tend to move outward from the center of rotation.

Body forces can be neglected in scenarios where they are small compared to surface forces (traction forces) or when they do not significantly affect the overall stress distribution within the material. For instance:

- **Thin Structures:** In thin structures like beams or shells, the body forces may be negligible compared to the surface traction forces, especially when the thickness is much smaller than the other dimensions.
- **Microscale Systems:** In microelectromechanical systems (MEMS), the gravitational forces are often negligible compared to surface forces due to the small size and mass of the components.

In the static case, the local form simplifies to:

$$\text{div}(\boldsymbol{\sigma}) + \rho \mathbf{b} = 0$$

Body forces other than gravity include electromagnetic and centrifugal forces. Body forces can often be neglected in thin structures or microscale systems where surface forces dominate.

## 2 Task 2.1

Write eq. (19) in index notation and denote for which indices the Einstein summation convention applies. How many components does  $C$  have in general in 3 dimensions? Show that if one uses that the stress and strain tensors are symmetric ( $\sigma = \sigma^T$ ;  $\epsilon = \epsilon^T$ ), the number of independent components reduces to 36.

The general form of Hooke's law in tensor notation is given by eq. (19) as follows:

$$\sigma = C : \epsilon$$

In index notation, this becomes:

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl}$$

where:

- $\sigma_{ij}$  is the stress tensor,
- $C_{ijkl}$  is the stiffness tensor (elasticity tensor),
- $\epsilon_{kl}$  is the strain tensor.

Using the Einstein summation convention, the repeated indices  $k$  and  $l$  imply summation over those indices. Therefore, the equation becomes:

$$\sigma_{ij} = C_{ijkl}\epsilon_{kl}$$

In 3D, each index  $i, j, k$ , and  $l$  can take the values 1, 2, or 3. Thus, the stiffness tensor  $C_{ijkl}$  has  $3^4 = 81$  components in total.

Considering the symmetries of the stress and strain tensors:

$$\sigma_{ij} = \sigma_{ji}, \quad \epsilon_{kl} = \epsilon_{lk}, \quad C_{ijkl} = C_{jikl} = C_{ijlk} = C_{klij}$$

These symmetries reduce the number of independent components in  $C_{ijkl}$  from 81 to 36.

The final result is that the stiffness tensor  $C_{ijkl}$  has 36 independent components due to the inherent symmetries of the stress, strain, and stiffness tensors.

## 3 Task 2.2

Show that the stiffness tensor has 21 independent components, if we take the symmetries from Task 2.1 and eq. (21) into account. (Hint: use Schwarz's theorem)

In this task, we demonstrate that the stiffness tensor  $C_{ijkl}$  has 21 independent components by considering the symmetries identified in Task 2.1 and utilizing equation (21).

Initially, the stiffness tensor  $C_{ijkl}$  in three dimensions is a fourth-rank tensor where each index  $i, j, k, l$  can independently take values from 1 to 3. Therefore, the total number of components is:

$$3 \times 3 \times 3 \times 3 = 81$$

From Task 2.1, we know that both the stress tensor  $\sigma_{ij}$  and the strain tensor  $\epsilon_{kl}$  are symmetric:

$$\sigma_{ij} = \sigma_{ji}, \quad \epsilon_{kl} = \epsilon_{lk}$$

These symmetries impose corresponding symmetries on the stiffness tensor:

$$C_{ijkl} = C_{jikl} \quad (\text{Symmetry in the first two indices})$$

$$C_{ijkl} = C_{ijlk} \quad (\text{Symmetry in the last two indices})$$

Additionally, equation (21) defines the stiffness tensor as the second derivative of the strain energy density function  $W$  with respect to the strain tensor  $\epsilon$ :

$$C = \frac{\partial^2 W}{\partial \epsilon \partial \epsilon}$$

This relation implies that the stiffness tensor  $C_{ijkl}$  is also symmetric with respect to the interchange of the first and second pair of indices:

$$C_{ijkl} = C_{klij} \quad (\text{Major symmetry})$$

Taking all these symmetries into account, the stiffness tensor satisfies:

$$C_{ijkl} = C_{jikl} = C_{ijlk} = C_{jilk} = C_{klij} = C_{lki j} = C_{klji} = \dots$$

These symmetry properties significantly reduce the number of independent components of  $C_{ijkl}$ .

Applying the minor symmetries  $C_{ijkl} = C_{jikl}$  and  $C_{ijkl} = C_{ijlk}$ , the number of unique components reduces from 81 to 36. This reduction accounts for the symmetry within the first two indices and within the last two indices.

Further applying the major symmetry  $C_{ijkl} = C_{klij}$ , which accounts for the symmetry between the first and second pairs of indices, the number of independent components is further reduced from 36 to 21.

