Software Development for Scientific Computing
Programming Assignment 2

Agrim Jain - 210010003
Srihari K G - 210030035
Vivek Pillai - 210010058

November 2023

1 Method: Mathematical Formulation/Calculation

The global stiffness matrix of a rod can be obtained by assembling the element stiffness matrices of the subdomains. For a rod broken into 3 elements, we use 1 point gauss quadrature, the element stiffness matrices are given by:

$$K_1 = \frac{EA}{L_1}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$
$$K_2 = \frac{EA}{L_2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$
$$K_3 = \frac{EA}{L_3}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

where E is the Young's modulus, A is the cross-sectional area, and L_i is the length of the i-th element.

The global stiffness matrix can be obtained by adding the contributions of each element at the corresponding degrees of freedom. For example, the first element contributes to the first and second rows and columns of the global matrix, the second element contributes to the second and third rows and columns, and the third element contributes to the third and fourth rows and columns. The global stiffness matrix is then given by:

$$K = \begin{bmatrix} K_1(1,1) & K_1(1,2) & 0 & 0 \\ K_1(2,1) & K_1(2,2)+K_2(1,1) & K_2(1,2) & 0 \\ 0 & K_2(2,1) & K_2(2,2)+K_3(1,1) & K_3(1,2) \\ 0 & 0 & K_3(2,1) & K_3(2,2) \end{bmatrix}$$
$$= \frac{EA}{L_1}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \frac{EA}{L_2}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \frac{EA}{L_3}\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

Now, the displacement vector is given by,

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

where u_1, u_2, u_3 and u_4 are displacements at the 4 nodes.
The force vector is given as,

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}$$

The equation that we are expected to solve is the following,

$$\begin{bmatrix} K_1(1,1) & K_1(1,2) & 0 & 0 \\ K_1(2,1) & K_1(2,2)+K_2(1,1) & K_2(1,2) & 0 \\ 0 & K_2(2,1) & K_2(2,2)+K_3(1,1) & K_3(1,2) \\ 0 & 0 & K_3(2,1) & K_3(2,2) \end{bmatrix} * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}$$

All intermediate forces i.e. F_2 and F_3 evaluate to 0 as all internal forces cancel out resulting in a net 0 force. $F_1 =$ f and $F_4 =$ -f where f is the applied force at the free end of the rod. Putting F_2 and F_3 to be 0, F_1 to be -f and F_4 to f(given force), the equation reduces to,

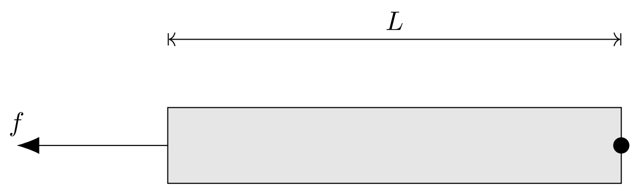
$$\begin{bmatrix} K_1(1,1) & K_1(1,2) & 0 & 0 \\ K_1(2,1) & K_1(2,2)+K_2(1,1) & K_2(1,2) & 0 \\ 0 & K_2(2,1) & K_2(2,2)+K_3(1,1) & K_3(1,2) \\ 0 & 0 & K_3(2,1) & K_3(2,2) \end{bmatrix} * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} -f \\ 0 \\ 0 \\ f \end{bmatrix}$$

Now, we know that $u_4 = 0$ as it is a stationary point fixed at an end. So, we need not calculate that and hence we can ignore the first row and first column of the global stiffness matrix and also ignore the first row of the force vector. Hence the resultant equation that we are expected to solve boils down to the following,

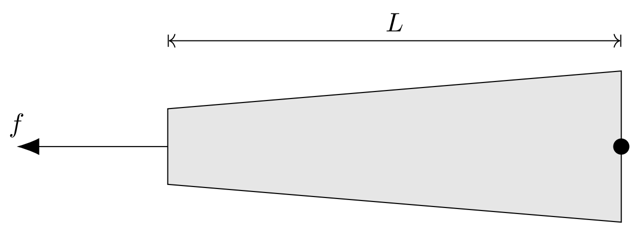
$$\begin{bmatrix} K_1(1,1) & K_1(1,2) & 0 \\ K_1(2,1) & K_1(2,2)+K_2(1,1) & K_2(1,2) \\ 0 & K_2(2,1) & K_2(2,2)+K_3(1,1) \end{bmatrix} * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix}$$

