REPUBLIC OF TÜRKİYE
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

DEVELOPMENT OF AN INTERACTIVE SOFTWARE FOR STRUCTURAL ANALYSIS OF PLANE FRAME SYSTEMS

Ahmet YAŞAR

MASTER OF SCIENCE THESIS
Department of Civil Engineering
Program of Civil Engineering

Supervisor
Asst. Prof. Dr. Orkun YILMAZ

February, 2026

REPUBLIC OF TÜRKİYE
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

DEVELOPMENT OF AN INTERACTIVE SOFTWARE FOR STRUCTURAL ANALYSIS OF PLANE FRAME SYSTEMS

A thesis submitted by Ahmet YAŞAR in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** is approved by the committee on 25.02.2026 in Department of Civil Engineering, Program of Civil Engineering.

Asst. Prof. Dr. Orkun YILMAZ
Yildiz Technical University
Supervisor

Approved By the Examining Committee

Asst. Prof. Dr. Orkun YILMAZ, Supervisor
Yildiz Technical University _____

Prof. Dr. Murat ALTEKİN, Member
Yildiz Technical University _____

Asst. Prof. Dr. Zeynep ÜNSAL ASLAN, Member
Batman University _____

I hereby declare that I have obtained the required legal permissions during data collection and exploitation procedures, that I have made the in-text citations and cited the references properly, that I haven't falsified and/or fabricated research data and results of the study and that I have abided by the principles of the scientific research and ethics during my Thesis Study under the title of Development of an interactive software for structural analysis of plane frame systems supervised by my supervisor, Asst. Prof. Dr. Orkun YILMAZ. In the case of a discovery of false statement, I am to acknowledge any legal consequence.

Ahmet YAŞAR

Signature

Dedicated to my family

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor, Asst. Prof. Dr. Orkun Yılmaz, for his valuable guidance, constructive feedback, and continuous support throughout the preparation of this thesis. His expertise and academic insight have been instrumental in shaping the scope and quality of this study.

I am also deeply grateful to my family for their patience, encouragement, and unwavering support during this challenging process.

Special thanks are extended to my department manager, Müge Karataş Vezir, and to all my colleagues who supported me throughout this process, for their understanding, motivation, and encouragement, which greatly contributed to the successful completion of this work.

Finally, I would like to thank everyone who contributed directly or indirectly to this study.

Ahmet YAŞAR

TABLE OF CONTENTS

LIST OF SYMBOLS viii

LIST OF ABBREVIATIONS x

LIST OF FIGURES xi

LIST OF TABLES xiii

ABSTRACT xiv

ÖZET xvi

1 INTRODUCTION 1

1.1 Literature Review 2

1.1.1 Historical Development of the Finite Element Method . . . 2

1.1.2 Theoretical Background and Element Formulations 3

1.1.3 FEM-Based Software and Algorithm Development 4

1.1.4 Structural Analysis, Simulation, and Environmental Load Studies 5

1.1.5 Scope of This Study 6

2 THEORETICAL BACKGROUND 8

2.1 Frame System Kinematics 8

2.2 Degrees of Freedom of the Frame Element 8

2.3 Local Stiffness Matrix of the Frame Element 9

2.4 Local-to-Global Coordinate Transformation 12

2.5 Assembly of the Global Stiffness Matrix 14

2.6 Load Vector Formulation 16

2.6.1 Nodal Loads 17

2.6.2 Element Loads 17

2.7 Boundary Conditions 22

2.7.1 Support Conditions 22

2.7.2 Prescribed Displacements 23

2.8 Solution of the Global System 25

2.8.1 Element End Forces 26
2.8.2 Internal Force and Moment Diagrams 27

3 SOFTWARE DEVELOPMENT PROCESS 28
3.1 Programming Language and Tools 29
3.1.1 Programming Language 29
3.1.2 Development Environment 30
3.2 Software Architecture and Class Structure 30
3.2.1 Core Classes 30
3.2.2 Advantages Of Object-Oriented Programming 32
3.2.3 Visualization of Structural Analysis Results 32
3.3 User Interface 32
3.3.1 Creating Coordinate System 32
3.3.2 Material Definition 33
3.3.3 Cross-Section Definition 34
3.3.4 Generation of the frame model 36
3.3.5 Cross-Section Assignment 36
3.3.6 Restriction Assignment 37
3.3.7 Frame Release Assignment 38
3.3.8 Load Assignments 39
3.3.9 Node Displacement Assignments 41
3.3.10 Diagram Visualization 41

4 VERIFICATION AND COMPARATIVE CASE STUDIES WITH SAP2000 43
4.1 Verification Methodology 43
4.2 Case Studies 44
4.2.1 Case Study A: Frame System Subjected to Mechanical Loads, Thermal Effects and Hinged Connections 44
4.2.2 Case Study B: Frame System with Prescribed Nodal Displacements 50
4.3 Error Analysis and Discussion 56

5 CONCLUSION 58

REFERENCES 59

A PROGRAM LISTINGS 61
A.1 DrawRestrictions 61
A.2 Line Element 62
A.3 Transformation Matrix 62

A.4 Deformed Shape 63
A.5 Restriction Assignment 65
A.6 Frame Release 66
A.7 Load Assignments 69

B Numerical Results of Case Study: A **79**
B.1 Element Stiffness Matrices 79
B.2 Reduced Global Stiffness Matrix 80

C Numerical Results of Case Study: B **81**
C.1 Element Stiffness Matrices 81
C.2 Reduced Global Stiffness Matrix 83

PUBLICATIONS FROM THE THESIS **84**

LIST OF SYMBOLS

$N_{(x)}$	Axial Force
k_{axial}	Axial Stiffness Matrix
ϵ	Axial Strain
U_M	Bending Deformation
$M_{(x)}$	Bending Moment
$L^{(i)}$	Boolean Topology Matrix
α	Coefficient of Thermal Expansion
A	Cross-Section Area
h	Cross-Section Height
D	Displacement Vector
E	Elastic Modulus
k_g	Element Stiffness Matrix in Global Coordinate System
F	Force Vector
K_{system}	Global Stiffness Matrix
d_e	Local Displacement Vector of an Element
I	Moment of Inertia
$S_{prescribed}$	Prescribed Degree of Freedom
θ	Rotation
N_i	Shape Functions
B	Strain-Displacement Relation
k_{el}	Stiffness Matrix of an Element
U_N	Strain Energy for Axial Deformation
ΔT	Temperature Change

T	Transformation Matrix
v, u	Transverse Displacements

LIST OF ABBREVIATIONS

CST	Constant Strain Triangle
DOF	Degree of Freedom
FEM	Finite Element Method
X-FEM	Extended Finite Element Method
IGA	Isogeometric Analysis
CAD	Computer Aided Design
FEA	Finite Element Analysis
OOP	Object-Oriented Programming
PeriFEM	Peridynamics-Based Finite Element Methods
UPF	User Programmable Features
UEL	User Defined Element

LIST OF FIGURES

Figure 2.1 Local Coordinates **13**
Figure 2.2 Topology Matrix **15**
Figure 2.3 Example System **15**
Figure 2.4 Point Load Example **18**
Figure 2.5 Distributed Load Example **20**
Figure 2.6 Reduction of Global Stiffness Matrix **23**
Figure 3.1 Workflow **29**
Figure 3.2 Support Reactions **31**
Figure 3.3 Coordinate System Form **33**
Figure 3.4 Material Define Form **34**
Figure 3.5 Section Define Form **35**
Figure 3.6 Section Import Form **35**
Figure 3.7 Generating the Frame Model **36**
Figure 3.8 Assigning Cross-Section **37**
Figure 3.9 Assigning Restrictions **38**
Figure 3.10 Assigning Frame Releases **39**
Figure 3.11 Assigning Loads **40**
Figure 3.12 Temperature Loads **40**
Figure 3.13 Assigning Node Displacements **41**
Figure 3.14 Axial Force **41**
Figure 3.15 Shear Force **42**
Figure 3.16 Moment Diagram **42**
Figure 3.17 Deflected Shape **42**
Figure 4.1 Example System **44**
Figure 4.2 Comparison of deformed shapes **46**
Figure 4.3 Comparison of axial force diagrams **49**
Figure 4.4 Comparison of shear force diagrams **49**
Figure 4.5 Comparison of bending moment diagrams **49**
Figure 4.6 Example System **50**
Figure 4.7 Comparison of deformed shapes **52**
Figure 4.8 Comparison of axial force diagrams **55**
Figure 4.9 Comparison of shear force diagrams **56**

Figure 4.10 Comparison of bending moment diagrams **56**

LIST OF TABLES

Table 3.1 Project-specific and Third-party Libraries **30**
Table 4.1 Geometric and Directional Properties of Frame Elements **45**
Table 4.2 Material and Cross-Section Properties of Frame Elements **45**
Table 4.3 Applied Loads on Frame Elements **46**
Table 4.4 Comparison of Nodal Displacements **47**
Table 4.5 Comparison of Internal Forces at Element Ends **48**
Table 4.6 Geometric and Directional Properties of Frame Elements **51**
Table 4.7 Material and Cross-Section Properties of Frame Elements **51**
Table 4.8 Applied Loads on Frame Elements **52**
Table 4.9 Comparison of Nodal Displacements **53**
Table 4.10 Comparison of Internal Forces at Element Ends **54**

ABSTRACT

Development of an interactive software for structural analysis of plane frame systems

Ahmet YAŞAR

Department of Civil Engineering
Master of Science Thesis

Supervisor: Asst. Prof. Dr. Orkun YILMAZ

In this thesis, a computer program based on the Finite Element Method (FEM) is developed for the static analysis of frame structures. The developed software is designed to compute nodal displacements, internal forces, and moment distributions of two-dimensional frame elements. The accuracy and reliability of the program are evaluated through comparative analyses conducted using SAP2000, a widely used commercial structural analysis software.

Within the scope of the study, a representative frame system is analyzed both with the developed program and SAP2000 under identical geometric, material, loading, and boundary conditions. Nodal displacements and element internal forces obtained from both solutions are compared, and the results are presented in tabular and graphical forms. The findings indicate a high level of agreement between the developed program and SAP2000 results, with only negligible numerical differences observed. These differences are attributed to numerical rounding, element formulation, and the implementation of boundary conditions.

In conclusion, the developed program is shown to produce reliable results for the static analysis of frame structures and can be used as an alternative analysis tool for academic studies and engineering applications.

Keywords: Finite element method, Planar frame analysis, Numerical structural analysis, Linear static analysis, Software development.

YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

ÖZET

Düzlem Çerçeve Sistemlerin Yapısal Analizi İçin İnteraktif Bir Yazılım Geliştirilmesi

Ahmet YAŞAR

İnşaat Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Danışman: Dr.Öğretim Üyesi Orkun YILMAZ

Bu tez çalışmasında, çerçeve sistemlerin statik analizi için geliştirilen bir sonlu elemanlar yöntemi (SEM) tabanlı bilgisayar programı sunulmuştur. Geliştirilen program, iki boyutlu çerçeve elemanlarının yer değiştirme, iç kuvvet ve moment dağılımlarını hesaplayabilecek şekilde tasarlanmıştır. Programın doğruluğu ve güvenilirliği, literatürde yaygın olarak kullanılan ticari bir yapısal analiz yazılımı olan SAP2000 ile yapılan karşılaştırmalı analizler aracılığıyla değerlendirilmiştir.

Çalışma kapsamında seçilen örnek bir çerçeve sistem, hem geliştirilen program hem de SAP2000 ortamında aynı geometrik, malzeme ve yükleme koşulları altında analiz edilmiştir. Düğüm noktası yer değiştirmeleri ile eleman iç kuvvetleri karşılaştırılmış ve sonuçlar tablo ve grafikler halinde sunulmuştur. Elde edilen bulgular, geliştirilen programın SAP2000 sonuçları ile yüksek uyum gösterdiğini, yalnızca ihmal edilebilir düzeyde sayısal farklar bulunduğunu ortaya koymuştur. Bu farkların, sayısal yuvarlama, eleman formülasyonu ve sınır koşullarının uygulanış biçimlerinden kaynaklandığı değerlendirilmiştir.

Sonuç olarak, geliştirilen programın çerçeve sistemlerin statik analizinde güvenilir sonuçlar ürettiği ve akademik çalışmalar ile mühendislik uygulamalarında alternatif bir analiz aracı olarak kullanılabileceği gösterilmiştir.

Anahtar Kelimeler: Sonlu elemanlar yöntemi, Düzlem çerçeve analizi, Sayısal yapısal analiz, Doğrusal statik analiz, Yazılım geliştirme

1
INTRODUCTION

Structural analysis plays a crucial role in the design and assessment of engineering structures. Accurate prediction of structural responses such as displacements, internal forces, and stresses is essential to ensure safety, serviceability, and economic efficiency. Among various numerical techniques, the finite element method (FEM) has become one of the most widely used and reliable approaches for solving complex structural problems.

With the advancement of computational power, numerous commercial finite element software packages, such as SAP2000, ANSYS, and ABAQUS, have been developed and are extensively used in both academic and professional practice. Although these software packages provide highly accurate and efficient solutions, they often operate as black-box tools, limiting the user's understanding of the underlying numerical procedures. This limitation can be particularly restrictive in educational environments and during the development of customized analysis approaches.

For this reason, the development of in-house numerical analysis programs based on fundamental FEM principles remains an important research topic. Such programs not only contribute to a deeper understanding of the theoretical background of the finite element method but also allow greater flexibility in implementing problem-specific formulations and boundary conditions.

The primary objective of this study is to develop a finite element-based structural analysis software capable of performing linear static analysis of frame and beam elements. The developed program is designed to compute nodal displacements and internal force components, including axial force, shear force, and bending moment. To verify the accuracy and reliability of the proposed software, the numerical results are compared with those obtained from the commercial software SAP2000 through an example.

The structure of this thesis is organized as follows. First, the theoretical background and numerical formulation of the finite element method are presented. Then, the implementation details of the developed software are explained. Subsequently, the numerical results are compared with reference solutions, and the observed discrepancies are discussed. Finally, conclusions and potential directions for future work are provided.

1.1 Literature Review

1.1.1 Historical Development of the Finite Element Method

The historical development of the Finite Element Method (FEM) is generally divided into distinct periods, beginning with its inception in the mid-twentieth century driven by the demands of aeronautical and aerospace engineering. The origins of the method are traced to the early 1940s; Hrennikoff introduced an early discretization approach by representing a continuous medium as a lattice framework, while Courant laid the mathematical foundation for piecewise polynomial approximation through variational principles. By the 1950s, these concepts were further advanced by Argyris at Imperial College and Turner at Boeing, who developed the 'Matrix Stiffness Method,' a direct precursor to modern FEM applied to complex aircraft structures.

The term "Finite Element" was formally coined by Clough in 1960 in his pioneering work on plane stress analysis, marking a significant milestone in the method's evolution. Following Clough's contributions, FEM rapidly expanded into civil engineering. A pivotal moment in this expansion was the analysis of the Norfork Dam in the early 1960s, which demonstrated the method's capability to solve complex continuum mechanics problems and predict realistic stresses in large structural systems.

The period between 1966 and 1991 is often characterized as the "Golden Age" of FEM, where the method evolved from a structural analysis tool into a rigorous mathematical discipline. During this era, mathematical foundations regarding error bounds and convergence were solidified by researchers such as Babuška and Zienkiewicz, and the method was extended to fluid dynamics and heat transfer. In the modern era, the development of the Extended Finite Element Method (X-FEM) for fracture mechanics and Isogeometric Analysis (IGA) to integrate CAD with FEA has further broadened the method's applicability across geotechnical and industrial engineering disciplines [1].

1.1.2 Theoretical Background and Element Formulations

FEM is based on the fundamental principle of modeling a continuum by discretizing it into a finite number of smaller and simpler subregions, known as finite elements. This method operates by transforming the partial differential equations governing the physical problem into a system of algebraic equations defined at discrete nodal points [2]. As a numerical analysis technique employed to obtain approximate solutions to engineering problems, FEM provides a piecewise representation of complex geometries by modeling the solution domain in an approximate analytical manner.

In structural mechanics, this formulation establishes a direct relationship between nodal displacements and nodal forces through element stiffness matrices. The element stiffness matrix $\mathbf{K}$ is generally derived using the principle of virtual work and expressed in terms of the strain–displacement matrix $\mathbf{B}$ and the material constitutive matrix $\mathbf{C}$ (or $\mathbf{E}$), in the general form [3]:

$$\mathbf{K} = \int_V \mathbf{B}^T \mathbf{E} \mathbf{B} \, dV \tag{1.1}$$

Various element formulations have been developed in the literature to address different classes of structural problems. For the analysis of frame and skeletal systems, one-dimensional truss and beam elements are commonly employed. In these elements, the geometry and displacement field are expressed in the natural coordinate system (r) using interpolation functions. For plane stress and plane strain problems, two-dimensional elements such as triangular and quadrilateral elements are widely preferred. In the plane strain condition, displacements through the thickness (in the z-direction) are assumed to be zero $(u_z = 0)$, whereas in the plane stress condition, the normal stress perpendicular to the plane is assumed to vanish $(\sigma_{zz} = 0)$.

Linear triangular elements, also known as Constant Strain Triangle (CST) elements, are particularly advantageous for mesh generation in complex geometries due to their simplicity and adaptability. Owing to their ability to approximate curved boundaries using straight-line segments, finite element models are capable of capturing boundary conditions of complex geometric shapes more effectively than finite difference methods [4]. However, numerous studies have demonstrated that mesh refinement is required to achieve accurate results in regions with high stress gradients. Furthermore, the use of second-order basis functions instead of first-order basis functions in element formulations can improve computational accuracy and convergence rates without excessively increasing the number of elements.

For three-dimensional analyses, tetrahedral and hexahedral solid elements are commonly adopted. In the mathematical modeling of these elements, the isoparametric formulation is frequently employed; in this approach, the same shape functions are used to interpolate both the element geometry and the displacement field ($\mathbf{x} = \mathbf{NX}$ and $\mathbf{u} = \mathbf{NU}$). This formulation enables the transformation between the natural coordinate system (r, s, t) and the global coordinate system through the Jacobian matrix $\mathbf{J}$, thereby allowing the analysis of complex three-dimensional geometries and nonstandard element shapes.

1.1.3 FEM-Based Software and Algorithm Development

With the rapid increase in computational power, a wide range of commercial FEM software packages, such as SAP2000, ANSYS, ABAQUS, and COMSOL Multiphysics, have been developed and are now widely used in both academic research and professional engineering practice. These programs enable engineers to analyze complex structures with high accuracy by providing efficient solution algorithms and user-friendly interfaces.

Despite their advantages, commercial FEM software generally operates as *black-box* systems, offering limited transparency regarding the underlying numerical formulations and solution procedures. This limitation may hinder a deep understanding of FEM principles, particularly in educational environments and during the development of customized numerical formulations. Consequently, many researchers tend to develop their own codes in order to obtain full control over the implemented algorithms. For instance, Sumets [5] notes that frameworks developed in programming languages such as MATLAB and Python provide a flexible environment for coding and validating FEM algorithms, ranging from one-dimensional boundary value problems to complex geometries.

During the development of in-house software, code sustainability and modularity are of critical importance. Nikishkov [6] emphasizes that, in a Java-based FEM design employing object-oriented programming (OOP) techniques, organizing the preprocessing, processing, and postprocessing stages into dedicated packages such as `fea`, `model`, `elem`, and `util` significantly reduces code duplication and enhances modularity. Such software architectures facilitate the integration of complex algorithms into custom FEM codes, including advanced procedures such as three-dimensional mesh generation using the sweep method.

In this context, local studies reported in the literature demonstrate the effectiveness of customized FEM implementations. Alyavuz [7] developed a computer program

based on triangular elements for plane elasticity problems and highlighted the importance of mesh refinement in critical regions by comparing the results with those obtained from ANSYS. Similarly, Gazi [8] implemented plane stress and plane strain analysis modules using Constant Strain Triangle (CST) elements and validated the results against SAP2000, concluding that CST elements are effective for problems with low stress variation. More advanced numerical studies have focused on nonlinear material behavior; in this regard, Hayrioğlu [9] developed a MATLAB-based algorithm for reinforced concrete systems, incorporating advanced constitutive models.

On the other hand, rather than writing codes entirely from scratch, researchers also take advantage of the extensibility features provided by commercial FEM software. Han [10] report that, in cases where standard software packages prove inadequate for simulating brittle fracture phenomena—such as in peridynamics-based finite element methods (PeriFEM)—user subroutines offered by commercial codes provide an effective solution. In their work, newly developed fracture models were successfully integrated into existing commercial software infrastructures by utilizing User Programmable Features (UPF) in ANSYS and the User Defined Element (UEL) interface in ABAQUS. This approach allows researchers to benefit from the robust preprocessing and solver capabilities of commercial software while implementing original theoretical models.

1.1.4 Structural Analysis, Simulation, and Environmental Load Studies

Beyond linear static structural analysis, FEM has been extensively applied to investigate structural behavior under mechanical, dynamic, and environmental loading conditions. As an effective numerical technique for handling complex geometries and irregular boundary conditions, FEM has been employed across a broad spectrum of problems, ranging from nonlinear analyses to dynamic simulations. In this context, Aydoğmuş [11] examined the nonlinear behavior of shear connectors in composite beams using ANSYS, addressing challenges associated with concrete nonlinearity and contact modeling. Similarly, Abdulkhalek [12] analyzed simply supported beams under moving loads, while Ahmed investigated monopile foundations of offshore wind turbines subjected to lateral loading conditions, demonstrating the effectiveness of FEM in dynamic analysis and soil–structure interaction problems.

Modal analysis plays a critical role in understanding structural behavior under dynamic loading. Wei [13] emphasized that FEM-based modal analysis is a vital tool for evaluating the dynamic characteristics of engineering structures and

enhancing their structural safety. This approach, which is employed to determine natural frequencies and vibration modes, is based on the formulation of mass and stiffness matrices and the solution of the associated eigenvalue problems. According to Wei, FEM simulations are essential for preventing resonance phenomena that may arise when the natural frequencies of a structure coincide with the frequencies of external excitations, particularly under broadband dynamic inputs such as seismic activity and wind loads in civil engineering applications.

Environmental loading effects, especially thermal actions, have also been widely investigated using FEM. Al-Hatmey [14] analyzed temperature distributions and thermal gradients in precast reinforced concrete bridge girders using ANSYS and COMSOL Multiphysics. The study demonstrated that environmental factors such as solar radiation, ambient temperature, and wind speed can induce significant thermal stresses, highlighting the importance of accurately modeling thermal effects in structural analysis. In this regard, Wei noted that future research is expected to focus on the integration of multiple physical fields rather than isolated single-physics domains, such as coupled thermo–structural–acoustic analyses or aerodynamic thermal loading. Such multiphysics-based modal analyses require the use of more advanced algorithms to simulate complex environmental interactions, including the influence of thermal loads on structural vibration modes and aerodynamic heating effects in high-speed vehicles.

1.1.5 Scope of This Study

The literature review indicates that the FEM has been extensively studied and successfully applied to a wide range of structural analysis problems. Commercial software packages offer powerful and efficient tools for such analyses; however, their black-box nature often limits transparency and reduces their effectiveness for educational purposes and for gaining a detailed understanding of the underlying numerical procedures.

Within this context, the scope of the present study is focused on the development of a finite element–based structural analysis software dedicated to linear static analysis of frame and beam elements. The primary objective is not to introduce new numerical formulations, but rather to implement fundamental FEM principles in a transparent and modular computational framework. The developed software enables direct observation of the relationships between element stiffness formulations, global system assembly, boundary condition enforcement, and resulting structural responses.

The program is designed to compute nodal displacements and internal force components, including axial force, shear force, and bending moment. The accuracy and reliability of the proposed implementation are assessed through comparison with results obtained from the commercial software SAP2000. By emphasizing clarity, modularity, and consistency with established FEM theory, this study aims to provide a practical and educationally valuable tool for understanding and applying linear static finite element analysis in structural engineering.

2
THEORETICAL BACKGROUND

2.1 Frame System Kinematics

The kinematic description of a frame system defines how the structure deforms under applied loads and boundary conditions. In the context of the finite element method, the frame system is assumed to undergo small displacements and rotations, allowing the use of linear kinematic relations. Each node of the frame structure is characterized by a set of translational and rotational degrees of freedom, which together describe the overall deformation state of the system.

The global displacement vector of the frame system is assembled by collecting all nodal degrees of freedom in a consistent order. These nodal displacements are related to element-level deformations through kinematic compatibility conditions, which ensure continuity of displacements and rotations between adjacent elements. By establishing these relationships, the deformation of each element can be expressed in terms of the global nodal displacement vector.