This reduction is justified by Schwarz's theorem, which states that the order of differentiation does not matter for continuously differentiable functions. In this context, it ensures that:

$$\frac{\partial^2 W}{\partial \epsilon_{ij} \partial \epsilon_{kl}} = \frac{\partial^2 W}{\partial \epsilon_{kl} \partial \epsilon_{ij}}$$

Thus, the stiffness tensor  $C_{ijkl}$  inherits these symmetry properties, leading to a total of 21 independent components.

In conclusion, by systematically applying the symmetries from Task 2.1 and utilizing equation (21), we establish that the stiffness tensor  $C_{ijkl}$  has 21 independent components.

## 4 Task 2.3

Consider a one-dimensional bar of length  $l$ , clamped at  $x = 0$  and loaded with a prescribed force  $F$  at  $x = l$ . The bar is made of an isotropic material with constant mass density  $\rho$  and cross-sectional area  $A$ . Furthermore, the cross-sectional area is assumed to be constant and does not change under load, implying  $\nabla \cdot u = 0$ .

The local balance equation of linear momentum in one dimension is given by:

$$\rho \frac{\partial^2 u(x, t)}{\partial t^2} = \frac{\partial \sigma(x, t)}{\partial x} + \rho b(x)$$



where  $u(x, t)$  is the displacement,  $\sigma(x, t)$  is the stress, and  $b(x)$  is the body force per unit mass.

In the static case (no time dependence), this simplifies to:

$$0 = \frac{d\sigma(x)}{dx} + \rho b(x)$$

For an isotropic material, the stress-strain relationship is given by Hooke's law:

$$\sigma(x) = E\epsilon(x) = E \frac{du(x)}{dx}$$

where  $E$  is the Young's modulus.

Substituting this into the balance equation gives:

$$0 = \frac{d}{dx} \left( E \frac{du(x)}{dx} \right) + \rho b(x)$$

Considering the constant cross-sectional area  $A$ , we multiply by  $A$  to obtain the final form:

$$\frac{d}{dx} \left( EA \frac{du(x)}{dx} \right) + A\rho b(x) = 0$$

which is the required equation (25) in the problem statement.

## 5 Task 3.1

Show that the weak form of the example problem (25) from Task 2.3 is given by

$$\frac{d}{dx} \left( EA \frac{du(x)}{dx} \right) + A\rho b(x) = 0$$

To derive the weak form, we follow these steps:

We begin by multiplying the differential equation by a test function  $\delta u(x)$  and integrating over the domain  $[0, l]$ :

$$\int_0^l \delta u(x) \left[ \frac{d}{dx} \left( EA \frac{du(x)}{dx} \right) + A\rho b(x) \right] dx = 0$$

We perform integration by parts on the first term to reduce the order of the derivative on  $u(x)$ :

$$\int_0^l \delta u(x) \frac{d}{dx} \left( EA \frac{du(x)}{dx} \right) dx = \left[ \delta u(x) \left( EA \frac{du(x)}{dx} \right) \right]_0^l - \int_0^l \frac{d\delta u(x)}{dx} \left( EA \frac{du(x)}{dx} \right) dx$$

Next, we apply the boundary conditions:

- At  $x = 0$ ,  $u(0) = 0$ , so  $\delta u(0) \left( EA \frac{du(x)}{dx} \right)_{x=0} = 0$ .
- At  $x = l$ , the term  $\delta u(l) \left( EA \frac{du(x)}{dx} \right)_{x=l} = \delta u(l) F$ .

Thus, the weak form is:

$$\delta u(l)F - \int_0^l \frac{d\delta u(x)}{dx} \left( EA \frac{du(x)}{dx} \right) dx + \int_0^l \delta u(x) A \rho b(x) dx = 0$$

Rearranging the terms, the weak form of the problem becomes:

$$\int_0^l \frac{du(x)}{dx} EA \frac{d\delta u(x)}{dx} dx = \int_0^l A \rho b(x) \delta u(x) dx + F \delta u(l)$$

## 6 Task 4.1

### 6.1 Introduction

In this task, we are required to sketch 6 elements and the linear shape functions, implement shape functions using Eq. (38), and test the numerical approximation with three different functions. This report contains the sketches, implementation, and detailed answers to the questions asked.

### 6.2 Sketch of 6 Elements with Shape Functions

The figure below illustrates the 6 elements used, with linear shape functions  $N_1$  and  $N_2$  for each element drawn on top of it.

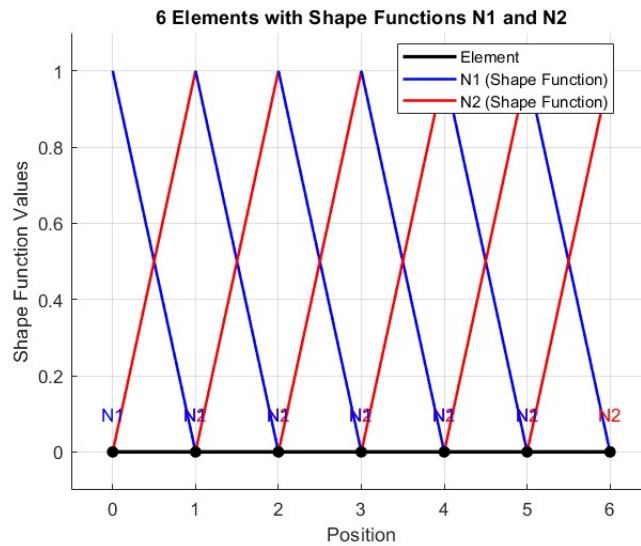


Figure 1: Sketch of 6 elements and the corresponding shape functions  $N_1$  and  $N_2$ .

