

Elastic analysis of 3D frames using MATLAB software



A Part of the FINAL YEAR PROJECT UG - 2017

By

Naimat Ali (G.L)	00000210272
Tahir Ali Khan	00000219592
Usama Fazal	00000237572
Hafiz Muhammad Ahmad	00000220559

NUST Institute of Civil Engineering (NICE)
School of Civil and Environmental Engineering (SCEE)
National University of Sciences and Technology (NUST)
Islamabad, Pakistan
Year 2021

This is to certify
that the Final
Year Project
Titled

Elastic analysis of 3D frames using MATLAB software

submitted by

Naimat Ali – 210272

Tahir Ali Khan – 219592

Usama Fazal – 237572

Muhammad Ahmad – 220559

has been accepted towards the
requirements for the undergraduate
degree

in

CIVIL ENGINEERING

Samiullah Khan Bangash

Lecturer

NUST Institute of Civil Engineering

School of Civil and Environmental Engineering

National University of Sciences
and Technology, Islamabad, Pakistan

ABSTRACT

People looking for software to solve linear elastic problems in structures face myriad challenges. The graphical user interface of the available software – such as ETABs – has a steep learning curve. If someone wants quick results – e.g., about deflections in a linear elastic structure – the procedure is laborious and time consuming.

While it is imperative to have a firm grasp on these structural softwares, sometimes a tool that does just one job but does it faster and in a simpler and more direct way is what is needed. After consulting with various students, it was ascertained that the initial learning curve of the existing structural softwares deterred many of them from even getting started. This is exactly where our software comes in and fills the gap. Quicker linear elastic structure analysis and in a way that is user friendly to even an undergrad student.

The quickness of the MATLAB based software depends on the way it is coded. Smart and efficient coding is the core of our project. Further, the friendliness of the GUI was determined by various experiments involving many trial runs – which were conducted with the help of our fellow batchmates.

The final aim of our project was to provide a pathway for students who want to develop a similar software. The steps of creating the code, the GUI, and the content used is provided in this thesis. It was imperative to our initial goals to have a project which contributed to the expansion of knowledge for anyone who wants to delve into programming as it is related to civil engineering. The mechanics of the code, the appearance of the GUI and the trials performed, they are all listed here in detail to help people interested in learning how to create a similar software.

COPYRIGHT NOTICE

In the spirit of knowledge and our love for civil engineering, we make the entire work done in this thesis to be available for use by the general public. The very purpose of our project was to make a product that was accessible to the general student – not only when it comes to a user-friendly interface, but also the code. Keeping in mind our main target that we initially set out, it is declared that the contents of this thesis may be used as by anyone. However, it is a reasonable expectation that our work be cited whenever it is used in another academic paper.

Further, it is stated that the work done in this dissertation is entirely our own. From the code to the explanations written here in this thesis. Any references taken from research papers, books, and journals are appropriately mentioned.

DEDICATION

The primary dedication of this project was to facilitate engineers interested in understanding the underlying codes and working of structural problem-solving software. The motivation for this project came when we faced inaccessibility to the underlying workings of structural software. The software that are commercially available, their source codes are not available to the public.

Further, the interface of these software is not user friendly to someone who is looking to get started. The dedication was to create a simpler and straight forward experience for users, who would obtain quick results for their linear elastic structures.

ACKNOWLEDGEMENTS

First and foremost, we would like to acknowledge the role played by our advisor during this project. It was due to his profound insights, sharp intellect, openness in communication, and dedication to the art that is civil engineering that we were able to accomplish the goals we set out to achieve.

There were many moments during our project where we needed direction. And it is our pleasure and good fortune that Engr. Samiullah Bangash was able to guide us. Be it the efficiency of the MATLAB code, of the way the graphical user interface (GUI) appeared, his insights were always valued and gave the direction that we needed in achieving the final product.

TABLE OF CONTENTS

1.0	INTRODUCTION	12
1.1	General	12
1.2	Problem Statement	12
2.0	LITERATURE REVIEW	13
2.1	DSM Method.....	13
2.1.1	System Stiffness Matrix.....	19
2.2	Geometric Stiffness	19
2.3	Element Stiffness Matrix for Axial Effects.....	22
2.4	Coordinate Transformation	23
2.5	Local and Global Coordinate Systems.....	26
2.6	Torsional Effects	27
2.7	Stiffness Matrix: Grid Element	28
2.8	Element Stiffness Matrix.....	29
2.9	Transformation of Coordinates	30
3.0	METHODOLOGY	35
3.1	Initialize Matrices and Degree of Freedom:.....	37
3.2	Transformation Matrix:	37
3.3	Stiffness Matrix:.....	38
3.4	Compartmentalizing the KC matrix:	39
3.5	Finding Displacements:.....	40
3.6	Finding Support Reactions:	40
3.7	Finding Member Forces:	41
3.8	Plotting the frame:.....	41
3.9	Guide To Make Application Using App Designer.....	45
3.9.1	Creating a Main GUI	45
3.9.1.1	MATLAB code.....	45
3.9.1.2	APP designer and MATLAB code	45
3.9.1.3	Design view of main GUI.....	46
3.9.1.4	Properties	49
3.9.1.5	Start Up Function.....	49