We have a rod with cross sectional area A(x) and length L. The rod is subjected to a constant force P = 5000 N at x = 0. At x = L the rod is fixed. The length of the rod is 0.5 m and the Young's modulus of the material of the rod is 70 GPa. For the given subproblems:

1. $A(x) = A_0 = 12.5 * 10^4 m^2$



2. $A(x) = A_0(1 + x/L)$
   Here the cross section is not uniform, it increases linearly with x. 'L' here is the total length of the rod. For any element number 'i', we have taken the area of cross section(A) to be the area at the left end of the element. Let 'n' be the number of elements, then length per element is L/n. i * (L/n) is the value of 'x' for any element.



2 Experimental Results

The following is a plot corresponding to constant area and 2 elements:

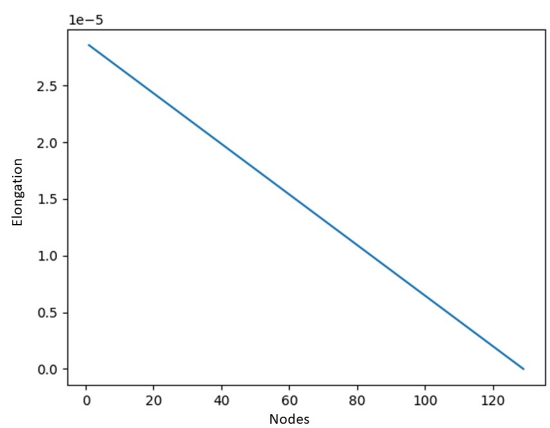


Figure 1: PROB=1, N=2

The following is a plot corresponding to constant area and 8 elements:

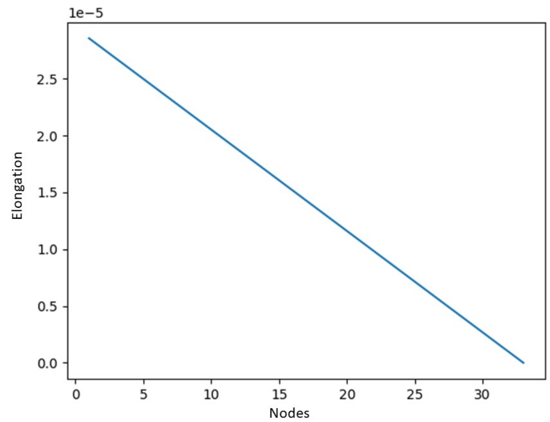


Figure 2: PROB=1, N=8

The following is a plot corresponding to constant area and 32 elements:

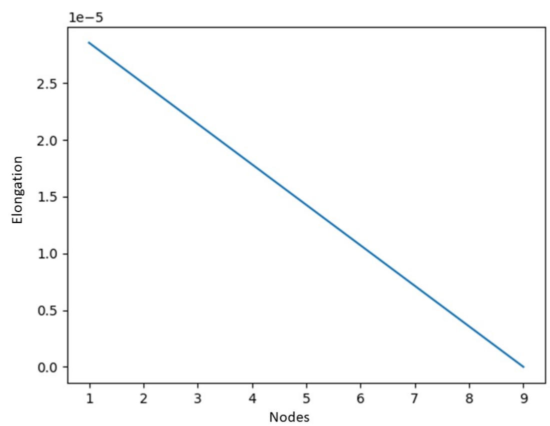


Figure 3: PROB=1, N=32

The following is a plot corresponding to constant area and 128 elements:

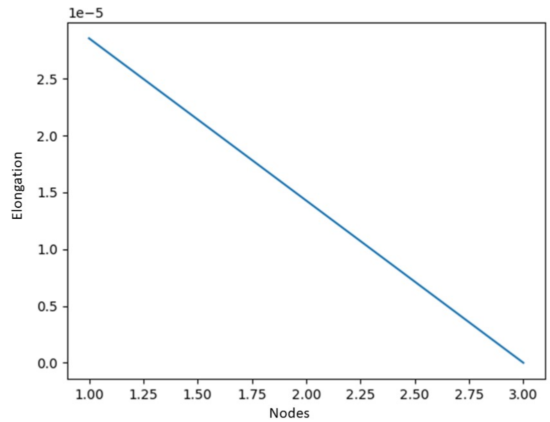


Figure 4: PROB=1, N=128

We can observe that the displacements vary linearly in case of constant area and the plot is approximately a line. The curve remains the same on increasing the number of elements because of an increase in accuracy of the predicted displacements.