The shape functions  $N_1$  and  $N_2$  for each element are linear and vary between the element nodes as per the finite element method formulation.

### 6.3 Shape Functions for One Element

Below is the sketch of the shape functions  $N_1$  and  $N_2$  for a single element. This is based on the two nodes of an element.

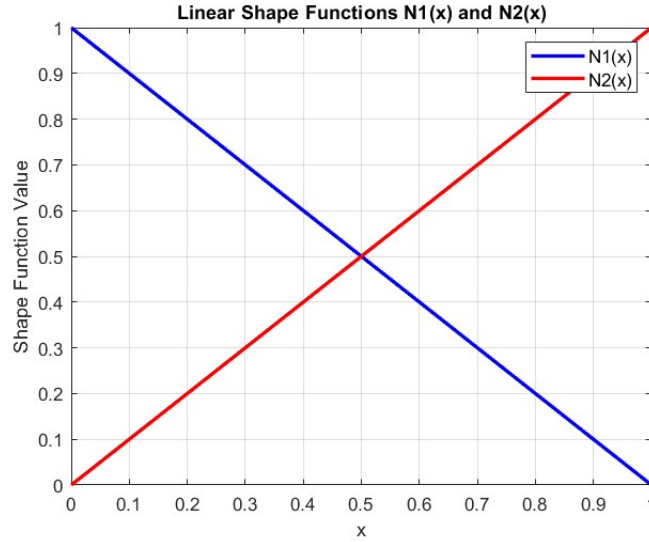


Figure 2: Shape functions  $N_1$  and  $N_2$  for a single element.

The shape functions  $N_1(x)$  and  $N_2(x)$  can be computed using Eq. (38):

$$N_1(x) = \frac{x_2 - x}{x_2 - x_1}, \quad N_2(x) = \frac{x - x_1}{x_2 - x_1}$$

Where  $x_1$  and  $x_2$  are the coordinates of the two element nodes.

## 6.4 Numerical Approximation with Different Functions

We approximated three different functions using the shape functions implemented for the finite element analysis. The following plot shows the approximation results for a linear, quadratic, and cubic function.

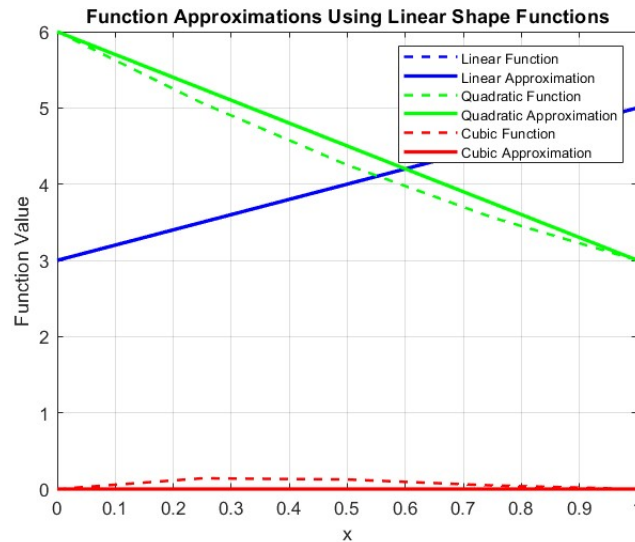


Figure 3: Approximation of linear, quadratic, and cubic functions using linear shape functions.

The functions used for the approximation were: -  $f_{\text{linear}}(x) = 2x + 3$  -  $f_{\text{quadratic}}(x) = x^2 - 4x + 6$  -  $f_{\text{cubic}}(x) = x^3 - 2x^2 + x$

## 6.5 Exact Representation of Functions

Linear shape functions can exactly represent the following types of functions:

- Linear functions: Since the shape functions are linear, any linear function (such as  $f_{\text{linear}}(x) = 2x + 3$ ) will be exactly represented.
- Higher-order functions (quadratic, cubic) are only approximated and not exactly represented, as the linear shape functions cannot capture their non-linear variations.

## 6.6 Effect on Displacement and Stress Fields

### 6.6.1 Displacement Fields

Since the displacement field in finite element analysis is approximated using the shape functions, a linear shape function leads to a linear approximation of the displacement field across each element. This means that only linearly varying displacements are exactly captured. Non-linear displacement fields will be approximated, with accuracy depending on the mesh density.

