CS601: Software Development for Scientific Computing

Programming assignment 2: Finite Element Method

November 10, 2023

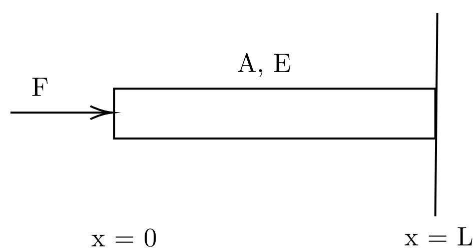
1 Problem statement

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 $0.5m$ and the Young's modulus of the material of the rod is $E = 70GPa$. Consider two sub problems:

1. The cross section of the rod is uniform with area $A(x) = A_0 = 12.5 * 10^{-4}m^2$.
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.
3. 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.
4. Plot your numerical slution. Write your observations. Apart from this, the source code must be reorganized into folders. Report throughput and execution time for each implementation.

2 Method

This is a 1D structural problem. The rod is fixed at one end and a force is applied on the other end. The goal is to find approximate displacement of different points (denoted by $u(x)$) on the rod.



Strong From PDE

$$EA\frac{d^2u}{dx^2} + F = 0$$

$$u = \begin{cases} EA\frac{du}{dx} & \text{if } x = 0, \text{Neumann} \\ 0 & \text{if } x = L, \text{Dirichlet} \end{cases}$$

These are equilibrium equations and boundary conditions for the rod. These PDEs can be converted to weak from using Galerkin approach.

$$EA\frac{d^2\tilde{u}}{dx^2} + F = R$$

$$\int \omega R = 0$$

$$\int \omega[EA\frac{d^2\tilde{u}}{dx^2} + F] = 0$$

here ω is called weight function, $\tilde{u}$ is approximate displacement, and R is residual due to approximation.

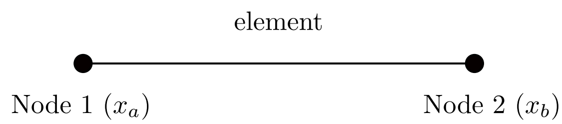
Weak from PDE .

By rearranging and simplifying above equations we get weak from of PDE. This will be used for FEM calculations.

$$\left[\omega EA\frac{d\tilde{u}}{dx}\right]_0^L + \int_0^L \omega F\,dx = \int_0^L EA\frac{d\omega}{dx}\frac{d\tilde{u}}{dx}\,dx$$

Discretization (Element)

Consider an element with 2 nodes, let u^e be approximate solution for the element and $h^e = x_b - x_a$ be the length of element.



$$u^e(x) = c_1 + c_2 x$$
$$u^e(x) = N_i u^e(x_a) + N_j u^e(x_b)$$
$$N_i(x) = \frac{x_b - x}{h^e}$$
$$N_j(x) = \frac{x - x_a}{h^e}$$

Now on simplifying weak from equations for for each node in the i^{th} element we get,

$$\begin{bmatrix} K_{ii} & K_{ij} \\ K_{ji} & K_{jj} \end{bmatrix} \times \begin{bmatrix} u_i \\ u_j \end{bmatrix} = \begin{bmatrix} g(i) \\ g(j) \end{bmatrix}$$

Here $j = i+1$, $K_{pq} = \int_{x_a}^{x_b} EA(x_a)\frac{N_p}{dx}\frac{N_q}{dx}dx$ and $g(p) = \left[N_p EA(x_a)\frac{d\tilde{u}}{dx}\right]_{x_a}^{x_b} + \int_{x_a}^{x_b} N_p F dx$

Area is approximated as $A = A(x_a)$, *i.e.* area of left end of the element. Now using 1 point Gauss quadrature we can approximate integrals as follows

$$K_{pq} = EA(x_a)\sum_{t=1}^{1} w_t(-1)^{p+q}, \quad \text{where} \quad w = \{1\}$$

Assembling

Computing and simplifying weak form equations for every element we the global stiffness matrix, displacement vector, and force vector as follows. There are total $n+1$ nodes and n elements.