The kinematic formulation provides the foundation for deriving element strain-displacement relations and, consequently, the element stiffness matrices. Therefore, an accurate definition of the frame system kinematics is a crucial step in the formulation of the global equilibrium equations of the structure.

2.2 Degrees of Freedom of the Frame Element

A two-dimensional frame element connects two nodes and is capable of transmitting axial force, shear force, and bending moment. Accordingly, each node of a planar frame element possesses three degrees of freedom: one axial displacement, one transverse displacement, and one rotational displacement. These degrees of freedom are typically defined in the local coordinate system of the element.

For a frame element with nodes i and j, the local displacement vector can be written

as

$$\mathbf{d}_e = \begin{Bmatrix} u_i & v_i & \theta_i & u_j & v_j & \theta_j \end{Bmatrix}^T \tag{2.1}$$

where u and v denote the axial and transverse displacements, respectively, and θ represents the rotation of the cross-section.

The element degrees of freedom are related to the global degrees of freedom through a coordinate transformation matrix, which accounts for the orientation of the element in the global coordinate system. This transformation allows the element stiffness matrix and force vector to be consistently assembled into the global system. The clear definition of the frame element degrees of freedom ensures compatibility and equilibrium throughout the entire structural model.

2.3 Local Stiffness Matrix of the Frame Element

A two-node planar frame element possesses a total of six degrees of freedom. Two axial displacement and one rotational degree of freedom at each node. This element can be regarded as the superposition of a truss element which provides axial rigidity and a beam element which provides flexural rigidity. In other words the axial force component N acting in the local longitudinal direction of the element and the local bending moment component M are mutually independent. Consequently, the overall stiffness matrix is obtained by combining the axial stiffness terms (EA/L) with the flexural stiffness terms(EI).

The stiffness matrix of the elements is typically derived using the principle of virtual work or energy methods.

For axial deformation, the strain energy is expressed as;

$$U_N = \frac{1}{2} \int_0^L EA\, \varepsilon^2 \, dx \tag{2.2}$$

Since a bar element has two nodes and undergoes only axial deformation, the displacement field is assumed to vary linearly along the element:

$$u(x) = N_1(x)\, u_i + N_2(x)\, u_j \tag{2.3}$$

The corresponding shape functions are defined as [3].

$$N_1(x) = 1 - \frac{x}{L}, \qquad N_2(x) = \frac{x}{L} \tag{2.4}$$

The axial strain ε is obtained by differentiating the displacement field with respect to the spatial coordinate:

$$\varepsilon = \frac{du}{dx} = \frac{d}{dx}\left[\left(1 - \frac{x}{L}\right) u_i + \frac{x}{L} u_j\right] = -\frac{1}{L} u_i + \frac{1}{L} u_j \tag{2.5}$$

In matrix notation, the strain–displacement relation can be written using the matrix $\mathbf{B}$ as:

$$\varepsilon = \begin{bmatrix} -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \end{Bmatrix} \tag{2.6}$$

Substituting the expression for ε into the axial strain energy formulation, the stiffness matrix $\mathbf{k}_{\text{axial}}$ is obtained from:

$$\mathbf{k}_{\text{axial}} = \int_0^L \mathbf{B}^T E A \mathbf{B}\, dx \tag{2.7}$$

Since $\mathbf{B}$, E, and A are constants along the element, they may be taken outside the integral.

$$\mathbf{k}_{\text{axial}} = EA \cdot L \cdot \begin{bmatrix} -\frac{1}{L} \\ \frac{1}{L} \end{bmatrix} \begin{bmatrix} -\frac{1}{L} & \frac{1}{L} \end{bmatrix} = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.8}$$

and for bending deformation the corresponding energy is

$$U_M = \frac{1}{2} \int_0^L EI\, \kappa^2\, dx \tag{2.9}$$

A beam element possesses both transverse displacement (v) and rotation (θ) degrees of freedom at each of its two nodes (a total of four unknowns). Therefore, a third-degree (cubic) polynomial with four coefficients must be used.

$$v(x) = N_1 v_i + N_2 \theta_i + N_3 v_j + N_4 \theta_j \tag{2.10}$$

The corresponding shape functions are defined as,

$$
\begin{aligned}
N_1(x) &= 1 - 3\left(\frac{x}{L}\right)^2 + 2\left(\frac{x}{L}\right)^3 \\
N_2(x) &= x\left(1 - \frac{x}{L}\right)^2 \\
N_3(x) &= 3\left(\frac{x}{L}\right)^2 - 2\left(\frac{x}{L}\right)^3 \\
N_4(x) &= \frac{x^2}{L}\left(\frac{x}{L} - 1\right)
\end{aligned}
\tag{2.11}
$$

The curvature κ is defined as the second derivative of the transverse displacement. By taking the second derivatives of the shape functions, the bending strain–displacement matrix $\mathbf{B}_{\text{bending}}$ is obtained.

$$
\int_0^L EI \left(\frac{d^2N}{dx^2}\right)^T \left(\frac{d^2N}{dx^2}\right) dx \tag{2.12}
$$

By substituting the obtained $\mathbf{B}$ matrix into Equation 2.9 and evaluating the integral, the bending stiffness matrix is obtained.

$$
\mathbf{k}_{\text{bending}} = \begin{bmatrix}
\frac{12EI}{L^3} & \frac{6EI}{L^2} & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\
\frac{6EI}{L^2} & \frac{4EI}{L} & -\frac{6EI}{L^2} & \frac{2EI}{L} \\
-\frac{12EI}{L^3} & -\frac{6EI}{L^2} & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\
\frac{6EI}{L^2} & \frac{2EI}{L} & -\frac{6EI}{L^2} & \frac{4EI}{L}
\end{bmatrix} \tag{2.13}
$$

By expressing the internal deformation measures in terms of nodal displacements using appropriate shape functions, the total potential energy of the system is obtained as the sum of the axial and bending contributions. Differentiating the total potential energy with respect to the nodal degrees of freedom yields the axial and bending element stiffness matrices.

Ultimately, these matrices are assembled into a single 6×6 stiffness matrix for the frame element. The axial element stiffness matrix occupies the rows and columns corresponding to the axial displacements, while the bending element stiffness matrix occupies the rows and columns corresponding to the transverse displacements and rotations.

The element degrees of freedom vector is given as:

$$\mathbf{d}' = \{u_i, v_i, \theta_i, u_j, v_j, \theta_j\}^T \tag{2.14}$$

where u, v, and θ denote the axial displacement, transverse displacement, and rotation at nodes i and j, respectively.

The overall stiffness matrix of the element is obtained by assembling the axial and bending contributions corresponding to these degrees of freedom:

$$\mathbf{k}_{\text{el}} = \mathbf{k}_{\text{axial}} + \mathbf{k}_{\text{bending}}. \tag{2.15}$$

Explicitly, the 6×6 element stiffness matrix can be expressed as:

$$\mathbf{k}_{\text{el}} = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{2.16}$$

2.4 Local-to-Global Coordinate Transformation

Structural elements within a structural system may be oriented at various angles with respect to one another. Although the local coordinate system of each element is used to formulate and evaluate its degree of freedom at the element-level, these quantities must ultimately be transformed into the global coordinate system to perform the overall structural analysis in a consistent framework.

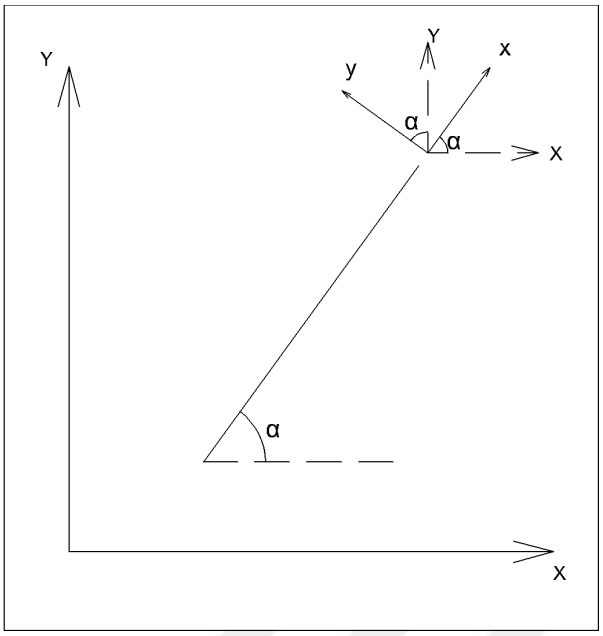


Figure 2.1 Local Coordinates

As illustrated in Figure 2.1 , the local axes of an element may be positioned at an arbitrary angle relative to the global coordinate axes. If the nodal displacements in the global X and Y directions are denoted by D_X and D_Y, respectively, then the corresponding global nodal displacements can be expressed in terms of their horizontal and vertical components as follows:

$$\begin{aligned} u &= \mathbf{D}_X \cos\alpha + \mathbf{D}_Y \sin\alpha \\ v &= -\mathbf{D}_X \sin\alpha + \mathbf{D}_Y \cos\alpha \end{aligned} \tag{2.17}$$

The degree of freedom of rotation at the node is independent of the choice of coordinate system; therefore, the rotational component θ can be taken as unchanged (invariant) under transformation.

$$\theta = 0 \tag{2.18}$$

Rewriting Equations 2.17 and 2.18 in matrix form, the associated rotational transformation matrix $\boldsymbol{r}$ is derived.

$$\mathbf{r} \quad \Rightarrow \quad \begin{Bmatrix} u \\ v \\ \theta' \end{Bmatrix} = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} D_X \\ D_Y \\ \theta \end{Bmatrix} \tag{2.19}$$

Where:
c: cos$a$, s: sin$a$

Since a frame element has two nodes (i and j), the full 6×6 transformation matrix ($\mathbf{T}$) is constructed by placing the $\mathbf{r}$ matrices in a block-diagonal form:

$$\mathbf{T} = \begin{bmatrix} \mathbf{r} & \mathbf{0} \\ \mathbf{0} & \mathbf{r} \end{bmatrix} = \begin{bmatrix} c & s & 0 & 0 & 0 & 0 \\ -s & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & c & s & 0 \\ 0 & 0 & 0 & -s & c & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

Once the transformation matrix is established, the element stiffness matrix computed in the local coordinate system is transformed to the global coordinate system using the following relation:

$$\mathbf{k}_{\text{g}} = \mathbf{T}\, \mathbf{k}_{\text{el}}\, \mathbf{T}^{T} \tag{2.21}$$

2.5 Assembly of the Global Stiffness Matrix

In finite element analysis, the stiffness matrices formulated in local coordinate systems for each element must be systematically assembled to construct the Global Stiffness Matrix, which characterizes the overall structural response. The mathematical foundation of this assembly process lies in defining the topological relationship between the degrees of freedom of individual elements and the corresponding global degrees of freedom of the structural system. In this context, the topology matrix functions as a mapping tool that specifies how each local degree of freedom is associated with its counterpart in the global system.[15]

In the topology matrix, the rows correspond to the elements, while the columns represent the nodes and possible releases. The matrix identifies the nodes to which each element is connected, together with any releases assigned to that element. Releases are incorporated by introducing additional columns and marking the corresponding entries in the relevant rows.[16]

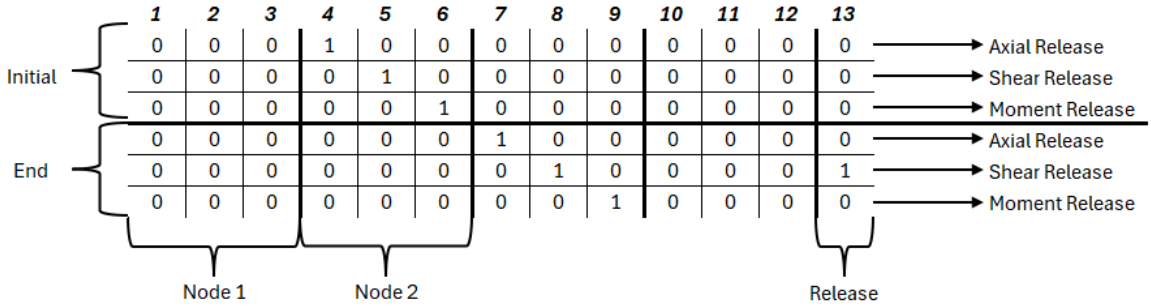


Figure 2.2 Topology Matrix

The topology matrix illustrated in Figure 2.2 corresponds to the system shown in Figure 2.3 and represents the second element of a simple three-element framework, where a shear release is defined at the right end of the element.

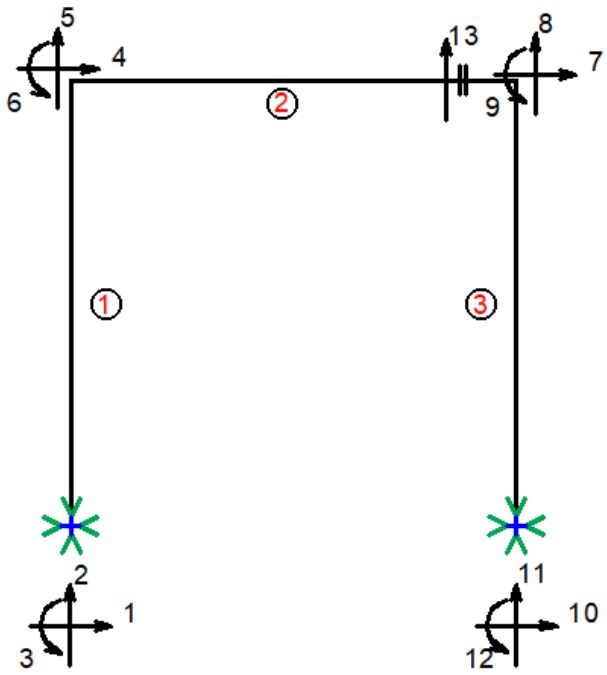


Figure 2.3 Example System

The stiffness matrices of the elements have previously been computed in the global coordinate system. These matrices are now expanded to the dimensions of the system using the topology matrix.

$$\mathbf{K}_{\text{system}} = \sum_{i=1}^{n} \left(\mathbf{L}^{(i)T} \, \mathbf{k}_{\text{global}}^{(i)} \, \mathbf{L}^{(i)} \right) \tag{2.22}$$

Where:

$\mathbf{K}_{\text{system}}$: Global stiffness matrix of the system,

n: Total number of elements,

$\mathbf{k}_{\text{global}}^{(i)}$: Stiffness matrix of the i th element expressed in the global coordinate system after coordinate transformation,

$\mathbf{L}^{(i)}$: Boolean topology matrix of the i th element.

In the example system in Figure 2.3, each matrix of stiffness of the element is of size 6×6, corresponding to the degrees of freedom of the element. Due to the presence of releases, the associated element topology matrices are of size 6×13. Using formulation 2.22, the global stiffness matrix of the system is obtained with dimensions 13×13, representing the total number of active degrees of freedom in the system.

2.6 Load Vector Formulation

The fundamental principle in the formation of the global load vector is that all physical effects acting on the structure must be reduced to the nodes of the system. Since the finite element method performs the matrix-based solution with respect to the degrees of freedom at the nodes, loads acting along the spans or bodies of the elements must also be mathematically transferred to the nodal points.

In this context, loads are generally classified into two main categories based on their mode of application to the system, and the global load vector is formed by superposition of these contributions.

$$\mathbf{F} = F^{\text{nodal}} + F^{\text{equiv}} \tag{2.23}$$

Where:

F^{nodal}: Forces and moments applied directly at node, F^{equiv}: Equivalent nodal forces and moments obtained from element loads through work-equivalent (consistent) nodal load vectors.

2.6.1 Nodal Loads

These consist of concentrated forces or moments that act directly on the predefined nodes of the system. The transfer of such loads to the global load vector is a direct process that does not require additional transformation; the load value is simply assigned to the vector entry corresponding to the relevant degree of freedom. For a two-dimensional frame system with three degrees of freedom per node, the nodal load vector can be expressed as:

$$\mathbf{F}^{\text{nodal}} = \begin{bmatrix} F_{x1}^{\text{nodal}} \\ F_{y1}^{\text{nodal}} \\ M_{z1}^{\text{nodal}} \\ F_{x2}^{\text{nodal}} \\ F_{y2}^{\text{nodal}} \\ M_{z2}^{\text{nodal}} \end{bmatrix} \tag{2.24}$$

2.6.2 Element Loads

In the formulation of the global load vector, different types of external effects acting on the structure require a distinct treatment depending on their mode of application. Loads such as point loads and distributed loads act along the element length and must therefore be transferred to the nodal degrees of freedom using shape functions and equivalent nodal force formulations. However, Temperature-induced effects are incorporated into the system through equivalent nodal forces or moments derived from analytically, depending on whether the temperature change is uniform or varies across the depth of the element. These approaches ensure that all external actions are consistently represented at the nodal level and can be directly included in the global equilibrium equations.

$$\mathbf{F}^{\text{equiv}} = \begin{bmatrix} F_1^{\text{equiv}} \\ F_2^{\text{equiv}} \\ F_3^{\text{equiv}} \\ F_4^{\text{equiv}} \\ F_5^{\text{equiv}} \\ F_6^{\text{equiv}} \end{bmatrix} = \int q_i(x) \begin{bmatrix} N_1(x) \\ N_2(x) \\ N_3(x) \\ N_4(x) \\ N_5(x) \\ N_6(x) \end{bmatrix} dx \tag{2.25}$$

2.6.2.1 Point Loads

For loads acting on structural elements at arbitrary orientations the resulting response of the element depends on the decomposition of the load with respect to the local element coordinate system. Since such loads are generally applied along the element span rather than directly at the nodal points, they cannot be incorporated into the global equilibrium equations in their original form. Instead, the load components are transferred to the nodal degrees of freedom through equivalent nodal force formulations. This transformation is carried out using appropriate shape functions associated with the relevant deformation modes of the element, ensuring that the effects of loads acting in different directions are consistently represented within the finite element framework.

Consider a beam element of length L subjected to a point load q acting in the local transverse (z) direction at a distance x from the start node as illustrated in the figure 2.4

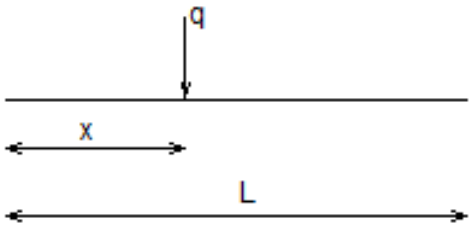


Figure 2.4 Point Load Example

The cubic Hermite shape functions used for transverse loading are given as:

$$\begin{aligned} N_1(x) &= 0, \\ N_2(x) &= 1 - 3\frac{x^2}{L^2} + 2\frac{x^3}{L^3}, \\ N_3(x) &= x - 2\frac{x^2}{L} + \frac{x^3}{L^2}, \\ N_4(x) &= 0, \\ N_5(x) &= 3\frac{x^2}{L^2} - 2\frac{x^3}{L^3}, \\ N_6(x) &= -\frac{x^2}{L} + \frac{x^3}{L^2}. \end{aligned} \tag{2.26}$$

These shape functions correspond to:

- N_2: transverse displacement at the start node,
- N_3: rotation at the start node,
- N_5: transverse displacement at the end node,
- N_6: rotation at the end node.

The equivalent nodal force vector due to a point load q applied at position x is computed as:

$$\mathbf{F}^{\text{equiv}} = -q \begin{bmatrix} N_1(x) \\ N_2(x) \\ N_3(x) \\ N_4(x) \\ N_5(x) \\ N_6(x) \end{bmatrix}. \tag{2.27}$$

If multiple point loads act on the same element, the total equivalent nodal force vector is obtained by superposition:

$$\mathbf{F}_{\text{total}} = \sum_{i=1}^{n_p} \mathbf{F}_{(i)}^{equiv} \tag{2.28}$$

where n_p denotes the number of point loads acting on the element.

Finally, the resulting equivalent nodal force vector is added to the element load vector and subsequently assembled into the global load vector through the element topology and transformation matrices.

2.6.2.2 Distributed Loads

In contrast to point loads, distributed loads act continuously along the element length and therefore contribute to the nodal equilibrium through an integral formulation. Once the load components are expressed in the local coordinate system of the element, the equivalent nodal forces are obtained by integrating the product of the distributed load function and the corresponding shape functions over the element domain. This approach ensures that the work done by the distributed load is consistently transferred to the nodal degrees of freedom.

For an element subjected to a distributed load $q(x)$ acting along an arbitrary direction, the load distribution is first represented as a continuous function along the element span. In practical implementations, this function may be defined in a

piecewise manner, allowing nonuniform load intensities to be described through linear interpolation between specified points, as illustrated in Figure 2.5. Each segment of the load distribution is treated independently and its contribution to the nodal force vector is evaluated through numerical integration.

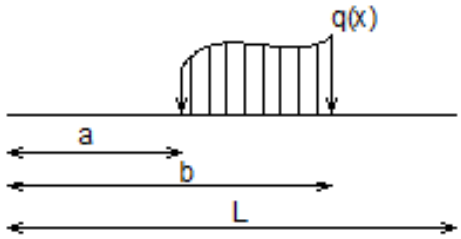


Figure 2.5 Distributed Load Example

The equivalent nodal force corresponding to a given degree of freedom is obtained by integrating the product of the distributed load and the associated shape function over the segment length. For transverse loading, cubic Hermite shape functions are employed to account for both nodal displacements and rotations. The resulting nodal force components are computed as

$$F_i = \int_a^b q(x)\, N_i(x)\, \mathrm{d}x, \tag{2.29}$$

where $N_i(x)$ denotes the relevant shape function and $[a, b]$ defines the limits of the load segment. Numerical integration schemes, such as Gauss–Legendre quadrature, are used to accurately evaluate these integrals.

The equivalent nodal force vectors obtained from each load segment are then superimposed to form the total element load vector in the local coordinate system. This vector is subsequently combined with other element load contributions and transformed to the global coordinate system through the element transformation and topology matrices before being assembled into the global load vector.

Through this integral-based formulation, distributed loads of arbitrary magnitude, orientation, and variation along the element length can be systematically incorporated into the finite element model while maintaining consistency with the nodal equilibrium equations.

2.6.2.3 Temperature Loads

Thermal loads constitute a special class of element loads, as they do not arise from externally applied forces but from restrained thermal deformations of the structural element. Temperature variations induce axial strains and bending curvatures depending on whether the temperature change is uniform across the section or varies through its depth. Since these effects do not act directly at the nodes, their contribution to the equilibrium equations is introduced through equivalent nodal force and moment vectors.

For a beam element subjected to a uniform temperature change ΔT, the free thermal strain is given by

$$\varepsilon_{\text{th}} = \alpha \Delta T, \tag{2.30}$$

where α denotes the coefficient of thermal expansion. If the axial deformation associated with this strain is restrained, an equivalent axial force develops within the element. The corresponding thermal axial force is expressed as

$$F_{\text{th}} = EA\alpha \Delta T, \tag{2.31}$$

where E is the Young's modulus and A is the cross-sectional area of the element. This force is introduced into the element load vector as equal and opposite axial forces acting at the start and end nodes, ensuring internal equilibrium.

In the case of a temperature gradient through the element depth, bending effects arise due to differential thermal expansion. Assuming a linear temperature variation across the section height h, the thermal curvature is proportional to the temperature gradient

$$\frac{\mathrm{d}T}{\mathrm{d}z} = \frac{\Delta T}{h}. \tag{2.32}$$

The resulting equivalent thermal bending moment is given by

$$M_{\text{th}} = EI\alpha \frac{\Delta T}{h}, \tag{2.33}$$

where I denotes the second moment of area of the cross-section. This thermal moment is incorporated into the element load vector as nodal moments of equal magnitude and opposite sign at the start and end nodes.

The equivalent nodal load vectors derived from uniform temperature change and temperature gradients are directly added to the element load vector in the local coordinate system. These contributions are then assembled into the global load vector through the standard transformation and topology operations used in

the finite element formulation. In this manner, thermal effects are consistently accounted for within the global equilibrium equations without the need for explicit shape function integration.

2.7 Boundary Conditions

After the global stiffness matrix ($\mathbf{K}$) and the global load vector ($\mathbf{F}$) of the system are assembled, the equilibrium equation of the structure can be written as

$$\mathbf{K} \cdot \mathbf{D} = \mathbf{F} \tag{2.34}$$

At this stage, the matrix $\mathbf{K}$ is still singular, meaning that its determinant is zero ($|\mathbf{K}| = 0$). From a physical standpoint, this indicates that the structure has no restraints in space and is therefore capable of undergoing rigid body motion. In order for the system to be statically determinate and stable, and hence for the equilibrium equations to be solvable, it is necessary to impose appropriate boundary conditions on the unknown displacement vector $\mathbf{D}$.