The following is a plot corresponding to variable area and 2 elements:

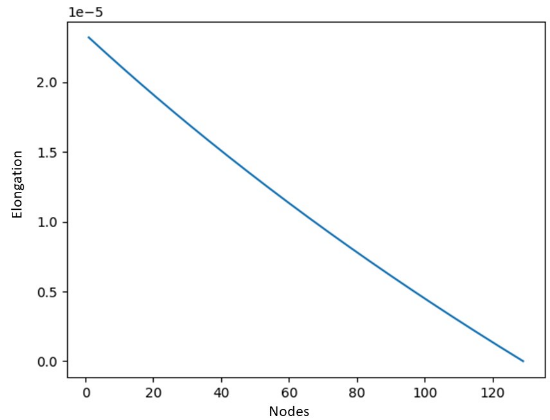


Figure 5: PROB=2, N=2

The following is a plot corresponding to variable area and 8 elements:

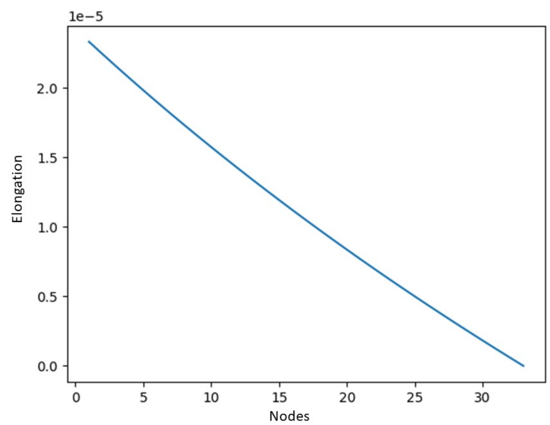


Figure 6: PROB=2, N=8

The following is a plot corresponding to variable area and 32 elements:

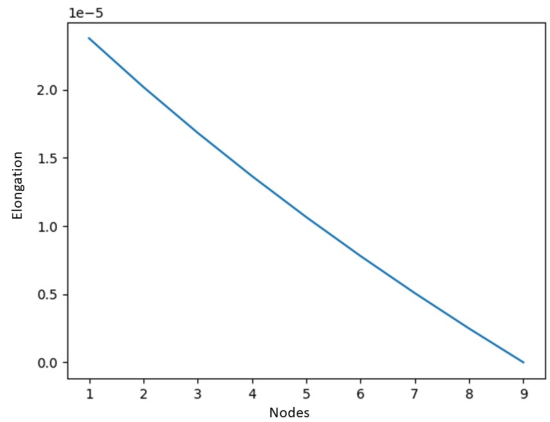


Figure 7: PROB=2, N=32

The following is a plot corresponding to variable area and 128 elements:

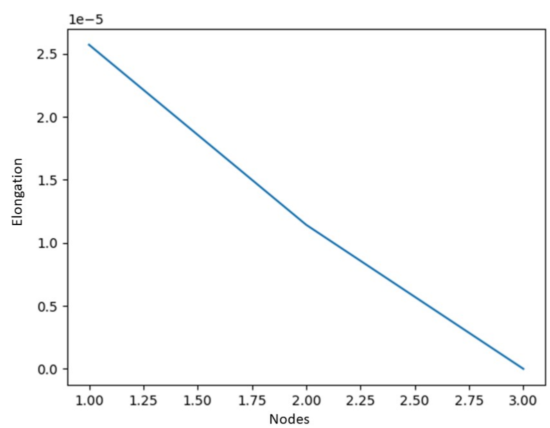


Figure 8: PROB=2, N=128

We can observe that the displacements do not vary linearly in case of variable area and the plot is a curve now. Further, the curve smoothens on increasing the number of elements because of an increase in accuracy of the predicted displacements.

