CS601 - Software Development for Scientific Computing
Programming Assignment 2

210010001 - Aditya Zanjurne
210010008 - Balaji N K
210010055 - Cebajel Tanan

November 10, 2023

Contents

1 Contribution break-up

Cebajel Tanan

- Makefile
- Code

Balaji NK

- Readme file
- Report: Contents, Contribution, Mathematical formulation of Problem 1

Aditya Sujeet Zanjurne

- Mathematical formulation of Problem 2 and Observations

2 Objective

The objective of this assignment is to implement Finite Element Method for a 1D rod problem. The problem description is as follows
Consider a rod with cross sectional area $A(x)$ and length L. The rod is subjected to a constant load $P = 5000N$ 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. Consider two subproblems:

- The cross section of the rod is uniform with area:
  $A(x) = A_0 = 12.5 \times 10^4 m^2$.
- The cross sectional area is given by the formula

  $$A(x) = A_0(1 + x/L)$$

  Here the cross section is not uniform, it increases linearly with x.
- Write an Finite Element code to find the displacement at the nodal points on the rod. You need to discretize the rod into N = 2, 8, 32, 128 elements of equal length for problems in 1 and 2.
- Plot your numerical solution. Write your observations.

3 Method - Mathematical Formulation

3.1 Problem 1

In our solution we have assumed that a compressive force acts on the rod which causes displacement at various points on the rod. We have done this so that we would not have to change any signs in our calculations.

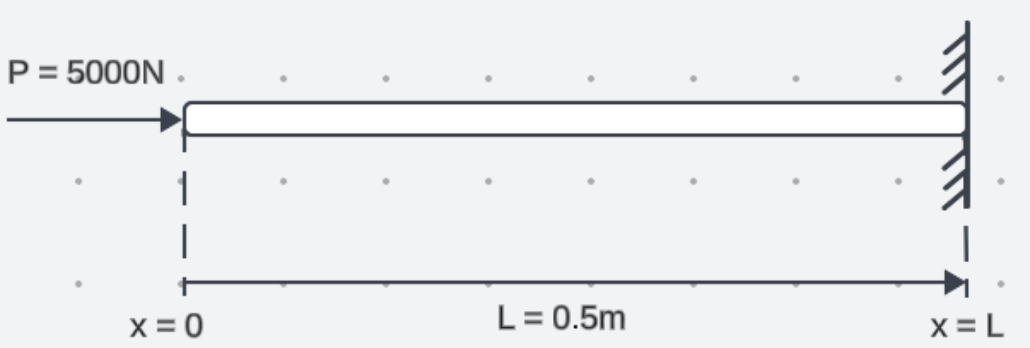

Figure 1: Compressive force acting on the rod which causes stress and hence the displacement of points on the rod

In this problem the area of cross-section of the rod remains constant throughout the length of the rod. Hence we can use the stiffness matrix which we had derived in the notes. Every elemental stiffness matrix will look like this:

$$\frac{EA_0}{L/N}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

where N = number of elements.
So the global stiffness matrix(in case of N = 2) will be:

$$\begin{bmatrix} \frac{A_0E}{L/2} & -\frac{A_0E}{L/2} & 0 \\ -\frac{A_0E}{L/2} & \frac{A_0E}{L/2}+\frac{A_0E}{L/2} & -\frac{A_0E}{L/2} \\ 0 & -\frac{A_0E}{L/2} & \frac{A_0E}{L/2} \end{bmatrix}$$

The force vector will have the first entry as 5000 and last entry as -5000 because those are the forces acting on the first and last node. On the other nodes the external force acting is 0 hence we put their entry as 0. So the force vector(for N = 2) will be:

$$\begin{bmatrix} 5000 \\ 0 \\ -5000 \end{bmatrix}$$

Now the matrix equation for N = 2 looks like this:

$$\begin{bmatrix} \frac{A_0E}{L/2} & -\frac{A_0E}{L/2} & 0 \\ -\frac{A_0E}{L/2} & \frac{A_0E}{L/2}+\frac{A_0E}{L/2} & -\frac{A_0E}{L/2} \\ 0 & -\frac{A_0E}{L/2} & \frac{A_0E}{L/2} \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ 0 \end{bmatrix} = \begin{bmatrix} 5000 \\ 0 \\ -5000 \end{bmatrix}$$

This can be generalized for N = 8, 32, 128.