2.7.1 Support Conditions

Support conditions correspond to cases in which the displacements associated with certain degrees of freedom of the system are constrained to be zero (homogeneous boundary conditions). For example, at a fixed support, the horizontal displacement, vertical displacement, and rotation are all zero ($u = 0$, $v = 0$, $\theta = 0$). In the case of a pinned support, only the translational degrees of freedom are constrained, while rotation is free.

To incorporate these boundary conditions into the numerical solution, the row and column elimination method (also referred to as matrix reduction) is employed. In this approach, the constrained degrees of freedom corresponding to zero displacements are explicitly removed from the system of equations.

In the present example, fixed supports are assumed at Nodes 1 and 4. Consequently, the first three degrees of freedom associated with Node 1 and the last three degrees of freedom associated with Node 4 are constrained, i.e., their displacements and rotations are prescribed as zero. The reduction procedure is carried out as follows:

- The constrained degrees of freedom for which the displacements are known to be zero ($D_r = 0$) are identified.

- The corresponding rows and columns of the global stiffness matrix $\mathbf{K}$ are removed.
- In a consistent manner, the same entries are eliminated from the global load vector $\mathbf{F}$ and the displacement vector $\mathbf{D}$.

A schematic illustration of this reduction process, showing the eliminated rows and columns of the global stiffness matrix due to the fixed supports at Nodes 1 and 4, is presented in Figure 2.6.

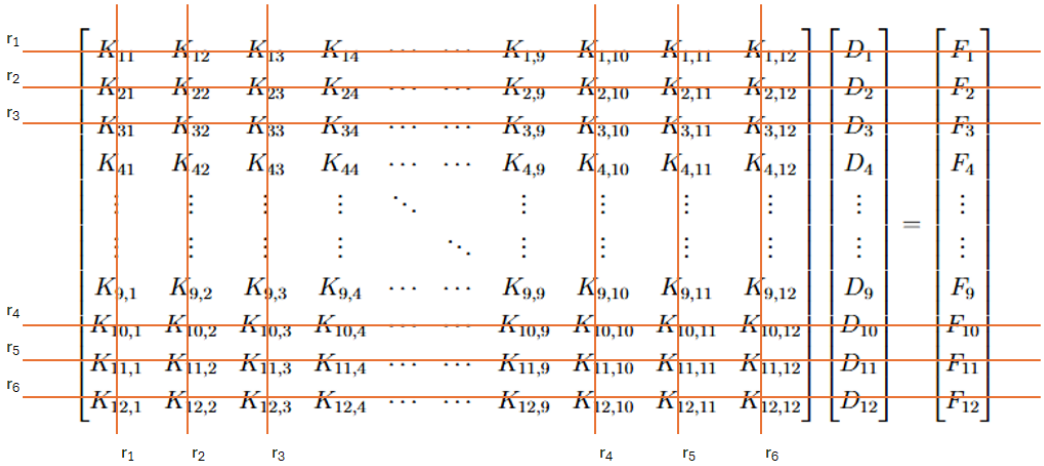


Figure 2.6 Reduction of Global Stiffness Matrix

As a result of this procedure, the size of the system of equations is reduced, and the stiffness matrix becomes non-singular and therefore invertible. The reduced equilibrium equations are written as

$$\mathbf{K}_r \cdot \mathbf{D}_r = \mathbf{F}_r,$$

where $\mathbf{K}_r$ contains only the stiffness terms associated with the unconstrained (free) degrees of freedom. The corresponding displacement vector $\mathbf{D}_r$ is then obtained as

$$\mathbf{D}_r = \mathbf{K}_r^{-1} \cdot \mathbf{F}_r.$$

2.7.2 Prescribed Displacements

In structural analysis, there are cases in which a prescribed amount of displacement or rotation is intentionally imposed on otherwise unconstrained (free) nodal degrees of freedom. Unlike support settlement, this procedure represents the application of an external kinematic constraint directly to the degrees of freedom of the system.

In the developed software, this functionality is implemented based on a systematic classification of the global degrees of freedom. The complete set of degrees of freedom (DOFs) of a structure is partitioned into three mutually exclusive subsets:

- **Fixed DOFs**: Degrees of freedom restrained by supports, whose values are known to be exactly zero ($u = 0$).
- **Prescribed DOFs**: Degrees of freedom that are normally free but are constrained by the user to specified nonzero values ($u = \delta$).
- **Free DOFs**: Remaining degrees of freedom whose values are unknown and determined as part of the structural analysis ($u = ?$).

2.7.2.1 Algorithm and Logical Filtering

To ensure the physical consistency of the prescribed displacements provided by the user, a preliminary logical filtering step is applied within the algorithm. The logical condition implemented in the code can be expressed as

$$S_{\text{prescribed}} = \{\text{dof} \in S_{\text{input}} \mid \text{dof} \notin S_{\text{fixed}}\}. \tag{2.35}$$

This operation prevents the assignment of prescribed displacements to degrees of freedom that are already restrained by supports. In other words, imposed displacements are allowed to act only on unrestrained (free) degrees of freedom. This filtering step eliminates physically inconsistent configurations, such as a degree of freedom being simultaneously fixed and allowed to move.

2.7.2.2 Mathematical Formulation (Matrix Modification)

Degrees of freedom subjected to prescribed displacements are mathematically removed from the set of unknowns and treated as known quantities. Accordingly, the global equilibrium equations are reformulated as

$$\begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fp} \\ \mathbf{K}_{pf} & \mathbf{K}_{pp} \end{bmatrix} \begin{Bmatrix} \mathbf{D}_f \\ \mathbf{D}_p \end{Bmatrix} = \begin{Bmatrix} \mathbf{F}_f \\ \mathbf{F}_p \end{Bmatrix}. \tag{2.36}$$

Here, the subscript p denotes *prescribed* degrees of freedom, while f refers to the remaining *free* degrees of freedom. The vector $\mathbf{D}_p$ consists of displacement values specified by the user.

Rearranging the system to solve for the unknown displacement vector $\mathbf{D}_f$ yields

$$\mathbf{K}_{ff}\mathbf{D}_f = \mathbf{F}_f - \mathbf{K}_{fp}\mathbf{D}_p. \tag{2.37}$$

In the software implementation, this formulation is realized through the following steps:

- The user-defined prescribed displacement vector $\mathbf{D}_p$ is assembled.
- The equivalent constraint-induced force vector is computed via the product $\mathbf{K}_{fp}\mathbf{D}_p$.
- This vector is subtracted from the original external load vector $\mathbf{F}_f$ to obtain the effective load acting on the free degrees of freedom.

The contribution of the prescribed displacements can be interpreted as an equivalent load acting on the free degrees of freedom. Accordingly, the effective load vector is defined as

$$\mathbf{F}_f^{\text{eff}} = \mathbf{F}_f - \mathbf{K}_{fp}\mathbf{D}_p \tag{2.38}$$

2.8 Solution of the Global System

Following the enforcement of support conditions and prescribed displacements, the global equilibrium equations are reduced to a system involving only the unknown (free) degrees of freedom. At this stage, all kinematic constraints have already been incorporated into the formulation, and the reduced stiffness matrix $\mathbf{K}_{ff}$ is guaranteed to be non-singular.

The structural response is therefore obtained by solving the reduced linear system

$$\mathbf{K}_{ff}\mathbf{D}_f = \mathbf{F}_f^{\text{eff}}, \tag{2.39}$$

where $\mathbf{D}_f$ represents the vector of unknown nodal displacements and $\mathbf{F}_f^{\text{eff}}$ denotes the effective load vector acting on the free degrees of freedom.

The solution of this system is carried out using standard numerical linear algebra techniques, yielding

$$\mathbf{D}_f = \mathbf{K}_{ff}^{-1}\mathbf{F}_f^{\text{eff}}. \tag{2.40}$$

Once the free displacement components are obtained, the complete global

displacement vector is reconstructed by combining the free, prescribed, and fixed degrees of freedom. These global displacements are then mapped to the element level using the element topology and coordinate transformation matrices.

Finally, internal element force vectors and reaction forces at constrained degrees of freedom are evaluated using the original (unreduced) stiffness relations. This post-processing stage enables the computation of internal force diagrams, stress resultants, and support reactions, thereby completing the structural analysis workflow.

2.8.1 Element End Forces

After the global displacement vector has been fully determined, the internal force state of each element is evaluated through a two-stage transformation procedure.

In the first stage, the global displacement vector is reduced to the element level while remaining in the global coordinate system. This is achieved by extracting the displacement components associated with the element degrees of freedom using the element topology matrices. The resulting vector represents the element displacement vector expressed in global coordinates,

$$\mathbf{d}_e^{(g)} = \mathbf{L}_e \mathbf{D} \tag{2.41}$$

where $\mathbf{D}$ is the global displacement vector and $\mathbf{L}_e$ denotes the element topology matrix that maps global degrees of freedom to the element-level degrees of freedom.

In the second stage, the element displacement vector is transformed from the global coordinate system to the local element coordinate system. This transformation is performed using the element transformation matrix,

$$\mathbf{d}_e^{(l)} = \mathbf{T}_e^{\mathrm{T}} \mathbf{d}_e^{(g)}, \tag{2.42}$$

where $\mathbf{T}_e$ is the transformation matrix relating local and global coordinates.

Once the local element displacement vector is obtained, the local element end force vector is computed using the standard stiffness relationship

$$\mathbf{f}_e^{(l)} = \mathbf{k}_e \mathbf{d}_e^{(l)} - \mathbf{f}_e^{\text{load}}, \tag{2.43}$$

where $\mathbf{k}_e$ is the local element stiffness matrix and $\mathbf{f}_e^{\text{load}}$ is the equivalent nodal load vector accounting for element-level loads such as point loads, distributed loads, and thermal effects.

The resulting vector $\mathbf{f}_e^{(l)}$ contains the axial forces, shear forces, and bending moments acting at the start and end nodes of the element in the local coordinate system. These element end forces constitute the primary input for subsequent post-processing steps, including the construction of internal force and moment diagrams.

2.8.2 Internal Force and Moment Diagrams

Although the end forces of the elements describe the state of the internal force at discrete nodal locations, engineering design and evaluation require knowledge of the variation of internal forces along the length of the element. Internal force and moment diagrams are therefore constructed by evaluating the sectional forces at arbitrary points within each element.

The internal force vector at a position x along an element is obtained by relating the element displacement field to the corresponding stress resultants through the element shape functions. Using displacement interpolation functions, the axial force $N(x)$, the shear force $V(x)$, and the bending moment $M(x)$ are calculated as continuous functions of the local coordinate x.

Element-level loads, including distributed loads, point loads, and thermal effects, are consistently incorporated into the sectional force expressions. This ensures that equilibrium is satisfied not only at the element nodes but also at any intermediate cross-section.

The resulting internal force and moment diagrams provide a complete representation of the structural response and serve as the main output of structural design checks, such as strength verification, serviceability assessment, and detail.

3
SOFTWARE DEVELOPMENT PROCESS

This thesis introduces a software developed for plane frame analysis based on the finite element method (FEM). The objectives of the development were twofold: to guarantee the accuracy and reliability of engineering calculations and to provide a seamless user interaction.

This section presents the programming languages, compilers, libraries, and class structures employed in the software development process, as well as the user interface and external integrations. The overall workflow diagram is provided below In Figure 3.1.

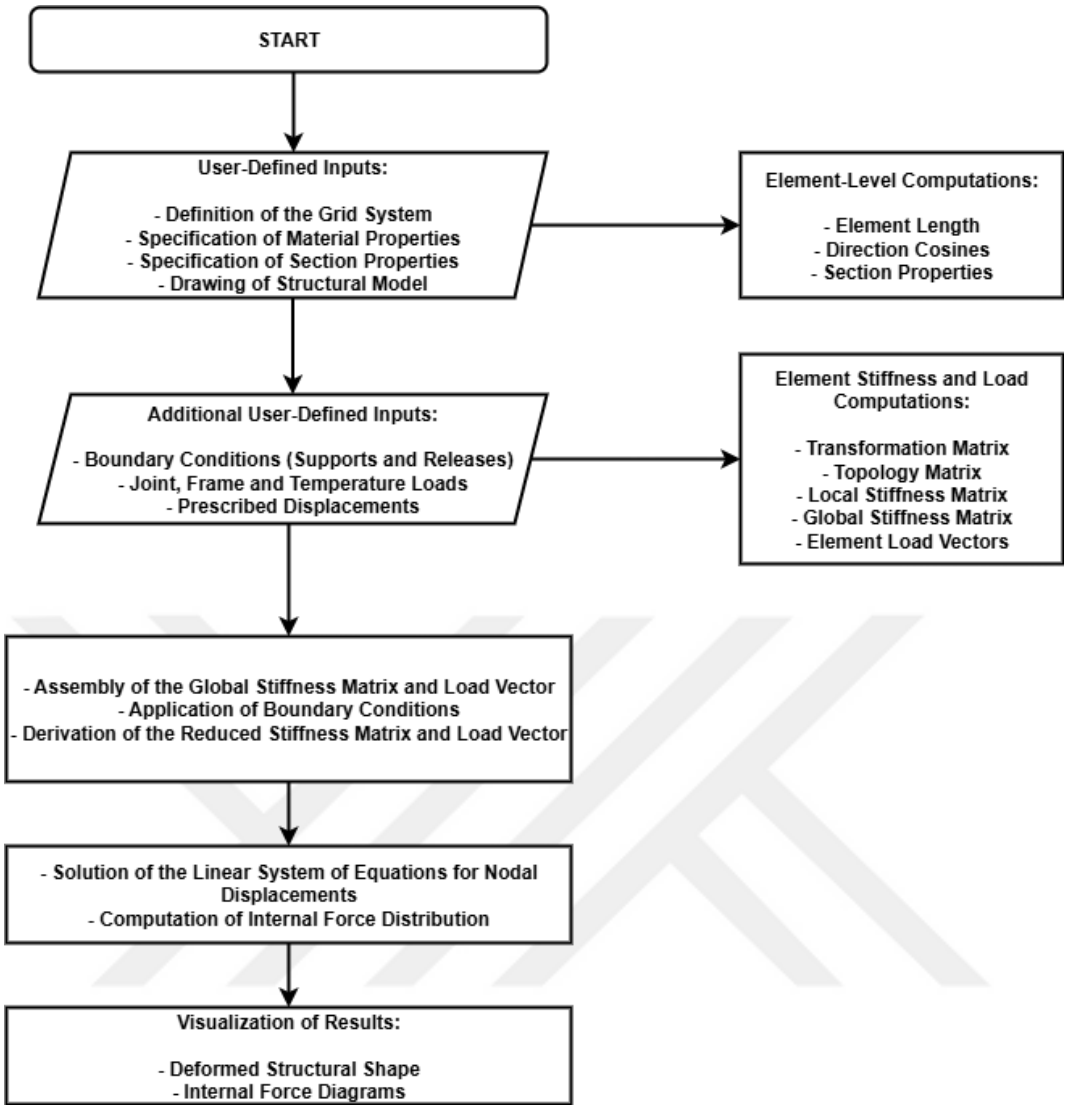


Figure 3.1 Workflow

3.1 Programming Language and Tools

3.1.1 Programming Language

For this thesis work, C# was chosen as the programming language due to its relatively easy usage, modern features, open-source nature and efficiency considering the intensity of mathematical operations. Alongside C#, SQLite was selected as the database management system to store user inputs. The C# libraries used in the project are presented in Table 3.1.

Table 3.1 Project-specific and Third-party Libraries

Libraries	Brief Description
OxyPlot	Graph plotting library for data visualization.
MathNet.Numerics	Mathematical operations, linear algebra, statistics, and optimization.
MathNet.Numerics. LinearAlgebra	Linear algebra classes (Vector, Matrix).
MathNet.Numerics. LinearAlgebra.Double	Double-precision matrix and vector operations.

3.1.2 Development Environment

The software developed in this study was implemented in the C# programming language using the Microsoft Visual Studio environment. The user interface of the application was designed based on Windows Forms and the program was tested on the Windows operating system. This development environment was selected due to its high performance, widespread adoption, extensive library support and user-friendly interface [17–19].

3.2 Software Architecture and Class Structure

3.2.1 Core Classes

The software developed within the scope of this thesis study is structured around three primary classes. The GeometryHelper class incorporates the drawing utilities, the LineElement class defines the characteristics of line elements and the Calcs class performs all mathematical computations required for the finite element analysis.

3.2.1.1 GeometryHelper Class

The GeometryHelper class defines all drawing tools utilized within the program. These tools are used for the graphical representation of support reactions, concentrated loads, distributed loads, and release assignments. Appendix A.1

presents the script of the support reactions. The support reactions drawn using the code provided in Listing A.1 are illustrated in Figure 3.2.

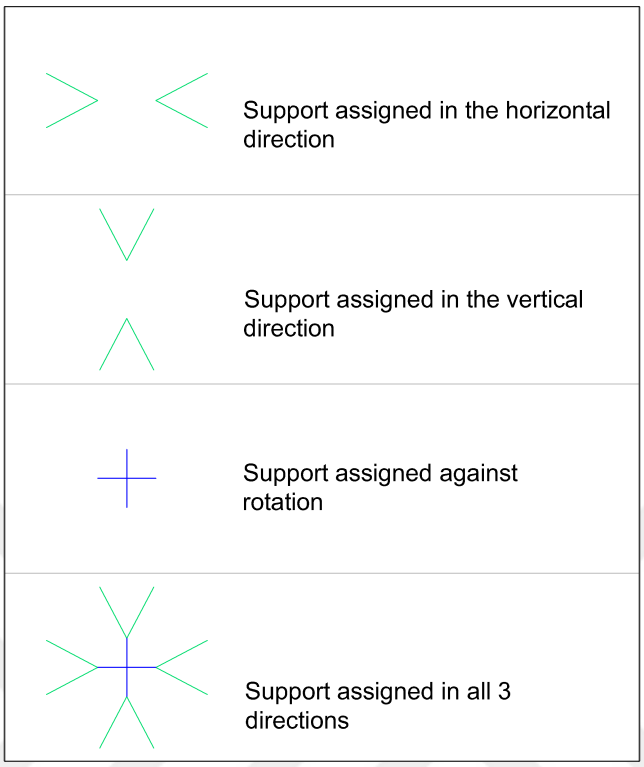


Figure 3.2 Support Reactions

3.2.1.2 LineElement Class

The LineElement class is a comprehensive structure that stores all characteristic data of the line elements. It contains information regarding the position of the elements in the global system, node and element numbers, element lengths and orientations, as well as their cross-sectional area, moment of inertia, and modulus of elasticity. In addition, it includes essential matrices such as the transformation matrix, topology matrix, and stiffness matrices. As shown in Appendix A.2

3.2.1.3 Calcs Class

The Calcs class serves as the central module that performs all mathematical operations required in the analysis. It is invoked during the computational procedures and provides methods for the generation of element transformation matrices, the topology matrix and stiffness matrices, as well as for displacement calculations and the solution of linear systems.

3.2.2 Advantages Of Object-Oriented Programming

Within the object-oriented programming approach, information such as node points, elements, material types and cross-sections are modularly stored in separate classes, allowing for repeated use in the computational process. In this way, the management of methods, classes and objects is facilitated, while code readability is substantially improved. More specifically, in the present study, the TransformationMatrix method within the Calcs class makes use of the ElementCos and ElementSin (the implementation of which is provided in Appendix A.3) methods to construct a transformation matrix for each element. This method has been employed 17 times throughout the calculations, whereby redundant lines of code are avoided and overall readability is significantly enhanced.

3.2.3 Visualization of Structural Analysis Results

In the visualization of structural analysis results, both external libraries and previously defined classes were utilized within a Windows Forms environment. The OxyPlot library (OxyPlot.Series, OxyPlot.Axes, OxyPlot.WindowsForms) was employed to generate graphical elements, with the overall chart structure defined using the PlotModel class, customizable axes created through LinearAxis, and structural data represented using LineSeries. This approach allowed the visualization of the system's undeformed and deformed configurations, as well as internal force diagrams.

The numerical data required for the diagrams were obtained during the analysis phase and stored within the relevant classes, from which they were later retrieved for plotting. The plotting process is carried out via methods specifically designed for either the deflected shape or the force diagrams.As an example, the implementation script of the DeformedShape form is presented in Appendix A.4

3.3 User Interface

3.3.1 Creating Coordinate System

The coordinate system is generated through the Create Grid System button, which allows the creation of horizontal and vertical axes in the desired number and spacing as shown in Figure 3.3

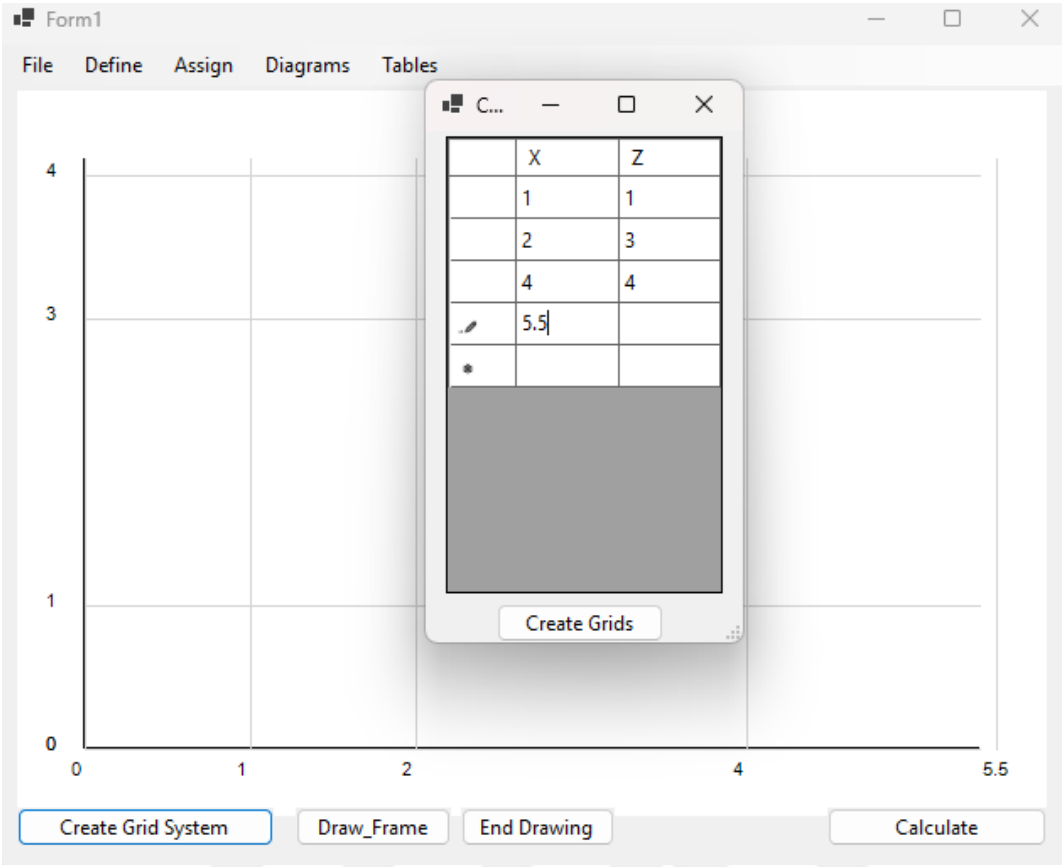


Figure 3.3 Coordinate System Form

3.3.2 Material Definition

The definition of structural elements is carried out in two stages. In the first stage, the material properties are specified, while in the second stage, the element cross-section is defined and the corresponding material is assigned. The material definition is accessed through the Define menu under the Material section. Within this interface, existing materials can be displayed, modified, or newly assigned as Shown in Figure 3.4. The information associated with the defined material is subsequently stored in a database created using SQLite [20].