$$\begin{bmatrix}
A_1\frac{E}{l} & -A_1\frac{E}{l} & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
-A_1\frac{E}{l} & A_1\frac{E}{l} + A_2\frac{E}{l} & -A_2\frac{E}{l} & 0 & \dots & 0 & 0 & 0 & 0 \\
0 & -A_2\frac{E}{l} & A_2\frac{E}{l} + A_3\frac{E}{l} & -A_3\frac{E}{l} & \dots & 0 & 0 & 0 & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \dots & -A_{n-2}\frac{E}{l} & A_{n-2}\frac{E}{l} + A_{n-1}\frac{E}{l} & -A_{n-1}\frac{E}{l} & 0 \\
0 & 0 & 0 & 0 & \dots & 0 & -A_{n-1}\frac{E}{l} & A_{n-1}\frac{E}{l} + A_n\frac{E}{l} & -A_n\frac{E}{l} \\
0 & 0 & 0 & 0 & \dots & 0 & 0 & -A_n\frac{E}{l} & A_n\frac{E}{l}
\end{bmatrix}$$

$$\times \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \\ u_{n+1} \end{bmatrix} = \begin{bmatrix} P \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -P \end{bmatrix}$$

Here, l = length of element,
u_i = displacement at i^{th} node,
A_i = Area at i^{th} node

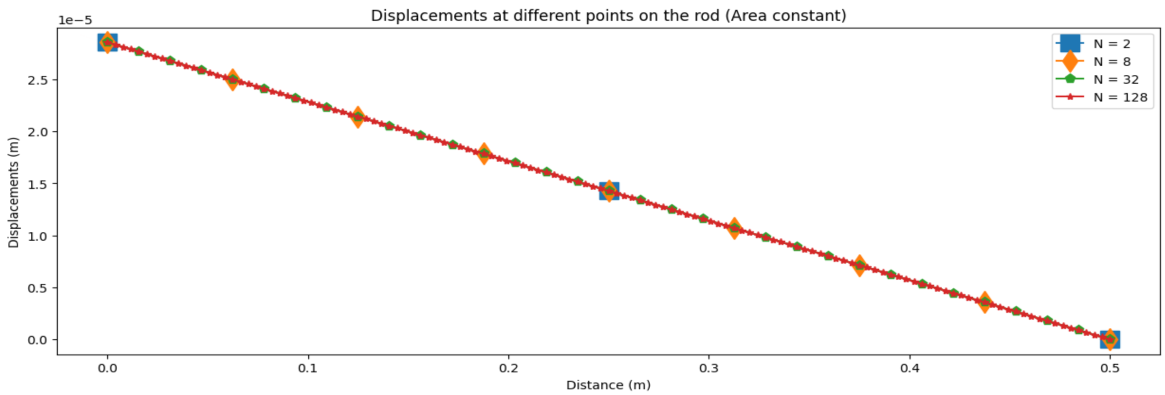
Above set of equations are of the form $Ax = b$, so a simple solution would be to compute displacements using $x = A^{-1}b$, but we know that $u_{n+1} = 0$ (boundary condition) and the matrix A is banded (tridiagonal). Thus we can get a linear time solution.

