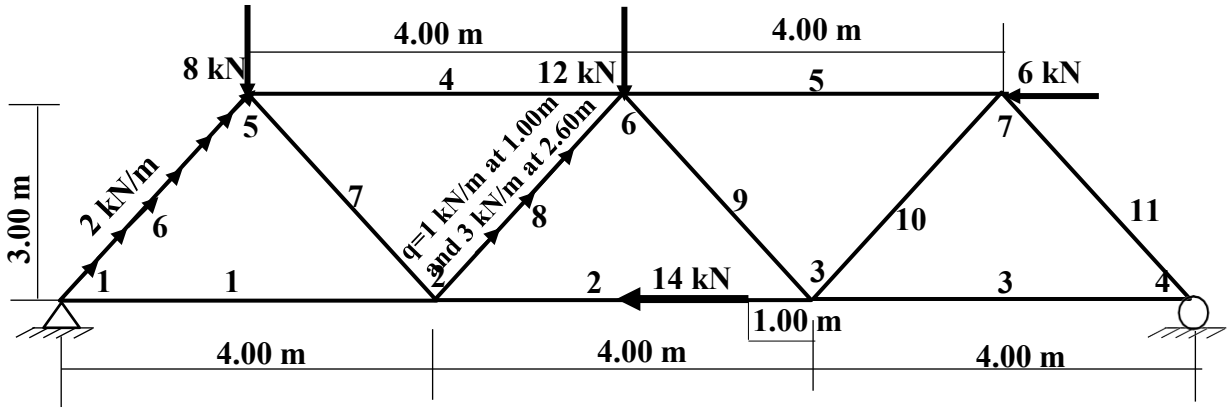
FEM Practical works

Part 1: Implementation of the finite element method

For a truss structure (see example below), loaded by nodal loads and/or concentrated or distributed element loads:



The following tables are defined:

- *Connectivity matrix defining the elements of the truss (For each element or bar we give node 1, node 2, and the property number):* ***connec***
- *Matrix of global coordinates (x, y) of the truss nodes:* ***gcor***
- *Matrix of properties (section A and Young's modulus E) of the bars:* ***prop***
- *Matrix of nodal forces of the structure (Number of the loaded node, the components of the force applied to the node for all loaded nodes):* ***nf***
- *Matrix of concentrated forces between nodes of the elements (Number of the loaded element, the abscissa - in local coordinates - of the point of application of the applied axial load and its algebraic value, for all the loaded elements):* ***clbn***
- *Matrix of loads distributed between nodes of the elements (Number of the loaded element, the abscissas delimiting the loaded length in local coordinates and the algebraic value of the load, for all the loaded elements):* ***dlbn***
- *Matrices of support conditions (number of the supported node, nature of the degrees of freedom in each direction x, y: 0 for free degree, 1 for blocked degree:* ***sup***

1. Data preparation :

- *Write the matrices **connec**, **gcor**, **prop**, **nf**, **clbn**, **dlbn** and **sup** of the truss above, the Young's modulus is 2×10^8 kN/m^2, the section is 0.0016 m^2 for the horizontal bars and 0.0036 m^2 for the oblique bars*
- *Run the schema function below to visualize the structure:*

```
import numpy as np; import matplotlib.pyplot as plt; import ex1
gcor, connec, prop, nf, clbn, dlbn, sup = ex1.don ()
conec = connec[:, 0:2]
def schema(gcor, conec):
    # Create figure
    plt.figure(figsize= (8, 6))
    # Draw lines
    for i, (N1, N2) in enumerate(conec):
        N1, N2 = int(N1), int(N2)
        x_coords = [gcor[N1][0], gcor[N2][0]]
        y_coords = [gcor[N1][1], gcor[N2][1]]
        plt.plot(x_coords, y_coords, 'b-', linewidth=2)
        mid_x = (gcor[N1][0] + gcor[N2][0]) / 2
        mid_y = (gcor[N1][1] + gcor[N2][1]) / 2
        plt.text(mid_x, mid_y + 0.15, f"{i}", color='b', fontsize=10,
                 bbox=dict(boxstyle='round, pad=0.3', facecolor='yellow', alpha=0.7))
    # Draw points
    for i, (x, y) in enumerate(gcor):
        plt.text(x + 0.1, y + 0.1, f'{i}', color='red', fontsize=10, \
                 fontweight='bold', bbox=dict(boxstyle='circle, pad=0.3',\
                                              facecolor='white', alpha=0.7))
    # Create legend
    plt.plot(gcor[conec[0, 0],0], gcor[conec[0, 1],1], color='blue', linewidth=2 , label='bars')
    plt.scatter(gcor[:, 0], gcor[:, 1], color='red', s=100, zorder=5, label='Nodes')
    # Graph settings
    plt.xlabel("X (m)", fontsize=12)
    plt.ylabel("Y (m)", fontsize=12)
    plt.title("Truss Structure", fontsize=14, fontweight='bold')
    plt.axhline(y=0, color='gray', linewidth=2, linestyle='-', alpha=0.5)
    plt.axvline(x=0, color='gray', linewidth=2, linestyle='-', alpha=0.5)
    plt.grid(True, linestyle='--', alpha=0.3)
    plt.legend(fontsize=10)
    plt.axis("equal")
    # Display graph
    plt.tight_layout()
    plt.show()

schema(gcor, conec)
```

From the data: connec, gcor, prop, nf, clbn, dlbn and sup, for each bar we can extract:

- *The coordinates of the nodes that we place in the **xy** matrix, which allows us to determine the length **L** of the bar, the cosines **(l, m)** that the local axis which is the axis of the bar makes with the global axes, and consequently the transformation matrix:*

$$[\boldsymbol{T}] = \begin{bmatrix} \boldsymbol{l} & \boldsymbol{m} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{l} & \boldsymbol{m} \end{bmatrix}$$

*The section **A** and the Young's modulus **E** of the bar, which allows to calculate the stiffness matrix of the bar in local coordinates $[\boldsymbol{ke}]$ and in global coordinates $[\boldsymbol{Ke}]$*

$$[ke] = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \qquad [Ke] = [T]^T[ke][T]$$

2. Element-level programming :

1. *Write a function called **egch** allowing the determination of a bar's length **L** and its transformation matrix **T***
2. *Write a function called **estm** using the **egch** function and allowing to determine the matrices* $[\boldsymbol{ke}]$ *and **[Ke]** in the local and global coordinates.*
3. *Write a function called **nelcl** using the **egch** function and allowing the calculation of equivalent loads of the concentrated loads applied between nodes of the element.*
4. *Write a function called **neldl** using the **egch** function and allowing the calculation of equivalent loads of the distributed loads applied between nodes of the element*

3. Programming at the global level:

1. *Write the function called **loc** which allows to replace the indices of the terms of the stiffness matrix or the force vector of the element with those concerning the entire structure, and then perform the assembly to form the global stiffness matrix and the global force vector.*
2. *Write the function called **supcon** which allows the introduction of the support conditions and subsequently forms the reduced global system*
3. *Write the main function named **Truss** using the different previous functions and allowing to calculate the displacements of the nodes and the forces in the bars*
4. *Complete the project with functions for presenting the program, printing the stiffness matrices of the elements, printing the global system of equations, and printing the calculation results: displacement and forces.*