CS 601: Software Development and Scientific Computing

Assignment-2

Siddharth Shankar, 200030056
Vidyasagar S Singadi, 200010056

`https://github.com/IITDhCSE/cs601pa2-Sid-Shankar`

Contents

1 Problem

A rod with length L = 0.5m and area of cross section A(x) and Young's modulus Y = 70GPa and is subjected to load P=5000N at x = 0 and fixed at x = L.

2 Mathematical Formulations

Images corresponding to our mathematical formulations can be found at `https://github.com/IITDhCSE/cs601pa2-Sid-Shanka tree/master/mathematical_formulations`

3 Analytical analysis

- Rod with uniform cross section, with $A(x) = A_0 = 12.5 \times 10^{-4} m^2$

$$u(x) = \frac{P}{A_0 E}(L - x) \tag{1}$$

- Rod with uniform cross section increasing linearly (non-uniform), with $A(x) = A_0(1 + x/L)$

$$u(x) = \frac{PL}{EA_0} \ln\left(\frac{2L}{L + x}\right) \tag{2}$$

4 Numerical analysis

- Rod with uniform cross section,
  - Weak form

$$EA_0 \int_{x_A}^{x_B} \frac{\partial w}{\partial x} \frac{\partial u_h^e}{\partial x} dx = wEA_0 \frac{\partial u_h^e}{\partial x}\Big|_{x_B}^{x_A} \tag{3}$$

  - Guass quadrature

$$k_{ij}^e = \sum_{k=1}^{n} w_i \left(\frac{x_b - x_a}{2} \phi \left[\left(\frac{x_b - x_a}{2} \right) \xi + \left(\frac{x_b + x_a}{2} \right) \right] \right) \tag{4}$$

  where,

$$\phi(x) = \frac{EA_0}{l_e^2}$$

$$w = \{1, 1\}$$

$$\xi = \left\{ \frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}} \right\}$$

- Rod with non-uniform cross section,
  - Weak form

$$EA_0 \int_{x_A}^{x_B} \left(1 + \frac{x}{L}\right) \frac{\partial w}{\partial x} \frac{\partial u_h^e}{\partial x} dx = wEA_0 \left(1 + \frac{x}{L}\right) \frac{\partial u_h^e}{\partial x}\Big|_{x_B}^{x_A} \tag{5}$$

  - Guass quadrature

$$k_{ij}^e = \sum_{k=1}^{n} w_i \left(\frac{x_b - x_a}{2} \phi \left[\left(\frac{x_b - x_a}{2} \right) \xi + \left(\frac{x_b + x_a}{2} \right) \right] \right) \tag{6}$$