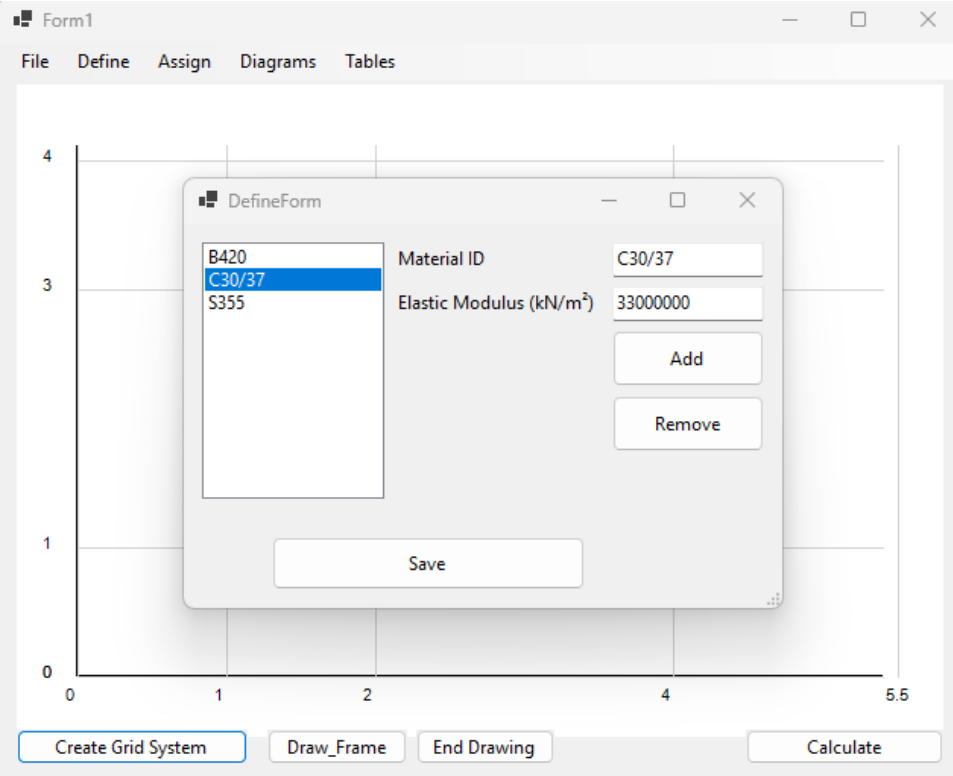


Figure 3.4 Material Define Form

3.3.3 Cross-Section Definition

In the second stage, the cross-section is assigned to the element. This procedure is accessed through the Define menu under the Section heading. Within this interface, cross-sections with various geometric configurations (rectangular, rectangular tube, circular, circular tube) can be defined as shown in Figure 3.5.

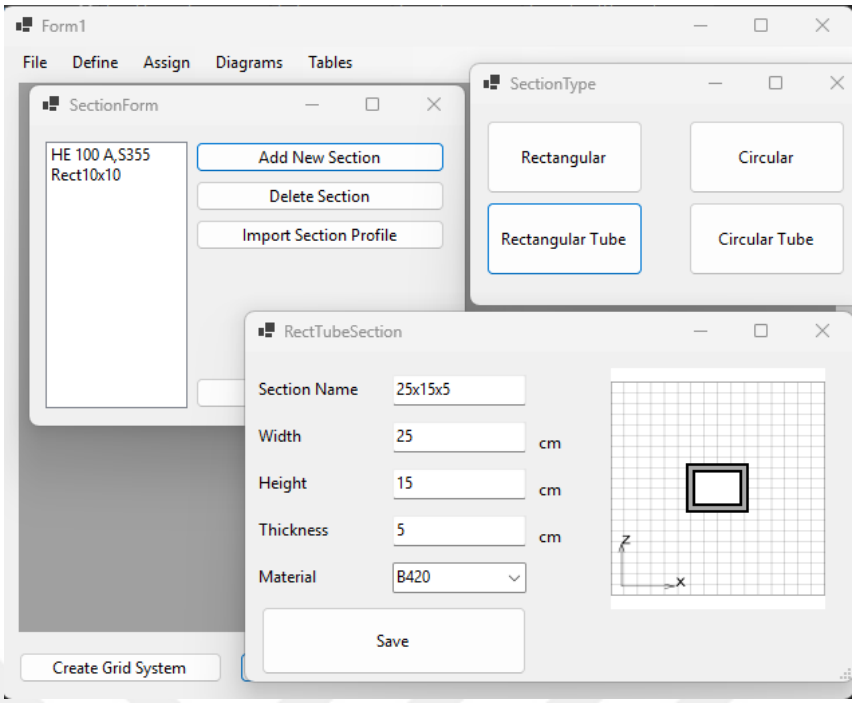


Figure 3.5 Section Define Form

Also specific profiles can be imported from the database as shown in Figure 3.6.

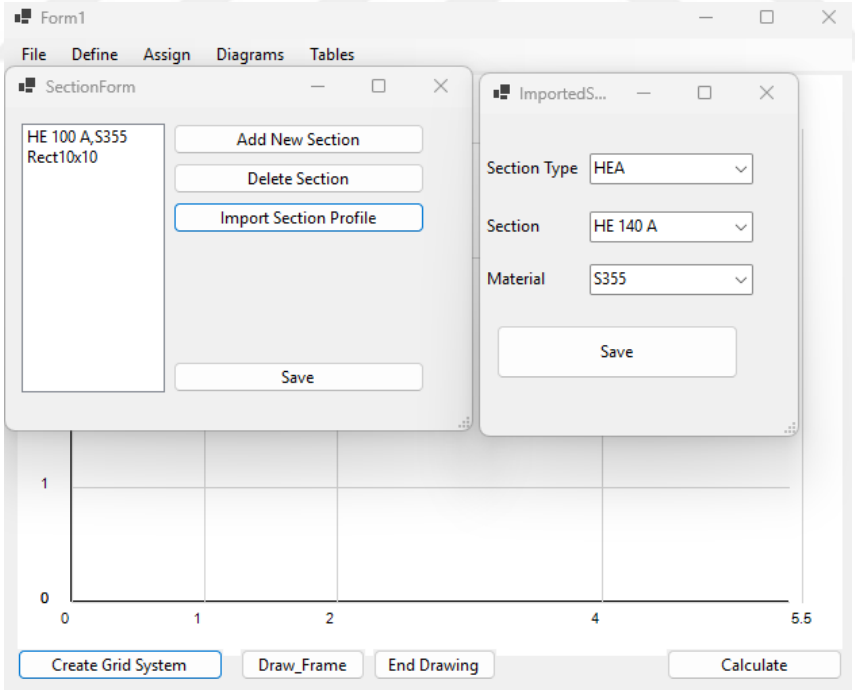


Figure 3.6 Section Import Form

3.3.4 Generation of the frame model

By clicking the Draw Frame button, the software enters the frame drawing mode. In this mode, nodes are defined by selecting points on the coordinate system, and line elements connecting these nodes are subsequently drawn. The drawing process can be temporarily paused by right-clicking and resumed by pressing the Draw Frame button again. Once the drawing is completed, it is finalized by selecting the End Drawing button. Upon completion of the frame generation process, the element and node numbers are visually displayed within the system as shown in Figure 3.7.

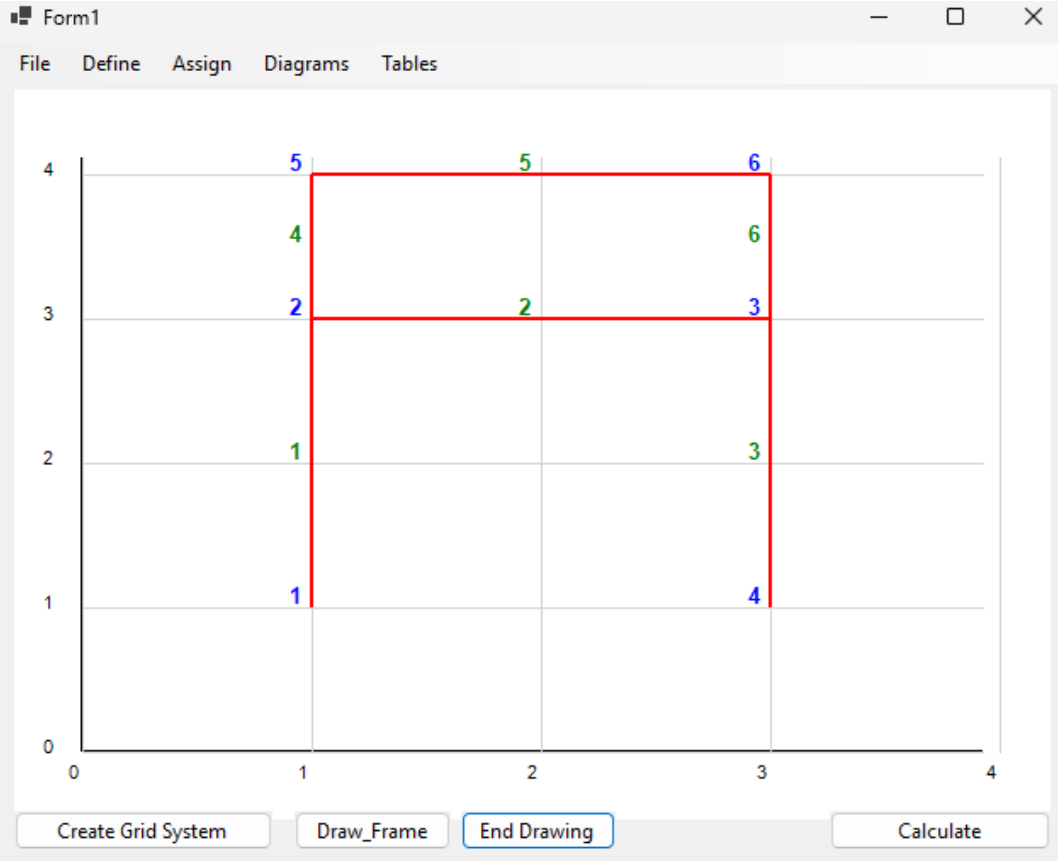


Figure 3.7 Generating the Frame Model

3.3.5 Cross-Section Assignment

Previously defined sections are assigned to the frame elements that have been drawn. At this stage, any desired section can be assigned to any selected element. The assignment process is carried out through the Assign Section form, where the target elements and the corresponding section are selected, and the Assign button is clicked. The section assignment procedure is illustrated in Figure 3.8.

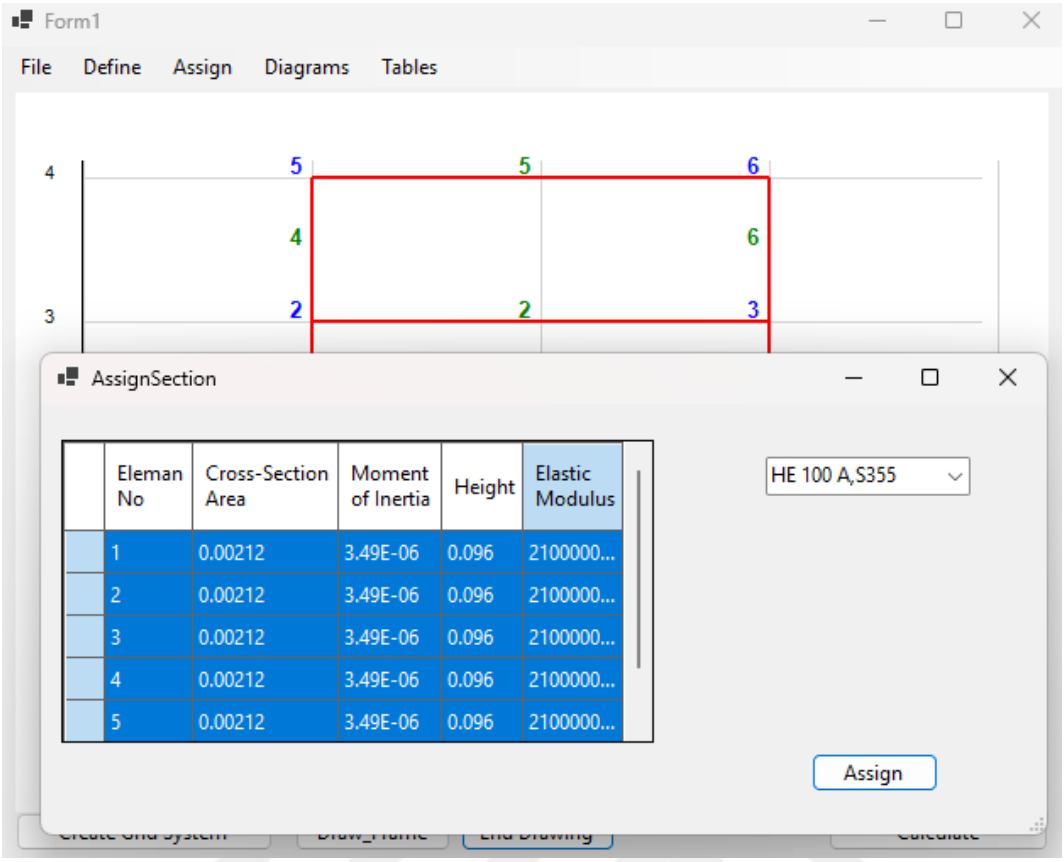


Figure 3.8 Assigning Cross-Section

3.3.6 Restriction Assignment

After the structural system is drawn, support assignments are applied to the model. This process is carried out by imposing constraints in any desired direction at any selected node. Unlike the conventional fixed, roller, or pinned support types, this approach provides a more flexible method for defining support conditions. The restriction assignment procedure is illustrated in Figure 3.9 and Appendix A.5.

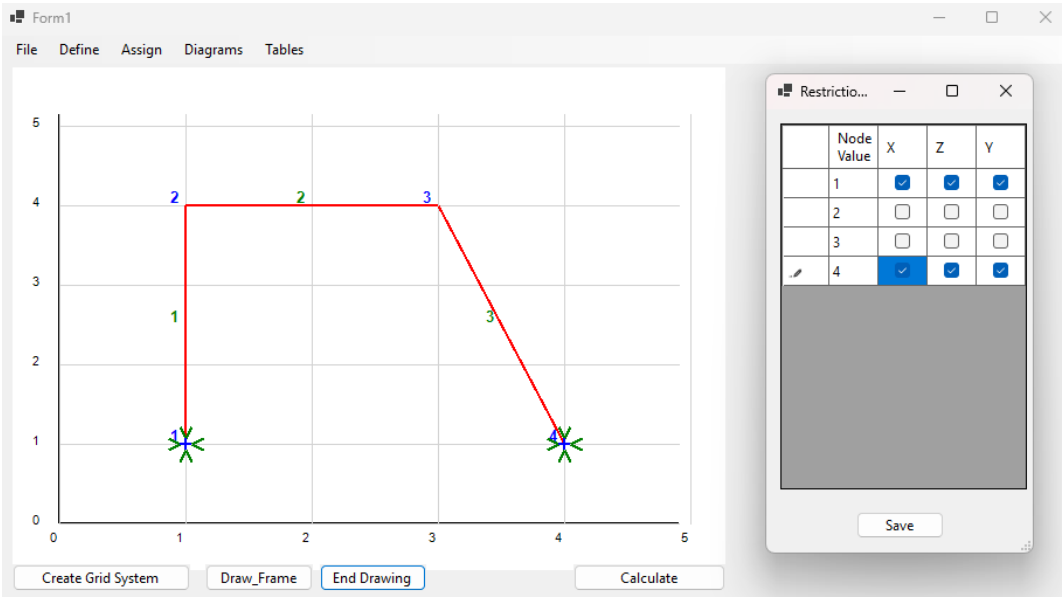


Figure 3.9 Assigning Restrictions

3.3.7 Frame Release Assignment

Under the Assign menu, frame releases can be applied to any desired element in any specified direction. Once this operation is performed, the element topology matrix is expanded and updated according to the number of assigned releases. In addition, the location and orientation of the applied releases are visually represented within the drawing interface. The frame release assignment procedure is illustrated in Figure 3.10 and Appendix A.6.

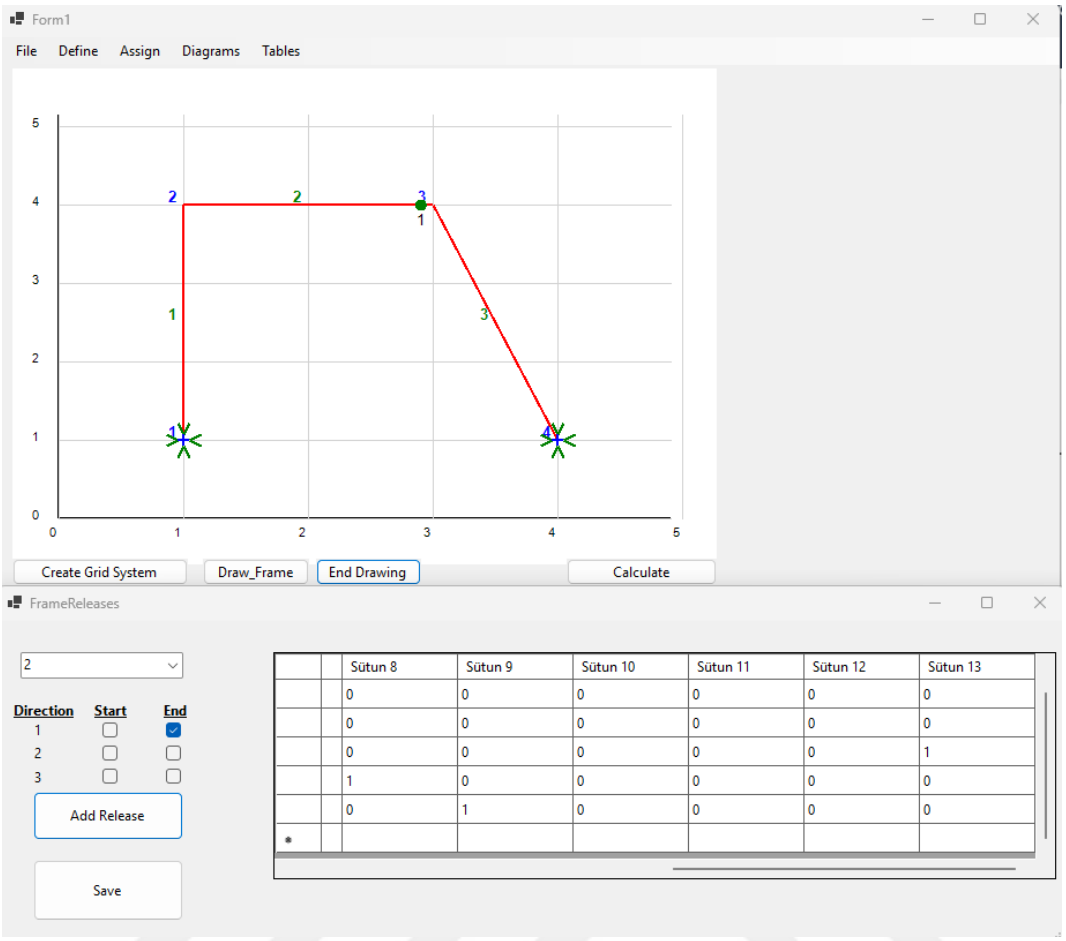


Figure 3.10 Assigning Frame Releases

3.3.8 Load Assignments

Load assignments can be performed through the Assign menu. The system allows the application of various types of loads, including point loads at nodes, concentrated and distributed loads acting on elements, and temperature loads. The loads assigned to the nodes are directly added to the load matrix, whereas the loads acting on the elements are converted into equivalent nodal loads before being incorporated into the matrix. This process is illustrated in Figure 3.11, which shows an applied nodal force at Node 3, a point load on Element 1, and a distributed load on Element 3. The code block used for joint loading is presented in Appendix A.7, while the code block used for concentrated loads applied to an element is presented in Appendix A.8.

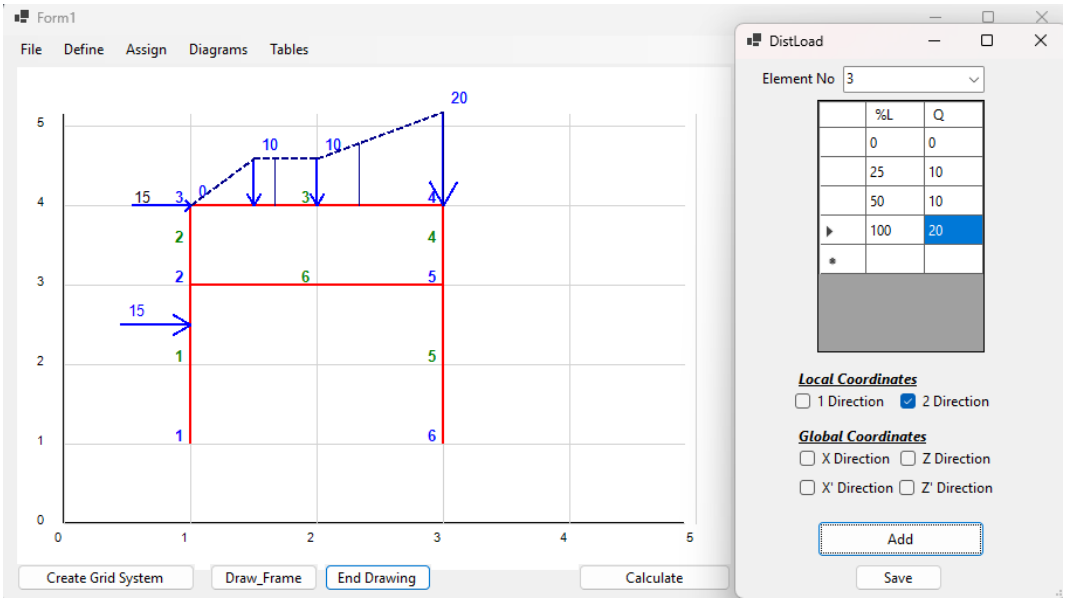


Figure 3.11 Assigning Loads

3.3.8.1 Temperature Load

The effect of temperature is treated as an equivalent axial force and can be applied through the Assign menu. The temperature load is computed by specifying the temperature difference between the top and bottom surfaces of the element, as illustrated in Figure 3.12 and the code block used for temperature load is presented in Appendix A.9.

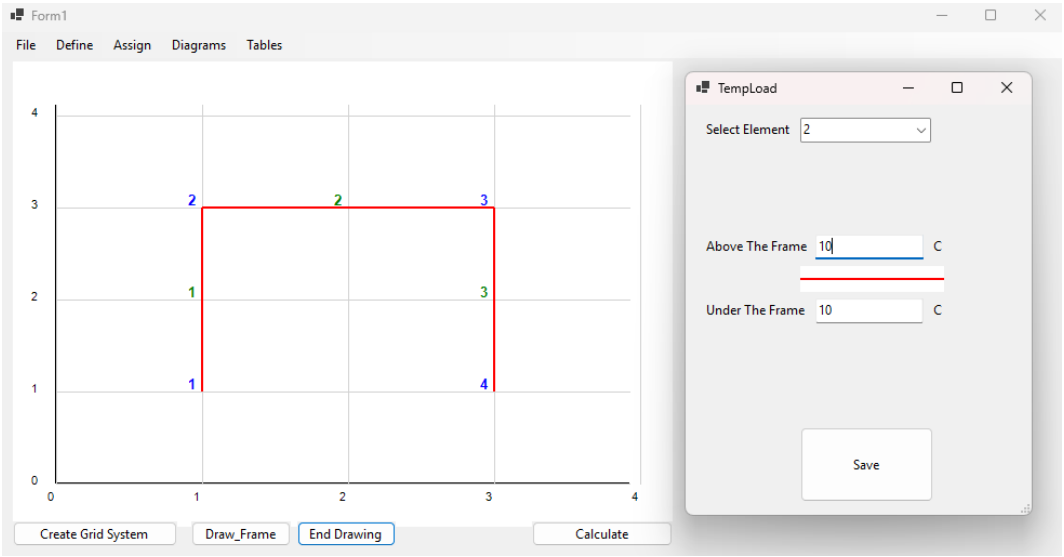


Figure 3.12 Temperature Loads

3.3.9 Node Displacement Assignments

The predefined nodal displacement assignment can also be accessed through the Assign menu. In the corresponding window, the desired node can be selected, and displacement or rotational constraints can be applied accordingly. This process is illustrated in Figure 3.13.