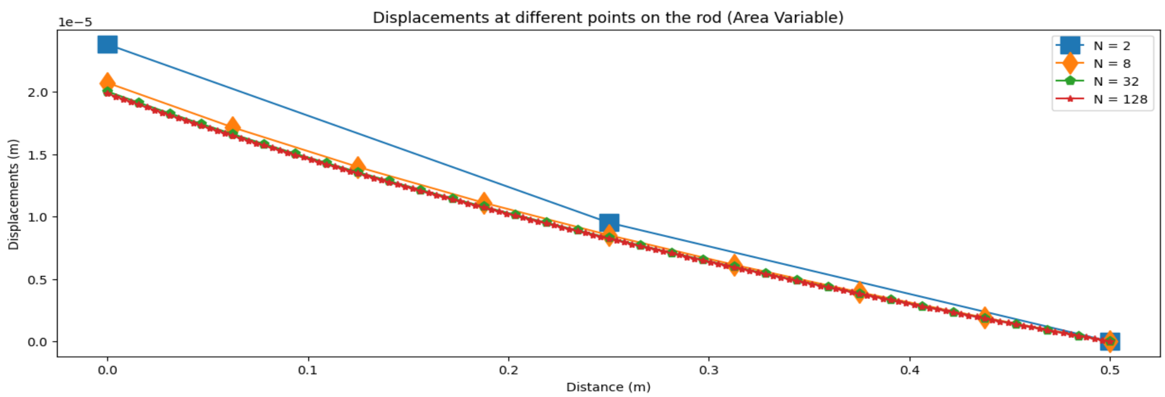
Algorithm 1 Calculating Displacements

Consider the rod divided into n elements (n+1 node points)
Let M be stiffness matrix, with entries $M_{i,j}$ ▷ it is a $n + 1 * n + 1$ matrix
Let F be force vector, with entries F_i ▷ length is n+1
Let u_i denote displacement value at i^{th} node.
$u_{n+1} \leftarrow 0$ ▷ Boundary condition
$u_n \leftarrow F_{n+1}/M_{n+1,n}$
$i \leftarrow n$
while $i \neq 0$ **do**
  $u_{i-1} \leftarrow \frac{F_i - u_i M_{i,i} - u_{i+1} M_{i,i+1}}{M_{i,i-1}}$
  $i \leftarrow i - 1$
end while
  return u