### 6.6.2 Stress Fields

The stress field is derived from the displacement field and is typically proportional to the derivative of the displacement. With linear shape functions, the displacement is linear, so the derivative (and therefore the stress field) will be constant across each element. This implies that the stress field will be approximated as constant within each element when using linear shape functions, which might not represent varying stress fields accurately unless a finer mesh is used.

## 7 Task 4.2

In this task, we implemented the derivatives of the linear shape functions  $N_1(x)$  and  $N_2(x)$ . The shape functions are linear, so their derivatives are constant within the element. The goal was to compute the derivatives of these shape functions and apply them to approximate the derivatives of test functions such as linear, quadratic, and cubic functions. We also visualized the results using plots of the approximated and exact derivatives.

### Derivatives of Shape Functions

The derivatives of the linear shape functions  $N_1(x)$  and  $N_2(x)$  are given by:

$$\frac{dN_1(x)}{dx} = \frac{-1}{x_2 - x_1}, \quad \frac{dN_2(x)}{dx} = \frac{1}{x_2 - x_1}$$

These derivatives are constant for each element because the shape functions are linear.

## Testing with Functions

We tested the derivative approximation using the following functions:

- Linear:  $f(x) = 2x + 3$
- Quadratic:  $f(x) = x^2 - 4x + 6$
- Cubic:  $f(x) = x^3 - 2x^2 + x$

The exact derivatives of these functions were compared with the approximated derivatives using the shape function derivatives. The linear function derivative was exactly represented by the approximation, while the quadratic and cubic functions showed some deviation.

The linear shape functions provide exact approximation for the derivatives of linear functions, while the quadratic and cubic functions have an inherent error due to the nature of the linear shape functions. This task demonstrates the limitations of linear shape functions for higher-order function approximation.

## Results

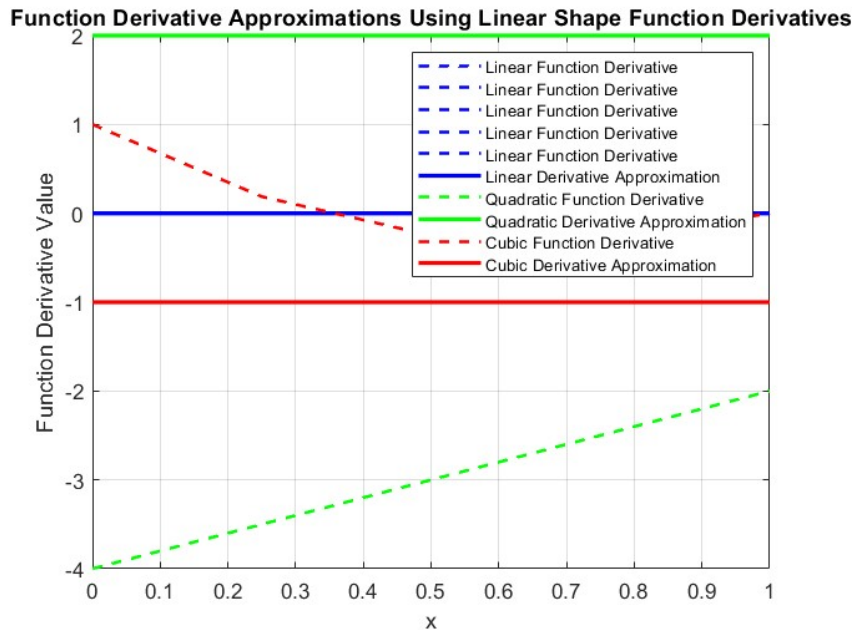


Figure 4: Comparison of Exact Derivatives and Approximate Derivatives for Linear, Quadratic, and Cubic Functions.

Figure 4 illustrates the comparison between the exact derivatives of the test functions and the approximate derivatives using shape function gradients.

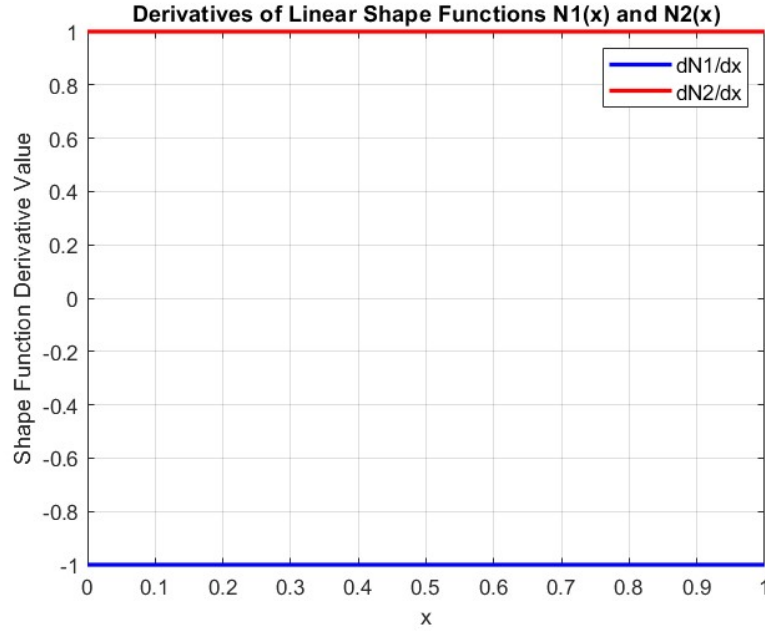


Figure 5: Derivatives of Shape Functions  $N_1(x)$  and  $N_2(x)$  for a single element.