3.2 Problem 2

In this case the area of the rod varies throughout the length of the rod as a linear function of x.

$$A(x) = A_0(1 + x/L)$$

However, the only change which occurs is in calculating the elemental stiffness matrix which contains this area term inside the integration. Let's look at it.
Suppose we have divided the rod in N equal parts, then let's first see what a Jacobian would look like in general.
$\frac{dx}{d\xi} = x_1 \frac{d}{d\xi} N_1 + x_2 \frac{d}{d\xi} N_2 = x_1 \frac{d}{d\xi}\left(\frac{1-\xi}{2}\right) + x_2 \frac{d}{d\xi}\left(\frac{1+\xi}{2}\right) = \frac{-x_1}{2} + \frac{x_2}{2} = \frac{x_2 - x_1}{2} = \frac{(x_1 + L/N) - x_1}{2} = \frac{L}{2N} = J$
We have our expression for area in terms of x. In our formulation we have taken area of an element as the area at the mid-point of the element. So the area expression which we will use in integration will be:

$$\begin{aligned} A &= \frac{A(x_i) + A(x_{i+1})}{2} \\ &= A_0 \frac{1 + \frac{x_i}{L} + 1 + \frac{x_i + L/N}{L}}{2} \\ &= A_0 \left(1 + \frac{x_i}{L} + \frac{1}{2N}\right) \end{aligned}$$

Now, we will compute all the elements of stiffness matrix. Let's look at K_{11} first.

$$\begin{aligned} K_{11} &= \int_{-1}^{1} EA \frac{dN_1}{d\xi} \frac{dN_1}{d\xi} [J]^{-1} d\xi \\ &= EA_0 \int_{-1}^{1} \left(1 + \frac{x_i}{L} + \frac{1}{2N}\right) \left(-\frac{1}{2}\right) \left(-\frac{1}{2}\right) \left(\frac{2N}{L}\right) d\xi \\ &= \frac{EA_0 N}{L} \left(1 + \frac{x_i}{L} + \frac{1}{2N}\right) \end{aligned}$$

Similarly we will get:

$$\begin{aligned} K_{12} &= -\frac{EA_0 N}{L} \left(1 + \frac{x_i}{L} + \frac{1}{2N}\right) \\ K_{21} &= -\frac{EA_0 N}{L} \left(1 + \frac{x_i}{L} + \frac{1}{2N}\right) \\ K_{22} &= \frac{EA_0 N}{L} \left(1 + \frac{x_i}{L} + \frac{1}{2N}\right) \end{aligned}$$