  where,

$$\phi(x) = \frac{EA_0}{l_e^2} \left(1 + \frac{x}{L}\right)$$

$$w = \{2\}$$

$$\xi = \{0\}$$

5 Graph plots

5.a Uniform cross section

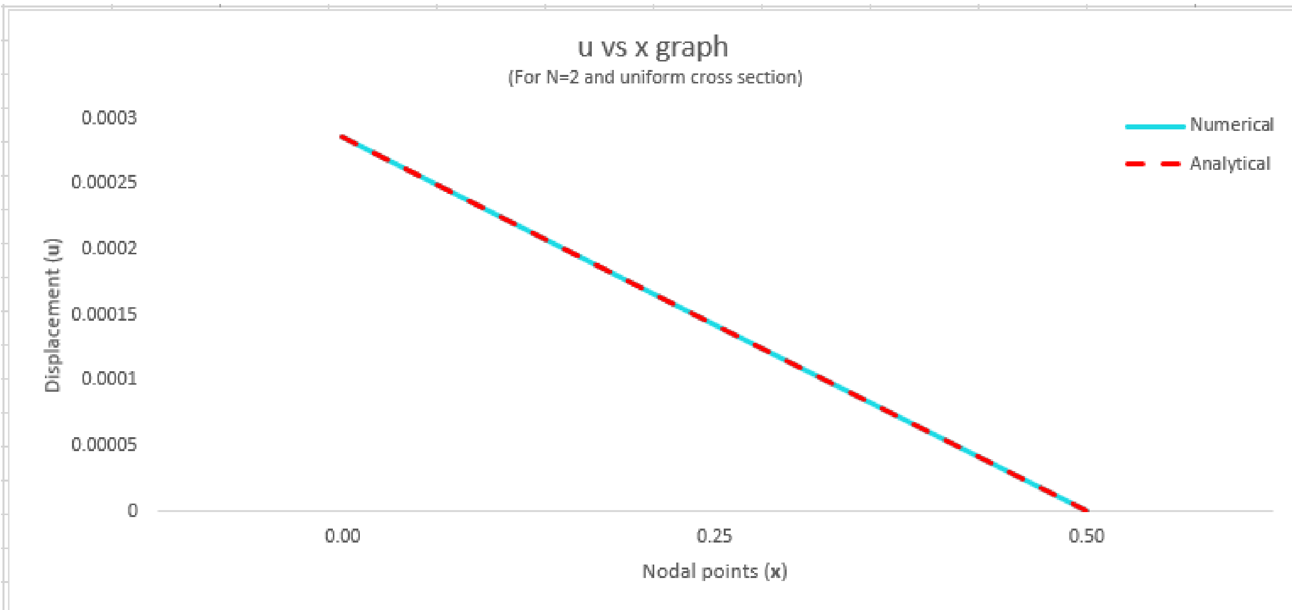


Figure 1: For N = 2

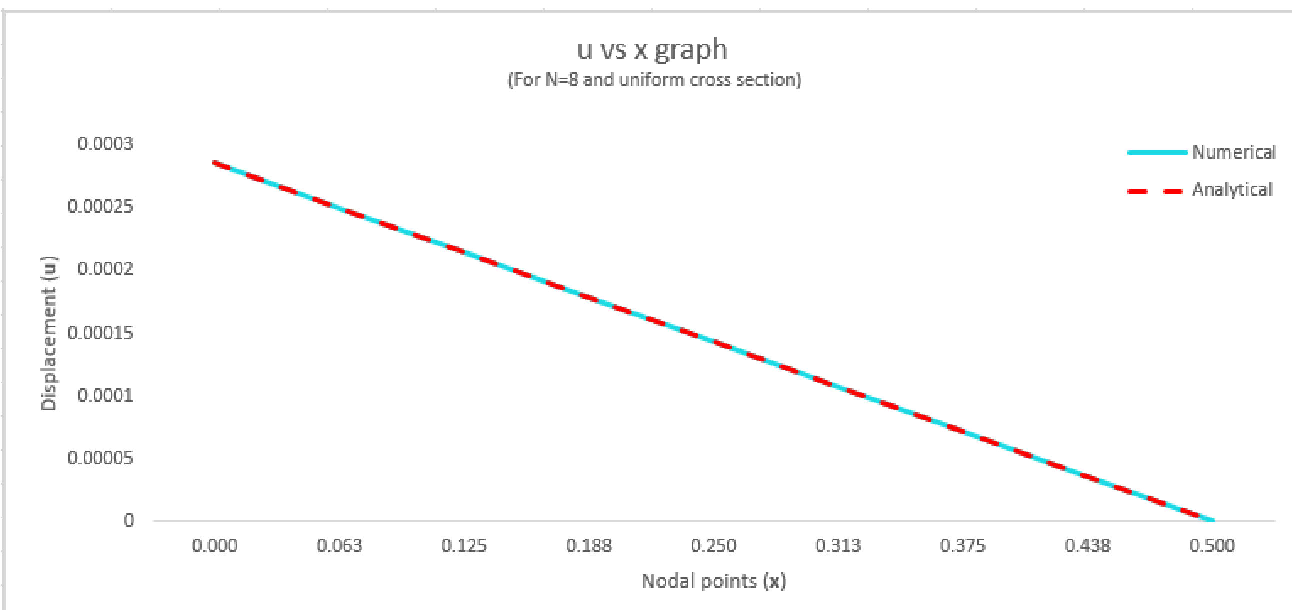


Figure 2: For N = 8

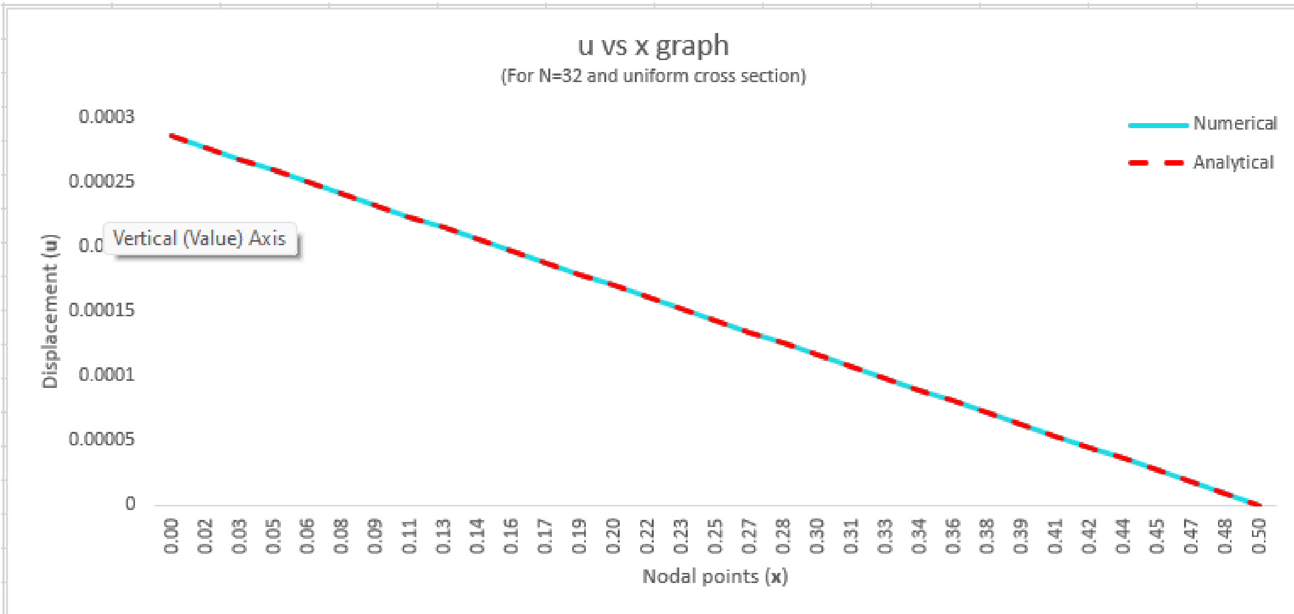


Figure 3: For N = 32

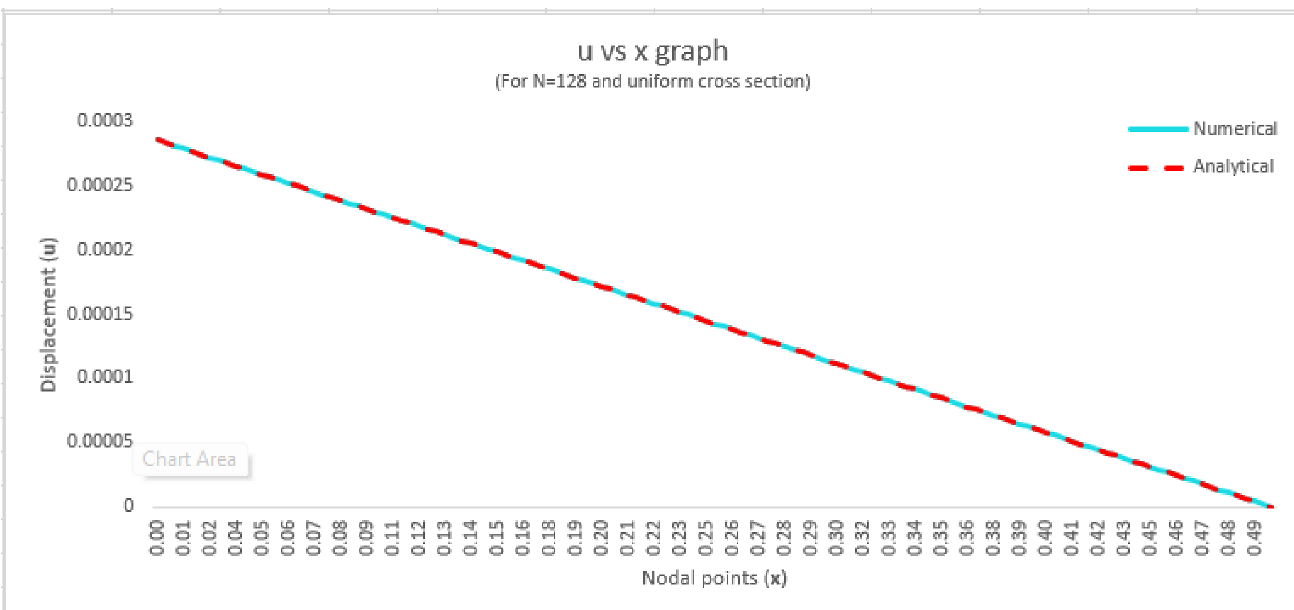


Figure 4: For N = 128

5.b Non-uniform cross section

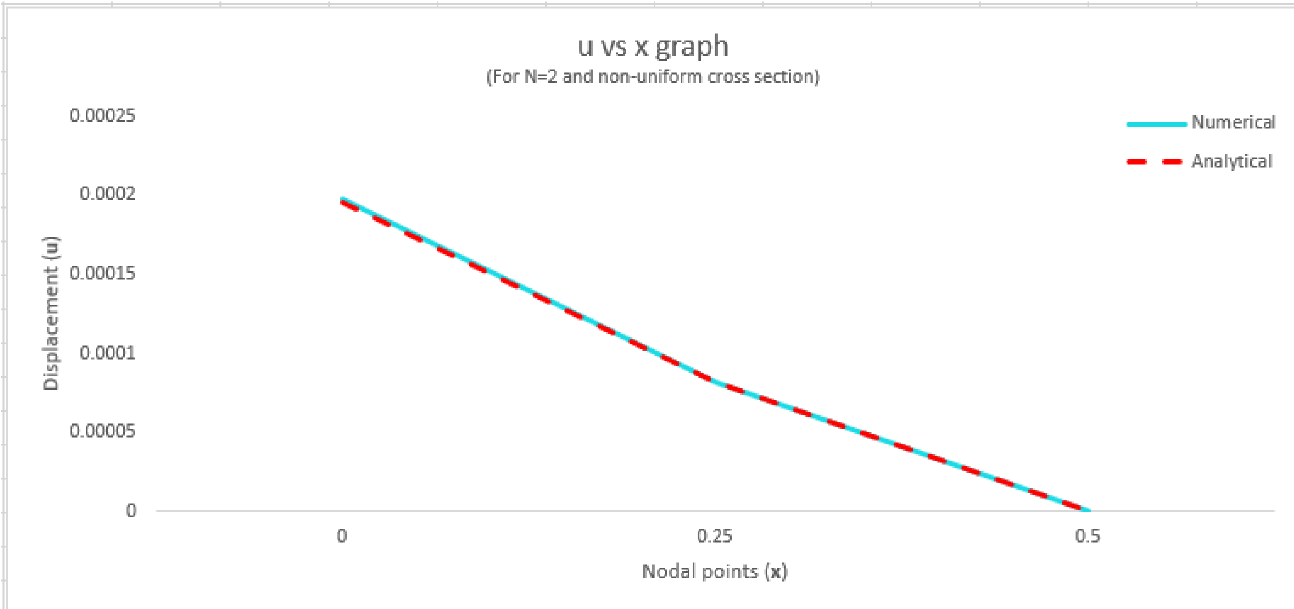


Figure 5: For N = 2

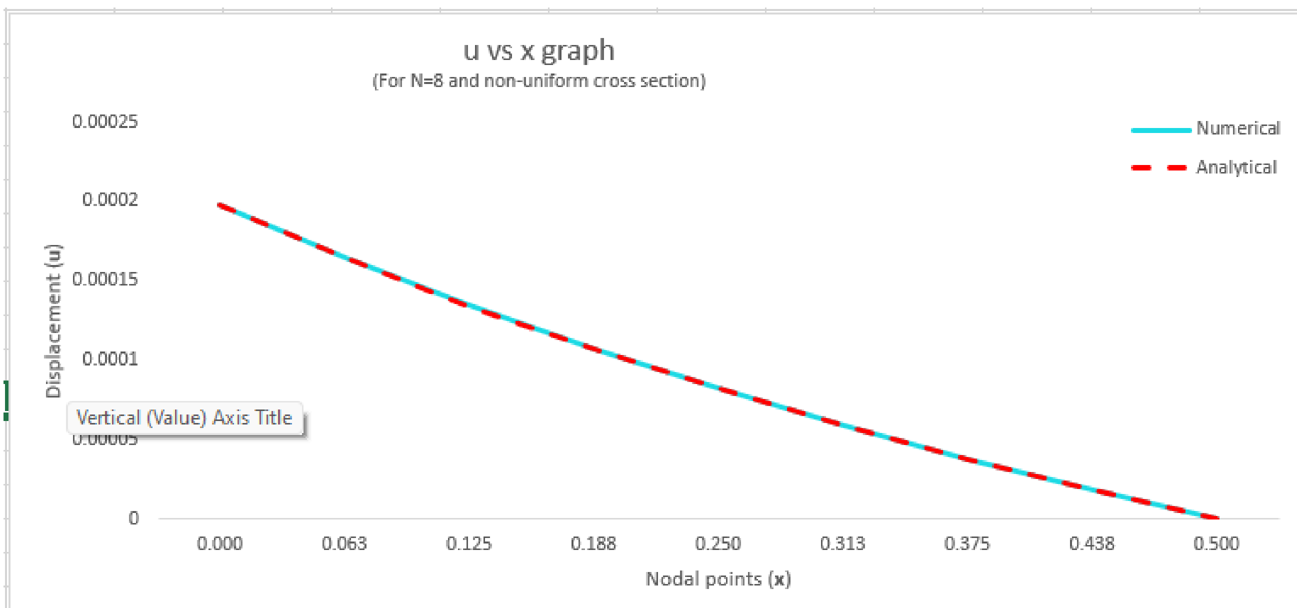


Figure 6: For N = 8

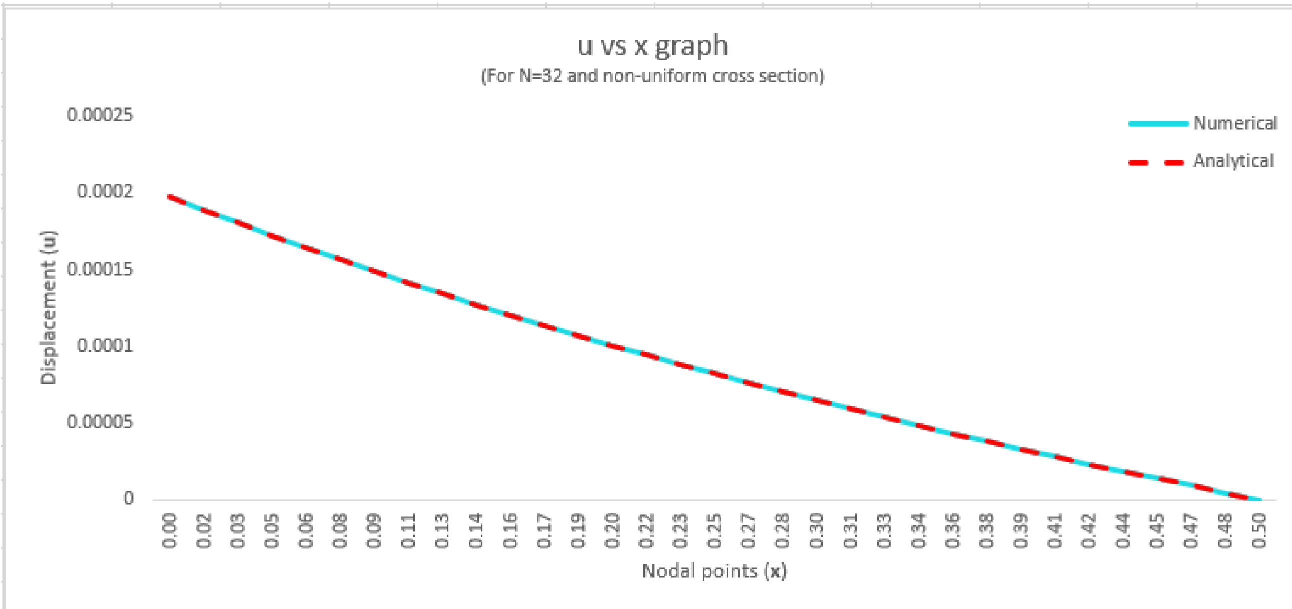


Figure 7: For N = 32

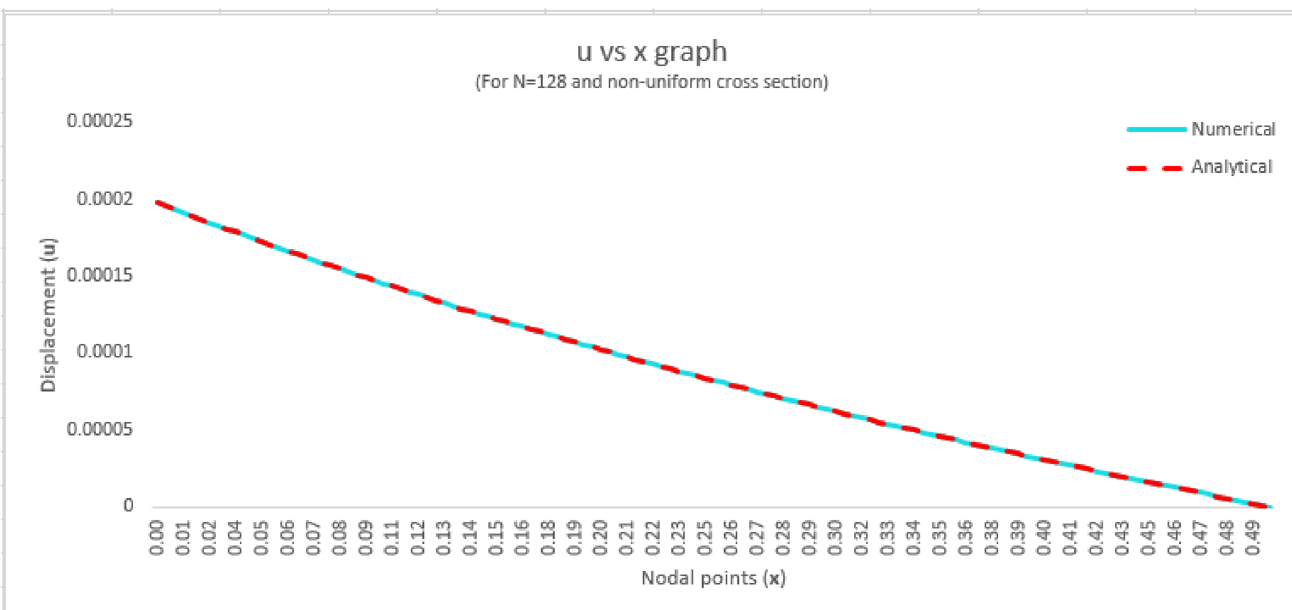


Figure 8: For N = 128

6 Error analysis

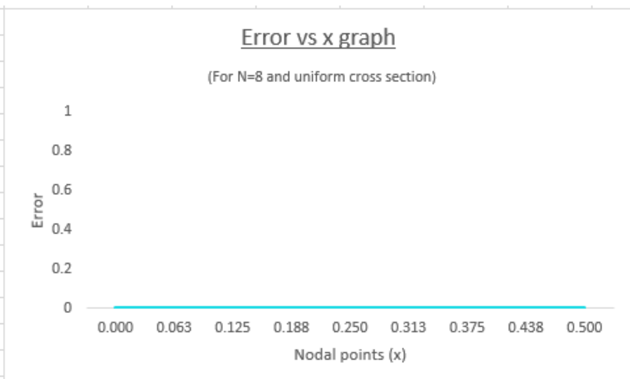
6.a Table

N	Uniform Area	Non-uniform Area
2	0	1.26852e-06
8	0	6.62244e-08
32	0	3.93794e-09
128	0	4.38469e-10

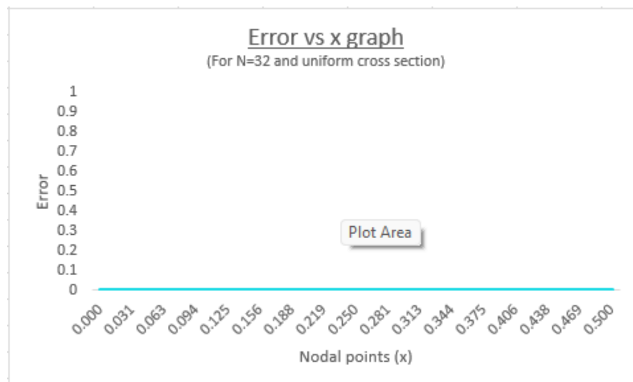
Table 1: RMS error

6.b Plots

- For uniform cross section

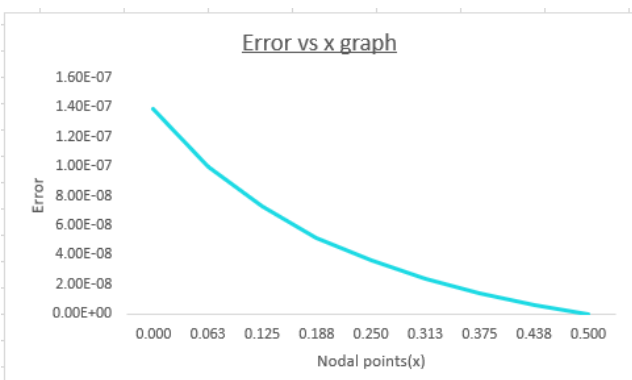


(a) For N = 8

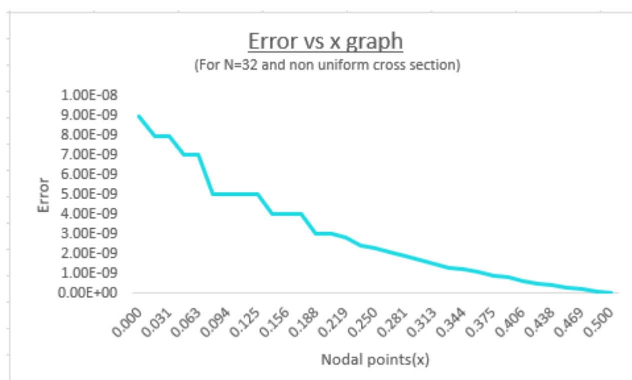


(b) For N = 32

- For non-uniform cross section



(a) For N = 8



(b) For N = 32

7 Conclusions