3 Results

Plots of displacement values for problem 1 and 2



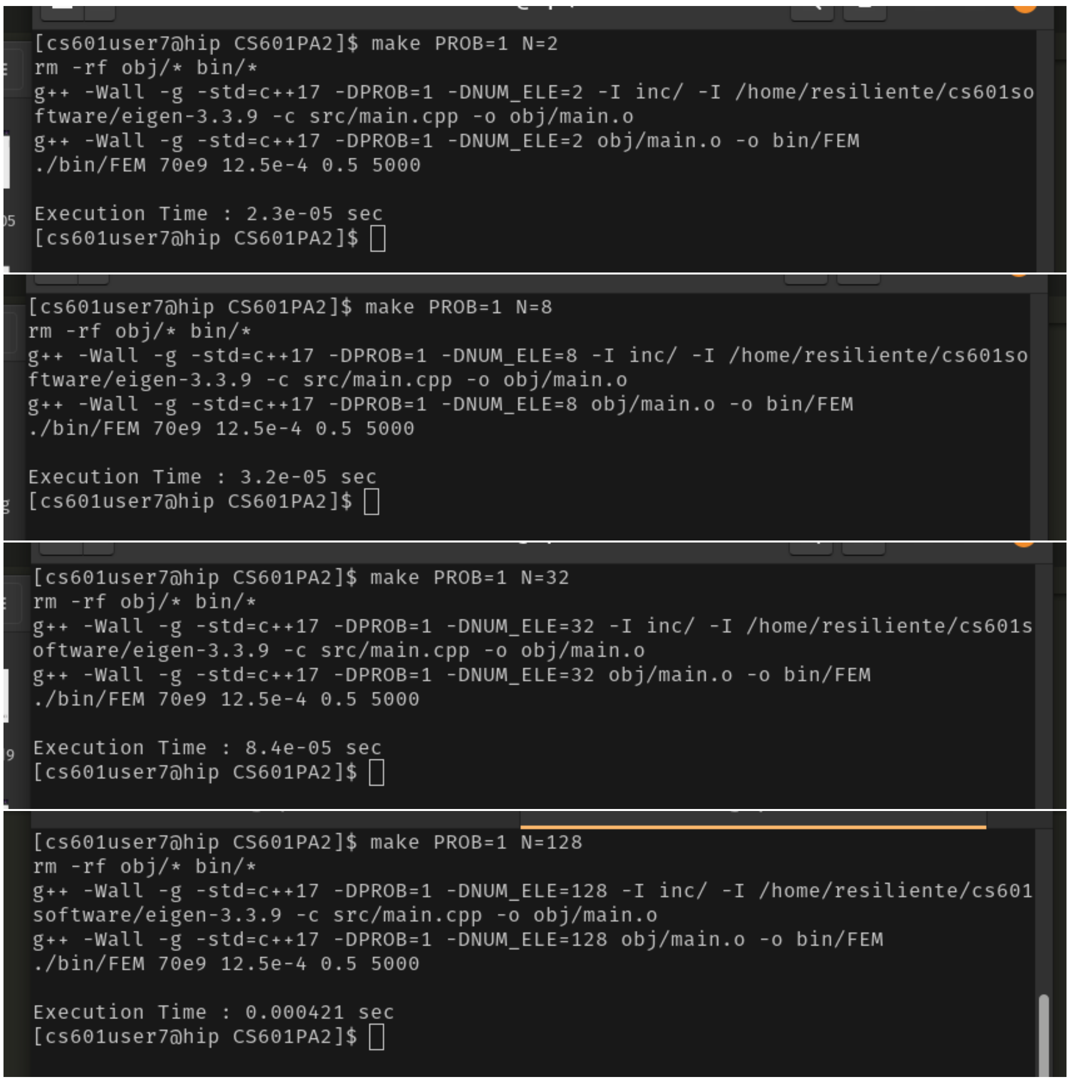


Observations

1. In problem 1 the displacement values remains more or less the same on increasing the number of nodes.
2. We see that as we increase the number of elements the accuracy of the solution increases (or the solution converges) in case of problem 2 where the area is linearly increasing.
3. This happens because, we consider more number of points and use their area in calculations.
4. Higher the number of elements, higher is the computation time and cost.

4 Appendix

4.1 Execution



```
[cs601user7@hip CS601PA2]$ make PROB=1 N=2
rm -rf obj/* bin/*
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=2 -I inc/ -I /home/resiliente/cs601so
ftware/eigen-3.3.9 -c src/main.cpp -o obj/main.o
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=2 obj/main.o -o bin/FEM
./bin/FEM 70e9 12.5e-4 0.5 5000

Execution Time : 2.3e-05 sec
[cs601user7@hip CS601PA2]$
```

```
[cs601user7@hip CS601PA2]$ make PROB=1 N=8
rm -rf obj/* bin/*
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=8 -I inc/ -I /home/resiliente/cs601so
ftware/eigen-3.3.9 -c src/main.cpp -o obj/main.o
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=8 obj/main.o -o bin/FEM
./bin/FEM 70e9 12.5e-4 0.5 5000

Execution Time : 3.2e-05 sec
[cs601user7@hip CS601PA2]$
```

```
[cs601user7@hip CS601PA2]$ make PROB=1 N=32
rm -rf obj/* bin/*
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=32 -I inc/ -I /home/resiliente/cs601s
oftware/eigen-3.3.9 -c src/main.cpp -o obj/main.o
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=32 obj/main.o -o bin/FEM
./bin/FEM 70e9 12.5e-4 0.5 5000

Execution Time : 8.4e-05 sec
[cs601user7@hip CS601PA2]$
```

```
[cs601user7@hip CS601PA2]$ make PROB=1 N=128
rm -rf obj/* bin/*
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=128 -I inc/ -I /home/resiliente/cs601
software/eigen-3.3.9 -c src/main.cpp -o obj/main.o
g++ -Wall -g -std=c++17 -DPROB=1 -DNUM_ELE=128 obj/main.o -o bin/FEM
./bin/FEM 70e9 12.5e-4 0.5 5000

Execution Time : 0.000421 sec
[cs601user7@hip CS601PA2]$
```