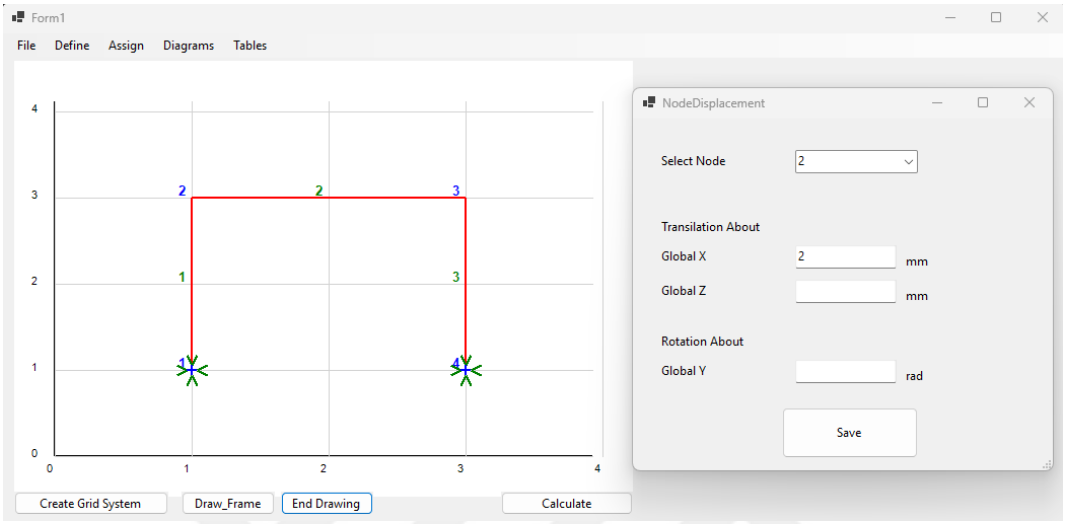


Figure 3.13 Assigning Node Displacements

3.3.10 Diagram Visualization

As output, the software provides the moment, shear, and axial force diagrams, along with the deformed shape of the structural system. These results can be conveniently accessed and visualized by the user through the Diagram menu, which displays each diagram in a clear and interpretable format as shown in Figure 3.14 - 3.17

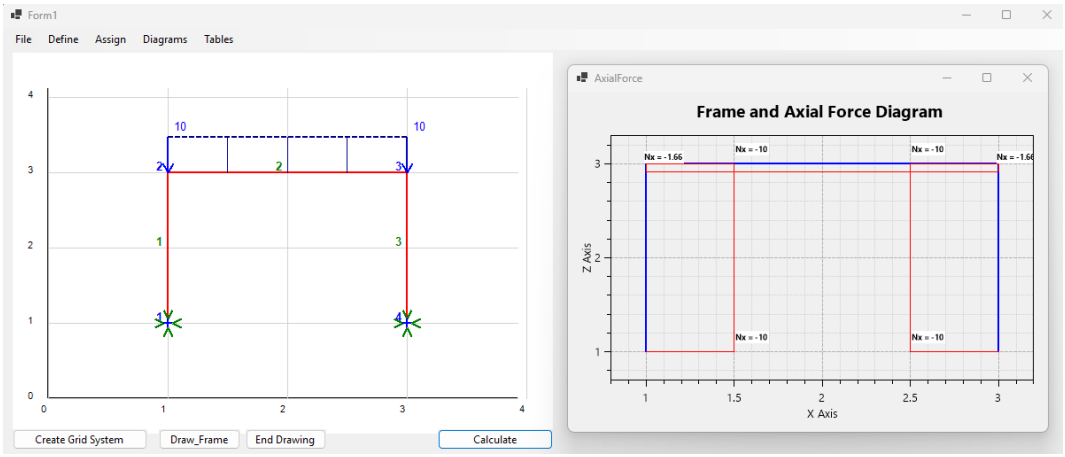


Figure 3.14 Axial Force

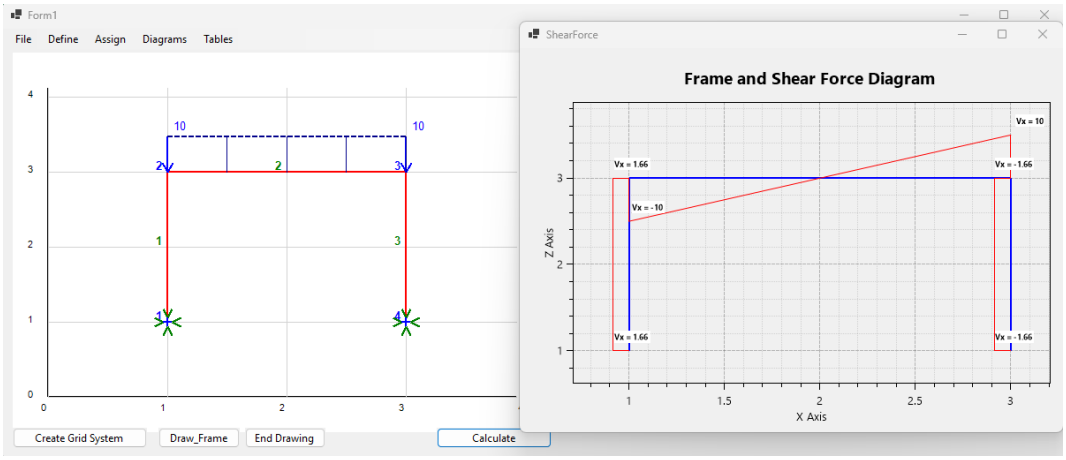


Figure 3.15 Shear Force

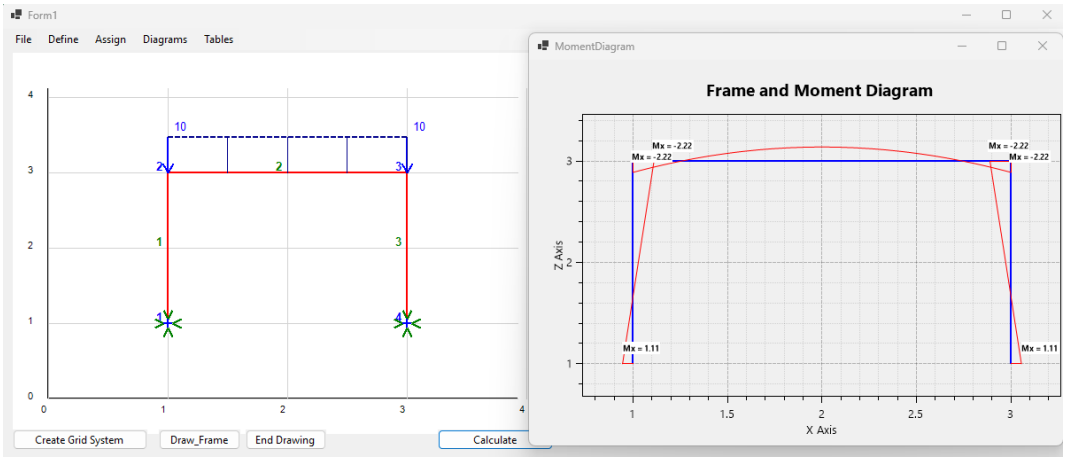


Figure 3.16 Moment Diagram

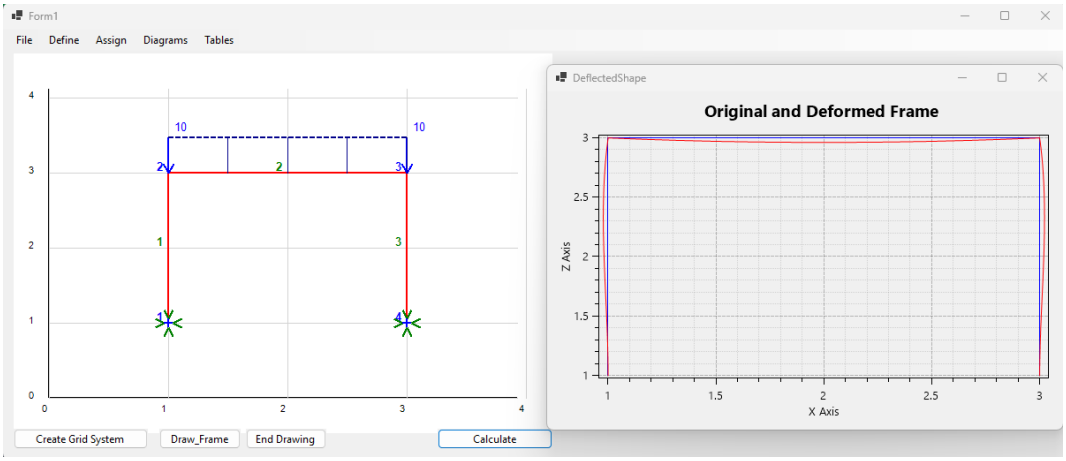


Figure 3.17 Deflected Shape

4
VERIFICATION AND COMPARATIVE CASE STUDIES WITH SAP2000

4.1 Verification Methodology

The verification of the developed frame analysis program is conducted through a systematic comparison with the commercial finite element software SAP2000, which is adopted as the reference solution tool. SAP2000 is widely used in structural engineering practice and has been extensively validated for linear elastic analysis of frame systems, making it a reliable benchmark for numerical verification[21].

For the verification process, identical structural models are established in both the developed program and SAP2000. This includes the use of the same geometric configuration, material properties, cross-sectional characteristics, boundary conditions, and loading scenarios. By ensuring complete consistency in the input data, any differences observed in the results can be directly attributed to the numerical formulation and implementation of the developed software rather than modeling discrepancies.

The comparison focuses on key structural response quantities, including nodal displacements, element end forces, and internal force resultants such as axial forces, shear forces, and bending moments. These quantities are evaluated at corresponding nodes and element locations in both models to ensure a meaningful and point-wise comparison.

To quantify the level of agreement between the two solutions, both absolute differences and relative (percentage) errors are computed. The relative error is evaluated with respect to the SAP2000 results, which are treated as the reference values. The accuracy of the developed program is assessed based on the magnitude and distribution of these discrepancies, allowing for an objective evaluation of its numerical performance and reliability.

4.2 Case Studies

4.2.1 Case Study A: Frame System Subjected to Mechanical Loads, Thermal Effects and Hinged Connections

4.2.1.1 Description of the Structural Model

The case study consists of a two-dimensional frame system composed of beam–column elements. The structure is modeled in the global Cartesian coordinate system, with all elements lying in the X–Z plane. Each node possesses three degrees of freedom, namely horizontal displacement, vertical displacement, and in-plane rotation.

The geometric configuration of the frame, including nodal coordinates and element connectivity, is summarized in Figure 4.1 and Table 4.1 .

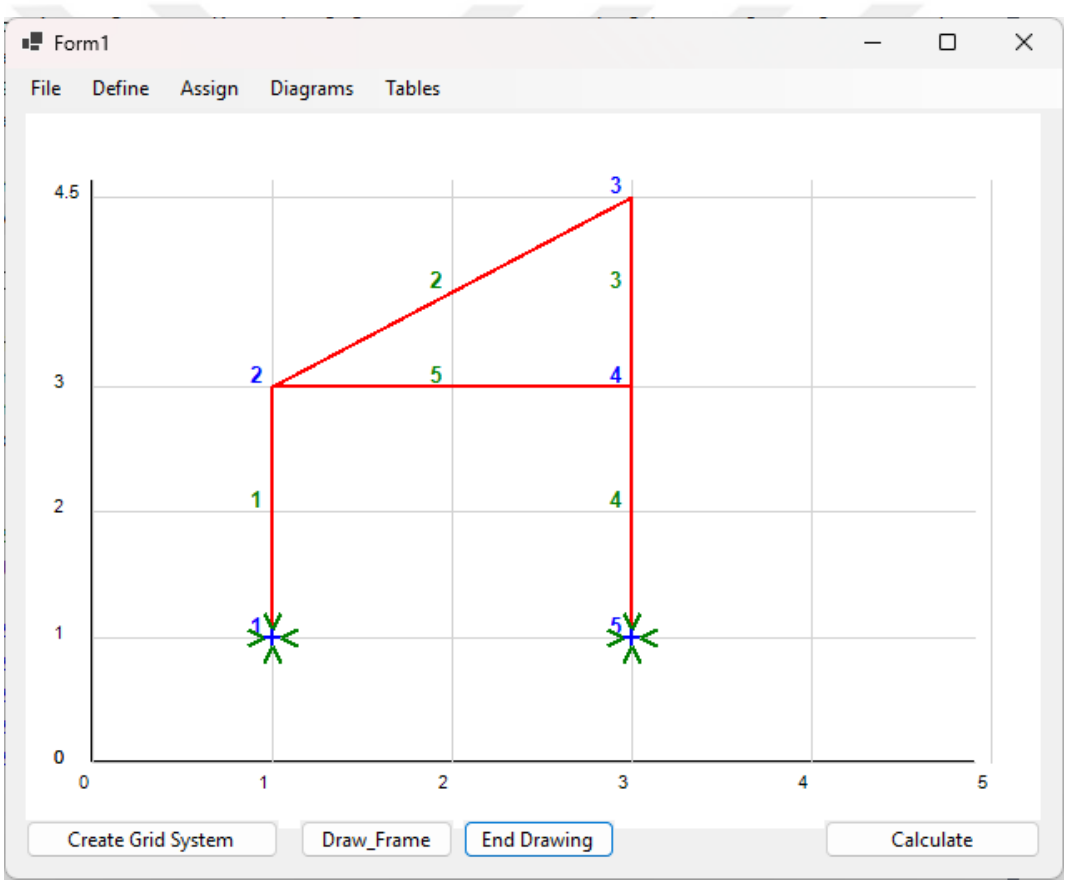


Figure 4.1 Example System

Table 4.1 Geometric and Directional Properties of Frame Elements

Elem.	Start Node	End Node	X_s	Z_s	X_e	Z_e	L	$(\cos\theta, \sin\theta)$
1	1	2	1.0	1.0	1.0	3.0	2.0000	(0.0000, 1.0000)
2	2	3	1.0	3.0	3.0	4.5	2.5000	(0.8000, 0.6000)
3	3	4	3.0	4.5	3.0	3.0	1.5000	(0.0000, -1.0000)
4	4	5	3.0	3.0	3.0	1.0	2.0000	(0.0000, -1.0000)
5	2	4	1.0	3.0	3.0	3.0	2.0000	(1.0000, 0.0000)

4.2.1.2 Material and Section Properties

Table 4.2 presents the geometric and material properties of the cross-sections used in the finite element model. The material was chosen as S355 structural steel; HE 100A sections were assigned to elements 1, 3, 4, and 5, while an IPE 100 section was assigned to element 2.

Table 4.2 Material and Cross-Section Properties of Frame Elements

Element No.	Cross-Section Area A (m^2)	Moment of Inertia I (m^4)	Section Height h (m)	Elastic Modulus E (MPa)
1	0.00212	3.49×10^{-6}	0.096	210000
2	0.00103	1.71×10^{-6}	0.100	210000
3	0.00212	3.49×10^{-6}	0.096	210000
4	0.00212	3.49×10^{-6}	0.096	210000
5	0.00212	3.49×10^{-6}	0.096	210000

4.2.1.3 Boundary Conditions

With regard to the boundary conditions, nodes 1 and 5 were modeled as fixed supports, while a shear release was defined at the right end of element 5.

4.2.1.4 Loading Conditions

In terms of loading conditions, a temperature difference of $5°\mathrm{C}$ was applied to the inner and outer faces of the column in element 1. The Element 2 was subjected to a triangular distribution load, while a concentrated load was applied to element 3. The detailed specifications of these loading conditions are summarized in Table 4.3.

Table 4.3 Applied Loads on Frame Elements

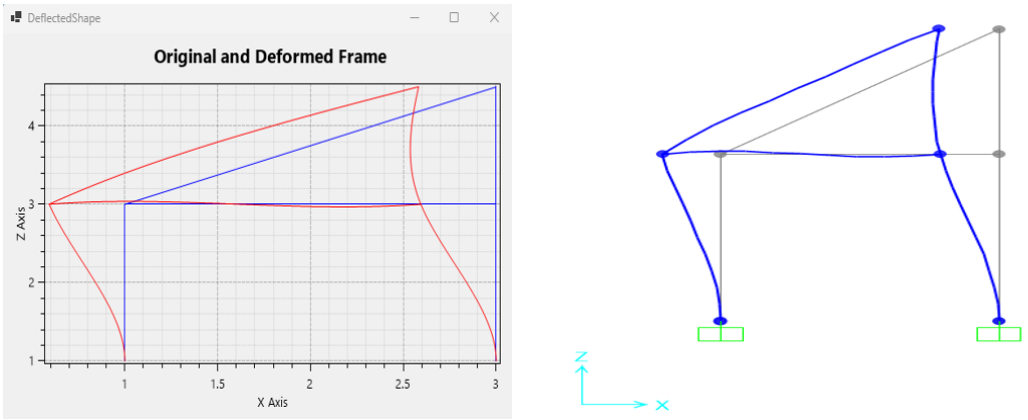
Element No	Loading
1	$(5°\text{C} - 5°\text{C})$ Thermal
2	$(0\%L - 10,\ 100\%L - 0)$ in Z' Axis
3	$(25\%L - 10)$ in Local 2-direction
4	–
5	–

4.2.1.5 Comparative Analysis of Results

This section presents a comparative evaluation of the numerical model by examining key structural response quantities. Nodal displacements are first compared with reference results in order to assess the accuracy of the developed formulation. Subsequently, internal force components will be investigated to further validate the numerical predictions. The system stiffness matrices employed in the numerical analysis are provided in Appendix B.

The maximum nodal displacements obtained from the numerical analysis and the reference solution are summarized in Table 4.9. The percentage differences are calculated based on the reference values. For rotational degrees of freedom, the comparison is performed using the magnitude of rotations, since the sign only indicates the direction of rotation.

In addition to the tabulated results, the displacement diagrams obtained from the developed software and SAP2000 are presented in Figure 4.2, respectively, in order to provide a visual comparison of the deformation patterns.



(a) Developed software (b) SAP2000

Figure 4.2 Comparison of deformed shapes

Table 4.4 Comparison of Nodal Displacements

Node	DOF	Developed software / Reference	Error (%)
1	X (mm)	0.000 / 0.000	0.00
	Z (mm)	0.000 / 0.000	0.00
	θ (rad)	0.000 / 0.000	0.00
2	X (mm)	-8.147 / -8.451	3.60
	Z (mm)	0.0452 / 0.0453	0.22
	θ (rad)	0.0036 / 0.0037	2.70
3	X (mm)	-8.356 / -8.660	3.51
	Z (mm)	0.0470 / 0.0468	0.43
	θ (rad)	-0.00187 / 0.00179	4.47
4	X (mm)	-8.119 / -8.425	3.63
	Z (mm)	0.0268 / 0.02678	0.08
	θ (rad)	0.0035 / -0.00357	1.96
5	X (mm)	0.000 / 0.000	0.00
	Z (mm)	0.000 / 0.000	0.00
	θ (rad)	0.000 / 0.000	0.00

In addition to nodal displacements, internal force components such as axial force, shear force, and bending moment are also compared and summarized in Table 4.5 to provide a comprehensive assessment of the developed software.

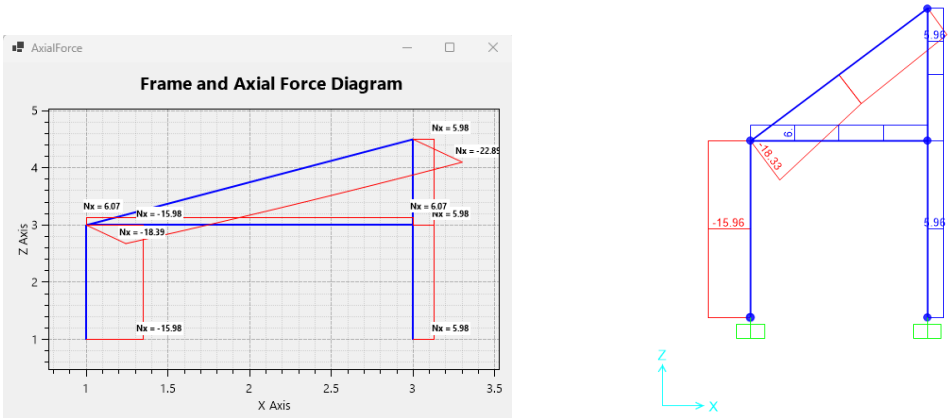
Furthermore, the corresponding internal force diagrams obtained from the developed software and SAP2000 are presented in Figures 4.3, 4.4 and 4.5 for visual comparison, allowing a qualitative evaluation of the distribution and consistency of the internal forces along the elements.

Table 4.5 Comparison of Internal Forces at Element Ends

Element	End	Force	Developed Software / Reference	Error (%)
1	i	N (kN)	-15.977 / -15.961	0.10
		V (kN)	4.926 / -4.946	0.40
		M (kNm)	6.269 / -6.301	0.51
	j	N (kN)	-15.977 / -15.961	0.10
		V (kN)	-4.926 / -4.946	0.40
		M (kNm)	-3.582 / 3.590	0.22
2	i	N (kN)	-18.387 / -18.33	0.31
		V (kN)	-6.181 / -6.203	0.35
		M (kNm)	-3.523 / -3.543	0.56
	j	N (kN)	-12.387 / -12.33	0.46
		V (kN)	1.818 / 1.797	1.17
		M (kNm)	-1.402 / -1.368	2.49
3	i	N (kN)	5.977 / 5.961	0.27
		V (kN)	-11.001 / -10.942	0.54
		M (kNm)	-1.402 / -1.368	2.49
	j	N (kN)	5.977 / 5.961	0.27
		V (kN)	-1.001 / -0.942	6.27
		M (kNm)	3.849 / 3.794	1.45
4	i	N (kN)	5.977 / 5.961	0.27
		V (kN)	5.073 / 5.054	0.38
		M (kNm)	3.789 / 3.746	1.15
	j	N (kN)	5.977 / 5.961	0.27
		V (kN)	5.073 / 5.054	0.38
		M (kNm)	-6.358 / -6.361	0.05
5	i	N (kN)	6.074 / 5.996	1.30
		V (kN)	0.000 / 0.000	0.00
		M (kNm)	-0.059 / -0.0479	23.17
	j	N (kN)	6.074 / 5.996	1.30
		V (kN)	0.000 / 0.000	0.00
		M (kNm)	-0.059 / -0.0479	23.17

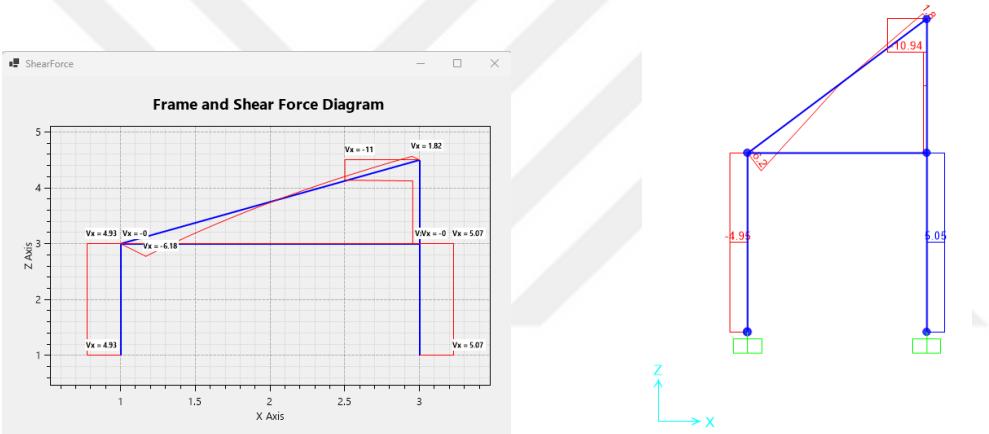
It should be noted that the relatively high percentage error observed in the bending moment results of element 5 does not indicate a significant discrepancy in structural response. This behavior is primarily attributed to the very small magnitude of the reference bending moment values at this element end. When the reference value approaches zero, even minor absolute differences lead to artificially large percentage errors.

Therefore, the reported percentage error in such cases should be interpreted with caution, and the absolute values of the internal forces should be considered as a more reliable indicator of agreement. Overall, the comparison demonstrates that the developed numerical model provides consistent and accurate predictions of internal forces.