We were able to deduce the following points from the above analyses,

- The displacements of nodal points are almost same for analytical and numerical approach, and we can have error as almost 0.
- We get almost 0 error as FEM approximation and analytical approximations are linear in nature.
- We can infer the above conclusions with plots of uniform cross section as both curve overlap throughout.
- The error in the nodal displacements decreases and accuracy of FEM increases as the number of elements increases.
- As the number of nodal points increases the curves for analytical numerical almost fuse into another, which shows that accuracy of FEM increases.
- We can infer the above conclusions with plots of non-uniform cross section as both curve start to overlap as N increases.
- As we see through the norms from x=0 to x=L it decreases, because given that x = L is a boundary condition, and as we move away from it, the errors in approximations start to build up.

8 Appendix

8.a Makefile

```
CXX = g++
CXFLAGS =-g -std=c++11
CFLAGS=-I ../../../../nfs_home/nikhilh/eigen-3.3.9

ANALYTICAL = src/analytical/src/
NUMERICAL = src/numerical/src/
SOLUTION = src/generate_solution/src/
BINARY = bin/
OUTPUTDIR = outputs_csv/
CLEANOBJ = $(BINARY)


sdsc: checkbin analytical numerical solution

analytical: $(ANALYTICAL)analytical1.cpp $(ANALYTICAL)analytical2.cpp
	$(CXX) $(ANALYTICAL)analytical1.cpp $(CXFLAGS) $(CFLAGS) -o $(BINARY)
	    analytical1.out
	$(CXX) $(ANALYTICAL)analytical2.cpp $(CXFLAGS) $(CFLAGS) -o $(BINARY)
	    analytical2.out


numerical: $(NUMERICAL)numerical1.cpp $(NUMERICAL)numerical2.cpp
	$(CXX) $(NUMERICAL)numerical1.cpp $(CXFLAGS) $(CFLAGS) -o $(BINARY)
	    numerical1.out
	$(CXX) $(NUMERICAL)numerical2.cpp $(CXFLAGS) $(CFLAGS) -o $(BINARY)
	    numerical2.out


solution: $(SOLUTION)solution_part1.cpp $(SOLUTION)solution_part2.cpp
	$(CXX) $(SOLUTION)solution_part1.cpp $(CXFLAGS) $(CFLAGS) -o $(BINARY)
	    solution_part1.out
```

```
	$(CXX) $(SOLUTION)solution_part2.cpp $(CXFLAGS) $(CFLAGS) -o $(BINARY)
	    solution_part2.out

checkbin:
	if [ ! -d $(BINARY) ]; then mkdir $(BINARY); fi

checkoutputdir:
	if [ ! -d $(OUTPUTDIR) ]; then mkdir $(OUTPUTDIR); fi

clean:
	rm $(CLEANOBJ)

team:
	@echo 200010056 - Vidyasagar Singadi
	@echo 200030056 - Siddharth Shankar
```