Figure 1: Problem 1

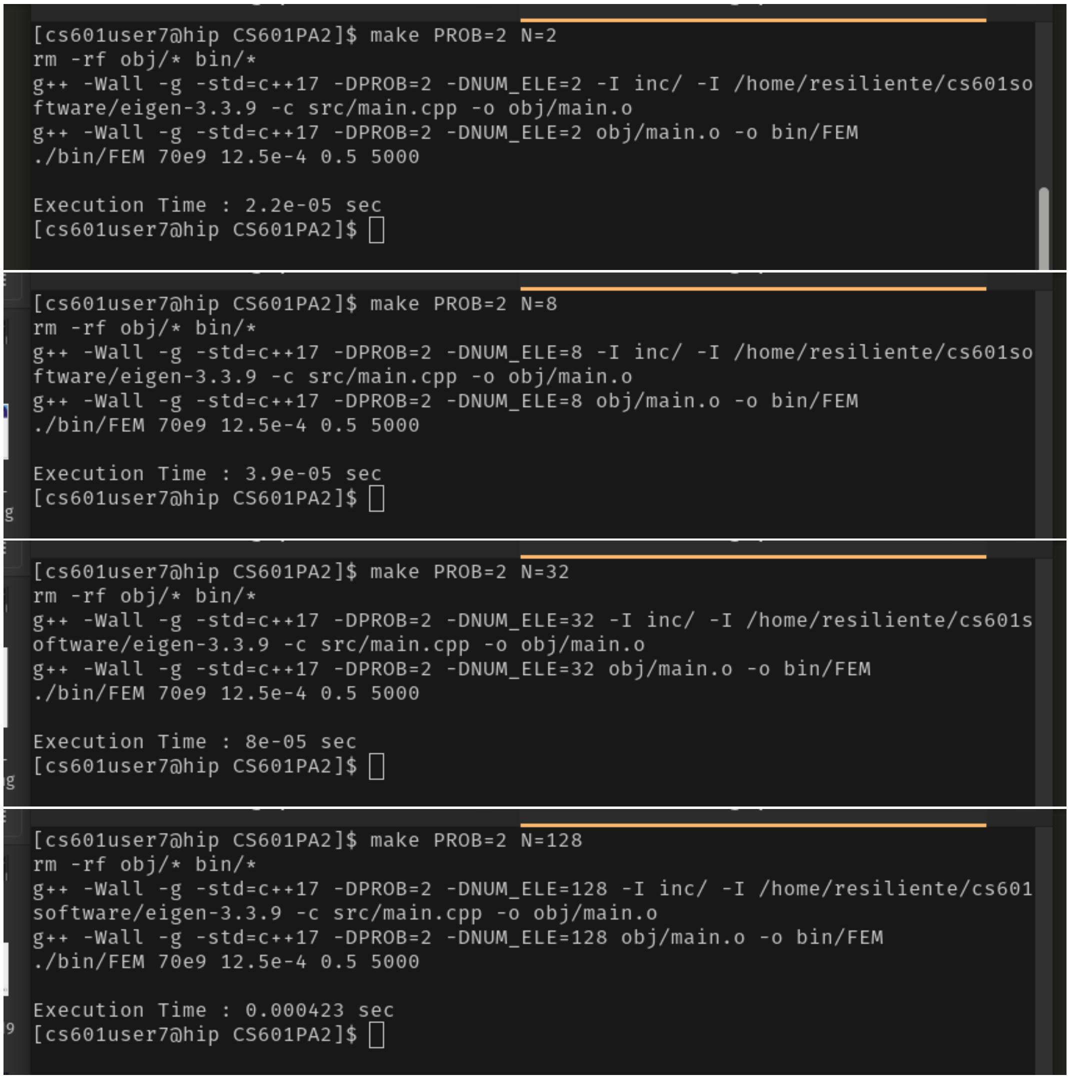


Figure 2: Problem 2

4.2 Code

4.2.1 main.cpp

```
#include<iostream>
#include "Solution.h"
#include "time.h"

int main(int argc, char* argv[]) {

    clock_t start, end;
    start = clock();

    if(argc != 5) {
        std::cerr << "E, A, L, P" << std::endl;
        exit(1);
    }

    int prob = 0, N = 0;
    #ifndef PROB
        std::cerr << "problem number not defined" << std::endl;
```

```
        exit(1);
    #else
        prob = (int)PROB;
    #endif

    #ifndef NUM_ELE
        std::cerr << "number of elements not defined" << std::endl;
        exit(1);
    #else
        N = (int)NUM_ELE;
    #endif

    double E, A, L, P;
    E = std::stod(argv[1]);
    A = std::stod(argv[2]);
    L = std::stod(argv[3]);
    P = std::stod(argv[4]);

    Domain<double> dom(N, E, A, L);
    Solution<double> sol(dom, P, prob);
    sol.solve();

    end = clock();
    double time_taken = double(end - start) / double(CLOCKS_PER_SEC);
    // sol.show_displacement_vector();

    std::cout << "\nExecution Time : "<< time_taken << " sec " << std::endl;
    return 0;
}
```