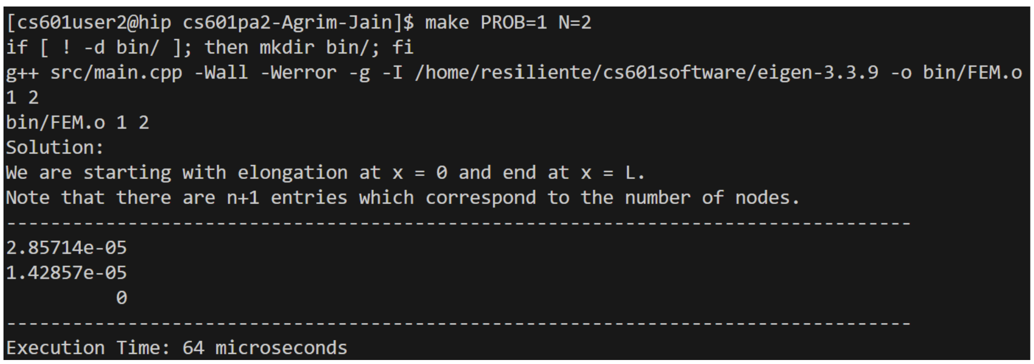
The observations of execution time for constant area and variable area with different number of elements are as follows:

Number of Elements	Time for Constant Area(in μ sec)	Time for Variable Area(in μ sec)
2	64	79
8	191	195
32	2157	2165
128	62842	63584

3 Appendix

3.1 Execution Snippets

PROB=1, N=2



```
[cs601user2@hip cs601pa2-Agrim-Jain]$ make PROB=1 N=2
if [ ! -d bin/ ]; then mkdir bin/; fi
g++ src/main.cpp -Wall -Werror -g -I /home/resiliente/cs601software/eigen-3.3.9 -o bin/FEM.o
1 2
bin/FEM.o 1 2
Solution:
We are starting with elongation at x = 0 and end at x = L.
Note that there are n+1 entries which correspond to the number of nodes.
--------------------------------------------------------------------------------
2.85714e-05
1.42857e-05
          0
--------------------------------------------------------------------------------
Execution Time: 64 microseconds
```

Figure 9: PROB=1, N=2

PROB=1, N=8

```
[cs601user2@hip cs601pa2-Agrim-Jain]$ make PROB=1 N=8
if [ ! -d bin/ ]; then mkdir bin/; fi
1 8
bin/FEM.o 1 8
Solution:
We are starting with elongation at x = 0 and end at x = L.
Note that there are n+1 entries which correspond to the number of nodes.
---------------------------------------------------------------------------------
2.85714e-05
    2.5e-05
2.14286e-05
1.78571e-05
1.42857e-05
1.07143e-05
7.14286e-06
3.57143e-06
          0
---------------------------------------------------------------------------------
Execution Time: 191 microseconds
```

Figure 10: PROB=1, N=8

PROB=1, N=32

```
   1.25e-05
1.16071e-05
1.07143e-05
9.82143e-06
8.92857e-06
8.03571e-06
7.14286e-06
   6.25e-06
5.35714e-06
4.46429e-06
3.57143e-06
2.67857e-06
1.78571e-06
8.92857e-07
          0
---------------------------------------------------------------------------------
Execution Time: 2157 microseconds
[cs601user2@hip cs601pa2-Agrim-Jain]$
```

Figure 11: PROB=1, N=32

PROB=1, N=128

```
  3.125e-06
2.90179e-06
2.67857e-06
2.45536e-06
2.23214e-06
2.00893e-06
1.78571e-06
 1.5625e-06
1.33929e-06
1.11607e-06
8.92857e-07
6.69643e-07
4.46429e-07
2.23214e-07
          0
--------------------------------------------------------------------------------
Execution Time: 62842 microseconds
[cs601user2@hip cs601pa2-Agrim-Jain]$
```

Figure 12: PROB=1, N=128

PROB=2, N=2

```
[cs601user2@hip cs601pa2-Agrim-Jain]$ make PROB=2 N=2
if [ ! -d bin/ ]; then mkdir bin/; fi
2 2
bin/FEM.o 2 2
Solution:
We are starting with elongation at x = 0 and end at x = L.
Note that there are n+1 entries which correspond to the number of nodes.
--------------------------------------------------------------------------------
2.57143e-05
1.14286e-05
          0
--------------------------------------------------------------------------------
Execution Time: 79 microseconds
```