8.b runme

```
make -f Makefile
echo "Enter the problem number (1 or 2)"
read problemNo
echo "Enter the P (load) value"
read pValue
echo "Enter the A0 (constant in area function) value"
read a0Value
echo "Enter the L(length of rod) value"
read lValue
echo "Enter the E (Young's Modulus) value"
read eValue
echo "Enter the N (no. of elements for FEM) value"
read nValue

if [[ $problemNo -eq 1 ]]
then
  bin/analytical1.out $pValue $a0Value $lValue $eValue $nValue
  bin/numerical1.out $pValue $a0Value $lValue $eValue $nValue
  mv "analytical1.csv" outputs_csv/
  mv "numerical1.csv" outputs_csv/
  bin/solution_part1.out
  cp solution_part1.csv file.txt
  mv "solution_part1.csv" outputs_csv/
  echo "file.txt created successfully for problem no. 1 !!"
elif [[ $problemNo -eq 2 ]]
then
  bin/analytical2.out $pValue $a0Value $lValue $eValue $nValue
  bin/numerical2.out $pValue $a0Value $lValue $eValue $nValue
  mv "analytical2.csv" outputs_csv/
  mv "numerical2.csv" outputs_csv/
  bin/solution_part2.out
  cp solution_part2.csv file.txt
  mv "solution_part2.csv" outputs_csv/
  echo "file.txt created successfully for problem no. 2 !!"
else
  echo "Error: Please enter a valid problem number(1 or 2)"
fi
```

8.c analytical.h

```
#ifndef ANALYTICAL_H
#define ANALYTICAL_H
#include <cstdio>
#include <iostream>

//#pragma once

/**
 * Domain class containing variables : P, N, A0, L, E
**/

template <typename T>
class Domain {
private:
        T P;
    T A0;
    T L;
    T E;
        int N;

public:
    //constructor function
        Domain(T P_arg, T A0_arg, T L_arg, T E_arg, int N_arg){
     P=P_arg;
    A0=A0_arg;
    L=L_arg;
    E=E_arg;
    N=N_arg;
    }
    // getter functions to return required private variables
    T getP(){
    return P;
    }
    T getA0(){
     return A0;
    }
    T getL(){
     return L;
    }
    T getE(){
    return E;
    }
    T getN(){
    return N;
    }
    // method to print all the private variables
    void print(){
    std::cout <<"P value: "<<P<<std::endl;
    std::cout <<"A0 value: "<<A0<<std::endl;
    std::cout <<"L value: "<<L<<std::endl;
    std::cout <<"E value: "<<E<<std::endl;
    std::cout <<"N value: "<<N<<std::endl;
    }
};
```

```
#endif
```

8.d analytical1.cpp

```
#include "../inc/analytical.h"
#include <cstdio>
#include <string>
#include<bits/stdc++.h>
#include <iostream>
#include <fstream>

/**
 * function to write the displacement (u) values for analytical solution
 * to a csv file called 'analytical1.csv'
**/
void write_csv(std::vector<double> vals)
{

    // Create an output filestream object
    std::fstream fout;
    fout.open("analytical1.csv", std::ios::out);

    // Send data to the stream
    //std::cout<<"vals size: "<<vals.size()<<"\n";

    for (int i = 0; i < vals.size(); ++i)
    {
        if(i==vals.size()-1){
            fout<<0;
        }else{
            fout << vals.at(i) << "\n";
        }
    }

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


/**
 * main function accepts command line arguments(P, A0, L, E, N)
 * Also, it calculates the value of displacement for all elements and write it to a
     csv file
**/
int main(int argc, char *argv[])
{
    //creating an object of template class Domain
    Domain<double> myDomain(atof(argv[1]), atof(argv[2]), atof(argv[3]), atof(argv
        [4]), atof(argv[5]));

    if (argc != 6)
    {
        std::cout<<"No. of arguments recieved does not match with that required by 
            program"<<std::endl;
        return 1;
```

```
    }

    //print the arguments received
    std::cout<<"\n\n————————Values received by analytical1.out————————\
        n";
    myDomain.print();

    // For N elements, total no. of nodes will be N+1

    //creating a N+1 by 1 u values array
    std::vector<double> u_array(myDomain.getN() + 1);

    //using the formula derived for displacement i.e. u(x) as mentioned in report,
        we find displacements
    for(int i=0; i<=myDomain.getN() ; i++){
    u_array[i]=((1-i/myDomain.getN())*myDomain.getP()*myDomain.getL())/(myDomain.
        getA0()*myDomain.getE());
    }

    //print the displacements(u_array)

//      for(int i=0; i<=myDomain.getN(); i++){
//          std::cout<<"u"<<i<<": "<<u_array[i]<<std::endl;
//      }

    // write the displacements calculated to 'analytical1.csv' file
    write_csv(u_array);

    return 0;
}
```

8.e analytical2.cpp

```
#include "../inc/analytical.h"
#include <cstdio>
#include <bits/stdc++.h>
#include <iostream>
#include <cmath>
#include <fstream>

/**
 * function to write the displacement (u) values for analytical solution
 * to a csv file called 'analytical2.csv'
**/
void write_csv(std::vector<double> vals)
{

    // Create an output filestream object
    std::fstream fout;
    fout.open("analytical2.csv", std::ios::out);


    // Send data to the stream
    //std::cout<<"vals size: "<<vals.size()<<"\n";

    for (int i = 0; i < vals.size(); ++i)
```

```
    {
        if(i==vals.size()-1){
            fout<<0;
        }else{
            fout << vals.at(i) << "\n";
        }
    }

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

/**
 * main function accepts command line arguments(P, A0, L, E, N)
 * Also, it calculates the value of displacement for all elements and write it to a
     csv file
**/
int main(int argc, char *argv[])
{
     //creating an object of template class Domain
    Domain<double> myDomain(atof(argv[1]), atof(argv[2]), atof(argv[3]), atof(argv
        [4]), atof(argv[5]));

    if (argc != 6)
    {
        std::cout << "No. of arguments recieved does not match with that required 
            by program" << std::endl;
        return 1;
    }

    // print the arguments received
    std::cout<<"\n\n------------------Values received by analytical2.out-----------------\
       n";
    myDomain.print();

    // For N elements, total no. of nodes will be N+1

    //creating a N+1 by 1 u values array
    std::vector<double> u_array(myDomain.getN() + 1);

    //using the formula derived for displacement i.e. u(x) as mentioned in report,
       we find displacements
    for (int i = 0; i <= myDomain.getN(); i++)
    {
        u_array[i] = ((myDomain.getP() * myDomain.getL()) / (myDomain.getE() *
            myDomain.getA0())) * (log((2 * myDomain.getL()) / (myDomain.getL() + (i
            * myDomain.getL() / myDomain.getN()))));
    }

    //print the displacements(u_array)
    // for (int i = 0; i <= myDomain.getN(); i++)
    // {
    //      std::cout << u_array[i] << std::endl;
    // }

    // write the displacements calculated to 'analytical1.csv' file
```

```
    write_csv(u_array);

    return 0;
}
```

8.f numerical.h

```
#ifndef NUMERICAL_H
#define NUMERICAL_H
#include <cstdio>
#include <iostream>

//#pragma once
/**
 * Domain class containing variables : P, N, A0, L, E
**/

template <typename T>
class Domain {
private:
        T P;
    T A0;
    T L;
    T E;
        int N;

public:
    //constructor function
        Domain(T P_arg, T A0_arg, T L_arg, T E_arg, int N_arg){
     P=P_arg;
    A0=A0_arg;
    L=L_arg;
    E=E_arg;
    N=N_arg;
    }
    // getter functions to return required private variables
    T getP(){
    return P;
    }
    T getA0(){
     return A0;
    }
    T getL(){
     return L;
    }
    T getE(){
    return E;
    }
    T getN(){
    return N;
    }
    // method to print all the private variables
    void print(){
    std::cout <<"P value: "<<P<<std::endl;
    std::cout <<"A0 value: "<<A0<<std::endl;
    std::cout <<"L value: "<<L<<std::endl;
```

```
    std::cout <<"E value: "<<E<<std::endl;
    std::cout <<"N value: "<<N<<std::endl;
    }
};

#endif
```