3.9.1.6	Directory Function in File Menu:	50
3.9.1.7	Import Data Function in File Menu:	51
3.9.1.8	Quit Function in File Menu:	52
3.9.1.9	Define Nodes in Node Menu:	52
3.9.1.10	Settlement in Node Menu:	52
3.9.1.11	Connectivity Menu:	53
3.9.1.12	Material Properties Menu:	53
3.9.1.13	Sectional Properties Menu:	53
3.9.1.14	Nodal Load Menu:	54
3.9.1.15	Run Button:	54
3.9.1.16	Show Results Button:	58
3.9.1.17	Back to Figure Button:	58
3.9.2	CREATING A SUB GUI:	59
	Nodes GUI:	59
3.9.3	Settlement GUI:	64
3.9.4	Connection GUI:	69
3.9.5	Material GUI:	74
3.9.6	Sectional Properties GUI:	78
3.9.7	Nodal Load GUI:	83
3.10	Ways to Share App:	89
3.10.1	Share MATLAB Files Directly:	89
3.10.2	PACKAGING APPS IN APP-DESIGNER:	90
3.10.3	Creating a Web App:	91
3.10.4	Creating a Standalone Desktop Application:	92
4.0	RESULTS AND DISCUSSION	93
4.1	Comparison with MASTAN2:	93
4.2	Graphs:	95
5.0	CONCLUSION	96
6.0	References	97

LIST OF FIGURES

Figure 2.1 Beam segment showing forces and displacement at the nodal coordinates.....	13
Figure 2.2 Beam element displaying static deflection curves because of a unit displacement at one of the nodal coordinates.....	15
Figure 2.3 (a) “Beam element loaded with arbitrary distributed axial force, (b) Beam element acted on by nodal forces resulting for, displacement $\delta_2 = 1$ undergoing a virtual displacement $\delta_1 = 1$ ”	20
Figure 2.4 Differential segment of deflected beam in Fig. 2.3	21
Figure 2.5 Beam element showing nodal axial loads P_1 , P_2 , and corresponding nodal displacements δ_1 , δ_2	23
Figure 2.6 Beam element showing nodal forces P_i in local (x, y, z) and nodal forces P_1 , in global coordinate axes (X, Y, Z)	25
Figure 2.7 Components of nodal displacements for a grid member. (a) Local coordinate system. (b) Global coordinate system.....	27
Figure 2.8 Nodal torsional coordinates for a beam element.....	28
Figure 2.9 Beam segment of a space frame showing forces and displacements at the nodal coordinates	29
Figure 2.10 Components of a general vector A in local and global coordinates	31
Figure 3.1:MATLAB Code for getting data from input file	37
Figure 3.2: MATLAB Code for making Transformation matrix.....	38
Figure 3.3: MATLAB Code for making Local Stiffness matrix	39
Figure 3.4: MATLAB Code for making Global Stiffness matrix.....	39
Figure 3.5: MATLAB Code for finding restrained and unrestrained nodal co-ordinates	40
Figure 3.6: MATLAB Code for finding deflections	40
Figure 3.7 MATLAB Code for support reactions	41
Figure 3.8: MATLAB Code for finding member forces	41
Figure 3.9: MATLAB Code for plotting actual frame	42
Figure 3.10: General deformation functions for 3D frame element in x-direction	42
Figure 3.11: General deformation functions for 3D frame element in y-direction	43
Figure 3.12: General deformation functions for 3D frame element in z-direction	43
Figure 3.13: Deformed co-ordinates of frame elements	44
Figure 3.14: MATLAB Code for plotting defelected shape of frame.....	44
Figure 3.15: Interface of MATLAB APP designer	45
Figure 3.16: Call back option in MATLAB APP designer	46
Figure 3.17: Components of our GUI	48
Figure 3.18: Layout of Main GUI	48
Figure 3.19: Property Function in MATLAB APP designer	49
Figure 3.20: Start up function in MATLAB APP designer	50
Figure 3.21: Directory Function	50
Figure 3.22:Import Data.....	51
Figure 3.23:Quit Function	52
Figure 3.24Calling Nodes GUI	52
Figure 3.25: Calling Settlement GUI.....	53