Figure 13: PROB=2, N=2

PROB=2, N=8

```
[cs601user2@hip cs601pa2-Agrim-Jain]$ make PROB=2 N=8
if [ ! -d bin/ ]; then mkdir bin/; fi
2 8
bin/FEM.o 2 8
Solution:
We are starting with elongation at x = 0 and end at x = L.
Note that there are n+1 entries which correspond to the number of nodes.
-----------------------------------------------------------------------------------
 2.3775e-05
2.02036e-05
1.68422e-05
1.36676e-05
1.06601e-05
7.80296e-06
5.08187e-06
2.48447e-06
          0
-----------------------------------------------------------------------------------
Execution Time: 195 microseconds
```

Figure 14: PROB=2, N=8

PROB=2, N=32

```
9.76384e-06
9.05837e-06
8.36151e-06
7.67304e-06
6.99277e-06
 6.3205e-06
5.65605e-06
4.99923e-06
4.34988e-06
3.70783e-06
3.07291e-06
2.44496e-06
1.82385e-06
1.20941e-06
6.01504e-07
          0
-----------------------------------------------------------------------------------
Execution Time: 2165 microseconds
[cs601user2@hip cs601pa2-Agrim-Jain]$
```

Figure 15: PROB=2, N=32

PROB=2, N=128

```
2.27994e-06
2.12508e-06
1.97064e-06
1.81661e-06
  1.663e-06
1.50981e-06
1.35702e-06
1.20464e-06
1.05266e-06
9.01088e-07
7.49917e-07
5.99144e-07
4.48768e-07
2.98787e-07
1.49198e-07
          0
----------------------------------------------------------------------------------
Execution Time: 63584 microseconds
[cs601user2@hip cs601pa2-Agrim-Jain]$
```

Figure 16: PROB=2, N=128

3.2 Code Snippets

File Structure

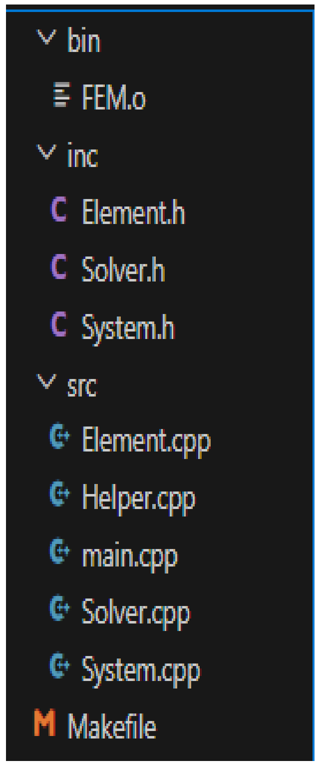


Figure 17: File Structure

Makefile

```
# Makefile for Finite Element Method (FEM) Simulation

# Compiler and Compilation Flags
CC = g++
CFLAGS = -Wall -Werror -g -I /home/resiliente/cs601software/eigen-3.3.9

# Source and Binary Directories
SRC = src/
BIN = bin/
CLEANOBJ = $(BIN)

# Default Problem and Number of Elements
PROB ?= 0
N ?= 0

# Target Rules
all: isBin bin/FEM.o run

compile: isBin bin/FEM.o

$(BIN)FEM.o: $(SRC)main.cpp
	$(CC) $^ $(CFLAGS) -o $(BIN)FEM.o

run:
	@echo $(PROB) $(N)
	$(BIN)FEM.o $(PROB) $(N)

isBin:
	if [ ! -d $(BIN) ]; then mkdir $(BIN); fi

# Clean Generated Files
clean:
	if [ -d $(CLEANOBJ) ]; then rm -r $(CLEANOBJ); fi

# Display Team Information
team:
	@echo 210030035 - Srihari K G
	@echo 210010058 - Vivek Pillai
	@echo 210010003 - Agrim Jain
```

Figure 18: Makefile

Element.h

inc > C Element.h

```cpp
#ifndef ELEMENT_H
#define ELEMENT_H

// Element class represents the properties of a finite element in a rod structure.
class Element {
private:
    double area;            // Cross-sectional area of the element
    double Youngs;          // Young's modulus of the material
    double element_length;  // Length of each element

public:
    // Constructor for the Element class, initializes the properties of the element.
    Element(double area, double Youngs, double element_length);

    // Computes and returns the coefficient at each index for a specific question number (qno).
    double getCoeff(int qno, int index);
};

#endif // ELEMENT_H
```