8.g numerical1.cpp

```
#include "../inc/numerical.h"
#include <cstdio>
#include<bits/stdc++.h>
#include <iostream>
#include <Eigen/Dense>
#include <fstream>

using namespace Eigen;

/**
 * function to write the displacement (u) values for analytical solution
 * to a csv file called 'numerical1.csv'
**/
void write_csv(Matrix<double, Dynamic,Dynamic> vals)
{

    // Create an output filestream object
    std::fstream fout;
    fout.open("numerical1.csv", std::ios::out);

   // Send data to the stream
    for (int i = 0; i < vals.size(); ++i)
    {
        fout << vals(i) << "\n";
    }

    // for last u , it's displacement is 0
    fout << 0;
    // Close the file
    fout.close();
}

/**
 * function to remove last row from a given matrix (required for removing
     singularity whlie finding inverse)
**/
void removeRow(Matrix<double, Dynamic,Dynamic>& matrix, unsigned int rowToRemove)
{
    unsigned int numRows = matrix.rows()-1;
    unsigned int numCols = matrix.cols();

    if( rowToRemove < numRows )
        matrix.block(rowToRemove,0,numRows-rowToRemove,numCols) = matrix.block(
            rowToRemove+1,0,numRows-rowToRemove,numCols);

    matrix.conservativeResize(numRows,numCols);
}
```

```
/**
 * function to remove last column from a given matrix (required for removing
     singularity whlie finding inverse)
**/
void removeColumn(Matrix<double, Dynamic,Dynamic>& matrix, unsigned int colToRemove
    )
{
    unsigned int numRows = matrix.rows();
    unsigned int numCols = matrix.cols()-1;

    if( colToRemove < numCols )
        matrix.block(0,colToRemove,numRows,numCols-colToRemove) = matrix.block(0,
            colToRemove+1,numRows,numCols-colToRemove);

    matrix.conservativeResize(numRows,numCols);
}


/**
 * Function to calculate the integral(I) for given parameters using Gauss
     Quadrature Method
 * For Numerical solution of Problem 1(constant area), we have used 2-point gauss
     quadrature
**/
double gaussQuadrature(int diff1, int diff2, double E, double A0, int N, double L){
        // defining phi(x), the function whose integral needs to be calculated
        double phi = (diff1*diff2)*(E*A0*N*N)/(L*L);
        // Initialise value of Integral of above function(phi(x)) with 0
        double I = 0;
        // 2-point gauss quadrature
        int n = 2;
        // lower limit of integral
        double a = 0;
        // upper limit of integral
        double b = L/N;
        // alpha array is same as the weights array (w)
        // alpha1 = 1 and alpha2 = 1 for n=2
        int alpha[] = {1, 1};
        // x1 = -0.5773502691896257  and x2 = 0.5773502691896257
        for(int i=0; i<n; i++){
        // phi(x) will be a constant polynomial for problem 1 (constant area), so
            the formula for I reduces to the following
            I = I + alpha[i]*(((b-a)/2)*phi);
        }
        return I;
    }



/**
 * main function accepts command line arguments(P, A0, L, E, N)
 * Also, it calculates the value of displacement for all elements and write it to a
      csv file
 *
**/
```

```
int main(int argc, char *argv[])
{
    //creating an object of template class Domain
    Domain<double> myDomain(atof(argv[1]), atof(argv[2]), atof(argv[3]), atof(argv
        [4]), atof(argv[5]));

    if (argc != 6)
    {
        std::cout<<"No. of arguments recieved does not match with that required by
            program"<<std::endl;
        return 1;
    }

    //print the arguments received
    std::cout<<"\n\n-----------------Values received by numerical1.out-----------------\n
        ";
    myDomain.print();

    // For N elements, total no. of nodes will be N+1

    /*
    Using the Eigen library (present at /nfs_home/nikhilh/eigen-3.3.9), we
    create and calculate stiffness matrix, displacement vector and force vector
    */

    //creating a N+1 by N+1 global stiffness matrix called k_matrix , initiaised to
        0
    Matrix<double, Dynamic,Dynamic> k_matrix=Matrix<double, Dynamic,Dynamic>::Zero(
        myDomain.getN() + 1, myDomain.getN() + 1);

    //creating a N by 4 elemental stiffness matrix called elemental_matrix ,
        initiaised to 0
    Matrix<double, Dynamic,Dynamic> elemental_matrix=Matrix<double, Dynamic,Dynamic
        >::Zero(myDomain.getN(), 4);

    //creating a N+1 by 1 displacement vector(u) called u_vector , initiaised to 0
    Matrix<double, Dynamic,Dynamic> u_vector=Matrix<double, Dynamic,Dynamic>::Zero(
        myDomain.getN() + 1, 1);

    //creating a N+1 by 1 force vector called f_vector , initiaised to 0
    Matrix<double, Dynamic,Dynamic> f_vector=Matrix<double, Dynamic,Dynamic>::Zero(
        myDomain.getN() + 1, 1);


    //first element of the f-vector should be equal to P and rest all elements
        should be 0
    f_vector(0)=myDomain.getP();


    /*
    calculating the values for each element and storing it in it's elemental matrix
    here i denotes the ith element
    ith row contains 4 columns that corresponds to the ith's element 11,12,21 and
        22 index K values
    */
    for(int i=0; i < myDomain.getN(); i++){
```

```
        elemental_matrix(i, 0) = gaussQuadrature(1, 1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL());
        elemental_matrix(i, 1) = gaussQuadrature(1, -1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL());
        elemental_matrix(i, 2) = gaussQuadrature(-1, 1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL());
        elemental_matrix(i, 3) = gaussQuadrature(-1, -1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL());
    }

    /*
    here i denotes the ith element
    we assemble the elemental stiffness matrices into a global stiffness matrix
    */
    for(int i = 0; i<myDomain.getN(); i++){
        k_matrix(i, i) += elemental_matrix(i, 0);
        k_matrix(i, i+1) += elemental_matrix(i, 1);
        k_matrix(i+1, i) += elemental_matrix(i, 2);
        k_matrix(i+1, i+1) += elemental_matrix(i, 3);
    }


    //print elemental_matrix
    //std::cout << elemental_matrix << std::endl<<std::endl<<std::endl;

    //print k_matrix
    //std::cout << k_matrix << std::endl<<std::endl<<std::endl;

    //print u_vector
    //std::cout << u_vector << std::endl<<std::endl<<std::endl;


    //print f_vector
    //std::cout << f_vector << std::endl<<std::endl<<std::endl;


   //removing last row and last column of k_matrix
    removeRow(k_matrix, myDomain.getN());
    removeColumn(k_matrix, myDomain.getN());

    //removing last row of u_vector
    removeRow(u_vector, myDomain.getN());

    //removing last row of f_vector
    removeRow(f_vector, myDomain.getN());



     /*
    calculating u_vector from KU=F matrix equation where K is global stiffness
        matrix,
    U is global displacement vector and F is global force vector
    */
    u_vector=k_matrix.inverse()*f_vector;

     //print u_vector
    //std::cout << u_vector << std::endl<<std::endl<<std::endl;
```

```
    //write the displacements calculated to 'numerical1.csv' file
    write_csv(u_vector);

    return 0;
}
```