4 Experimental Results

4.1 Plots

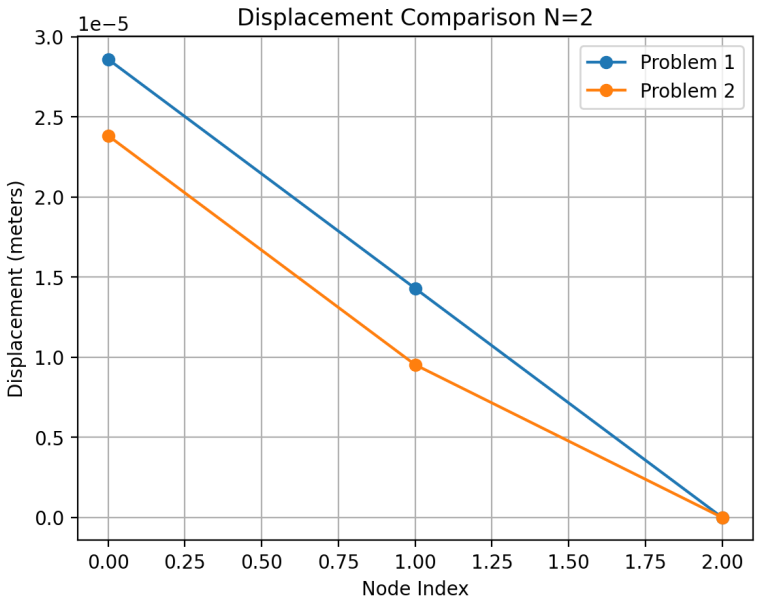

Figure 2: Plot of Displacement v/s Node Index for 2 elements

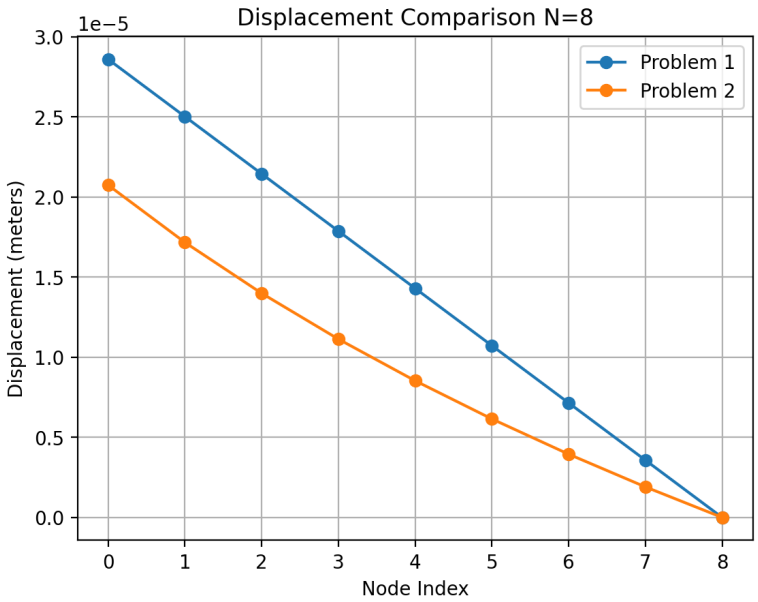

Figure 3: Plot of Displacement v/s Node Index for 8 elements

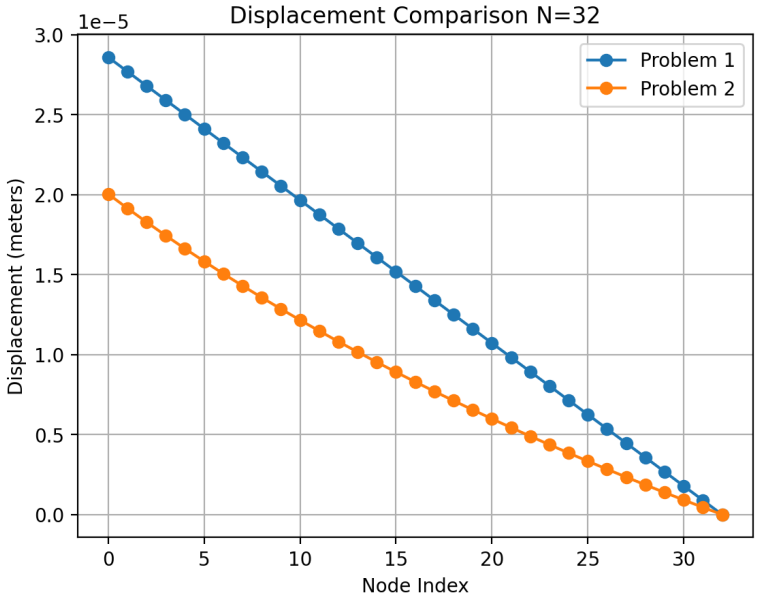

Figure 4: Plot of Displacement v/s Node Index for 32 elements

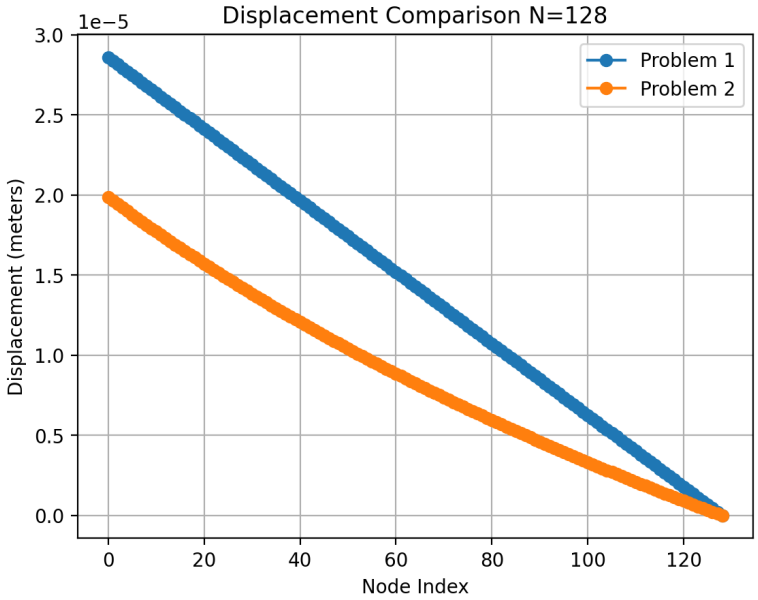

Figure 5: Plot of Displacement v/s Node Index for 128 elements

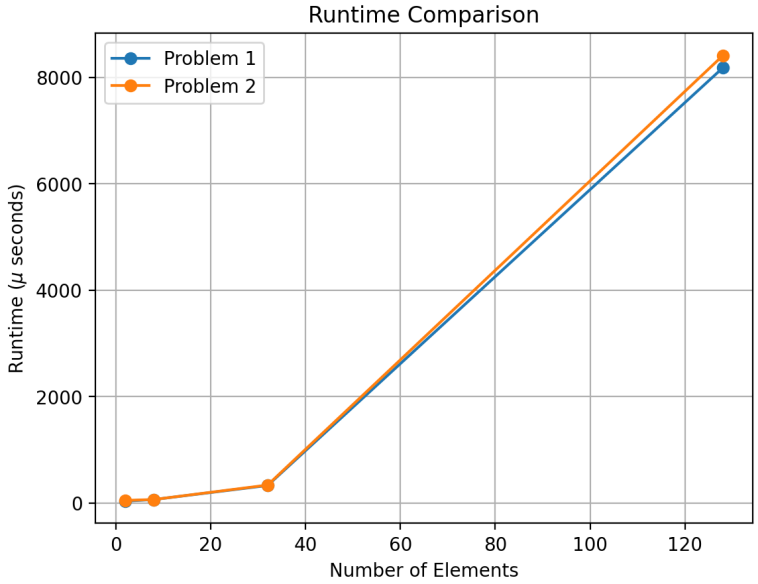

Figure 6: Plot of Runtime v/s Number of elements for both the problems

4.2 Observation

4.2.1 Displacement v/s Node Index

In each of the graphs we can clearly see that for Problem 1 we get a linear plot and it decreases with increasing node index while for Problem 2 it decreases non-linearly. But what is interesting to see is that both of them converge at the final node index as the displacement of final node is 0 because it is fixed.

4.2.2 Runtime v/s Number of Elements

The runtime for both the problems remains almost the same till N = 32, but as N approaches 128 the runtime for Problem 2 is more than that of Problem 1 which is expected as there are relatively more computations in Problem 2.

5 Appendix and Code

5.1 Appendix

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=2 PROB=1
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=2 -D PROB=1
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
2.86e-05
1.43e-05
       0
Time taken: 37 microseconds
```

Figure 7: Values for N=2 and Problem 1

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=8 PROB=1