Figure 19: Element.h

Solver.h

inc > C Solver.h

```cpp
#ifndef SOLVER_H
#define SOLVER_H

#include "Eigen/Dense"
#include "System.h"

// Solver class handles the solution of a linear system represented by a System object.

class Solver {
private:
    Eigen::MatrixXd global_stiffness_matrix;  // Stores the global stiffness matrix
    Eigen::MatrixXd force_vector;             // Stores the force vector

public:
    // Constructor for the Solver class, taking a System object as a parameter.
    Solver(System system);

    // Method to solve the linear system and return the solution matrix.
    Eigen::MatrixXd solve();
};

#endif // SOLVER_H

```

Figure 20: Solver.h

System.h

inc > C System.h

```
#ifndef SYSTEM_H
#define SYSTEM_H

#include "Eigen/Dense"
#include "Element.h"

// The System class represents a physical system with finite elements.
class System {
private:
    int num_element;                    // Number of elements in the system
    int qno;                            // Question number
    double force;                       // Applied force
    double Length;                      // Length of the rod
    Element element;                    // Element object representing properties of the rod
    Eigen::MatrixXd global_stiffness_matrix;   // Global Stiffness Matrix
    Eigen::MatrixXd force_vector;               // Force Vector

public:
    // Creates and returns the Global Stiffness Matrix
    Eigen::MatrixXd createGlobalStiffness(int qno, int num_element);

    // Creates and returns the Force Vector
    Eigen::MatrixXd createForceVector(int num_element, double force);

    // Constructor for the System class
    System(int qno, int num_element, double force, double Length, double area, double Youngs);

    // Returns the Global Stiffness Matrix
    Eigen::MatrixXd getGlobalStiffnessMatrix();

    // Returns the Force Vector
    Eigen::MatrixXd getForceVector();
};

#endif // SYSTEM_H

```

Figure 21: System.h

Element.cpp

src > Element.cpp

```cpp
#include "../inc/Element.h"
#include<iostream>
using namespace std;

// Constructor for the Element class, initializes the properties of the element
Element::Element(double area, double Youngs, double element_length)
    : area(area), Youngs(Youngs), element_length(element_length) {}

double Element::getCoeff(int qno, int index) {

    /*
    Method to calculate the coefficient based on specified parameters.

    Parameters:
        - qno (int): Question number
        - index (int): Index of the node element

    Returns:
        The coefficient value based on the question number and index.
    */

    if (qno == 2) {
        // Computes coefficient for the variable cross-section problem
        double area_left = area * (1 + (element_length * index));
        return ((Youngs * area_left) / element_length);
    }

    // Computes coefficient for the uniform cross-section problem
    return ((Youngs * area) / element_length);
}
```

Figure 22: Element.cpp

Helper.cpp

src > Helper.cpp

```cpp
#include <iostream>
#include <stdexcept>
#include <string>
#include "Eigen/Dense"

using namespace std;

// Print the correct usage of the program
void printUsage(const char* programName) {
    std::cerr << "Usage: " << programName << " PROB N" << std::endl;
}

// Parse and validate command line arguments
bool parseArguments(int argc, char* argv[], int& prob, int& n) {
    if (argc < 3) {
        printUsage(argv[0]);
        return false;
    }

    std::string strProb = argv[1];
    std::string strN = argv[2];

    try {
        prob = std::stoi(strProb);
        n = std::stoi(strN);

        return true;
    } catch (const std::invalid_argument& e) {
        std::cerr << "Invalid argument: " << e.what() << std::endl;
        return false;
    } catch (const std::out_of_range& e) {
        std::cerr << "Out of range: " << e.what() << std::endl;
        return false;
    }
}

// Validate the value of PROB
bool validateProb(int prob) {
    if (prob != 1 && prob != 2) {
        std::cerr << "Invalid value for PROB. Must be 1 or 2." << std::endl;
        return false;
    }
    return true;
}
```