8.h numerical2.cpp

```
#include "../inc/numerical.h"
#include <cstdio>
#include<bits/stdc++.h>
#include <iostream>
#include <Eigen/Dense>
#include <fstream>

using namespace Eigen;

/**
 * function to write the displacement (u) values for analytical solution
 * to a csv file called 'numerical2.csv'
**/
void write_csv(Matrix<double, Dynamic,Dynamic> vals)
{

    // Create an output filestream object
    std::fstream fout;
    fout.open("numerical2.csv", std::ios::out);

    // Send data to the stream
    for (int i = 0; i < vals.size(); ++i)
    {
        fout << vals(i) << "\n";
    }

    //for last u , it's displacement is 0
    fout << 0;

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

/**
 * function to remove last row from a given matrix (required for removing
    singularity whlie finding inverse)
**/
void removeRow(Matrix<double, Dynamic,Dynamic>& matrix, unsigned int rowToRemove)
{
    unsigned int numRows = matrix.rows()-1;
    unsigned int numCols = matrix.cols();

    if( rowToRemove < numRows )
        matrix.block(rowToRemove,0,numRows-rowToRemove,numCols) = matrix.block(
            rowToRemove+1,0,numRows-rowToRemove,numCols);

    matrix.conservativeResize(numRows,numCols);
```

```
}

/**
 * function to remove last column from a given matrix (required for removing
     singularity whlie finding inverse)
**/
void removeColumn(Matrix<double, Dynamic,Dynamic>& matrix, unsigned int colToRemove
    )
{
    unsigned int numRows = matrix.rows();
    unsigned int numCols = matrix.cols()-1;

    if( colToRemove < numCols )
        matrix.block(0,colToRemove,numRows,numCols-colToRemove) = matrix.block(0,
            colToRemove+1,numRows,numCols-colToRemove);

    matrix.conservativeResize(numRows,numCols);
}

// xA - i then xB - i+1
/**
 * Function to calculate the integral(I) for given parameters using Gauss
     Quadrature Method
 * For Numerical solution of Problem 2(variable area), we have used 1-point gauss
     quadrature
 * if xA = i then xB = i+1
**/
double gaussQuadrature(int diff1, int diff2, double E, double A0, int N, double L,
   double xA, double xB){
        // Initialise value of Integral of above function(phi(x)) with 0
        double I = 0;
        // 1-point gauss quadrature
        int n = 1;
        // lower limit of integral
        double a = xA*L/N;
        // upper limit of integral
        double b = xB*L/N;
        // defining phi(x), the function whose integral needs to be calculated
        double phi = (1+(a+b)/(2*L))*(diff1*diff2)*(E*A0*N*N)/(L*L);
        // alpha array is same as the weights array (w)
        // alpha1 = 2 for n=1
        int alpha[] = {2};
        // x1 = 0, so the formula for I reduces to the following
        for(int i=0; i<n; i++){
            I = I + alpha[i]*(((b-a)/2)*phi);
        }
        return I;
    }


/**
 * main function accepts command line arguments(P, A0, L, E, N)
 * Also, it calculates the value of displacement for all elements and write it to a
      csv file
 *
**/
```

```
int main(int argc, char *argv[])
{
    //creating an object of template class Domain
    Domain<double> myDomain(atof(argv[1]), atof(argv[2]), atof(argv[3]), atof(argv
        [4]), atof(argv[5]));

    if (argc != 6)
    {
        std::cout<<"No. of arguments recieved does not match with that required by
            program"<<std::endl;
        return 1;
    }

    //print the arguments received
    std::cout<<"\n\n----------------Values received by numerical2.out----------------\n
        ";
    myDomain.print();

   // For N elements, total no. of nodes will be N+1

    /*
    Using the Eigen library (present at /nfs_home/nikhilh/eigen-3.3.9), we
    create and calculate stiffness matrix, displacement vector and force vector
    */

   //creating a N+1 by N+1 global stiffness matrix called k_matrix , initiaised to
      0
    Matrix<double, Dynamic,Dynamic> k_matrix=Matrix<double, Dynamic,Dynamic>::Zero(
        myDomain.getN() + 1, myDomain.getN() + 1);

    //creating a N by 4 elemental stiffness matrix called elemental_matrix ,
        initiaised to 0
    Matrix<double, Dynamic,Dynamic> elemental_matrix=Matrix<double, Dynamic,Dynamic
        >::Zero(myDomain.getN(), 4);

    //creating a N+1 by 1 displacement vector(u) called u_vector , initiaised to 0
    Matrix<double, Dynamic,Dynamic> u_vector=Matrix<double, Dynamic,Dynamic>::Zero(
        myDomain.getN() + 1, 1);

    //creating a N+1 by 1 force vector called f_vector , initiaised to 0
    Matrix<double, Dynamic,Dynamic> f_vector=Matrix<double, Dynamic,Dynamic>::Zero(
        myDomain.getN() + 1, 1);


    //first element of the f-vector should be equal to P and rest all elements
        should be 0
    f_vector(0)=myDomain.getP();

     /*
    calculating the values for each element and storing it in it's elemental matrix
    here i denotes the ith element
    ith row contains 4 columns that corresponds to the ith's element 11,12,21 and
        22 index K values
    if xA = i then xB = i+1
    */
    for(int i=0; i < myDomain.getN(); i++){
```

```
        elemental_matrix(i, 0) = gaussQuadrature(1, 1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL(), i, i+1);
        elemental_matrix(i, 1) = gaussQuadrature(-1, 1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL(), i, i+1);
        elemental_matrix(i, 2) = gaussQuadrature(1, -1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL(), i, i+1);
        elemental_matrix(i, 3) = gaussQuadrature(-1, -1, myDomain.getE(), myDomain.
            getA0(), myDomain.getN(), myDomain.getL(), i, i+1);
    }

     /*
    here i denotes the ith element
    we assemble the elemental stiffness matrices into a global stiffness matrix
    */
    for(int i = 0; i<myDomain.getN(); i++){
        k_matrix(i, i) += elemental_matrix(i, 0);
        k_matrix(i, i+1) += elemental_matrix(i, 1);
        k_matrix(i+1, i) += elemental_matrix(i, 2);
        k_matrix(i+1, i+1) += elemental_matrix(i, 3);
    }


    //print elemental_matrix
    //std::cout << elemental_matrix << std::endl<<std::endl<<std::endl;

    //print k_matrix
    //std::cout << k_matrix << std::endl<<std::endl<<std::endl;

    //print u_vector
    //std::cout << u_vector << std::endl<<std::endl<<std::endl;


    //print f_vector
    //std::cout << f_vector << std::endl<<std::endl<<std::endl;

    //removing last row and last column of k_matrix
    removeRow(k_matrix, myDomain.getN());
    removeColumn(k_matrix, myDomain.getN());

    //removing last row of u_vector
    removeRow(u_vector, myDomain.getN());

    //removing last row of f_vector
    removeRow(f_vector, myDomain.getN());

    /*
    calculating u_vector from KU=F matrix equation where K is global stiffness
        matrix,
    U is global displacement vector and F is global force vector
    */
    u_vector=k_matrix.inverse()*f_vector;

    //print u_vector
    //std::cout << u_vector << std::endl<<std::endl<<std::endl;

    //write the displacements calculated to 'numerical2.csv' file
```

```
    write_csv(u_vector);
    return 0;
}
```