4.2.2 Solution.h

```
#include "Domain.h"
#include <Eigen/Dense>
using namespace Eigen;

template<class T>
class Solution {
public:
    Solution(const Domain<T>& dom, T load, int prob);
    void generate_global_stiffness_matrix();
    void generate_force_vector();
    void solve();
    void show_displacement_vector() const;
    ~Solution();

private:
    int prob;
    T load;
    Domain<T> dom;
    Matrix<T, Dynamic, Dynamic> global_stiffness_matrix;
    Matrix<T, Dynamic, 1> displacement_vector, force_vector;
    /* global_stiffness_matrix * displacement_vector = force_vector */
};


template<class T>
Solution<T>::Solution(const Domain<T>& dom, T load, int prob) {
    this->dom = dom;
    this->load = load;
    this->prob = prob;
    int n = dom.nodes_count();
    this->global_stiffness_matrix = Matrix<T, Dynamic, Dynamic>::Zero(n, n); // all values = 0
    this->force_vector = this->displacement_vector = Matrix<T, Dynamic, 1>::Zero(n, 1);

}

template<class T>
void Solution<T>::generate_global_stiffness_matrix() {
    for(int i = 0; i < dom.get_N(); i++) {
        /* first set the stiffness matrix for ith element */
        Element<T> element = this->dom.get_element(i);
        // printf("prob: %d\n", prob);
        if(this->prob == 1)
            element.set_stiffness_matrix(dom.get_E(), dom.get_A());
        else if(prob == 2)
            element.set_stiffness_matrix(dom.get_E(), dom.get_A() * (1 + i / T(dom.get_N())));

        // printf("\nvalue: %f", element.stiffness_matrix_cell(0,0));
        /* use this to find the global stiffness matrix */
        this->global_stiffness_matrix(i, i)     += element.stiffness_matrix_cell(0, 0);
        this->global_stiffness_matrix(i, i + 1) += element.stiffness_matrix_cell(0, 1);
        this->global_stiffness_matrix(i + 1, i) += element.stiffness_matrix_cell(1, 0);
        this->global_stiffness_matrix(i + 1, i + 1) += element.stiffness_matrix_cell(1, 1);
    }
}

/* For compression force at x = 0 */
template<class T>
void Solution<T>::generate_force_vector() {
    this->force_vector(0, 0) = this->load;
```

```
    this->force_vector(dom.nodes_count() - 1, 0) = -this->load;
}

/* finds the displacement vector */
/* For the given problem, it is not difficult to solve the system of linear equations */
template<class T>
void Solution<T>::solve() {
    this->generate_global_stiffness_matrix();
    this->generate_force_vector();
    int dim = this->displacement_vector.rows();
    /* boundary condition : u = 0 at x = L */
    displacement_vector(dim-1, 0) = 0 ;
    displacement_vector(dim-2, 0) = (this->force_vector(dim-1, 0) - 0)/
        this->global_stiffness_matrix(dim-1, dim-2);

    for (int i = this->displacement_vector.rows()-2; i > 0 ; i--) {
        displacement_vector(i-1, 0) =( this->force_vector(i, 0) -
            this->global_stiffness_matrix(i, i) * displacement_vector(i, 0) -
            this->global_stiffness_matrix(i, i+1)*this->displacement_vector(i+1)) /
            this->global_stiffness_matrix(i,i-1);
    }
}

template<class T>
void Solution<T>::show_displacement_vector() const {
    std::cout << "\nDISPLACEMENT VECTOR:\n" << this->displacement_vector << std::endl;
}

template<class T>
Solution<T>::~Solution() {}
```