Figure 5 shows the constant derivatives of the linear shape functions  $N_1(x)$  and  $N_2(x)$ .

## 8 Task 5.1

The objective of this task was to implement the trapezoidal rule for numerical integration of the function  $f(x) = x^2$  over the interval  $[0, 1]$ . The implementation involved creating a function to perform the integration and a separate script to test its correctness.

### 8.1 Implementation Details

The trapezoidal rule was implemented in the function `quadrature.m`, which computes the integral using the following parameters:

- **Integration Limits:**  $x_1 = 0, x_2 = 1$
- **Number of Points:**  $n = 2$
- **Integration Points:**  $\zeta = [x_1, x_2]$
- **Weights:**  $\lambda = [0.5, 0.5]$

### 8.2 Results

The output of the integration was as follows:

Result of integration: 0.5

This result matches the expected value, confirming that the trapezoidal rule correctly approximates the integral of  $f(x) = x^2$  over the specified interval.

## 9 Task 5.2

The objective of this task is to evaluate the performance of various Newton-Cotes quadrature methods in integrating polynomial functions. Specifically, we investigate the following schemes:

- **Trapezoidal Rule (n=2)**
- **Simpson's Rule (n=3)**
- **3/8 Rule (n=4)**

We assess each method's exactness for polynomials of different degrees (linear, quadratic, cubic, and quartic) by computing the integral of each polynomial over the interval  $[0, 1]$ . This helps us determine the highest polynomial degree for which each scheme provides an exact result, as well as the number of integration points required for exact integration of linear and quadratic shape functions.

### 9.1 Output Verification

To verify the output of the numerical integration, we compare the results obtained from the Newton-Cotes quadrature methods with the actual integral values of the polynomials over the interval  $[0, 1]$ :

- **Linear Function**  $f(x) = x$ :

$$\int_0^1 x \, dx = 0.5$$

- **Quadratic Function**  $f(x) = x^2$ :

$$\int_0^1 x^2 \, dx = \frac{1}{3} \approx 0.3333$$

- **Cubic Function**  $f(x) = x^3$ :

$$\int_0^1 x^3 \, dx = \frac{1}{4} = 0.25$$

- **Quartic Function**  $f(x) = x^4$ :

$$\int_0^1 x^4 \, dx = \frac{1}{5} = 0.2$$

Based on the numerical results, we summarize the exactness of the integration schemes in the following table.

## 9.2 Results Table

Integration Method	Polynomial Degree	Result
Trapezoidal Rule (n=2)	Linear (1)	0.5000
Trapezoidal Rule (n=2)	Quadratic (2)	0.5000
Trapezoidal Rule (n=2)	Cubic (3)	0.5000
Trapezoidal Rule (n=2)	Quartic (4)	0.5000
Simpson's Rule (n=3)	Linear (1)	0.5000
Simpson's Rule (n=3)	Quadratic (2)	0.3333
Simpson's Rule (n=3)	Cubic (3)	0.2500
Simpson's Rule (n=3)	Quartic (4)	0.2083
3/8 Rule (n=4)	Linear (1)	0.5000
3/8 Rule (n=4)	Quadratic (2)	0.3333
3/8 Rule (n=4)	Cubic (3)	0.2500
3/8 Rule (n=4)	Quartic (4)	0.2037

Table 1: Integration Results for Different Polynomials

## 9.3 Summary of Exactness

- **Trapezoidal Rule (n=2):**
  - Exact for linear polynomials (degree 1).
  - Not exact for quadratic (degree 2) and higher.
- **Simpson's Rule (n=3):**
  - Exact for linear (degree 1) and quadratic polynomials (degree 2).
  - Not exact for cubic (degree 3) and higher.
- **3/8 Rule (n=4):**
  - Exact for linear (degree 1), quadratic (degree 2), and cubic polynomials (degree 3).
  - Not exact for quartic (degree 4).

## 9.4 Integration Points Needed

- To exactly integrate **linear** shape functions: 2 points (Trapezoidal Rule).
- To exactly integrate **quadratic** shape functions: 3 points (Simpson's Rule).
- To exactly integrate **cubic** shape functions: 4 points (3/8 Rule).



## 10 Task 5.3

In this task, we aim to solve the weak form for a clamped 1D bar using the Finite Element Method (FEM). The bar is modeled with two nodes, and the stiffness matrix is derived using linear shape functions. The displacement field is approximated, and we solve the resulting system of linear equations. Finally, we compare the analytical solution to the MATLAB implementation and provide a handwritten derivation.