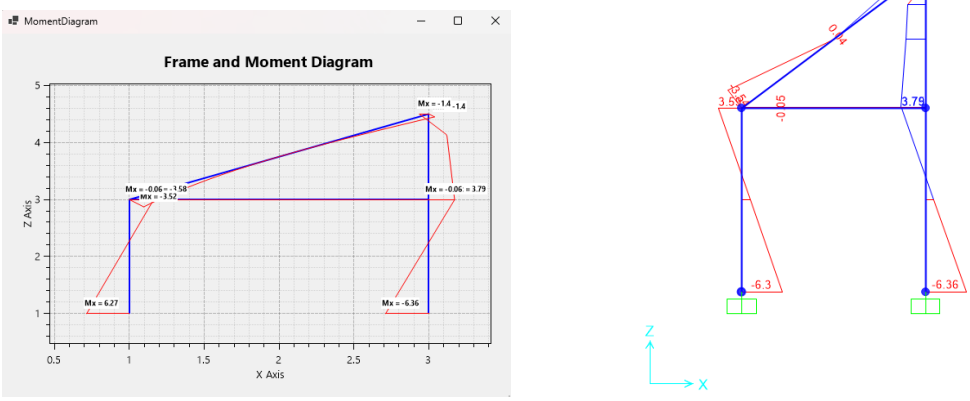
(a) Developed software (b) SAP2000

Figure 4.3 Comparison of axial force diagrams



(a) Developed software (b) SAP2000

Figure 4.4 Comparison of shear force diagrams



(a) Developed software (b) SAP2000

Figure 4.5 Comparison of bending moment diagrams

4.2.2 Case Study B: Frame System with Prescribed Nodal Displacements

4.2.2.1 Description of the Structural Model

The case study consists of a two-dimensional frame system composed of beam–column elements. The structure is modeled in the global Cartesian coordinate system, with all elements lying in the X–Z plane. Each node possesses three degrees of freedom, namely horizontal displacement, vertical displacement, and in-plane rotation.

The geometric configuration of the frame, including nodal coordinates and element connectivity, is summarized in Figure 4.6 and Table 4.6 .

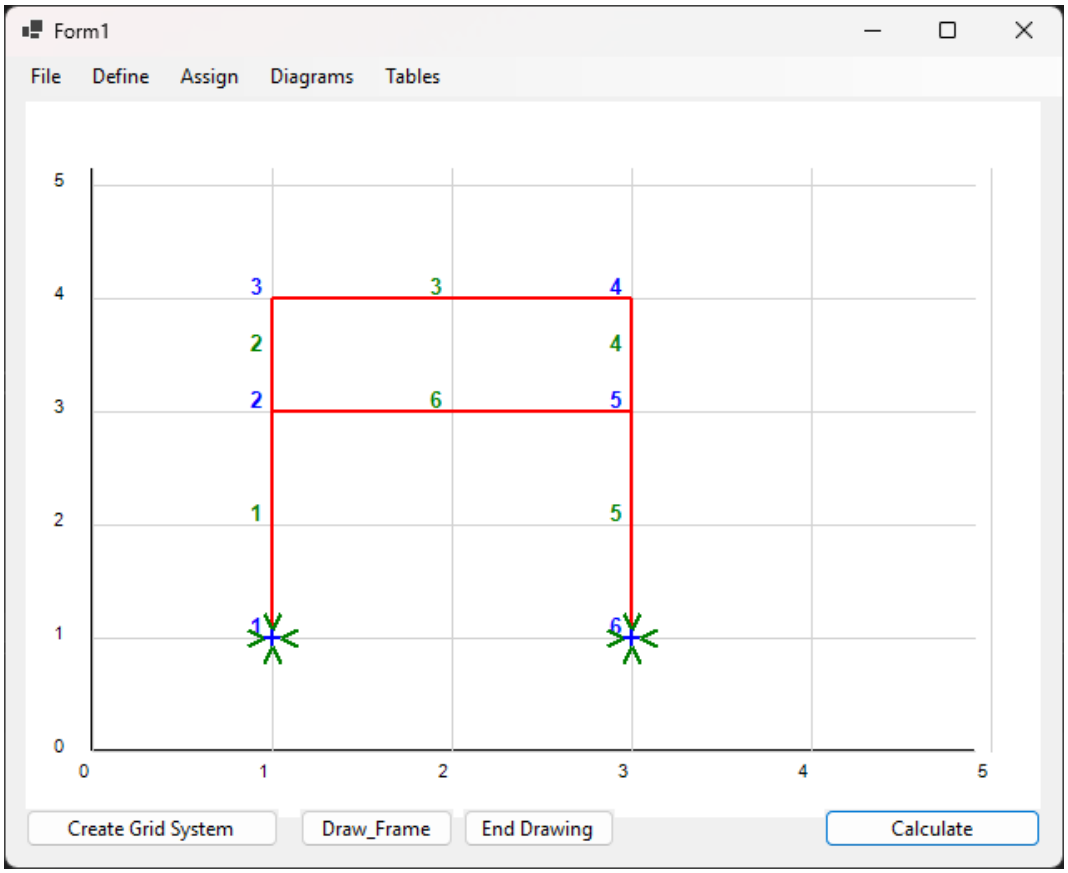


Figure 4.6 Example System

4.2.2.2 Material and Section Properties

Table 4.7 presents the geometric and material properties of the cross-sections used in the finite element model. The material was chosen as S355 structural steel; IPE 100 sections were assigned to elements.

Table 4.6 Geometric and Directional Properties of Frame Elements

Elem.	Start Node	End Node	X_s	Z_s	X_e	Z_e	L	$(\cos\theta, \sin\theta)$
1	1	2	1.0	1.0	1.0	3.0	2.0000	(0.0000, 1.0000)
2	2	3	1.0	3.0	1.0	4.0	1.0000	(0.0000, 1.0000)
3	3	4	1.0	4.0	3.0	4.0	2.0000	(1.0000, 0.0000)
4	4	5	3.0	4.0	3.0	3.0	1.0000	(0.0000, -1.0000)
5	5	6	3.0	3.0	3.0	1.0	2.0000	(0.0000, -1.0000)
6	2	5	1.0	3.0	3.0	3.0	2.0000	(1.0000, 0.0000)

Table 4.7 Material and Cross-Section Properties of Frame Elements

Element No.	Cross-Section Area A (m^2)	Moment of Inertia I (m^4)	Section Height h (m)	Elastic Modulus E (MPa)
1	0.00103	1.71×10^{-6}	0.10	210000
2	0.00103	1.71×10^{-6}	0.10	210000
3	0.00103	1.71×10^{-6}	0.10	210000
4	0.00103	1.71×10^{-6}	0.10	210000
5	0.00103	1.71×10^{-6}	0.10	210000
6	0.00103	1.71×10^{-6}	0.10	210000

4.2.2.3 Boundary Conditions

Regarding the boundary conditions, nodes 1 and 6 were modeled as fixed supports. In SAP2000, prescribed nodal displacements are applied by constraining the corresponding degrees of freedom, whereas in the developed program, prescribed displacements are imposed without constraining the associated degrees of freedom, and are instead incorporated directly into the solution procedure.

4.2.2.4 Loading Conditions

No external loads were applied to the system. Instead, a prescribed rotation of 0.01 rad about the global Y-axis was imposed at node 2, and a prescribed horizontal displacement of 0.02 m along the global X-axis was applied at node 4. The detailed specifications of these loading conditions are summarized in the Table 4.8.

Table 4.8 Applied Loads on Frame Elements

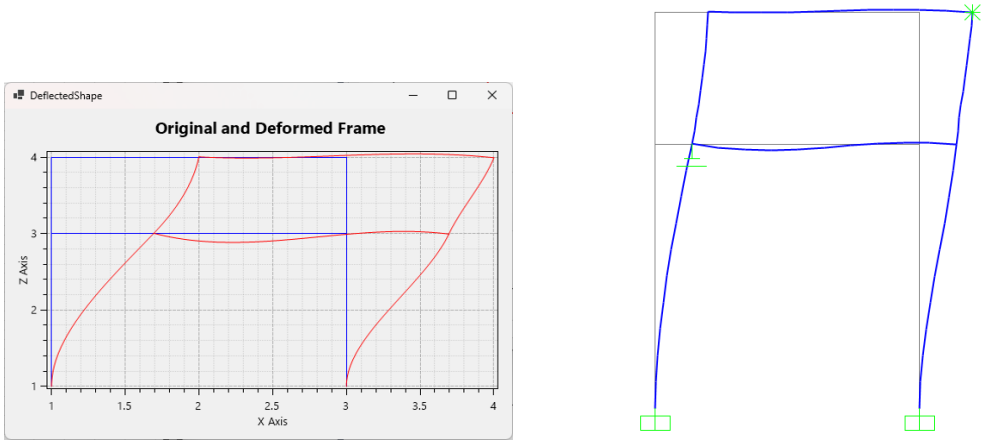
Element No	Loading
1	–
2	Rotation of 0.01 rad about the global Y-axis (R_Y)
3	–
4	Horizontal displacement of 0.02 m along the global X-axis (U_X)
5	–
6	–

4.2.2.5 Comparative Analysis of Results

This section presents a comparative evaluation of the numerical model by examining key structural response quantities. Nodal displacements are first compared with reference results in order to assess the accuracy of the developed formulation. Subsequently, internal force components will be investigated to further validate the numerical predictions.The system stiffness matrices employed in the numerical analysis are provided in Appendix C.

The maximum nodal displacements obtained from the numerical analysis and the reference solution are summarized in Table 4.9. The percentage differences are calculated based on the reference values. For rotational degrees of freedom, the comparison is performed using the magnitude of rotations, since the sign only indicates the direction of rotation.

In addition to the tabulated results, the displacement diagrams obtained from the developed software and SAP2000 are presented in Figure 4.7, respectively, in order to provide a visual comparison of the deformation patterns.



(a) Developed software (b) SAP2000

Figure 4.7 Comparison of deformed shapes

Table 4.9 Comparison of Nodal Displacements

Node	DOF	Developed software / Reference	Error (%)
1	X (mm)	0.000 / 0.000	0.00
	Z (mm)	0.000 / 0.000	0.00
	θ (rad)	0.000 / 0.000	0.00
2	X (mm)	13.875 / 13.81	0.47
	Z (mm)	0.103 / 0.100	3.00
	θ (rad)	-0.01 / 0.01	0.00
3	X (mm)	19.999 / 19.998	0.01
	Z (mm)	0.118 / 0.115	2.61
	θ (rad)	-0.002 / 0.002	0.00
4	X (mm)	20.000 / 20.000	0.00
	Z (mm)	-0.118 / -0.115	2.61
	θ (rad)	-0.004 / 0.004	0.00
5	X (mm)	13.892 / 13.826	0.48
	Z (mm)	-0.103 / -0.100	3.00
	θ (rad)	-0.005 / 0.0048	4.17
6	X (mm)	0.000 / 0.000	0.00
	Z (mm)	0.000 / 0.000	0.00
	θ (rad)	0.000 / 0.000	0.00

In addition to nodal displacements, internal force components such as axial force, shear force, and bending moment are also compared and summarized in Table 4.10 to provide a comprehensive assessment of the developed software.

Furthermore, the corresponding internal force diagrams obtained from the developed software and SAP2000 are presented in Figures 4.8, 4.9 and 4.10 for visual comparison, allowing a qualitative evaluation of the distribution and consistency of the internal forces along the elements.

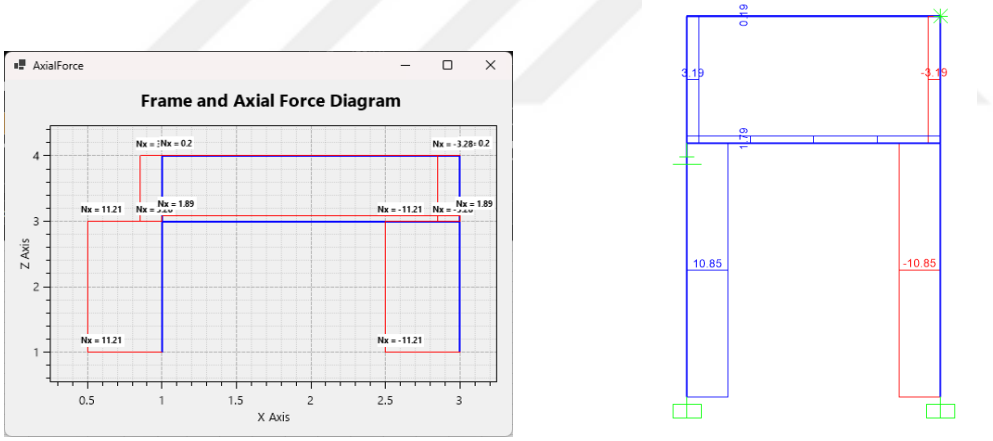
Table 4.10 Comparison of Internal Forces at Element Ends

Element	End	Force	Developed Software / Reference	Error (%)
1	i	N (kN)	11.207 / 10.851	3.28
		V (kN)	-2.087 / 1.988	4.98
		M (kNm)	-3.882 / 3.783	2.62
	j	N (kN)	11.207 / 10.851	3.28
		V (kN)	-2.087 / 1.988	4.98
		M (kNm)	0.291 / -0.192	51.56
2	i	N (kN)	3.275 / 3.195	2.50
		V (kN)	-0.195 / 0.193	1.04
		M (kNm)	2.719 / -2.677	1.57
	j	N (kN)	3.275 / 3.195	2.50
		V (kN)	-0.195 / 0.193	1.04
		M (kNm)	2.914 / -2.870	1.53
3	i	N (kN)	0.195 / 0.193	1.04
		V (kN)	3.275 / 3.195	2.50
		M (kNm)	2.914 / 2.870	1.53
	j	N (kN)	0.195 / 0.193	1.04
		V (kN)	3.275 / -3.195	2.50
		M (kNm)	-3.635 / -3.518	3.33
4	i	N (kN)	-3.275 / -3.195	2.50
		V (kN)	-6.718 / -6.463	3.95
		M (kNm)	-3.635 / -3.518	3.33
	j	N (kN)	-3.275 / -3.195	2.50
		V (kN)	-6.718 / -6.463	3.95
		M (kNm)	3.082 / 2.944	4.69
5	i	N (kN)	-11.207 / -10.851	3.28
		V (kN)	-4.825 / -4.668	3.36
		M (kNm)	-3.940 / -3.792	3.90
	j	N (kN)	-11.207 / -10.851	3.28
		V (kN)	-4.825 / -4.668	3.36
		M (kNm)	5.711 / 5.442	4.95
6	i	N (kN)	1.892 / 1.795	5.40
		V (kN)	7.932 / 7.657	3.59
		M (kNm)	8.841 / 8.576	3.09
	j	N (kN)	1.892 / 1.795	5.40
		V (kN)	7.932 / 7.657	3.59
		M (kNm)	-7.022 / -6.737	4.23

The comparison of internal force results obtained from the developed software and the reference solution indicates that the majority of discrepancies remain at relatively low levels. For most axial force (N) and shear force (V) components, the percentage error is generally below 5%, demonstrating a high level of consistency between the two solutions.

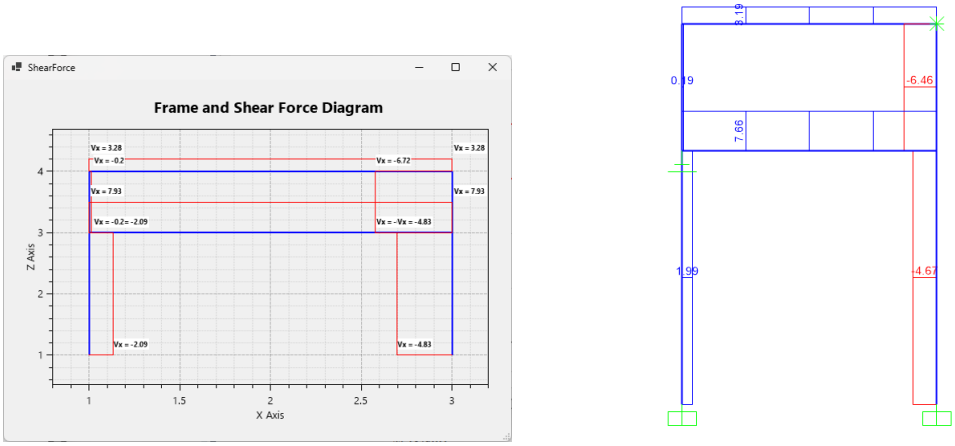
The maximum error, approximately 51.56%, is observed in the bending moment (M) at the j-end of Element 1. This relatively high percentage error is primarily attributed to the very small magnitude of the reference moment value. In such cases, even minor absolute differences may lead to disproportionately large percentage errors, which can be misleading when evaluated solely on a relative basis.

Overall, when both percentage and absolute differences are considered, the developed software shows results that are in close agreement with the reference solution. The observed discrepancies remain within acceptable limits for linear static analysis of planar frame systems, confirming the reliability and numerical robustness of the implemented finite element formulation for both educational and research-oriented applications.



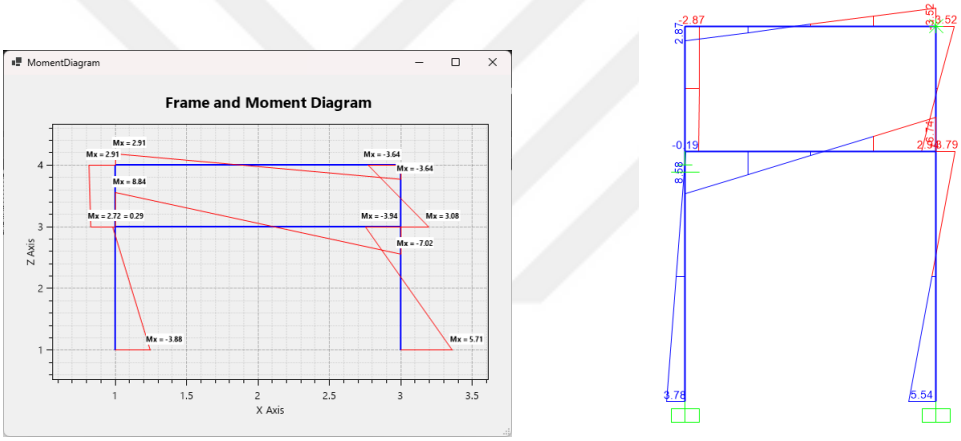
(a) Developed software (b) SAP2000

Figure 4.8 Comparison of axial force diagrams



(a) Developed software (b) SAP2000

Figure 4.9 Comparison of shear force diagrams



(a) Developed software (b) SAP2000

Figure 4.10 Comparison of bending moment diagrams

4.3 Error Analysis and Discussion

In this section, the discrepancies observed between the results obtained from the developed numerical software and the reference solution provided by SAP2000 are evaluated and discussed. The comparison focuses on key response quantities such as nodal displacements and internal force components, including axial force, shear force, and bending moment.

The percentage errors were calculated based on the reference values in order to quantify the level of agreement between the two approaches. For displacement components and internal forces, the error assessment was performed using the absolute values, since the sign of the results only indicates the direction of action

and does not affect the magnitude of the structural response.

Overall, the results demonstrate a very good agreement between the developed software and the reference solution. The majority of the displacement and internal force values exhibit percentage errors within a small and acceptable range, indicating that the numerical formulation and implementation of the developed software are consistent and reliable.

It is observed that relatively larger percentage errors occur in certain bending moment values, particularly in elements where the reference moment magnitudes are very small. In such cases, even minor absolute differences lead to disproportionately large percentage errors. This behavior is purely mathematical and does not represent a significant physical inconsistency in the structural response. Therefore, the interpretation of percentage errors in regions with near-zero reference values should be carried out with caution, and absolute differences should be considered as a more meaningful indicator[22].

The minor discrepancies observed in the results can be attributed to several factors, including numerical rounding effects, differences in load idealization, and the implementation of boundary conditions and hinge definitions in the numerical model. Additionally, assumptions such as linear elastic material behavior and idealized boundary conditions may contribute to small deviations between the two solutions.

Despite these minor differences, the close agreement in deformation shapes, displacement magnitudes, and internal force distributions confirms the accuracy and robustness of the developed numerical model. Accordingly, the developed software can be considered suitable for the analysis of frame structures under similar loading and boundary conditions.

5
CONCLUSION

In this study, a numerical analysis software based on the finite element method was developed and its performance was evaluated through a series of benchmark problems. The results obtained from the developed program were systematically compared with those generated by the commercial software SAP2000 in terms of nodal displacements and internal force components.

The comparative analyses demonstrated a very good agreement between the proposed software and the reference solutions. The maximum nodal displacements obtained from both approaches showed only minor discrepancies, and the overall deformation patterns were observed to be highly consistent. Similarly, internal force components, including axial force, shear force, and bending moment, exhibited close correspondence across all examined cases.

The observed differences between the numerical results were found to be relatively small and remained within acceptable engineering limits. These discrepancies can primarily be attributed to numerical precision, modeling assumptions, and minor differences in element formulation and solution procedures. Considering these factors, the level of accuracy achieved by the developed software is deemed satisfactory for practical structural analysis applications.

Based on the results of this study, it can be concluded that the developed finite element program is capable of accurately predicting structural responses and can serve as a reliable alternative tool for educational and preliminary engineering analyses. Future work may focus on extending the program to include more advanced element formulations, nonlinear material behavior, and dynamic analysis capabilities in order to further enhance its applicability.

REFERENCES

[1] W. K. Liu, S. Li, H. S. Park, "Eighty years of the finite element method: Birth, evolution, and future," *Archives of Computational Methods in Engineering*, vol. 29, no. 6, pp. 4431–4453, 2022.

[2] R. Li, "Research on application of finite element method in static analysis," *Journal of Physics: Conference Series*, IOP Publishing, vol. 1802, 2021, p. 042 008.

[3] C. S. Desai, T. Kundu, *Introductory finite element method*. Crc Press, 2017.

[4] V. Jagota, A. P. S. Sethi, K. Kumar, "Finite element method: An overview," *Walailak Journal of Science and Technology (WJST)*, vol. 10, no. 1, pp. 1–8, 2013.

[5] P. Sumets, *Computational framework for the finite element method in MATLAB® and python*. Chapman and Hall/CRC, 2022.

[6] G. Nikishkov, *Programming Finite Elements in Java™*. Springer, 2010.

[7] B. Alyavuz, "Boşluklu perde-çerçeve sistemlerin sonlu elemanlar metodu ile gerilme analizi," Ph.D. dissertation, Yüksek Lisans Tezi, Gazi Üniversitesi Fen Bilimleri Enstitüsü, Ankara, 6-38, 2003.

[8] H. Gazi, "Sonlu elemanlar yönemi ile çalışan bir bilgisayar programının geliştirilmesi ve düzlem gerilme—düzlem deformasyon problemlerinin analizi," Ph.D. dissertation, Istanbul University Istanbul, Turkey, 2008.

[9] S. Hayrioğlu, S. Bekiroğlu, "Modeling of plane elasto-plastic reinforced concrete frame system with developed finite element algorithm," 2023.

[10] F. Han, Z. Li, J. Zhang, Z. Liu, C. Yao, W. Han, "Abaqus and ansys implementations of peridynamics-based finite element method (perifem) for brittle fracture," 2022.

[11] C. Aydoğmuş, "Simulation of channel connector behavior using finite element method," M.S. thesis, Middle East Technical University (Turkey), 2013.

[12] H. ABDULKHALEK, "Numerical analysis for behavior of simply supported beams under moving loading using finite element method," M.S. thesis, Altınbaş University (Turkey), 2022.

[13] Y. Wei, "The study of numerical algorithms for modal partial differential equations based on finite element modelling," *Proceedings of the 2025 International Symposium on Artificial Intelligence and Computational Social Sciences*, 2025, pp. 28–33.

[14] I. A. AL-HATMEY, “Finite element analysis of temperature distribution in precast concret girders subjected to thermal loads,” M.S. thesis, Gaziantep University (Turkey), 2014.

[15] A. A. Kasımzade, *Sonlu elemanlar metodu: temelleri ve yapı mekaniğinde uygulamaları*. BirsenYayınevi, 2004.

[16] O. Yılmaz, A. Şahin, “A novel approach for static analysis of plane frame systems,” *Sigma Journal of Engineering and Natural Sciences*, vol. 33, no. 2, pp. 129–143, 2015.

[17] *C# programming guide*, Microsoft, 2024.

[18] *Oxyplot: A cross-platform plotting library for .net*, `https://oxyplot.readthedocs.io`, Accessed: 2025, 2024.

[19] *Math.net numerics*, `https://numerics.mathdotnet.com`, 2024.

[20] *Sqlite documentation*, SQLite Consortium, 2024.

[21] *Sap2000 integrated software for structural analysis and design*, Computers and Structures, Inc., 2021.

[22] M. Ainsworth, J. T. Oden, “A posteriori error estimation in finite element analysis,” *Computer methods in applied mechanics and engineering*, vol. 142, no. 1-2, pp. 1–88, 1997.