Figure 3.26: Calling Connectivity GUI.....	53
Figure 3.27: Calling Material properties GUI	53
Figure 3.28: Calling sectional properties GUI	54
Figure 3.29: Calling Nodal load GUI	54
Figure 3.30: Clearing all data plotted in our APP	55
Figure 3.31: APP designer code for Saving data from user in variable	55
Figure 3.32: APP designer code for pop-up window	55
Figure 3.33: APP designer code for pop-up window	56
Figure 3.34: Saving support reactions in property variable.....	56
Figure 3.35: Saving deformations in property variable	56
Figure 3.36: APP designer code to plotting frame	57
Figure 3.37: APP designer code for plotting deflected shape of frame.....	57
Figure 3.38: Show results function	58
Figure 3.39: Back to figure button	59
Figure 3.40: Design view of nodes GUI	60
Figure 3.41: Inspector window in APP designer	61
Figure 3.42: Property Function	61
Figure 3.43: Add to table function	62
Figure 3.44: Resetting values in add to table function	63
Figure 3.45: Delete row function	63
Figure 3.46: Done Button function	64
Figure 3.47: Design view of Settlement GUI.....	65
Figure 3.48: Property Function in Settlement GUI	65
Figure 3.49: Start up function for Settlement GUI.....	66
Figure 3.50: Add to table function for Settlement GUI	67
Figure 3.51: adding input data in variable	67
Figure 3.52:Resetting variable in add to table function	67
Figure 3.53: Delete row function for Settlement GUI.....	68
Figure 3.54: Done function for Settlement GUI.....	68
Figure 3.55: Design view of Connectivity GUI.....	70
Figure 3.56: : Start up function for Connectivity GUI	71
Figure 3.57: : Add to table function for Connectivity GUI	72
Figure 3.58: adding input data in variable	72
Figure 3.59: Resetting all variables	72
Figure 3.60: Delete Row function for Connectivity GUI.....	73
Figure 3.61: : Done function for Connectivity GUI.....	73
Figure 3.62: Design view of Material properies.....	74
Figure 3.63: Property Function	75
Figure 3.64: Startup functcion For Material properties GUI.....	75
Figure 3.65: Add to table function for Material properties GUI	76
Figure 3.66: : adding input data in variable	76
Figure 3.67: Resetting all values	77
Figure 3.68: Delete row function in Material properties function	77
Figure 3.69:Done function in Material properties GUI.....	78

Figure 3.70:Design view of Sectional Properties GUI	79
Figure 3.71: Property Function for Sectional Properties GUI	79
Figure 3.72: Startup function for Sectional Properties GUI	80
Figure 3.73: Add to table function for Sectional Properties GUI	81
Figure 3.74: Adding input data in variables	81
Figure 3.75: Resetting all values	81
Figure 3.76: Delete row function for Sectional Properties GUI	82
Figure 3.77: Done function for Sectional Properties GUI	82
Figure 3.78: Design view of Nodal Load GUI.....	84
Figure 3.79: Properties for Nodal Load GUI.....	84
Figure 3.80: Startup function for Nodal Load GUI	85
Figure 3.81: Add to table function for Nodal Load GUI	86
Figure 3.82: Adding input data in variables	86
Figure 3.83: Resetting all values	86
Figure 3.84: Delete row function for Nodal Load GUI	87
Figure 3.85: Done button function for Nodal Load GUI.....	87
Figure 3.86: Saving application Data in MATLAB APP designer	89
Figure 3.87: Ways to share APP	90
Figure 3.88: Window for creating MATLAB APP	91
Figure 3.89: Window for creating stand alone software	92
<i>Figure 4.1 3D fixed jointed frame in MASTAN2</i>	<i>93</i>
<i>Figure 4.2 3D fixed jointed frame deflection in MASTAN2.....</i>	<i>94</i>
<i>Figure 4.3 Linear Elastic Frame Solver (LEFS) Deflection.....</i>	<i>94</i>
<i>Figure 4.4 Reaction forces comparison of LEFS with MASTAN2.....</i>	<i>95</i>
<i>Figure 4.5 Deflection comparison of LEFS with MASTAN2</i>	<i>95</i>

CHAPTER 1

1.0 INTRODUCTION

1.1 General

Using the direct stiffness method, a software with a user-friendly GUI was created to solve linear elastic problems.