### 10.1 System of Linear Equations

For a 1D bar with two nodes, the stiffness matrix  $K$  is given by:

$$K = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

where:

- $E$  is Young's modulus,
- $A$  is the cross-sectional area,
- $L$  is the length of the bar.

The force vector is:

$$f = \begin{bmatrix} 0 \\ F \end{bmatrix}$$

where  $F$  is the applied force at node 2.

The system of equations becomes:

$$\frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ F \end{bmatrix}$$

Applying the boundary condition  $\hat{u}_1 = 0$  (since the bar is clamped at  $x = 0$ ), the system reduces to:

$$\frac{EA}{L} \hat{u}_2 = F$$

Thus, the displacement at node 2 is:

$$\hat{u}_2 = \frac{FL}{EA}$$

### 10.2 Comparison with MATLAB Implementation

The MATLAB implementation follows the same steps as the handwritten derivation. The stiffness matrix is constructed, the force vector is defined, and the system is solved with the boundary condition  $\hat{u}_1 = 0$ . The solution is consistent with the analytical result. Below is the MATLAB output for the displacement at each node and the stiffness matrix:

```

Command Window

Displacement at each node:
    1.0e-04 *

         0
    0.4762

Stiffness matrix:
    105000    -105000
   -105000     105000

fx >>

```

Figure 6: MATLAB Command Window Output

### 10.3 Handwritten Solution

The handwritten derivation of the system of linear equations, including the matrix form, is shown in the following image:

$$\begin{aligned}
 K &= \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad f = \begin{bmatrix} 0 \\ F \end{bmatrix} \\
 Ku &= f \\
 \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \hat{u}_1 \\ \hat{u}_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ F \end{bmatrix} \\
 \frac{EA}{L} (\hat{u}_1 - \hat{u}_2) &= 0 \quad \text{--- (1)} \\
 \frac{EA}{L} (-\hat{u}_1 + \hat{u}_2) &= F \quad \text{--- (2)} \\
 \text{From dirichlet boundary condition, } \hat{u}_1 &= 0 \\
 \therefore \hat{u}_2 &= \frac{FL}{EA} = \frac{5 \times 50}{210000 \times 25} = 4.76 \times 10^{-5} \text{ mm}
 \end{aligned}$$

Figure 7: Handwritten Solution of the System of Linear Equations

## 10.4 Choice of Solver

For this system, the best choice of a solver is a **direct solver** due to the small size and simplicity of the matrix (a 2x2 system). In MATLAB, we use the backslash operator '`\`' which automatically applies an efficient direct solver such as **LU decomposition** or **Cholesky decomposition** based on the properties of the matrix.

The stiffness matrix is symmetric and positive definite, making **Cholesky decomposition** an ideal choice. MATLAB's built-in solver recognizes this and solves the system efficiently. For larger systems or sparse matrices, **iterative solvers** such as **Conjugate Gradient (CG)** could be more appropriate, but they are not necessary here.

## 10.5 Solution Comparison

The MATLAB result for the displacement at node 2 is:

$$\hat{u}_2 = 4.762 \times 10^{-5} \text{ mm}$$

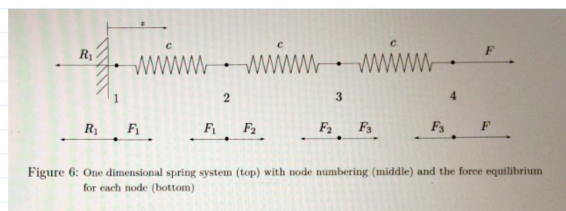
which matches the analytical solution:

$$\hat{u}_2 = \frac{5 \times 50}{210000 \times 25} = 4.762 \times 10^{-5} \text{ mm}$$

Thus, the analytical and numerical solutions are identical, confirming the correctness of the MATLAB implementation.

## 11 Task 6.1

Below are the handwritten calculations for task 6.1.



$$\begin{aligned} \text{Node 1: } & -R_1 + c(u_2 - u_1) = 0 \Rightarrow c(u_1 - u_2) = -R_1 \\ \text{Node 2: } & c(u_2 - u_1) + c(u_3 - u_2) = 0 \Rightarrow c(-u_1 + 2u_2 - u_3) = 0 \\ \text{Node 3: } & c(u_3 - u_2) + c(u_4 - u_3) = 0 \Rightarrow c(-u_2 + 2u_3 - u_4) = 0 \\ \text{Node 4: } & c(u_4 - u_3) = F \Rightarrow c(u_3 - u_4) = -F \end{aligned}$$

$$\therefore c \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -F \end{bmatrix} - \begin{bmatrix} R_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figure 8: Handwritten Calculations for Task 6.1

## 12 Task 6.2

In this task, the assembly routine for linear elements was implemented. The goal was to construct the global stiffness matrix by placing local element stiffness matrices into the global matrix according to the node connectivity. This process ensures that the contribution of each element is properly accounted for in the overall system behavior.

