**Space Truss Analysis Program**

ME5631, Spring 2023

Ravi Chaudhary

Civil Engineering

IIT, Palakkad

Table of Contents

[1.0 Introduction 3](#_Toc141080736)

[2.0 The CTruss Class 4](#_Toc141080737)

[2.1 PrepareIO () 5](#_Toc141080738)

[2.2 SetSize () 5](#_Toc141080739)

[2.3 ReadTrussModel () 5](#_Toc141080740)

[2.4 ReadProblemSize () 6](#_Toc141080741)

[2.5 Analyze () 6](#_Toc141080742)

[2.6 ConstructK () 6](#_Toc141080743)

[2.7 ImposeBC () 6](#_Toc141080744)

[2.8 SuppressDOF () 6](#_Toc141080745)

[2.9 Solve () 6](#_Toc141080746)

[2.10 Response () 7](#_Toc141080747)

[2.11 DisplayErrorMessage () 7](#_Toc141080748)

[2.12 TerminateProgram () 7](#_Toc141080749)

[2.13 CreateOutput () 7](#_Toc141080750)

[3.0 CNode Class 8](#_Toc141080751)

[4.0 CNodalResponse Class 9](#_Toc141080752)

[5.0 CElement Class 10](#_Toc141080753)

[6.0 CElementResponse Class 11](#_Toc141080754)

[7.0 The CMatToolBox Class 12](#_Toc141080755)

[8.0 Input File format 13](#_Toc141080756)

[9.0 Test Cases 13](#_Toc141080757)

[10.0 Concluding Remarks 13](#_Toc141080758)

[References 13](#_Toc141080759)

# 1.0 Introduction

The Finite Element Analysis (FEA) of space truss structures is a sophisticated engineering approach employed to assess the structural behavior and performance of these complex frameworks under diverse loading conditions. In pursuit of enhancing the efficiency and accuracy of such analyses, a specialized Space Truss Analysis Program has been meticulously crafted, utilizing the power of C++ programming language and the principles of finite element analysis. This project aims to cater to the needs of engineers and researchers seeking in-depth insights into the intricate behavior of space trusses.

Implemented as an object-oriented numerical analysis tool, the Space Truss Analysis Program boasts a modular architecture, comprising multiple C++ source files that synergistically collaborate to conduct a comprehensive structural analysis. The heart of the program lies in its ability to read intricate input data, define space truss models, and perform detailed analysis, ultimately yielding valuable output results.

The primary file, **main.cpp**, orchestrates the entire process, seamlessly calling functions defined in various other files, namely **truss.h**, **truss.cpp**, **node.h**, **node.cpp**, **nodalresponse.h**, **nodalresponse.cpp**, **element.h**, **element.cpp**, **elementresponse.h**, **elementresponse.cpp**, **parser.h**, and **parser.cpp**. Each of these files is dedicated to handling specific tasks and responsibilities crucial to the analysis.

The essence of the Space Truss Analysis Program lies in its capacity to accurately handle mechanical and thermal loading scenarios. The input file, meticulously structured, allows users to specify truss geometry, nodal coordinates, fixity conditions, applied loads, and thermal changes. The program proceeds to perform finite element analysis on the input data, invoking classes like **CTruss** to represent the truss model, and **CParser** to parse and interpret the input data correctly.

Through rigorous matrix computations and utilization of matrix toolbox functionalities, the program efficiently evaluates nodal displacements, internal forces, and stresses in each truss member. The results are then meticulously recorded in the output file, presenting a comprehensive overview of the truss's response under the applied loading conditions.