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=8 -D PROB=1
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
   2.86e-05
 2.5025e-05
  2.145e-05
 1.7875e-05
   1.43e-05
 1.0725e-05
7.1501e-06
  3.575e-06
         0
Time taken: 66 microseconds
```

Figure 8: Values for N=8 and Problem 1

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=32 PROB=1
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=32 -D PROB=1
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
   2.86e-05
2.77063e-05
2.68125e-05
2.59188e-05
 2.5025e-05
2.41313e-05
2.32375e-05
2.23438e-05
  2.145e-05
2.05563e-05
1.96625e-05
1.87688e-05
 1.7875e-05
1.69813e-05
1.60875e-05
1.51938e-05
   1.43e-05
1.34063e-05
1.25125e-05
1.16188e-05
 1.0725e-05
9.83126e-06
8.93751e-06
8.04376e-06
7.15001e-06
6.25626e-06
5.36251e-06
4.46875e-06
  3.575e-06
2.68125e-06
 1.7875e-06
8.93751e-07
         0
Time taken: 330 microseconds
```

Figure 9: Values for N=32 and Problem 1

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=128 PROB=1
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=128 -D PROB=1
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
   2.86e-05
2.83766e-05
2.81532e-05
2.79297e-05
2.77063e-05
2.74828e-05
2.72594e-05
 2.7036e-05
2.68125e-05
2.65891e-05
2.63657e-05
2.61422e-05
2.59188e-05
2.56953e-05
2.54719e-05
2.52485e-05
 2.5025e-05
2.48016e-05
2.45781e-05
2.43547e-05
2.41313e-05
2.39078e-05
2.36844e-05
 2.3461e-05
2.32375e-05
2.30141e-05
2.27906e-05
2.25672e-05
2.23438e-05
2.21203e-05
2.18969e-05
2.16735e-05
   2.145e-05
2.12266e-05
2.10031e-05
2.07797e-05
2.05563e-05
```