A PROGRAM LISTINGS

A.1 DrawRestrictions

Listing A.1 GeometryHelper Class

```
1  public static void DrawRestrictions(Vector<double> resultMatrix,
       List<PointF> coordsList, List<PointF> NodeCoords, float scaleX,
        float scaleY, int margin, int bitmapHeight, Graphics g)
2  {
3  using (Pen xPen = new Pen(Color.Green, 2))
4  using (Pen zPen = new Pen(Color.Green, 2))
5  using (Pen yPen = new Pen(Color.Blue, 2))
6  {
7  foreach (var restrictionIndex in resultMatrix)
8  {
9  int index = (int)restrictionIndex;
10 int nodeIndex = (index - 1) / 3;
11 int restrictionType = (index - 1) % 3;
12 float minX = coordsList.Min(p => p.X);
13 float minY = coordsList.Min(p => p.Y);
14 PointF nodeCoord = NodeCoords[nodeIndex];
15 int x = (int)((nodeCoord.X - minX) * scaleX) + margin;
16 int y = bitmapHeight - (int)((nodeCoord.Y - minY) * scaleY) +
       margin;
17 switch (restrictionType)
18 {
19 case 0: // X
20 g.DrawLine(xPen, x - 15, y - 5, x - 5, y);
21 g.DrawLine(xPen, x - 5, y, x - 15, y + 5);
22 g.DrawLine(xPen, x + 15, y + 5, x + 5, y);
23 g.DrawLine(xPen, x + 5, y, x + 15, y - 5);
24 break;
25 case 1: // Z
26 g.DrawLine(zPen, x - 5, y - 15, x, y - 5);
27 g.DrawLine(zPen, x, y - 5, x + 5, y - 15);
28 g.DrawLine(zPen, x + 5, y + 15, x, y + 5);
29 g.DrawLine(zPen, x, y + 5, x - 5, y + 15);
30 break;
31 case 2: // Y
32 g.DrawLine(yPen, x, y - 5, x, y + 5);
33 g.DrawLine(yPen, x - 5, y, x + 5, y);
34 break;
35 default:
36 break;
37 }
38 }
```

```
}
}
```

A.2 Line Element

Listing A.2 LineElement Class

```
public LineElement(int elementNo, PointF startPoint, PointF
    endPoint, int initialNode, int endNode, double elementLength,
    double elementCos, double elementSin, double area = 0.0, double
     ix = 0.0, double elasticity = 0.0)
{
    StartPoint = startPoint;
    EndPoint = endPoint;
    InitialNode = initialNode;
    ElementNo = elementNo;
    EndNode = endNode;
    ElementLength = elementLength;
    ElementCos = elementCos;
    ElementSin = elementSin;
    Area = area;
    Ix = ix;
    Elasticity = elasticity;
    StiffnessMatrix = null;
    SStiffnessMatrix = null;
    TransformationMatrix = null;
    El_GStiffnessMatrix = null;
    El_GGStiffnessMatrix = null;
    El_TopologyMatrix = null;
    El_LoadVector = Vector<double>.Build.Dense(6, 0.0);
    El_TempLoadVector = Vector<double>.Build.Dense(6, 0.0);
    El_GradTempLoadVector = Vector<double>.Build.Dense(6, 0.0);
}
```

A.3 Transformation Matrix

Listing A.3 Calcs Class

```
public static double ElementCos(PointF p1, PointF p2, double
    elementLength)
{
    return (p2.X - p1.X) / elementLength;
}
public static double ElementSin(PointF p1, PointF p2, double
    elementLength)
{
    return (p2.Y - p1.Y) / elementLength;
}
public static Matrix<double> TransformationMatrix(double
    elementCos, double elementSin)
{
    double[,] tData = {
    { elementCos, -elementSin, 0, 0, 0, 0 },
    { elementSin, elementCos, 0, 0, 0, 0 },
    { 0, 0, 1, 0, 0, 0 },
```

```
        { 0, 0, 0, elementCos, -elementSin, 0 },
        { 0, 0, 0, elementSin, elementCos, 0 },
        { 0, 0, 0, 0, 0, 1 }
        };

        return DenseMatrix.OfArray(tData);
    }
```

A.4 Deformed Shape

Listing A.4 DeformedShapeForm

```
using OxyPlot.Series;
using OxyPlot.WindowsForms;
using OxyPlot;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Tez5.Classes;
using OxyPlot.Axes;
using MathNet.Numerics.LinearAlgebra;
using System.Diagnostics;

namespace Tez5.Forms.Diagrams
{
    public partial class DeflectedShape : Form
    {
        private List<LineElement> lineElements;
        private List<double> XDef;
        private List<double> YDef;
        private int nPoint;

        public DeflectedShape(List<LineElement> lineElements,
            Vector<double> U,List<double> xDef,List<double> yDef,
            int nPoint)
        {
            InitializeComponent();
            this.lineElements = lineElements;
            this.XDef = xDef;
            this.YDef = yDef;
            this.nPoint = nPoint;
            DrawCombinedFrameWithDashedGrids();

        }
        private void DrawCombinedFrameWithDashedGrids()
        {
            Debug.WriteLine($"Deflected Shape Form: XDef.Count = {
                XDef.Count}, YDef.Count = {YDef.Count}");
            if (plotView1 == null) return;
            var plotModel = new PlotModel { Title = "Original and
                Deformed Frame" };
```

```
            var xAxis = new LinearAxis
            {
                Position = AxisPosition.Bottom,
                MajorGridlineStyle = LineStyle.Dash,
                MinorGridlineStyle = LineStyle.Dot,
                Title = "X Axis"
            };
            var zAxis = new LinearAxis
            {
                Position = AxisPosition.Left,
                MajorGridlineStyle = LineStyle.Dash,
                MinorGridlineStyle = LineStyle.Dot,
                Title = "Z Axis"
            };
            plotModel.Axes.Add(xAxis);
            plotModel.Axes.Add(zAxis);
            foreach (var element in lineElements)
            {
                var originalElementSeries = new LineSeries
                {
                    LineStyle = LineStyle.Solid,
                    Color = OxyColors.Blue,
                    StrokeThickness = 1.5,
                    Title = $"Original {element.ElementNo}"
                };
                originalElementSeries.Points.Add(new DataPoint(
                    element.StartPoint.X, element.StartPoint.Y));
                originalElementSeries.Points.Add(new DataPoint(
                    element.EndPoint.X, element.EndPoint.Y));
                plotModel.Series.Add(originalElementSeries);
            }
            foreach (var element in lineElements)
            {
                int pointIndex = (element.ElementNo-1)*(nPoint);
                var deformedElementSeries = new LineSeries
                {
                    LineStyle = LineStyle.Solid,
                    Color = OxyColors.Red,
                    StrokeThickness = 1.5,
                    Title = $"Deformed Element {element.ElementNo}
                        "
                };
                for (int i = 0; i < nPoint; i++)
                {
                    double deformedX = XDef[pointIndex];
                    double deformedY = YDef[pointIndex];
                    deformedElementSeries.Points.Add(new DataPoint
                        (deformedX, deformedY));
                    pointIndex++;
                }
                plotModel.Series.Add(deformedElementSeries);
            }
            plotModel.InvalidatePlot(true);
            plotView1.Model = plotModel;
        }
    }
}
```

A.5 Restriction Assignment

Listing A.5 Restriction Assignment

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;
using MathNet.Numerics.LinearAlgebra;

namespace Tez5
{
    public partial class RestrictionForm : Form
    {
        public Vector<double> RestMatrix { get; private set; }
        private readonly List<int> Nodes;
        public RestrictionForm(List<int> nodes)
        {
            InitializeComponent();
            Nodes = nodes;
        }
        private void RestrictionForm_Load_1(object sender,
            EventArgs e)
        {
            var table = new DataTable();
            table.Columns.Add("Node_Value", typeof(int));
            table.Columns.Add("X", typeof(bool));
            table.Columns.Add("Z", typeof(bool));
            table.Columns.Add("Y", typeof(bool));
            foreach (var node in Nodes)
            table.Rows.Add(node, false, false, false);
            dataGridView1.DataSource = table;
            dataGridView1.AllowUserToAddRows = false;
            dataGridView1.AutoSizeColumnsMode =
                DataGridViewAutoSizeColumnsMode.Fill;
        }
        private void btnSave_Click(object sender, EventArgs e)
        {
            var restrictions = new List<double>();
            foreach (DataGridViewRow row in dataGridView1.Rows)
            {
                var nodeValue = row.Cells["Node_Value"].Value;
                bool isXRestricted = Convert.ToBoolean(row.Cells["
                    X"].Value);
                bool isZRestricted = Convert.ToBoolean(row.Cells["
                    Z"].Value);
                bool isYRestricted = Convert.ToBoolean(row.Cells["
                    Y"].Value);
                if (nodeValue != null)
                {
                    int nodeIndex = Convert.ToInt32(nodeValue);
                    restrictions.AddRange(GetRestrictionValues(
                        nodeIndex, isXRestricted, isZRestricted,
                        isYRestricted));
                }
            }
            RestMatrix = CreateVector(restrictions);
            this.DialogResult = DialogResult.OK;
            this.Close();
```

```
        }
        private IEnumerable<double> GetRestrictionValues(int
            nodeIndex, bool isX, bool isZ, bool isY)
        {
            int baseIndex = (nodeIndex - 1) * 3;
            var restrictionList = new List<double>();
            if (isX) restrictionList.Add(baseIndex + 1);
            if (isZ) restrictionList.Add(baseIndex + 2);
            if (isY) restrictionList.Add(baseIndex + 3);
            return restrictionList;
        }
        private Vector<double> CreateVector(List<double> values)
        {
            return Vector<double>.Build.Dense(values.ToArray());
        }
    }
}
```

A.6 Frame Release

Listing A.6 Frame Release

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Printing;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;
using MathNet.Numerics.LinearAlgebra;
using OxyPlot;
using Tez5.Classes;
using Tez5.Matrix_Tables;
namespace Tez5.Forms
{
    public partial class FrameReleases : Form
    {
        private Form1 _form1;
        private List<LineElement> _lineElements;
        public FrameReleases(List<LineElement> lineElements, Form1
            form1)
        {
            InitializeComponent();
            _form1 = form1;
            _lineElements = lineElements;
            comboBox1.DisplayMember = "ElementNo";
            comboBox1.ValueMember = "ElementNo";
            comboBox1.DataSource = _lineElements;
        }
        private void button1_Click(object sender, EventArgs e)
        {
```

```
35              if (comboBox1.SelectedItem is LineElement
                    selectedElement)
36              {
37                  int currentColumnCount = selectedElement.
                        El_TopologyMatrix?.ColumnCount ?? 12;
38                  int newColumnCount = currentColumnCount + 1;
39                  AddVectorIfChecked(checkBox1, 0, selectedElement,
                        newColumnCount);
40                  AddVectorIfChecked(checkBox2, 1, selectedElement,
                        newColumnCount);
41                  AddVectorIfChecked(checkBox3, 2, selectedElement,
                        newColumnCount);
42                  AddVectorIfChecked(checkBox4, 3, selectedElement,
                        newColumnCount);
43                  AddVectorIfChecked(checkBox5, 4, selectedElement,
                        newColumnCount);
44                  AddVectorIfChecked(checkBox6, 5, selectedElement,
                        newColumnCount);
45                  foreach (var element in _lineElements)
46                  {
47                      if (element != selectedElement)
48                      {
49                          ExpandTopologyMatrixWithZeros(element,
                                newColumnCount);
50                      }
51                  }
52                  UpdateDataGridView(selectedElement.
                        El_TopologyMatrix);
53                  if (checkBox1.Checked)
54                  {
55                      _form1.DrawRelease1(selectedElement);
56                  }
57                  if (checkBox2.Checked)
58                  {
59                      _form1.DrawRelease2(selectedElement);
60                  }
61                  if (checkBox3.Checked)
62                  {
63                      _form1.DrawRelease3(selectedElement);
64                  }
65                  if (checkBox4.Checked)
66                  {
67                      _form1.DrawRelease4(selectedElement);
68                  }
69                  if (checkBox5.Checked)
70                  {
71                      _form1.DrawRelease5(selectedElement);
72                  }
73                  if (checkBox6.Checked)
74                  {
75                      _form1.DrawRelease6(selectedElement);
76                  }
77              }
78          }
79          private void AddVectorIfChecked(CheckBox checkBox, int
                index, LineElement element, int newColumnCount)
80          {
81              if (checkBox.Checked)
```

```
82              {
83                  var newMatrix2 = Matrix<double>.Build.Dense(
                        element.El_TopologyMatrix?.RowCount ?? 6,
                        newColumnCount, 0);
84                  var newMatrix3 = Matrix<double>.Build.Dense(
                        element.El_TopologyMatrix?.RowCount ?? 6,
                        newColumnCount, 0);
85                  if (element.El_TopologyMatrix != null)
86                  {
87                      newMatrix3.SetSubMatrix(0, 0, element.
                            El_TopologyMatrix);
88                  }
89                  if (element.El_TopologyMatrix2 != null)
90                  {
91                      newMatrix2.SetSubMatrix(0, 0, element.
                            El_TopologyMatrix2);
92                  }
93                  if (element.El_TopologyMatrix3 != null)
94                  {
95                      newMatrix3.SetSubMatrix(0, 0, element.
                            El_TopologyMatrix3);
96                  }
97                  newMatrix2[index, newColumnCount - 1] = 1;
98                  element.El_TopologyMatrix = (element.
                        TransformationMatrix * newMatrix2) + newMatrix3
                        ;
99                  element.El_TopologyMatrix2 = element.
                        TransformationMatrix * newMatrix2;
100                 element.El_TopologyMatrix3 = newMatrix3;
101             }
102         }
103         private void ExpandTopologyMatrixWithZeros(LineElement
                element, int newColumnCount)
104         {
105             var newMatrix = Matrix<double>.Build.Dense(element.
                    El_TopologyMatrix?.RowCount ?? 6, newColumnCount,
                    0);
106             var newMatrix2 = Matrix<double>.Build.Dense(element.
                    El_TopologyMatrix?.RowCount ?? 6, newColumnCount,
                    0);
107             var newMatrix3 = Matrix<double>.Build.Dense(element.
                    El_TopologyMatrix?.RowCount ?? 6, newColumnCount,
                    0);
108             if (element.El_TopologyMatrix != null)
109             {
110                 newMatrix.SetSubMatrix(0, 0, element.
                        El_TopologyMatrix);
111             }
112             if (element.El_TopologyMatrix2 != null)
113             {
114                 newMatrix2.SetSubMatrix(0, 0, element.
                        El_TopologyMatrix2);
115             }
116             if (element.El_TopologyMatrix3 != null)
117             {
118                 newMatrix3.SetSubMatrix(0, 0, element.
                        El_TopologyMatrix3);
119             }
```

```
            element.El_TopologyMatrix = newMatrix;
            element.El_TopologyMatrix2 = newMatrix2;
            element.El_TopologyMatrix3 = element.El_TopologyMatrix
                -element.El_TopologyMatrix2;
        }
        private void UpdateDataGridView(Matrix<double> matrix)
        {
            if (matrix == null) return;
            dataGridView1.Rows.Clear();
            dataGridView1.Columns.Clear();
            int rowCount = matrix.RowCount;
            int colCount = matrix.ColumnCount;
            for (int col = 0; col < colCount; col++)
            {
                dataGridView1.Columns.Add($"Col{col}", $"S tun␣{
                    col␣+␣1}");
            }
            for (int row = 0; row < rowCount; row++)
            {
                var rowData = new object[colCount];
                for (int col = 0; col < colCount; col++)
                {
                    rowData[col] = matrix[row, col];
                }
                dataGridView1.Rows.Add(rowData);
            }
        }
        private void comboBox1_SelectedIndexChanged(object sender,
            EventArgs e)
        {
            if (comboBox1.SelectedItem is LineElement
                selectedElement)
            {
                UpdateDataGridView(selectedElement.
                    El_TopologyMatrix);
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.OK;
            this.Close();
        }
    }
}
```

A.7 Load Assignments

Listing A.7 Joint Load Assignment

```

public void Canvas_MouseClick(object sender, MouseEventArgs e)
{

    if (e.Button == MouseButtons.Right)
    {
        drawingMode = false;
        pictureBox1.MouseClick -= Canvas_MouseClick;
```

```
9           currentPoint = null; //   karken   sadece ge ici noktay
                s f r la
10          return;
11      }
12
13      float coordX = (e.X - margin) / scaleX + coordsList.Min(p => p
            .X);
14      float coordY = (bitmapHeight - (e.Y - margin)) / scaleY +
            coordsList.Min(p => p.Y);
15      Point clickedPoint = new Point((int)Math.Round(coordX), (int)
            Math.Round(coordY));
16      int closestNodeIndex = -1;
17      double minDistance = double.MaxValue;
18      for (int i = 0; i < Nodes.Count; i++)
19      {
20          PointF node = NodeCoords[i];
21          double distance = Math.Sqrt(Math.Pow(clickedPoint.X - node
                .X, 2) + Math.Pow(clickedPoint.Y - node.Y, 2));
22          if (distance < minDistance)
23          {
24              minDistance = distance;
25              closestNodeIndex = i + 1;
26          }
27      }
28      if (closestNodeIndex != -1)
29      {
30          ShowLoadInputDialog(closestNodeIndex);
31      }
32  }
33  private void InitializeLoadsMatrix()
34  {
35      if (loadsMatrix == null || loadsMatrix.Count != Nodes.Count *
            3)
36      {
37          loadsMatrix = Vector<double>.Build.Dense(Nodes.Count * 3,
                0.0);
38      }
39      if (loadsMatrixG == null || loadsMatrixG.Count != Nodes.Count
            * 3)
40      {
41          loadsMatrixG = Vector<double>.Build.Dense(Nodes.Count * 3,
                0.0);
42      }
43  }
44  private void ShowLoadInputDialog(int nodeIndex)
45  {
46      InitializeLoadsMatrix();
47      Form loadInputForm = new Form
48      {
49          Text = $"Node␣{nodeIndex}",
50          Size = new Size(250, 200),
51          StartPosition = FormStartPosition.CenterScreen
52      };
53      Label fxLabel = new Label { Text = "Fx:", Location = new Point
            (10, 10), AutoSize = true };
54      TextBox fxTextBox = new TextBox { Location = new Point(50, 10)
            , Width = 100 };
```

```
55      Label fyLabel = new Label { Text = "Fz:", Location = new Point
            (10, 40), AutoSize = true };
56      TextBox fyTextBox = new TextBox { Location = new Point(50, 40)
            , Width = 100 };
57      Label mLabel = new Label { Text = "Fy:", Location = new Point
            (10, 70), AutoSize = true };
58      TextBox mTextBox = new TextBox { Location = new Point(50, 70),
             Width = 100 };
59      Button submitButton = new Button { Text = "Submit", Location =
             new Point(50, 110) };
60      submitButton.Click += (sender, e) =>
61      {
62          double.TryParse(fxTextBox.Text, out double fx);
63          double.TryParse(fyTextBox.Text, out double fy);
64          double.TryParse(mTextBox.Text, out double m);
65          AssignLoadToNode(nodeIndex, fx, fy, m);
66          loadInputForm.Close();
67          Bitmap bitmap = new Bitmap(pictureBox1.Image);
68          using (Graphics g = Graphics.FromImage(bitmap))
69          {
70              GeometryHelper.DrawNodeForces(NodeCoords, loadsMatrixG
                    , scaleX, scaleY, margin, bitmapHeight, Nodes, g);
71          }
72          pictureBox1.Image = bitmap;
73      };
74      loadInputForm.Controls.Add(fxLabel);
75      loadInputForm.Controls.Add(fxTextBox);
76      loadInputForm.Controls.Add(fyLabel);
77      loadInputForm.Controls.Add(fyTextBox);
78      loadInputForm.Controls.Add(mLabel);
79      loadInputForm.Controls.Add(mTextBox);
80      loadInputForm.Controls.Add(submitButton);
81      loadInputForm.ShowDialog();
82  }
83  private void AssignLoadToNode(int nodeIndex, double fx, double fz,
        double fy)
84  {
85      int baseIndex = (nodeIndex - 1) * 3;
86      loadsMatrix[baseIndex] += fx;
87      loadsMatrix[baseIndex + 1] += fz;
88      loadsMatrix[baseIndex + 2] += fy;
89      loadsMatrixG[baseIndex] += fx;
90      loadsMatrixG[baseIndex + 1] += fz;
91      loadsMatrixG[baseIndex + 2] += fy;
92  }
```