The assembly was performed by:

- Initializing the global stiffness matrix as a zero matrix.
- Iterating through the elements and inserting each element's local stiffness matrix at the appropriate positions corresponding to its connected nodes.
- Ensuring continuity between elements, preparing the system for applying boundary conditions.

## 13 Task 6.3

### 13.1 Global Matrix Output

The global stiffness matrix was constructed for three elements and four nodes. The expected matrix, assuming a stiffness constant  $c = 1$ , is:

$$K = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

This matrix is symmetric and positive semi-definite, reflecting the physical properties of the system.

### 13.2 Properties of the Global Stiffness Matrix

- **Symmetry:** The global stiffness matrix is symmetric, indicating that the force between any two nodes is reciprocal.
- **Singularity:** The matrix is singular because no boundary conditions (Dirichlet conditions) have been applied. As a result, the system allows for rigid body motions (translations/rotations), leading to infinite solutions.
- **Positive Semi-Definite:** The matrix has non-negative eigenvalues, corresponding to the physical reality that the system cannot have negative stiffness, implying stability.

### 13.3 Why the Global System Cannot Be Solved Yet

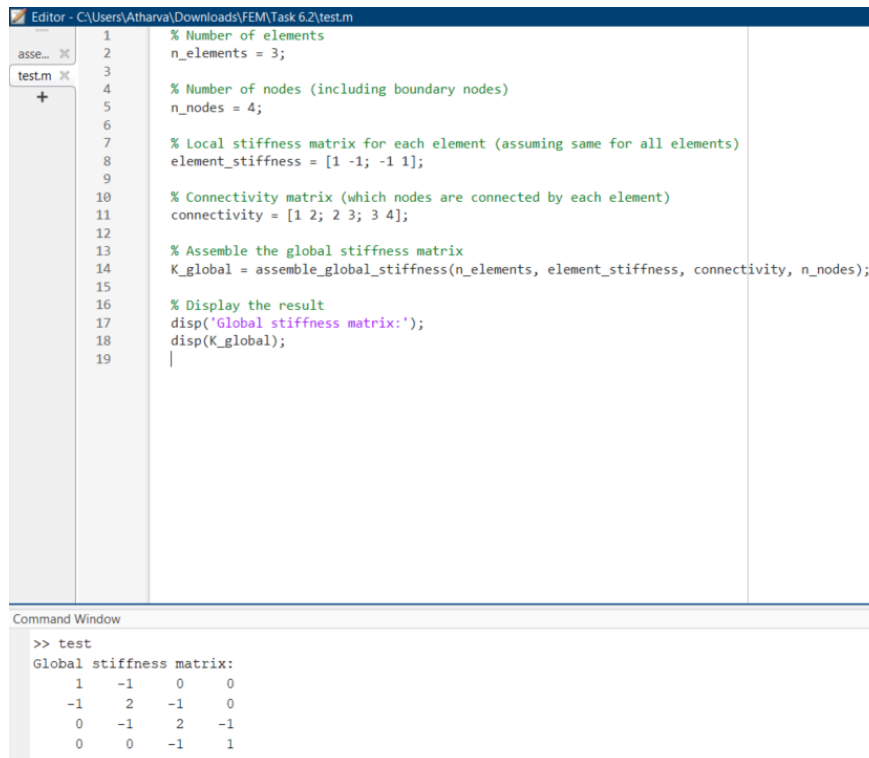
The system cannot be solved as it is because it is singular. Physically, this means the system can move freely in space without any resistance, as no nodes are constrained. To solve the system, boundary conditions must be applied (e.g., fixing a displacement at one or more nodes), which will make the matrix non-singular.

## 13.4 Physical Explanation/Interpretation

The singularity arises because, without constraints, the structure can undergo rigid body motion. This is equivalent to the system having an infinite number of solutions, where any displacement of the entire structure does not result in internal forces. Fixing nodes removes these degrees of freedom and allows the system to have a unique solution.

## 13.5 MATLAB Output

The following figure shows the global stiffness matrix obtained from MATLAB for this system.