Figure 10: Values for N=128 and Problem 1 (1)

```
1.00547e-05
9.83126e-06
9.60782e-06
9.38438e-06
9.16095e-06
8.93751e-06
8.71407e-06
8.49063e-06
 8.2672e-06
8.04376e-06
7.82032e-06
7.59688e-06
7.37344e-06
7.15001e-06
6.92657e-06
6.70313e-06
6.47969e-06
6.25626e-06
6.03282e-06
5.80938e-06
5.58594e-06
5.36251e-06
5.13907e-06
4.91563e-06
4.69219e-06
4.46875e-06
4.24532e-06
4.02188e-06
3.79844e-06
  3.575e-06
3.35157e-06
3.12813e-06
2.90469e-06
2.68125e-06
2.45781e-06
2.23438e-06
2.01094e-06
 1.7875e-06
1.56406e-06
1.34063e-06
1.11719e-06
8.93751e-07
6.70313e-07
4.46875e-07
2.23438e-07
         0
Time taken: 8184 microseconds
```

Figure 11: Values for N=128 and Problem 1 (2)

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=2 PROB=2
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=2 -D PROB=2
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
2.38334e-05
9.53334e-06
          0
Time taken: 50 microseconds
```

Figure 12: Values for N=2 and Problem 2

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=8 PROB=2
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=8 -D PROB=2
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
2.07457e-05
1.71707e-05
1.39929e-05
1.11329e-05
8.53287e-06
6.14953e-06
3.94953e-06
1.90667e-06
          0
Time taken: 66 microseconds
```

Figure 13: Values for N=8 and Problem 2

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=32 PROB=2
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=32 -D PROB=2
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
2.00492e-05
1.91555e-05
1.82888e-05
1.74476e-05
1.66305e-05
 1.5836e-05
1.50631e-05
1.43104e-05
1.35771e-05
1.28621e-05
1.21645e-05
1.14836e-05
1.08185e-05
1.01685e-05
 9.5329e-06
8.91116e-06
8.30265e-06
7.70681e-06
7.12314e-06
6.55114e-06
5.99035e-06
5.44035e-06
4.90073e-06
 4.3711e-06
 3.8511e-06
3.34038e-06
2.83863e-06
2.34553e-06
1.86078e-06
1.38411e-06
9.15259e-07
4.53969e-07
          0
Time taken: 338 microseconds
```

Figure 14: Values for N=32 and Problem 2

```
[cs601user10@hip cs601pa2-Aditya-iitdh]$ make N=128 PROB=2
g++ -c -O3 -g -Wall src/fem.cpp -o ./obj/fem.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/ -D NUMBER_OF_ELEMENTS=128 -D PROB=2
g++ -c -O3 -g -Wall src/element.cpp -o ./obj/element.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -c -O3 -g -Wall src/rod.cpp -o ./obj/rod.o -I./inc -I /home/resiliente/cs601software/eigen-3.3.9/
g++ -O3 -g -Wall ./obj/* -o ./bin/fem -I./inc
./bin/fem

Hello!

Computed Displacement vector x:
  1.988e-05
1.96566e-05
1.94349e-05
1.92149e-05
1.89965e-05
1.87799e-05
1.85648e-05
1.83514e-05
1.81395e-05
1.79292e-05
1.77205e-05
1.75132e-05
1.73075e-05
1.71032e-05
1.69004e-05
 1.6699e-05
 1.6499e-05
1.63003e-05
1.61031e-05
1.59072e-05
1.57127e-05
1.55196e-05
```

Figure 15: Values for N=128 and Problem 2 (1)

```
3.5722e-06
3.44565e-06
3.31966e-06
3.19422e-06
3.06933e-06
2.94499e-06
2.82118e-06
 2.6979e-06
2.57515e-06
2.45293e-06
2.33123e-06
2.2104e-06
2.08937e-06
 1.9692e-06
1.84953e-06
1.73037e-06
1.61169e-06
1.49351e-06
1.37582e-06
 1.2586e-06
1.14187e-06
1.02561e-06
9.09818e-07
7.94495e-07
6.79636e-07
5.65236e-07
4.51291e-07
3.37799e-07
2.24756e-07
1.12157e-07
         0
Time taken: 8413 microseconds
```

Figure 16: Values for N=128 and Problem 2 (2)

5.2 Code

5.2.1 inc/element.h