# 2.0 The CTruss Class

class CTruss

{

public:

// ctor and dtor

CTruss ();

~CTruss ();

void Banner (std::ostream& OF) const;

void PrepareIO (int argc, char\* argv[]);

void ReadTrussModel ();

void Analyze ();

void TerminateProgram ();

void DisplayErrorMessage (CLocalErrorHandler::ERRORCODE);

private:

int m\_nNodes; // number of nodes

int m\_nElements; // number of elements

int m\_nDOF; // total degrees-of-freedom

int m\_nDebugLevel; // debugging level

CParser m\_Parser; // to parse input lines

int m\_nLineNumber; // current line number in input file

int m\_nTokens; // number of tokens read

std::vector<std::string>

m\_strVTokens; // stores the tokens read

std::string m\_strComment; // comment characters in the input file

std::string m\_strDelimiters; // delimiters in the input file

CLocalErrorHandler m\_LEH; // for handling errors detected by program

// data storage for

CVector<CNode> m\_NodalData; // nodal data

CVector<CElement> m\_ElementData; // element data

CVector<CNodalResponse> m\_NodalResponseData; // nodal response

CVector<CElementResponse> m\_ElementResponseData;// element response

std::ifstream m\_FileInput; // File Input

std::ofstream m\_FileOutput; // File Output

CMatrix<double> m\_dSSM; // structural stiffness matrix

CVector<double> m\_dSND; // structural nodal displacements

CVector<double> m\_dSNF; // structural nodal forces

CMatToolBox<double> m\_MTBDP; // double precision toolbox

void ConstructK ();

void CreateOutput ();

void ImposeBC ();

void Response ();

void ReadProblemSize ();

void SetSize ();

void Solve ();

void SuppressDOF (const int);

void ErrorHandler (CLocalErrorHandler::ERRORCODE); // gateway to local error handler

void ErrorHandler (CGlobalErrorHandler::ERRORCODE) const; // gateway to global error handler

};

The CTruss class is defined in the **truss.h** header file and implemented in the **truss.cpp** file.

The following variables of their respective classes are created in the CTruss class, which are used to store the data of the truss structure as per the name indicates, more details about these classes are described later in the report.

1. m\_NodalData vector of **CNode** class
2. m\_ElementData vector of **CElement** class
3. m\_NodalResponseData vector of **CNodalResponse** class
4. m\_ElementResponseData vector of **CElementResponse** class

Many Other types of variables are also created,

1. m\_dSSM which is the structural stiffness matrix
2. m\_dSND vector which is for the structural nodal displacements
3. m\_dSNF vector which is for the structural nodal forces
4. m\_MTBDP is a double precision toolbox of class **CMatToolBox**

Other variables such as **m\_FileInput** and **m\_FileOutput** which are used to input and output the file.

Variable **m\_Parser** of class **CParser** is also created which is used to parse input lines, it helps identify the current line, number of arguments/tokens and stores them, comments (here as, \*\*) the character in the input file, identify the delimiters in the input file, local error handler is also called to identify if any error is detected by the program.

The CTruss Class Contains all the major functions for the working of the program,

## 2.1 PrepareIO ()

The PrepareIO function manages input and output file operations in the Space Truss Analysis Program. It allows users to specify input and output file names either through the command line or interactively. The function opens the files, performs error checks, and prints a program banner to the output file, providing essential information about the analysis program.

## 2.2 SetSize ()

This function is responsible for memory allocation for all the major arrays used in the truss finite element analysis. It allocates space for storing nodal data, nodal response data, element data, and element response data. Additionally, it allocates and initializes major matrices used in the analysis, such as the structural stiffness matrix (m\_dSSM), nodal displacements vector (m\_dSND), and nodal forces vector (m\_dSNF). The memory allocation and initialization prepare the data structures for subsequent calculations and analysis in the truss analysis.

## 2.3 ReadTrussModel ()

The ReadTrussModel function reads the truss model data from the input file, including nodal coordinates, fixity conditions, nodal loads, and element data, populating the program's data structures. It ensures error-free parsing and validation of the input data.

## 2.4 ReadProblemSize ()

The ReadProblemSize function reads the problem size from the input file, comprising the number of nodes, elements, and debug level. It performs essential error checks and dynamically allocates memory for data storage.