8.i solution_part1.cpp

```
#include <cstdio>
#include<bits/stdc++.h>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <cmath>


/**
 * function to write the node numbers, analytical solution, numerical solution and
     error(analytical-numerical)
 * to a csv file called 'solution_part1.csv' for problem 1
**/
void write_csv(int size, std::vector<double> analytical, std::vector<double>
   numerical, std::vector<double> error_vals, double rms_error)
{

    // Create an output filestream object
    std::fstream fout;
    fout.open("solution_part1.csv", std::ios::out);

     fout<<"Node number, Analytical Solution, Numerical Solution, Error"<<"\n";
    // Send data to the stream
    for (int i = 0; i < size; ++i)
    {
        fout<<i<<","<<analytical[i]<<","<<numerical[i]<<","<<error_vals[i]<<"\n";
    }

    fout<<"\n"<<"RMS Error: "<<rms_error;
    // Close the file
    fout.close();
}


/**
 * function to find the root mean square error from the vector called error_vals
**/
double findRMS(std::vector<double> error_vals){
    double square = 0;
    double mean = 0.0, root = 0.0;

    // Calculate square.
    for (int i = 0; i < error_vals.size(); i++) {
        square += pow(error_vals[i], 2);
    }

    // Calculate Mean.
    mean = (square / (double)(error_vals.size()));
```

```cpp
    // Calculate Root.
    root = sqrt(mean);

    return root;
}


/**
 * main function uses the fstream library to read and write to csv files
 * Also, it calculates the node wise error values and also the overall RMS error
**/
int main(int argc, char *argv[])
{

    std::fstream analytical1;
    std::fstream numerical1;

    analytical1.open("outputs_csv/analytical1.csv", std::ios::in);

    numerical1.open("outputs_csv/numerical1.csv", std::ios::in);

    std::vector<std::string> analytical1_rows_strings;
    std::vector<std::string> numerical1_rows_strings;
    std::string line, temp;

    //reading analytical solution (displacement values) for problem 1
    while(!analytical1.eof()){
        getline(analytical1, line);
        analytical1_rows_strings.push_back(line);
    }

    //reading numerical solution (displacement values) for problem 1
    while(!numerical1.eof()){
        getline(numerical1, line);
        numerical1_rows_strings.push_back(line);
    }

        analytical1.close();
        numerical1.close();

    // this will remove double quotes from the string if present
    for (auto &s : analytical1_rows_strings)
        s.erase(remove(s.begin(), s.end(), '\"'), s.end());

    // this will remove double quotes from the string if present
    for (auto &s : numerical1_rows_strings)
        s.erase(remove(s.begin(), s.end(), '\"'), s.end());


    std::vector<double> analytical1_rows_doubles;
    std::vector<double> numerical1_rows_doubles;

    //converting string values to double for error calculation
    for (auto it = analytical1_rows_strings.begin(); it != analytical1_rows_strings
        .end(); it++) {
        analytical1_rows_doubles.push_back(stod(*it));
```

```
    }

    //converting string values to double for error calculation
    for (auto it = numerical1_rows_strings.begin(); it != numerical1_rows_strings.
        end(); it++) {
         numerical1_rows_doubles.push_back(stod(*it));
    }

    //storing error values(numerical - analytical)
    // analytical_rows_doubles.size() == numerical1_rows_doubles.size()
    int size=analytical1_rows_doubles.size();
    std::vector<double> error_vals;

    for(int i=0; i < size; i++){
        error_vals.push_back(analytical1_rows_doubles[i]-numerical1_rows_doubles[i
            ]);
    }

    //finding the Root Mean Square Error
   double rms_error=findRMS(error_vals);

    // write the final solution to 'solution_part1.csv' file
    write_csv(size, analytical1_rows_doubles, numerical1_rows_doubles, error_vals,
        rms_error);

    return 0;
}
```

8.j solution_part2.cpp

```
#include <cstdio>
#include<bits/stdc++.h>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <sstream>


/**
 * function to write the node numbers, analytical solution, numerical solution and
     error(analytical-numerical)
 * to a csv file called 'solution_part2.csv' for problem 2
**/
void write_csv(int size, std::vector<double> analytical, std::vector<double>
    numerical, std::vector<double> error_vals, double rms_error)
{

    // Create an output filestream object
    std::fstream fout;
    fout.open("solution_part2.csv", std::ios::out);

     fout<<"Node_number, Analytical_Solution, Numerical_Solution, Error"<<"\n";
    // Send data to the stream
    for (int i = 0; i < size; ++i)
    {
        fout<<i<<","<<analytical[i]<<","<<numerical[i]<<","<<error_vals[i]<<"\n";
```

```
    }

    fout<<"\n"<<"RMS Error: "<<rms_error;
    // Close the file
    fout.close();
}

/**
 * function to find the root mean square error from the vector called error_vals
**/
double findRMS(std::vector<double> error_vals){
    double square = 0;
    double mean = 0.0, root = 0.0;

    // Calculate square.
    for (int i = 0; i < error_vals.size(); i++) {
        square += pow(error_vals[i], 2);
    }

    // Calculate Mean.
    mean = (square / (double)(error_vals.size()));

    // Calculate Root.
    root = sqrt(mean);

    return root;
}

/**
 * main function uses the fstream library to read and write to csv files
 * Also, it calculates the node wise error values and also the overall RMS error
**/
int main(int argc, char *argv[])
{

    std::fstream analytical2;
    std::fstream numerical2;

    analytical2.open("outputs_csv/analytical2.csv", std::ios::in);
    numerical2.open("outputs_csv/numerical2.csv", std::ios::in);

    std::vector<std::string> analytical2_rows_strings;
    std::vector<std::string> numerical2_rows_strings;
    std::string line, temp;

    //reading analytical solution (displacement values) for problem 2
    while(!analytical2.eof()){
        getline(analytical2, line);
        analytical2_rows_strings.push_back(line);
    }

    //reading numerical solution (displacement values) for problem 1
    while(!numerical2.eof()){
        getline(numerical2, line);
        numerical2_rows_strings.push_back(line);
    }
```

```
        analytical2.close();
        numerical2.close();

    // this will remove double quotes from the string if present
    for (auto &s : analytical2_rows_strings)
        s.erase(remove(s.begin(), s.end(), '\"'), s.end());

    // this will remove double quotes from the string if present
    for (auto &s : numerical2_rows_strings)
        s.erase(remove(s.begin(), s.end(), '\"'), s.end());

    std::vector<double> analytical2_rows_doubles;
    std::vector<double> numerical2_rows_doubles;

    //converting string values to double for error calculation
    for (auto it = analytical2_rows_strings.begin(); it != analytical2_rows_strings
        .end(); it++) {
    analytical2_rows_doubles.push_back(stod(*it));
     }

    //converting string values to double for error calculation
      for (auto it = numerical2_rows_strings.begin(); it != numerical2_rows_strings
          .end(); it++) {
    numerical2_rows_doubles.push_back(stod(*it));
     }


    //storing error values(numerical - analytical)
    // analytical_rows_doubles.size() == numerical1_rows_doubles.size()
    int size=analytical2_rows_doubles.size();
    std::vector<double> error_vals;

    for(int i=0; i < size; i++){
        error_vals.push_back(analytical2_rows_doubles[i]-numerical2_rows_doubles[i
            ]);
    }

    //finding the Root Mean Square Error
    double rms_error=findRMS(error_vals);

     // write the final solution to 'solution_part2.csv' file
    write_csv(size, analytical2_rows_doubles, numerical2_rows_doubles, error_vals,
       rms_error);

    return 0;
}
```