```
#include <Eigen/Sparse>
/*
This class contains necessary attributes and fuctions to
define a 1D rod element in Finite Element Analysis.
Parameters :
    xa : Coordinates of starting point of the element
    xb : Coordinates of ending point of the element
    Area : Area of the element
    stiffness : Local stiffness matrix of the element

Functions :
    get_area() : returns area of the element
    get_xa() : returns starting point coordinates of the element
    get_stiffness() : returns local stiffness matrix of the element
    get_stiffness_index(int i, int j) : returns value at ith row
         and jth column of the local stiffness matrix of the element
*/
#ifndef _Element_h_
#define _Element_h_
class Element {
    private:
        double xa;
        double xb;
        double area;
        Eigen::SparseMatrix<double> stiffness;
    public:
    Element(){};
    Element(double XA, double XB, double Area, double young_modulus, double element_length);
    double get_area();
    const Eigen::SparseMatrix<double> get_stiffness();
    double get_stiffness_index(int index_i, int index_j);
    ~Element(){};
};
#endif
```

5.2.2 inc/rod.h

```
#include "element.h"
#include <Eigen/Sparse>
/*
This class contains necessary attributes and functions to define a 1D rod Fininte Element Analysis.
Parameters:
    elements : it is an array of elements of the rod
    global_stiffness : it is a 2D matrix of global stiffness matrix

Functions:
    get_elements() : returns an array of elements of the rod
    get_element(int i) : returns ith index element from "elements" array
    get_stiffness() : returns gloabal stiffness matrix of the rod
    ~Rod() : Destructor of class Rod
*/
#ifndef _Rod_h_
#define _Rod_h_
class Rod {
    private:
        double length;
        double young_modulus;
        double element_length;
        int number_of_elements;
        int number_of_nodes;
        double a_0;
        bool variable_area;
        Element * elements;
        Eigen::SparseMatrix<double> global_stiffness;
    public:
    Rod(double Len, int num_elements, double Young_Modulus, double a_0, bool variable_area);
    const Element * get_elements();
    const Element get_element(int index);
    const Eigen::SparseMatrix<double> get_stiffness();
    ~Rod();
};
#endif
```

5.2.3 src/element.cpp

```
#include "../inc/element.h"
#include <Eigen/Sparse>

// Constructor of the class Element
Element::Element(double XA, double XB, double Area, double young_modulus, double element_length){
    // initializing values of xa, xb, and area of the class element
    this->xa = XA;
    this->xb = XB;
    this->area = Area;

    // Allocating memory to stiffness matrix
    this->stiffness.resize(2, 2);

    // Initializing values of stiffness matrix
    for (int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
            if(i == j){
                this->stiffness.insert(i, j) = (area*young_modulus)/element_length;
            }
            else{
                this->stiffness.insert(i, j) = -(area*young_modulus)/element_length;
            }
        }
    }

}

// get_area() : returns area of the element
double Element::get_area(){
    return this->area;
}

// get_stiffness() : returns local stiffness matrix of the element
const Eigen::SparseMatrix<double> Element::get_stiffness(){
    return this->stiffness;
}

// get_stiffness_index(int i, int j) : returns value at ith row and jth column of the local stiffness m
double Element::get_stiffness_index(int index_i, int index_j){
    return this->stiffness.coeff(index_i, index_j);
}
```

5.2.4 src/rod.cpp

```
#include "../inc/element.h"
#include "../inc/rod.h"
#include <Eigen/Sparse>

// Constructor of the class rod
Rod::Rod(double Len, int num_elements, double Young_Modulus, double a_0, bool variable_area){
    // Allocating memory and initializing global stiffness matrix with zeros
    this->length = Len;
    this->number_of_elements = (int) num_elements;
    this->young_modulus = Young_Modulus;
    this->element_length = this->length/this->number_of_elements;
    this->number_of_nodes = this->number_of_elements + 1;
    this->a_0 = a_0;
    this->variable_area = variable_area;

    this->global_stiffness.resize(this->number_of_nodes, this->number_of_nodes);
    for(int i = 0; i < this->number_of_nodes; i++){
        for (int j = 0; j < this->number_of_nodes; j++){
            this->global_stiffness.insert(i, j) = 0;
        }
    }