## 2.5 Analyze ()

The Analyze function implements the Finite Element Analysis (FEA) steps for a space truss system. The function first handles the thermal loading by calculating thermal forces induced by temperature changes at each element's nodes and then applies these loads to the respective nodes.

Next, the function constructs the structural nodal load vector based on the applied forces at each node, considering both the thermal and external loads. It then proceeds to construct the structural stiffness matrix, which represents the relationship between nodal displacements and applied forces.

The function further imposes boundary conditions to account for fixed or constrained nodes in the truss system. It solves for the nodal displacements using matrix equations and computes the responses of individual truss elements.

Finally, the function creates the output file to store the analysis results, including nodal displacements, element responses, and other relevant data for further evaluation and visualization.

## 2.6 ConstructK ()

The ConstructK function calculates and assembles the structural stiffness matrix for a truss model. It iterates through all elements, computes element stiffness matrices, and combines them to form the global stiffness matrix (m\_dSSM). The function takes into account element properties, nodal coordinates, and local-to-global transformations to construct the stiffness matrix.

## 2.7 ImposeBC ()

Implements imposition of essential boundary conditions in a truss finite element analysis, constraining nodes based on their fixity in X, Y, and Z directions to minimize unnecessary calculations and accurately analyze the structure.

## 2.8 SuppressDOF ()

Implements essential boundary conditions by suppressing a degree of freedom (DOF) at a given equation number in the structural stiffness matrix (SSM) and sets the corresponding right-hand side (RHS) value to zero. This ensures that the constrained DOF does not affect the analysis and is fixed at zero displacement.

## 2.9 Solve ()

Solves the system equations for the nodal displacements using the Cholesky decomposition technique. It performs LDLT factorization of the structural stiffness matrix (SSM) and then solves the system equations to obtain the nodal displacements. The resulting displacements are stored in the m\_NodalResponseData array, representing the computed displacements at each node of the truss structure.

## 2.10 Response ()

Computes the element response in a truss finite element analysis, including strain, stress, and force for each element. The function calculates the nodal displacements, transforms them to local coordinates, and then determines the strain, stress, and force for each element. The results are stored in the element response data for further analysis or output.

## 2.11 DisplayErrorMessage ()

The DisplayErrorMessage function is used to display an error message based on the error code passed as an argument. It calls the ErrorHandler function from the CLocalErrorHandler class and provides the error code along with the current line number where the error occurred as additional information to handle the error appropriately.

## 2.12 TerminateProgram ()

The TerminateProgram function is responsible for closing the input and output files used during the execution of the program. After closing the files, it displays a message indicating that the execution completed successfully.

## 2.13 CreateOutput ()

Generates an output file containing the results of the finite element analysis for a truss structure. It includes information about the problem size, nodal coordinates, nodal fixities, nodal forces, nodal displacements, element responses (strain, stress, and force), and nodal reactions. The output file provides valuable insights into the behavior of the truss under various loads and constraints.

# 3.0 CNode Class

The **CNode** class represents a node in a planar truss structure. It is used to store and manage information related to a node, including its coordinates, fixity conditions (constraints), and applied loads. The class contains accessor and modifier functions to get and set these properties. Specifically, the **CNode** class has the following member variables:

1. **m\_fXCoor**, **m\_fYCoor**, **m\_fZCoor**: These variables store the x, y, and z coordinates of the node, respectively.
2. **m\_nXFC**, **m\_nYFC**, **m\_nZFC**: These variables store the fixity codes for the x, y, and z directions, respectively. The fixity codes indicate whether the node is fixed (constraint) in each direction.
3. **m\_fXForce**, **m\_fYForce**, **m\_fZForce**: These variables store the applied forces at the node in the x, y, and z directions, respectively.
4. **m\_fdelT**: This variable stores the temperature change at the node.

The class provides member functions to access and modify these properties, allowing other parts of the program to interact with and manage the node data effectively. Overall, the **CNode** class serves as a fundamental building block for representing individual nodes in the planar truss structure and plays a vital role in the analysis and design of the truss.