Figure 23: Helper.cpp

Helper.cpp

```cpp
// Print the solution matrix to the terminal
void printSolnTerminal(Eigen::MatrixXd solution) {
    /*
        Print the solution matrix to the terminal.

        Parameters:
            - solution (Eigen::MatrixXd): Matrix containing the solution values.

        Note:
            - The matrix represents elongation at each node.
            - The solution starts at x = 0 and ends at x = L.
            - There are n+1 entries corresponding to the number of nodes.
    */
    cout << "Solution:" << endl;
    cout << "We are starting with elongation at x = 0 and end at x = L." << endl
         << "Note that there are n+1 entries which correspond to the number of nodes." << endl;
    cout << "----------------------------------------------------------------------------------" << endl;
    cout << solution << endl;
    cout << "----------------------------------------------------------------------------------" << endl;
}

// Create and write the solution matrix to an output text file
void createOutputtxt(Eigen::MatrixXd solution) {
    /*
        Create and write the solution matrix to an output text file.

        Parameters:
            - solution (Eigen::MatrixXd): Matrix containing the solution values.

        Note:
            - The matrix represents elongation at each node.
            - The solution starts at x = 0 and ends at x = L.
            - There are n+1 entries corresponding to the number of nodes.
    */
    std::ofstream outputFile("output.txt");

    // Write to the file
    outputFile << "We are starting with elongation at x = 0 and end at x = L." << endl
```

Figure 24: Helper.cpp

Helper.cpp

src > Helper.cpp

```cpp
    // Write to the file
    outputFile << "We are starting with elongation at x = 0 and end at x = L." << endl
                << "Note that there are n+1 entries which correspond to the number of nodes." << endl;
    outputFile << "-----------------------------------------------------------------------------------------" << endl;
    outputFile << solution << std::endl;
    outputFile << "-----------------------------------------------------------------------------------------" << endl;

    // Close the file
    outputFile.close();
}

```

Figure 25: Helper.cpp

main.cpp

src > main.cpp

```cpp
#include <iostream>
#include <fstream>

#include "Eigen/Dense"
#include "Element.cpp"
#include "Solver.cpp"
#include "System.cpp"
#include "Helper.cpp"
#include <chrono>

using namespace std;

int main(int argc, char* argv[]) {
    // Initialize variables for the problem and number of elements
    int prob, n;

    // Check for the validity of inputs
    if (!parseArguments(argc, argv, prob, n) || !validateProb(prob)) {
        return 1;
    }

    // Domain parameters
    double area = 0.00125;
    double Youngs = 70000000000;
    double Length = 0.5;
    int num_element = n;
    double force = 5000;
    int qno = prob;

```

Figure 26: main.cpp

main.cpp

```cpp
    double force = 5000;
    int qno = prob;

    // Start the solution process
    auto start = std::chrono::high_resolution_clock::now();
    System system(qno, num_element, force, Length, area, Youngs);
    Solver solver(system);
    Eigen::MatrixXd solution = solver.solve();
    // The solution is a vector containing the elongation at each node.
    auto stop = std::chrono::high_resolution_clock::now();

    // Print the solution to the terminal
    printSolnTerminal(solution);
    // Output the solution into a text file
    createOutputtxt(solution);

    // Calculate and print the execution time
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(stop - start);
    std::cout << "Execution Time: " << duration.count() << " microseconds" << std::endl;

    return 0;
}
```

Figure 27: main.cpp

Solver.cpp

src > Solver.cpp

```cpp
#include "../inc/Solver.h"
#include <iostream>

// Constructor for Solver class, takes a System object as a parameter
Solver::Solver(System system) {
    // Initialize private member global_stiffness_matrix with the Eigen matrix from the System object
    global_stiffness_matrix = system.getGlobalStiffnessMatrix();
    // Initialize private member force_matrix with the Eigen force matrix from the System object
    force_vector = system.getForceVector();
}

// Solve method for Solver class, returns the solution matrix
Eigen::MatrixXd Solver::solve() {

    /*
    Function to compute the vector containing elongations.

    Returns:
        - Eigen::MatrixXd: Vector containing elongations of each node.

    Procedure:
        1. Calculate the pseudo-inverse of the global_stiffness_matrix using complete orthogonal decomposition.
        2. Calculate the solution by multiplying the pseudo-inverse with the force_vector.
        3. Resize the solution matrix to add a row and set the values in the added row to zero (elongation at the point where it is hinged).
        4. Return the final solution matrix.
    */

    // Calculate the pseudo-inverse of the global_stiffness_matrix using complete orthogonal decomposition
    Eigen::MatrixXd pseudoinverse = this->global_stiffness_matrix.completeOrthogonalDecomposition().pseudoInverse();