Listing A.8 Point Load Assignment

```
1
2   using System;
3   using System.Collections.Generic;
4   using System.ComponentModel;
5   using System.Data;
6   using System.Drawing;
7   using System.Linq;
8   using System.Text;
9   using System.Threading.Tasks;
10  using System.Windows.Forms;
```

```
using Tez5.Classes;
using MathNet.Numerics.Integration;
using System.Diagnostics;
using System.Reflection;
using MathNet.Numerics.LinearAlgebra;
using System.Security.Policy;
using System.Drawing.Printing;

namespace Tez5.Forms.LoadForms
{
public partial class PointLoad : Form
{
private Form1 _form1;
public Vector<double> PointLoadVector { get; private set; }
public PointLoad(List<LineElement> lineElements, List<int> Nodes,
    List<Loads> Pload, Form1 form1)
{
    InitializeComponent();
    _form1 = form1;
    comboBox1.DisplayMember = "ElementNo";
    comboBox1.ValueMember = "ElementNo";
    comboBox1.DataSource = lineElements;
    PointLoadVector = Vector<double>.Build.Dense((Nodes.Count) *
        3);
}
private void button1_Click(object sender, EventArgs e)
{
    if (checkBox1.Checked)
    {
        var selectedElement = (LineElement)comboBox1.SelectedItem;
        double L = selectedElement.ElementLength;
        List<double> ll1 = new List<double>();
        List<double> q = new List<double>();
        ll1.Clear();
        q.Clear();
        foreach (DataGridViewRow row in dataGridView1.Rows)
        {
            if (row.Cells[0].Value != null && row.Cells[1].Value
                != null)
            {
                if (double.TryParse(row.Cells[0].Value.ToString(),
                    out double ll1Value) &&
                    double.TryParse(row.Cells[1].Value.ToString(),
                        out double qValue))
                {
                    ll1.Add(ll1Value);
                    q.Add(qValue);
                    selectedElement.PointLoadLocations = ll1;
                    selectedElement.PointLoadValuesNx = q.Select(
                        val => 1 * val).ToList();
                    selectedElement.PointLoadValuesVx = q.Select(
                        val => 0 * val).ToList();
                 }
            }
        }
        Func<double, double> N1 = (x) => (1 - x / L);
        Func<double, double> N2 = (x) => 0;
        Func<double, double> N3 = (x) => 0;
```

```
62          Func<double, double> N4 = (x) => x / L;
63          Func<double, double> N5 = (x) => 0;
64          Func<double, double> N6 = (x) => 0;
65          List<double> ll = ll1.Select(x => (x / 100.0) * L).ToList
                ();
66          List<double[]> nodalForces = new List<double[]>();
67          for (int i = 0; i < q.Count; i++)
68          {
69              double x = ll[i];
70              double F1 = q[i] * N1(x); // N1
71              double F2 = q[i] * N2(x); // N2
72              double F3 = q[i] * N3(x); // N3
73              double F4 = q[i] * N4(x); // N4
74              double F5 = q[i] * N5(x); // N5
75              double F6 = q[i] * N6(x); // N6
76              nodalForces.Add(new double[] { F1, -F2, -F3, F4, -F5,
                    -F6 });
77          }
78          Matrix<double> topologyMatrix = selectedElement.
                El_TopologyMatrix;
79          Matrix<double> TransformationMatrix = selectedElement.
                TransformationMatrix;
80          double[] totalNodalForcesArray = new double[6];
81          foreach (var forces in nodalForces)
82          {
83              totalNodalForcesArray[0] += forces[0];
84              totalNodalForcesArray[1] += forces[1];
85              totalNodalForcesArray[2] += forces[2];
86              totalNodalForcesArray[3] += forces[3];
87              totalNodalForcesArray[4] += forces[4];
88              totalNodalForcesArray[5] += forces[5];
89          }
90          Vector<double> totalNodalForces = Vector<double>.Build.
                Dense(totalNodalForcesArray);
91          selectedElement.El_LoadVector += totalNodalForces;
92          _form1.DrawPointLoad1(selectedElement, ll1, q);
93      }
94      if (checkBox2.Checked)
95      {
96          var selectedElement = (LineElement)comboBox1.SelectedItem;
97          double L = selectedElement.ElementLength;
98          List<double> ll1 = new List<double>();
99          List<double> q = new List<double>();
100         ll1.Clear();
101         q.Clear();
102         foreach (DataGridViewRow row in dataGridView1.Rows)
103         {
104             if (row.Cells[0].Value != null && row.Cells[1].Value
                    != null)
105             {
106                 if (double.TryParse(row.Cells[0].Value.ToString(),
                        out double ll1Value) &&
107                     double.TryParse(row.Cells[1].Value.ToString(),
                            out double qValue))
108                 {
109                     ll1.Add(ll1Value);
110                     q.Add(qValue);
111                     selectedElement.PointLoadLocations = ll1;
```

```
                    selectedElement.PointLoadValuesNx = q.Select(
                        val => 0 * val).ToList();
                    selectedElement.PointLoadValuesVx = q;
                }
            }
        }
        Func<double, double> N1 = (x) => 0;
        Func<double, double> N2 = (x) => 1 - 3 * Math.Pow(x, 2) /
            Math.Pow(L, 2) + 2 * Math.Pow(x, 3) / Math.Pow(L, 3);
        Func<double, double> N3 = (x) => x - 2 * Math.Pow(x, 2) /
            L + Math.Pow(x, 3) / Math.Pow(L, 2);
        Func<double, double> N4 = (x) => 0;
        Func<double, double> N5 = (x) => 3 * Math.Pow(x, 2) / Math
            .Pow(L, 2) - 2 * Math.Pow(x, 3) / Math.Pow(L, 3);
        Func<double, double> N6 = (x) => -Math.Pow(x, 2) / L +
            Math.Pow(x, 3) / Math.Pow(L, 2);
        List<double> ll = ll1.Select(x => (x / 100.0) * L).ToList
            ();
        List<double[]> nodalForces = new List<double[]>();
        for (int i = 0; i < q.Count; i++)
        {
            double x = ll[i];
            double F1 = q[i] * N1(x); // N1
            double F2 = q[i] * N2(x); // N2
            double F3 = q[i] * N3(x); // N3
            double F4 = q[i] * N4(x); // N4
            double F5 = q[i] * N5(x); // N5
            double F6 = q[i] * N6(x); // N6
            nodalForces.Add(new double[] { -F1, -F2, -F3, -F4, -F5
                , -F6 });
        }
        Matrix<double> topologyMatrix = selectedElement.
            El_TopologyMatrix;
        Matrix<double> TransformationMatrix = selectedElement.
            TransformationMatrix;
        double[] totalNodalForcesArray = new double[6];
        foreach (var forces in nodalForces)
        {
            totalNodalForcesArray[0] += forces[0];
            totalNodalForcesArray[1] += forces[1];
            totalNodalForcesArray[2] += forces[2];
            totalNodalForcesArray[3] += forces[3];
            totalNodalForcesArray[4] += forces[4];
            totalNodalForcesArray[5] += forces[5];
        }
        Vector<double> totalNodalForces = Vector<double>.Build.
            Dense(totalNodalForcesArray);
        selectedElement.El_LoadVector += totalNodalForces;
        _form1.DrawPointLoad2(selectedElement, ll1,q);
    }
    if (checkBox3.Checked)
    {
        var selectedElement = (LineElement)comboBox1.SelectedItem;
        double sin = selectedElement.ElementSin;
        double cos = selectedElement.ElementCos;
        double L = selectedElement.ElementLength;
        List<double> ll1 = new List<double>();
        List<double> q = new List<double>();
```

```
160         ll1.Clear();
161         q.Clear();
162         foreach (DataGridViewRow row in dataGridView1.Rows)
163         {
164             if (row.Cells[0].Value != null && row.Cells[1].Value
                    != null)
165             {
166                 if (double.TryParse(row.Cells[0].Value.ToString(),
                        out double ll1Value) &&
167                     double.TryParse(row.Cells[1].Value.ToString(),
                            out double qValue))
168                 {
169                     ll1.Add(ll1Value);
170                     q.Add(qValue);
171                     selectedElement.PointLoadLocations = ll1;
172                     selectedElement.PointLoadValuesNx = q.Select(
                            val => selectedElement.ElementCos * val).
                            ToList();
173                     selectedElement.PointLoadValuesVx = q.Select(
                            val => selectedElement.ElementSin * val).
                            ToList();
174                 }
175             }
176         }
177         Func<double, double> N1 = (x) => (1 - x / L) * cos;
178         Func<double, double> N2 = (x) => (1 - 3 * Math.Pow(x, 2) /
                Math.Pow(L, 2) + 2 * Math.Pow(x, 3) / Math.Pow(L, 3))
                * sin;
179         Func<double, double> N3 = (x) => (x - 2 * Math.Pow(x, 2) /
                L + Math.Pow(x, 3) / Math.Pow(L, 2)) * sin;
180         Func<double, double> N4 = (x) => (x / L) * cos;
181         Func<double, double> N5 = (x) => (3 * Math.Pow(x, 2) /
                Math.Pow(L, 2) - 2 * Math.Pow(x, 3) / Math.Pow(L, 3)) *
                 sin;
182         Func<double, double> N6 = (x) => (-Math.Pow(x, 2) / L +
                Math.Pow(x, 3) / Math.Pow(L, 2)) * sin;
183         List<double> ll = ll1.Select(x => (x / 100.0) * L).ToList
                ();
184         List<double[]> nodalForces = new List<double[]>();
185         for (int i = 0; i < q.Count; i++)
186         {
187             double x = ll[i];
188             double F1 = q[i] * N1(x); // N1
189             double F2 = q[i] * N2(x); // N2
190             double F3 = q[i] * N3(x); // N3
191             double F4 = q[i] * N4(x); // N4
192             double F5 = q[i] * N5(x); // N5
193             double F6 = q[i] * N6(x); // N6
194             nodalForces.Add(new double[] { F1, -F2, -F3, F4, -F5,
                    -F6 });
195         }
196         Matrix<double> topologyMatrix = selectedElement.
                El_TopologyMatrix;  // Topoloji matrisini al yoruz
197         Matrix<double> TransformationMatrix = selectedElement.
                TransformationMatrix;
198         double[] totalNodalForcesArray = new double[6];  // Kuvvet
                 vekt r
199         foreach (var forces in nodalForces)
```

```
200         {
201             totalNodalForcesArray[0] += forces[0];
202             totalNodalForcesArray[1] += forces[1];
203             totalNodalForcesArray[2] += forces[2];
204             totalNodalForcesArray[3] += forces[3];
205             totalNodalForcesArray[4] += forces[4];
206             totalNodalForcesArray[5] += forces[5];
207         }
208         Vector<double> totalNodalForces = Vector<double>.Build.
               Dense(totalNodalForcesArray);
209         selectedElement.El_LoadVector += totalNodalForces;
210         _form1.DrawPointLoad3(selectedElement, ll1, q);
211     }
212     if (checkBox5.Checked)
213     {
214         var selectedElement = (LineElement)comboBox1.SelectedItem;
215         double L = selectedElement.ElementLength;
216         double sin = selectedElement.ElementSin;
217         double cos = selectedElement.ElementCos;
218         List<double> ll1 = new List<double>();
219         List<double> q = new List<double>();
220         ll1.Clear();
221         q.Clear();
222         foreach (DataGridViewRow row in dataGridView1.Rows)
223         {
224             if (row.Cells[0].Value != null && row.Cells[1].Value
                   != null)
225             {
226                 if (double.TryParse(row.Cells[0].Value.ToString(),
                        out double ll1Value) &&
227                     double.TryParse(row.Cells[1].Value.ToString(),
                            out double qValue))
228                 {
229                     ll1.Add(ll1Value);
230                     q.Add(qValue);
231                     selectedElement.PointLoadLocations = ll1;
232                     selectedElement.PointLoadValuesNx = q.Select(
                           val => selectedElement.ElementSin * val).
                           ToList();
233                     selectedElement.PointLoadValuesVx = q.Select(
                           val => selectedElement.ElementCos * val).
                           ToList();
234                 }
235             }
236         }
237         Func<double, double> N1 = (x) => (1 - x / L) * sin;
238         Func<double, double> N2 = (x) => (1 - 3 * Math.Pow(x, 2) /
                Math.Pow(L, 2) + 2 Math.Pow(x, 3) / Math.Pow(L, 3)) *
               cos;
239         Func<double, double> N3 = (x) => (x - 2 * Math.Pow(x, 2) /
                L + Math.Pow(x, 3) / Math.Pow(L, 2)) * cos;
240         Func<double, double> N4 = (x) => (x / L) * sin;
241         Func<double, double> N5 = (x) => (3 * Math.Pow(x, 2) /
               Math.Pow(L, 2) - 2 * Math.Pow(x, 3) / Math.Pow(L, 3)) *
                cos;
242         Func<double, double> N6 = (x) => (-Math.Pow(x, 2) / L +
               Math.Pow(x, 3) / Math.Pow(L, 2)) * cos;
243         List<double> ll = ll1.Select(x => (x / 100.0) * L).ToList
```

```
            ();
244         List<double[]> nodalForces = new List<double[]>();
245         for (int i = 0; i < q.Count; i++)
246         {
247             double x = ll[i];
248             double F1 = q[i] * N1(x); // N1
249             double F2 = q[i] * N2(x); // N2
250             double F3 = q[i] * N3(x); // N3
251             double F4 = q[i] * N4(x); // N4
252             double F5 = q[i] * N5(x); // N5
253             double F6 = q[i] * N6(x); // N6
254             nodalForces.Add(new double[] { -F1, -F2, -F3, -F4, -F5
                    , -F6 });
255         }
256         Matrix<double> topologyMatrix = selectedElement.
                El_TopologyMatrix;
257         Matrix<double> TransformationMatrix = selectedElement.
                TransformationMatrix;
258         double[] totalNodalForcesArray = new double[6];
259         foreach (var forces in nodalForces)
260         {
261             totalNodalForcesArray[0] += forces[0];
262             totalNodalForcesArray[1] += forces[1];
263             totalNodalForcesArray[2] += forces[2];
264             totalNodalForcesArray[3] += forces[3];
265             totalNodalForcesArray[4] += forces[4];
266             totalNodalForcesArray[5] += forces[5];
267         }
268         Vector<double> totalNodalForces = Vector<double>.Build.
                Dense(totalNodalForcesArray);
269         selectedElement.El_LoadVector += totalNodalForces;
270         _form1.DrawPointLoad5(selectedElement, ll1, q);
271     }
272 }
273 private void Save_Click(object sender, EventArgs e)
274 {
275     this.DialogResult = DialogResult.OK;
276     this.Close();
277 }
278 }
279 }
```

Listing A.9 Temperature Load Assignment

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Data.SQLite;
6  using System.Diagnostics;
7  using System.Drawing;
8  using System.Linq;
9  using System.Text;
10 using System.Threading.Tasks;
11 using System.Windows.Forms;
12 using Tez5.Classes;
13 namespace Tez5.Forms.LoadForms
14 {
```

```
15      public partial class TempLoad : Form
16      {
17          private Form1 _form1;
18          public TempLoad(List<LineElement> lineElements, Form1
                form1)
19          {
20              InitializeComponent();
21              _form1 = form1;
22              comboBox1.DisplayMember = "ElementNo";
23              comboBox1.ValueMember = "ElementNo";
24              comboBox1.DataSource = lineElements;
25          }
26          private void save_Click(object sender, EventArgs e)
27          {
28              var selectedElement = (LineElement)comboBox1.
                    SelectedItem;
29              if (TempA.Text == TempU.Text)
30              {
31                  double E = selectedElement.Elasticity;
32                  double A = selectedElement.Area;
33                  double a = 0.00001170;
34                  double T = double.Parse(TempU.Text);
35                  double Fthermal = E * A * a * T;
36                  selectedElement.El_TempLoadVector[0] += -1 *
                        Fthermal;
37                  selectedElement.El_TempLoadVector[3] += Fthermal;
38                  selectedElement.El_LoadVector += selectedElement.
                        El_TempLoadVector;
39                  MessageBox.Show("El_TempLoadVector:␣" + string.
                        Join(",␣", selectedElement.El_LoadVector));
40              }
41              else
42              {
43                      double E = selectedElement.Elasticity;
44                      double A = selectedElement.Area;
45                      double a = 0.00001170;
46                      double I = selectedElement.Ix;
47                      double h = selectedElement.Height;
48                      double T = (double.Parse(TempA.Text)-double.
                            Parse(TempU.Text))/h;
49                      double Mthermal = E * I * a * T;
50                      selectedElement.El_GradTempLoadVector[2] += -1
                             * Mthermal;
51                      selectedElement.El_GradTempLoadVector[5] +=
                            Mthermal;
52                      selectedElement.El_LoadVector +=
                            selectedElement.El_GradTempLoadVector;
53                      MessageBox.Show("El_GradTempLoadVector:␣" +
                            string.Join(",␣", selectedElement.
                            El_LoadVector));
54              }
55          }
56      }
57  }
```

B Numerical Results of Case Study: A

B.1 Element Stiffness Matrices

$$\mathbf{k}_{el1} = \begin{bmatrix} 1099.35 & 0.00 & -1099.35 & -1099.35 & 0.00 & -1099.35 \\ 0.00 & 222600.00 & 0.00 & 0.00 & -222600.00 & 0.00 \\ -1099.35 & 0.00 & 1465.80 & 1099.35 & 0.00 & 732.90 \\ -1099.35 & 0.00 & 1099.35 & 1099.35 & 0.00 & 1099.35 \\ 0.00 & -222600.00 & 0.00 & 0.00 & 222600.00 & 0.00 \\ -1099.35 & 0.00 & 732.90 & 1099.35 & 0.00 & 1465.80 \end{bmatrix}$$

$$\mathbf{k}_{el2} = \begin{bmatrix} 55472.08 & 41397.22 & -206.84 & -55472.08 & -41397.22 & -206.84 \\ 41397.22 & 31323.70 & 275.79 & -41397.22 & -31323.70 & 275.79 \\ -206.84 & 275.79 & 574.56 & 206.84 & -275.79 & 287.28 \\ -55472.08 & -41397.22 & 206.84 & 55472.08 & 41397.22 & 206.84 \\ -41397.22 & -31323.70 & -275.79 & 41397.22 & 31323.70 & -275.79 \\ -206.84 & 275.79 & 287.28 & 206.84 & -275.79 & 574.56 \end{bmatrix}$$

$$\mathbf{k}_{el3} = \begin{bmatrix} 2605.87 & 0.00 & 1954.40 & -2605.87 & 0.00 & 1954.40 \\ 0.00 & 296800.00 & 0.00 & 0.00 & -296800.00 & 0.00 \\ 1954.40 & 0.00 & 1954.40 & -1954.40 & 0.00 & 977.20 \\ -2605.87 & 0.00 & -1954.40 & 2605.87 & 0.00 & -1954.40 \\ 0.00 & -296800.00 & 0.00 & 0.00 & 296800.00 & 0.00 \\ 1954.40 & 0.00 & 977.20 & -1954.40 & 0.00 & 1954.40 \end{bmatrix}$$

$$\mathbf{k}_{el4} = \begin{bmatrix} 1099.35 & 0.00 & 1099.35 & -1099.35 & 0.00 & 1099.35 \\ 0.00 & 222600.00 & 0.00 & 0.00 & -222600.00 & 0.00 \\ 1099.35 & 0.00 & 1465.80 & -1099.35 & 0.00 & 732.90 \\ -1099.35 & 0.00 & -1099.35 & 1099.35 & 0.00 & -1099.35 \\ 0.00 & -222600.00 & 0.00 & 0.00 & 222600.00 & 0.00 \\ 1099.35 & 0.00 & 732.90 & -1099.35 & 0.00 & 1465.80 \end{bmatrix}$$

$$\mathbf{k}_{el5} = \begin{bmatrix} 222600.00 & 0.00 & 0.00 & -222600.00 & 0.00 & 0.00 \\ 0.00 & 1099.35 & 1099.35 & 0.00 & -1099.35 & 1099.35 \\ 0.00 & 1099.35 & 1465.80 & 0.00 & -1099.35 & 732.90 \\ -222600.00 & 0.00 & 0.00 & 222600.00 & 0.00 & 0.00 \\ 0.00 & -1099.35 & -1099.35 & 0.00 & 1099.35 & -1099.35 \\ 0.00 & 1099.35 & 732.90 & 0.00 & -1099.35 & 1465.80 \end{bmatrix}$$

B.2 Reduced Global Stiffness Matrix

$$\mathbf{K} = \begin{bmatrix} 279171.43 & 41397.22 & 892.51 & -55472.08 & -41397.22 & -206.84 & -222600.00 & 0.00 & 0.00 & 0.00 \\ 41397.22 & 255023.05 & 1375.14 & -41397.22 & -31323.70 & 275.79 & 0.00 & -1099.35 & 1099.35 & -1099.35 \\ 892.51 & 1375.14 & 3506.16 & 206.84 & -275.79 & 287.28 & 0.00 & -1099.35 & 732.90 & -1099.35 \\ -55472.08 & -41397.22 & 206.84 & 58077.95 & 41397.22 & 2161.24 & -2605.87 & 0.00 & 1954.40 & 0.00 \\ -41397.22 & -31323.70 & -275.79 & 41397.22 & 328123.70 & -275.79 & 0.00 & -296800.00 & 0.00 & 0.00 \\ -206.84 & 275.79 & 287.28 & 2161.24 & -275.79 & 2528.96 & -1954.40 & 0.00 & 977.20 & 0.00 \\ -222600.00 & 0.00 & 0.00 & -2605.87 & 0.00 & -1954.40 & 226305.22 & 0.00 & -855.05 & 0.00 \\ 0.00 & -1099.35 & -1099.35 & 0.00 & -296800.00 & 0.00 & 0.00 & 520499.35 & -1099.35 & 1099.35 \\ 0.00 & 1099.35 & 732.90 & 1954.40 & 0.00 & 977.20 & -855.05 & -1099.35 & 4886.00 & -1099.35 \\ 0.00 & -1099.35 & -1099.35 & 0.00 & 0.00 & 0.00 & 0.00 & 1099.35 & -1099.35 & 1099.35 \end{bmatrix}$$

C Numerical Results of Case Study: B

C.1 Element Stiffness Matrices

$$\mathbf{k}_{el1} = \begin{bmatrix} 538.65 & 0.00 & -538.65 & -538.65 & 0.00 & -538.65 \\ 0.00 & 108150.00 & 0.00 & 0.00 & -108150.00 & 0.00 \\ -538.65 & 0.00 & 718.20 & 538.65 & 0.00 & 359.10 \\ -538.65 & 0.00 & 538.65 & 538.65 & 0.00 & 538.65 \\ 0.00 & -108150.00 & 0.00 & 0.00 & 108150.00 & 0.00 \\ -538.65 & 0.00 & 359.10 & 538.65 & 0.00 & 718.20 \end{bmatrix}$$

$$\mathbf{k}_{el2} = \begin{bmatrix} 4309.20 & 0.00 & -2154.60 & -4309.20 & 0.00 & -2154.60 \\ 0.00 & 216300.00 & 0.00 & 0.00 & -216300.00 & 0.00 \\ -2154.60 & 0.00 & 1436.40 & 2154.60 & 0.00 & 718.20 \\ -4309.20 & 0.00 & 2154.60 & 4309.20 & 0.00 & 2154.60 \\ 0.00 & -216300.00 & 0.00 & 0.00 & 216300.00 & 0.00 \\ -2154.60 & 0.00 & 718.20 & 2154.60 & 0.00 & 1436.40 \end{bmatrix}$$

$$\mathbf{k}_{el3} = \begin{bmatrix} 108150.00 & 0.00 & 0.00 & -108150.00 & 0.00 & 0.00 \\ 0.00 & 538.65 & 538.65 & 0.00 & -538.65 & 538.65 \\ 0.00 & 538.65 & 718.20 & 0.00 & -538.65 & 359.10 \\ -108150.00 & 0.00 & 0.00 & 108150.00 & 0.00 & 0.00 \\ 0.00 & -538.65 & -538.65 & 0.00 & 538.65 & -538.65 \\ 0.00 & 538.65 & 359.10 & 0.00 & -538.65 & 718.20 \end{bmatrix}$$

$$\mathbf{k}_{el4} = \begin{bmatrix} 4309.20 & 0.00 & 2154.60 & -4309.20 & 0.00 & 2154.60 \\ 0.00 & 216300.00 & 0.00 & 0.00 & -216300.00 & 0.00 \\ 2154.60 & 0.00 & 1436.40 & -2154.60 & 0.00 & 718.20 \\ -4309.20 & 0.00 & -2154.60 & 4309.20 & 0.00 & -2154.60 \\ 0.00 & -216300.00 & 0.00 & 0.00 & 216300.00 & 0.00 \\ 2154.60 & 0.00 & 718.20 & -2154.60 & 0.00 & 1436.40 \end{bmatrix}$$

$$\mathbf{k}_{el5} = \begin{bmatrix} 538.65 & 0.00 & 538.65 & -538.65 & 0.00 & 538.65 \\ 0.00 & 108150.00 & 0.00 & 0.00 & -108150.00 & 0.00 \\ 538.65 & 0.00 & 718.20 & -538.65 & 0.00 & 359.10 \\ -538.65 & 0.00 & -538.65 & 538.65 & 0.00 & -538.65 \\ 0.00 & -108150.00 & 0.00 & 0.00 & 108150.00 & 0.00 \\ 538.65 & 0.00 & 359.10 & -538.65 & 0.00 & 718.20 \end{bmatrix}$$

$$\mathbf{k}_{el6} = \begin{bmatrix} 108150.00 & 0.00 & 0.00 & -108150.00 & 0.00 & 0.00 \\ 0.00 & 538.65 & 538.65 & 0.00 & -538.65 & 538.65 \\ 0.00 & 538.65 & 718.20 & 0.00 & -538.65 & 359.10 \\ -108150.00 & 0.00 & 0.00 & 108150.00 & 0.00 & 0.00 \\ 0.00 & -538.65 & -538.65 & 0.00 & 538.65 & -538.65 \\ 0.00 & 538.65 & 359.10 & 0.00 & -538.65 & 718.20 \end{bmatrix}$$

C.2 Reduced Global Stiffness Matrix

$$
\mathbf{K} = \begin{bmatrix}
112997.85 & 0.00 & -1615.95 & -4309.20 & 0.00 & -2154.60 & 0.00 & 0.00 & 0.00 & -108150.00 \\
0.00 & 0.00 & & & & & & & & \\
0.00 & 324988.65 & 538.65 & 0.00 & -216300.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-538.65 & 538.65 & & & & & & & & \\
-1615.95 & 538.65 & 2872.80 & 2154.60 & 0.00 & 718.20 & 0.00 & 0.00 & 0.00 & 0.00 \\
-538.65 & 359.10 & & & & & & & & \\
-4309.20 & 0.00 & 2154.60 & 112459.20 & 0.00 & 2154.60 & -108150.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & & & & & & & & \\
0.00 & -216300.00 & 0.00 & 0.00 & 216838.65 & 538.65 & 0.00 & -538.65 & 538.65 & 0.00 \\
0.00 & 0.00 & & & & & & & & \\
-2154.60 & 0.00 & 718.20 & 2154.60 & 538.65 & 2154.60 & 0.00 & -538.65 & 359.10 & 0.00 \\
0.00 & 0.00 & & & & & & & & \\
0.00 & 0.00 & 0.00 & -108150.00 & 0.00 & 0.00 & 112459.20 & 0.00 & 2154.60 & -4309.20 \\
0.00 & 2154.60 & & & & & & & & \\
0.00 & 0.00 & 0.00 & 0.00 & -538.65 & -538.65 & 0.00 & 216838.65 & -538.65 & 0.00 \\
-216300.00 & 0.00 & & & & & & & & \\
0.00 & 0.00 & 0.00 & 0.00 & 538.65 & 359.10 & 2154.60 & -538.65 & 2154.60 & -2154.60 \\
0.00 & 718.20 & & & & & & & & \\
-108150.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -4309.20 & 0.00 & -2154.60 & 112997.85 \\
0.00 & -1615.95 & & & & & & & & \\
0.00 & -538.65 & -538.65 & 0.00 & 0.00 & 0.00 & 0.00 & -216300.00 & 0.00 & 0.00 \\
324988.65 & -538.65 & & & & & & & & \\
0.00 & 538.65 & 359.10 & 0.00 & 0.00 & 0.00 & 2154.60 & 0.00 & 718.20 & -1615.95 \\
-538.65 & 2872.80 & & & & & & & &
\end{bmatrix}
$$

PUBLICATIONS FROM THE THESIS

Conference Papers

1. A. Yaşar and O. Yılmaz, "Development of a Structural Analysis Software for Plane Frame Systems," in *Proc. International Graduate Studies Congress (IGSC)*, pp. 300–310, 2025.