The figure displays a MATLAB script in the Editor window and its execution output in the Command Window. The script defines the number of elements (3) and nodes (4), sets a local stiffness matrix for each element, defines a connectivity matrix, and then assembles the global stiffness matrix. The output shows the resulting 4x4 global stiffness matrix.

```
1 % Number of elements
2 n_elements = 3;
3
4 % Number of nodes (including boundary nodes)
5 n_nodes = 4;
6
7 % Local stiffness matrix for each element (assuming same for all elements)
8 element_stiffness = [1 -1; -1 1];
9
10 % Connectivity matrix (which nodes are connected by each element)
11 connectivity = [1 2; 2 3; 3 4];
12
13 % Assemble the global stiffness matrix
14 K_global = assemble_global_stiffness(n_elements, element_stiffness, connectivity, n_nodes);
15
16 % Display the result
17 disp('Global stiffness matrix:');
18 disp(K_global);
19 |
```

```
>> test
Global stiffness matrix:
     1     -1     0     0
    -1     2    -1     0
     0     -1     2    -1
     0     0    -1     1
```

Figure 9: Global Stiffness Matrix Output for Task 6.3

## 14 Task 7.1

In this task, we are required to write down the permutation matrix for a 4×4 system with Dirichlet boundary conditions applied to the first node. The permutation matrix rearranges the system such that the boundary condition at the first node is handled appropriately. The handwritten solution is shown below:

$$u_1 = 0$$

$$K = \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{14} \\ K_{21} & K_{22} & K_{23} & K_{24} \\ K_{31} & K_{32} & K_{33} & K_{34} \\ K_{41} & K_{42} & K_{43} & K_{44} \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 10: Permutation matrix for a 4x4 system with Dirichlet boundary condition at the first node.

## 15 Task 7.2

In this task, we programmatically apply Dirichlet boundary conditions using MATLAB. The code modifies the global stiffness matrix and force vector to impose the boundary condition at the specified node. The MATLAB output for this task, after imposing the Dirichlet boundary condition, is shown below:

```
Modified stiffness matrix:
    1     0     0     0
    0     2    -1     0
    0    -1     2    -1
    0     0    -1     1

Modified force vector:
    0
    0
    0
   -5

Solution vector:
    0
  -5.0000
 -10.0000
 -15.0000

Modified stiffness matrix for multiple Dirichlet conditions:
    1     0     0     0
    0     2    -1     0
    0    -1     2     0
    0     0     0     1

Modified force vector for multiple Dirichlet conditions:
    0
    0
    2
    2

Solution vector:
    0
  0.6667
  1.3333
  2.0000
```

Figure 11: MATLAB output

The modified stiffness matrix and force vector, after applying the Dirichlet boundary condition, were obtained and verified with the MATLAB output shown above.

## 16 Task 8.1

In this task, we perform a linear, one-dimensional finite element analysis (FEA) of a bar subjected to axial loading. The bar has a length of  $L = 50$  mm, a cross-sectional area of  $A = 25$  mm<sup>2</sup>, and a Young's modulus of  $E = 210000$  N/mm<sup>2</sup>. The right end of the bar is subjected to a force  $F = 5$  N, while the left end is fixed. The problem is discretized into  $n = 6$  elements.

The stiffness matrix for each element is derived from the relation:

$$k_e = \frac{EA}{L_e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

where  $L_e$  is the length of each element. The global stiffness matrix is assembled by summing the contributions of individual elements, and boundary conditions are applied by fixing the displacement at the left end.

The approximate displacement field is computed by solving the system of equations  $Ku = f$ , and the stress in each element is calculated using:

$$\sigma_i = \frac{E(u_{i+1} - u_i)}{L_e}$$

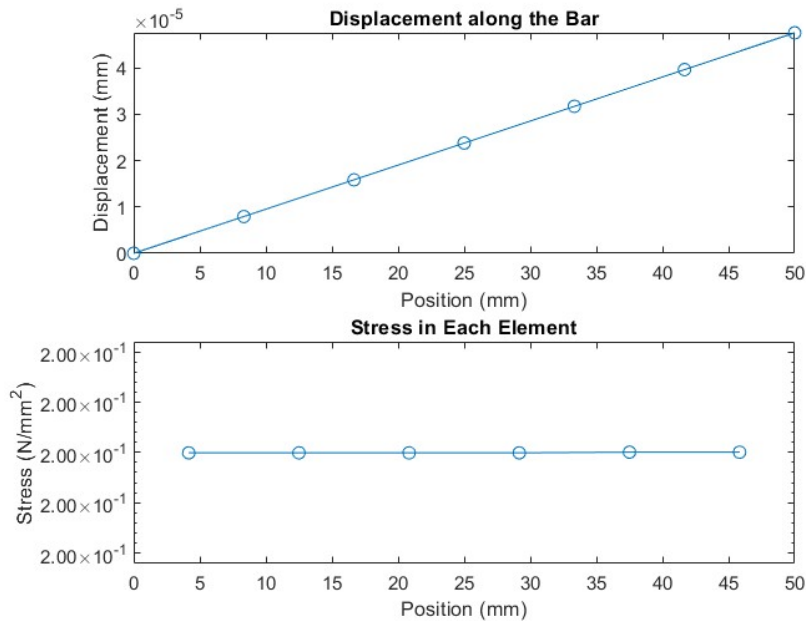


Figure 12: Displacement and stress distribution along the bar.

Figure 12 shows the computed displacement field along the length of the bar and the corresponding stress distribution in each element. The displacement increases linearly, as expected in a uniform bar under axial loading. The stress remains constant across the elements, reflecting the uniformity of the applied load and material properties.