    // Allocating memory and initializing elements array
    this->elements = new Element[this->number_of_elements];
    for (int i = 0; i < this->number_of_elements; i++){
        double area = this->a_0;
        if (this->variable_area){
            area = this->a_0 * (1 + ((i * this->element_length) / this->length) + 1 / (2 * this->number
        }
        this->elements[i] = Element(i*(this->element_length), (i+1)*(this->element_length), area, this-
    }

    // filling correct values in global stiffness matrix using local stiffness matrix of the containing
    for(int i = 0; i < this->number_of_elements; i++){
        this->global_stiffness.coeffRef(i, i) += this->elements[i].get_stiffness_index(0, 0);
        this->global_stiffness.coeffRef(i, i + 1) += this->elements[i].get_stiffness_index(0, 1);
        this->global_stiffness.coeffRef(i + 1, i) += this->elements[i].get_stiffness_index(1, 0);
        this->global_stiffness.coeffRef(i + 1, i + 1) += this->elements[i].get_stiffness_index(1, 1);
    }
}

// get_elements() : returns an array of elements of the rod
const Element * Rod::get_elements(){
    return this->elements;
}
// get_element(int i) : returns ith index element from "elements" array
const Element Rod::get_element(int index){
    return this->elements[index];
}
// get_stiffness() : returns gloabal stiffness matrix of the rod
const Eigen::SparseMatrix<double> Rod::get_stiffness(){
    return this->global_stiffness;
}
// destructor of Rod class: deallocated memory allocated to elements
Rod::~Rod() {
    delete[] this->elements;
}
```

5.2.5 src/fem.cpp

```
/*
This file contains code to discretize and solve a 1D rod
problem assigned in CS 601 assignment 2
using finite element analysis.
*/

#include "../inc/rod.h"
#include <Eigen/Sparse>
#include <iostream>
#include <chrono>

#ifndef NUMBER_OF_ELEMENTS
#define NUMBER_OF_ELEMENTS 2
#endif

#ifndef PROB
#define PROB 1
#endif

#ifndef VARIABLE_AREA
#if PROB == 1
#define VARIABLE_AREA false
#else
#define VARIABLE_AREA true
#endif
#endif

#ifndef LENGTH
#define LENGTH 0.5
#endif

#ifndef FORCE
#define FORCE 5000
#endif

#ifndef A_0
#define A_0 12.5e-4
#endif

#ifndef YOUNG_MODULUS
#define YOUNG_MODULUS 7e10
#endif

int main(){
    std::cout<<"\nHello!\n"<<std::endl;
    auto start_time = std::chrono::high_resolution_clock::now();
    // Declaring a rod object
    Rod rod = Rod(LENGTH, NUMBER_OF_ELEMENTS, YOUNG_MODULUS, A_0, VARIABLE_AREA);
    int number_of_nodes = NUMBER_OF_ELEMENTS + 1;
    // Declaring and allocating memory to force vector
    Eigen::VectorXd force(number_of_nodes);

    // Initializing force vector
    for (int i = 0; i < number_of_nodes; i++) {
        if (i == 0) {
            force(i) = FORCE;
        } else if (i == number_of_nodes - 1) {
            force(i) = -FORCE;
        } else {
```

```
            force(i) = 0;
        }
    }

    // Declaring a solver
    Eigen::SparseQR<Eigen::SparseMatrix<double>, Eigen::COLAMDOrdering<int>> solver;
    double shiftValue = 1e-3;
    int count = 0;
    bool decomposed = false;
    // iterating till matrix gets decomposed or upto max 10 times
    while((count<10) && !(decomposed)){
        // Solve the shifted linear system (A - shift * I) * x = b using SparseQR
        solver.compute(rod.get_stiffness() - shiftValue * Eigen::SparseMatrix<double>(rod.get_stiffness
        if (solver.info() == Eigen::Success) {
            decomposed = true;
        }
        // shifting the shift value if matrix does not get decomposed
        shiftValue *= 2;
        count++;
    }

    if(!decomposed){
        std::cout<<"Matrix could not be decomposed!"<<std::endl;
        return 1;
    }

    Eigen::VectorXd x = solver.solve(force);

    auto end_time = std::chrono::high_resolution_clock::now();
    std::cout << "Computed Displacement vector x:" << std::endl << x << std::endl;
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end_time - start_time);
    std::cout << "Time taken: " << duration.count() << " microseconds" << std::endl;

    return 0;
}
```