# 4.0 CNodalResponse Class

The **CNodalResponse** class is used to represent the nodal displacements of a node in a planar truss structure. It is responsible for storing and managing the x, y, and z displacements of a node after the truss analysis is performed. The class provides accessor and modifier functions to get and set these displacement values.

The **CNodalResponse** class has the following member variables:

1. **m\_fXDisp**: This variable stores the x-displacement of the node.
2. **m\_fYDisp**: This variable stores the y-displacement of the node.
3. **m\_fZDisp**: This variable stores the z-displacement of the node.

The class provides the following member functions:

1. **CNodalResponse()**: The default constructor that initializes the displacements to zero.
2. **~CNodalResponse()**: The destructor.
3. **GetDisplacements()**: An accessor function that retrieves the x, y, and z displacements of the node.
4. **SetDisplacements()**: A modifier function that sets the x, y, and z displacements of the node.

The **CNodalResponse** class is crucial for capturing and representing the results of the truss analysis, as it stores the nodal displacements that indicate how the nodes have deformed under the applied loads and constraints. This information is essential for further analysis, design, and visualization of the truss structure.

# 5.0 CElement Class

The **CElement** class is used to represent an individual element in a planar truss structure. It stores and manages data related to a truss element, including the start and end node numbers, cross-sectional area, modulus of elasticity, and coefficient of thermal expansion. The class provides accessor and modifier functions to get and set these element-related values.

The **CElement** class has the following member variables:

1. **m\_nSN**: This variable stores the node number of the start node of the truss element.
2. **m\_nEN**: This variable stores the node number of the end node of the truss element.
3. **m\_fArea**: This variable stores the cross-sectional area of the truss element.
4. **m\_fE**: This variable stores the modulus of elasticity of the truss material.
5. **m\_falp**: This variable stores the coefficient of thermal expansion of the truss material.

The class provides the following member functions:

1. **CElement()**: The default constructor that initializes the member variables to zero.
2. **~CElement()**: The destructor.
3. **GetData()**: An accessor function that retrieves the start node number, end node number, cross-sectional area, modulus of elasticity, and coefficient of thermal expansion of the truss element.
4. **SetData()**: A modifier function that sets the start node number, end node number, cross-sectional area, modulus of elasticity, and coefficient of thermal expansion of the truss element.

The **CElement** class is crucial for representing and managing the properties of each truss element. It is used to store data that describes the characteristics of the truss elements, enabling efficient computation of the truss analysis and providing valuable information for further structural analysis and design.

# 6.0 CElementResponse Class

The **CElementResponse** class is used to represent the response of an individual truss element in a planar truss structure. It stores and manages data related to the element's response, including the strain, stress, and internal force. This information is crucial in assessing the performance and behavior of the truss elements under various loading conditions and can be used for further structural analysis and design.

The **CElementResponse** class has the following member variables:

1. **m\_fStrain**: This variable stores the strain experienced by the truss element.
2. **m\_fStress**: This variable stores the stress developed in the truss element.
3. **m\_fForce**: This variable stores the internal force within the truss element.

The class provides the following member functions:

1. **CElementResponse()**: The default constructor that initializes the member variables to zero.
2. **~CElementResponse()**: The destructor.
3. **GetData()**: An accessor function that retrieves the element's strain, stress, and internal force.
4. **SetData()**: A modifier function that sets the element's strain, stress, and internal force.

The **CElementResponse** class is essential for storing and managing the response data of individual truss elements during the truss analysis process. It allows for the retrieval and modification of strain, stress, and internal force values for each element, providing valuable insights into the behavior of the truss structure and supporting various engineering calculations and assessments.

# 7.0 The CMatToolBox Class

The CMatToolBox class contains all the functions for performing vector and matrix operations such as addition, subtraction, dot product, cross product, scaling, normalization, LDLT Factorization.

All these functions perform the operations as their names indicate, and within CMatToolBox class they are in the MatToolBox.h header file.