4.2.3 Domain.h

```cpp
#include "Element.h"
#include <vector>

template<class T>
class Domain {
public:
    Domain();
    Domain(int N, T E, T A, T L);
    Element<T> get_element(int i) const;
    int get_N() const;
    int nodes_count() const;
    T get_E() const;
    T get_A() const;
    T get_L() const;
    ~Domain();
private:
    int N; // number of elements
    T E, A, L;
    std::vector<Element<T>> elements;
};

template<class T>
Domain<T>::Domain() {}

/* parameterized constructor */
template<class T>
Domain<T>::Domain(int N, T E, T A, T L) {
    this->N = N;
    this->E = E;
    this->A = A;
    this->L = L;
    this->elements.resize(N, Element<T>(L / N, 2));
}

/* fetches the ith element of the rod */
template<class T>
Element<T> Domain<T>::get_element(int i) const {
    return this->elements[i];
}

/* Get the number of elements */
template<class T>
int Domain<T>::get_N() const {
    return this->N;
}

/* count of total number of nodes */
template<class T>
int Domain<T>::nodes_count() const {
    return this->N + 1;
}

/* young's modulus of rod */
template<class T>
T Domain<T>::get_E() const {
    return this->E;
}

/* area constant A_0 */
```

```
template<class T>
T Domain<T>::get_A() const {
    return this->A;
}

/* length of the rod */
template<class T>
T Domain<T>::get_L() const {
    return this->L;
}

/* Destructor */
template<class T>
Domain<T>::~Domain() {}
```

4.2.4 Element.h

```
#include <Eigen/Dense>
using namespace Eigen;

template<class T>
class Element {
public:
Element();
Element(T length, int n);
T gauss_quad(T E, T A);
void set_stiffness_matrix(T E, T A);
T stiffness_matrix_cell(int i, int j) const;
~Element();

private:
    T length;
    Matrix<T, Dynamic, Dynamic> stiffness_matrix;
    int num_of_nodes;
};

template<class T>
Element<T>::Element() {}

template<class T>
Element<T>::Element(T length, int n) {
    this->length = length;
    this->num_of_nodes = n;
    stiffness_matrix = Matrix<T, Dynamic, Dynamic>::Zero(n, n);
}

/* one point gauss quadrature */
template<class T>
T Element<T>::gauss_quad(T E, T A) {
    return E * A / length;
}

template<class T>
void Element<T>::set_stiffness_matrix(T E, T A) {
    for(int i = 0; i < this->num_of_nodes; i++)
        for(int j = 0; j < this->num_of_nodes; j++)
            this->stiffness_matrix(i, j) = gauss_quad(E, A) * ((i + j) % 2 ? -1 : 1);
}

/* returns Kij */
template<class T>
T Element<T>::stiffness_matrix_cell(int i, int j) const {
    return this->stiffness_matrix(i, j);
}

template<class T>
Element<T>::~Element() {}
```

4.2.5 Makefile

```
CXX = g++

EIGEN_PATH = /home/resiliente/cs601software/eigen-3.3.9


# Variables for prob and N
PROB ?= 1
N ?= 2
E := 70e9
A := 12.5e-4
L := 0.5
P := 5000

# run 70 12.5e-4 0.5 5000

CXXFLAGS = -Wall -g -std=c++17 -DPROB=$(PROB) -DNUM_ELE=$(N)
INC_DIR = inc
SRC_DIR = src
OBJ_DIR = obj
BIN_DIR = bin

PROG_NAME = $(BIN_DIR)/FEM

# Default target
run: clean $(PROG_NAME)
   ./$(PROG_NAME) $(E) $(A) $(L) $(P)

$(PROG_NAME): $(OBJ_DIR)/main.o
   $(CXX) $(CXXFLAGS) $^ -o $@

$(OBJ_DIR)/main.o: $(SRC_DIR)/main.cpp $(INC_DIR)/Solution.h $(INC_DIR)/Domain.h
     $(INC_DIR)/Element.h
   $(CXX) $(CXXFLAGS) -I $(INC_DIR)/ -I $(EIGEN_PATH) -c $< -o $@

# Create obj and bin directories if they don't exist
$(OBJ_DIR) $(BIN_DIR):
   mkdir -p $@

team:
   @echo Team Members:
   @echo 210010015 - Divy Jain
   @echo 210010022 - Karthik Hegde

clean:
   rm -rf $(OBJ_DIR)/* $(BIN_DIR)/*

.PHONY: clean
```

5 Team Members

1. Divy Jain (210010015)
2. Karthik Hegde (210010022)