```

Figure 28: Solver.cpp

Solver.cpp

```cpp
    // Calculate the pseudo-inverse of the global_stiffness_matrix using complete orthogonal decomposition
    Eigen::MatrixXd pseudoinverse = this->global_stiffness_matrix.completeOrthogonalDecomposition().pseudoInverse();

    // Calculate the solution by multiplying the pseudo-inverse with the force_vector
    Eigen::MatrixXd solution = pseudoinverse * this->force_vector;

    // Resize the solution matrix to add a row and set the values in the added row to zero (elongation at the point where it is hinged)
    solution.conservativeResize(solution.rows() + 1, Eigen::NoChange);
    solution.row(solution.rows() - 1).setZero();

    // Return the final solution matrix
    return solution;
}

```

Figure 29: Solver.cpp

System.cpp

src > System.cpp

```cpp
#include "../inc/System.h"

Eigen::MatrixXd System::createGlobalStiffness(int qno, int num_element) {
    /*
    Creates the Global Stiffness Matrix for the system.

    Parameters:
        - qno (int): Question number
        - num_element (int): Number of elements in the system

    Returns:
        Eigen::MatrixXd: Global Stiffness Matrix
    */

    // Initialize the global stiffness matrix with dimensions (num_element+1, num_element+1).
    Eigen::MatrixXd global_stiffness_matrix = Eigen::MatrixXd::Zero(num_element + 1, num_element + 1);

    // Fill in the values of the global stiffness matrix
    for (int i = 0; i < num_element; i++) {
        double coeff = this->element.getCoeff(qno, i); // Determine the coefficient for the specific element

        // Update the global stiffness matrix using the local stiffness matrix.
        global_stiffness_matrix(i, i) += coeff;
        global_stiffness_matrix(i + 1, i) += -1 * coeff;
        global_stiffness_matrix(i, i + 1) += -1 * coeff;
        global_stiffness_matrix(i + 1, i + 1) += coeff;
    }

    // Return a submatrix corresponding to the actual system size
    return global_stiffness_matrix.block(0, 0, num_element, num_element);
}
```

Figure 30: System.cpp

System.cpp

```cpp
Eigen::MatrixXd System::createForceVector(int num_element, double force) {
    /*
    Creates a force vector for the system.

    Parameters:
        - num_element (int): Number of elements
        - force (double): Applied force

    Returns:
        Eigen::MatrixXd: Force vector
    */

    // Initialize the force vector with dimensions
    force_vector = Eigen::MatrixXd::Zero(num_element, 1);

    // Set the first entry of the vector as the applied force, leaving the rest as 0.
    force_vector(0, 0) = force;

    return force_vector;
}

System::System(int qno, int num_element, double force, double Length, double area, double Youngs) :
    num_element(num_element), qno(qno), force(force), Length(Length), element(area, Youngs, Length / num_element) {
    /*
    Constructor for System.

    Parameters:
        - num_element (int): Number of elements
        - force (double): Force
        - qno (int): Question number
        - Length (double): Length of the rod
        - area (T): Area of the rod
        - Youngs (double): Young's modulus of the rod
    */

    // Set the Global Stiffness Matrix and the Force vector of the system.
    global_stiffness_matrix = createGlobalStiffness(qno, num_element);
    force_vector = createForceVector(num_element, force);
}
```

Figure 31: System.cpp

System.cpp

```cpp

Eigen::MatrixXd System::getGlobalStiffnessMatrix() {
    /*
    Returns the Global Stiffness Matrix.
    */
    return global_stiffness_matrix;
}

Eigen::MatrixXd System::getForceVector() {
    /*
    Returns the Force Vector.
    */
    return force_vector;
}

```

Figure 32: System.cpp

Acknowledgment

We would like to acknowledge the valuable insights gained from the Finite Element Method (FEM) lecture series on NPTEL. The lecture is available on YouTube : FEM Lecture NPTEL.