Since vector and matrix operations are performed, they require CVector and CMatrix inputs which are used using CVector and CMatrix classes. All the required features of CVector and CMatrix classes are within the CMatToolBox class.

Error Handler function contains all the designated erros for each operation.

**CMatrixToolBox Class contains following functions:**

// vector-related functions

void Display (const std::string& strMessage,

const CVector<T>& A) const;

void Add (const CVector<T>& A, const CVector<T>& B,

CVector<T>& C);

void Subtract (const CVector<T>& A, const CVector<T>& B,

CVector<T>& C);

T DotProduct (const CVector<T>& A,

const CVector<T>& B);

void Normalize (CVector<T>& A);

void Scale (CVector<T>& A, const T factor);

T MaxValue (const CVector<T>& A) const;

T MinValue (const CVector<T>& A) const;

T TwoNorm (const CVector<T>& A);

T MaxNorm (const CVector<T>& A) const;

void CrossProduct (const CVector<T>& A,

const CVector<T>& B, CVector<T>& C);

// matrix-related functions

void Display (const std::string& strMessage,

const CMatrix<T>& A) const;

void Multiply (const CMatrix<T>& A,

const CMatrix<T>& B, CMatrix<T>& C);

T MaxNorm (const CMatrix<T>& A) const;

void Transpose (const CMatrix<T>& A,

CMatrix<T>& B);

void MatMultVec (const CMatrix<T>& A,

const CVector<T>& x,

CVector<T>& b);

void AxEqb (CMatrix<T>& A,

CVector<T>& x,

CVector<T>& b,

const T TOL);

void LDLTFactorization (CMatrix<T>& A, const T TOL);

void LDLTSolve (const CMatrix<T>& A, CVector<T>& x,

const CVector<T>& b);

// helper function

void ResidualVector (const CMatrix<T>& A, const CVector<T>& x,

const CVector<T>& b, CVector<T>& R,

T& AbsError, T& RelError);

bool IsEqual (const T d1, const T d2, T TOL = TOLDEF) const;

bool IsEqual (const CMatrix<T>& dMA,

const CMatrix<T>& dMB, T TOL = TOLDEF) const;

bool IsEqual (const CVector<T>& dVA,

const CVector<T>& dVB, T TOL = TOLDEF) const;

void GetFLOPStats (double& dAS, double& dM, double& dD) const;

# 8.0 Input File format

A screenshot of a computer

Description automatically generated

# 9.0 Test Cases

A screenshot of a computer program

Description automatically generated

Test Case 1:

Input File:

A computer screen shot of a program

Description automatically generated

Fig1.1.1 InputTestCase1

Output:

A black screen with white text

Description automatically generated

Fig1.2.1 OutputTestCase1

A screenshot of a computer

Description automatically generated

Fig1.2.2 OutputTestCase1

# 10.0 Concluding Remarks

Through the integration of finite element analysis principles and C++ programming, this project has yielded a versatile and efficient tool for analyzing space truss structures under mechanical and thermal loading conditions.

The program's modular design and object-oriented approach ensure code reusability, maintainability, and scalability. Its capability to accurately predict nodal displacements, element responses, and critical stress points empowers engineers and researchers to make informed decisions during the design, evaluation, and optimization of space truss structures.

Throughout the development process, a strong emphasis was placed on code organization, efficiency, and error handling. The robustness of the program, combined with its user-friendly interface, makes it accessible to engineering professionals, facilitating the analysis of complex space truss systems.

While settlement analysis could have further enhanced the program's capabilities, its current form already serves as a powerful tool for structural analysis. As with any software project, there are always opportunities for future improvements and expansions, and the inclusion of settlement analysis could be a valuable enhancement in subsequent iterations.

Overall, the Space Truss Analysis Program stands as a testament to the synergy between finite element analysis and C++ programming. Its contribution to the field of structural engineering fosters innovation and provides engineers with a powerful means to design safer and more resilient space truss structures.

# References

* Object-OrientedNumerical Methodsvia C++-2ndEdition, by S. D. Rajan.