The procedure of the development of the GUI has been included in detail in the thesis to help anyone make their own software.

Further, the results provided the software have been verified after comparison to other software.

1.2 Problem Statement

To solve linear elastic problems, using the manual method is time consuming. Further, as the number of variables increase, it gets cumbersome to keep all the calculations and to avoid errors. Finally, there is a lot of wastage of paper and the process of writing is laborious.

Secondly, the software available in the market do not have any open source code available. This is a problem because if someone wants to understand the operating principles of such software, they cannot use that software to learn that.

Lastly, there is no guide available to make a GUI for such software.

CHAPTER 2

2.0 LITERATURE REVIEW

2.1 DSM Method

If we consider a beam element of x-sectional MOI I , modulus of elasticity E , and length L as displayed in Fig. 2.1. It is possible to get a relation for the moments and forces named as P_1 , P_2 , P_3 and P_4 and the related linear and angular displacements δ_1 , δ_2 , δ_3 and δ_4 at the ends of the beam element as shown in Fig. 2.1. This relation that we get is the stiffness matrix eq. The displacements δ_i and forces P_i are at the nodal coordinates at the ends for the beam element (Paz & Kim, 2019).

The renowned equation for small transverse displacements of a beam element, is given by the following differential equation

$$EI \frac{d^2u}{dx^2} = M(x) \quad (2.1)$$

where u is the transverse displacement and $M(x)$ is the bending moment at a cross-section 'x' of the beam.

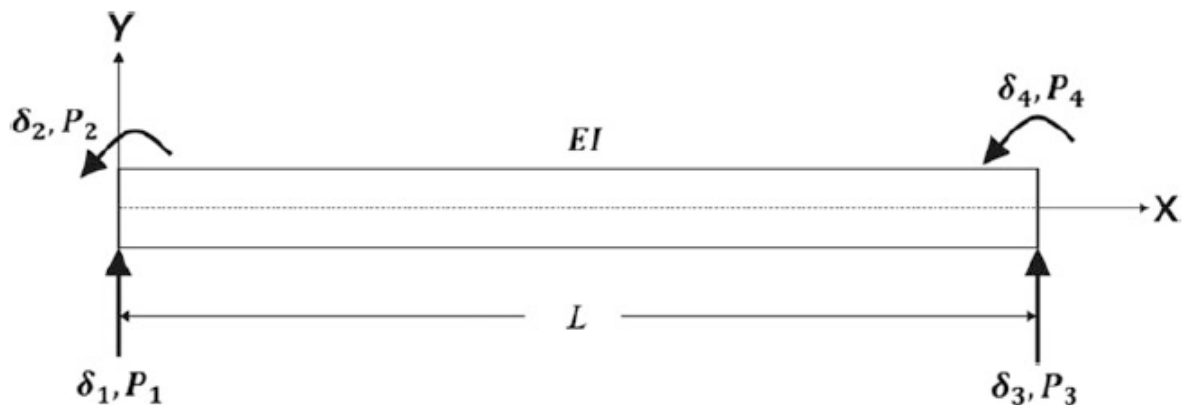


Figure 2.1 Beam segment showing forces and displacement at the nodal coordinates

The differential Equation (2.1) present for a uniform beam element is equal to

$$EI \frac{d^4u}{dx^4} = P(x) \quad (2.2)$$

since

$$EI \frac{M(x)}{dx} = V(x) \quad (2.3)$$

and

$$EI \frac{V(x)}{dx} = P(x) \quad (2.4)$$

In which $V(x)$ is the shear force and $p(x)$ is the load of the beam per unit of length.

First, we state the definition of the stiffness coefficient, designated by k_{ij} , that is, k_{ij} is the force at nodal coordinate i because of a unit displacement at nodal coordinate j while the rest of the nodal coordinates are kept at 0 displacement. “Figure 2.2 demonstrates the displacement curves and the equivalent stiffness coefficients because of a unit displacement at each of the 4 nodal coordinates of the beam element (Paz & Kim, 2019). For the determination of expressions for the stiffness coefficients k_{ij} , we start by the equations for displaced curves demonstrated in Figure 2.2. First, we take the beam element in Figure 2.1 with no loads [$p(x) = 0$], with the exception of the forces P_1 , P_2 , P_3 and P_4 applied at nodal coordinates” (Paz & Kim, 2019). Here, Eq. (2.2) is reduced:

$$\frac{d^4u}{dx^4} = 0 \quad (2.4)$$

Further integrations of Eq. (2.4) gives

$$\frac{d^3u}{dx^3} = C_1$$

$$\frac{d^2u}{dx^2} = C_1x + C_2$$

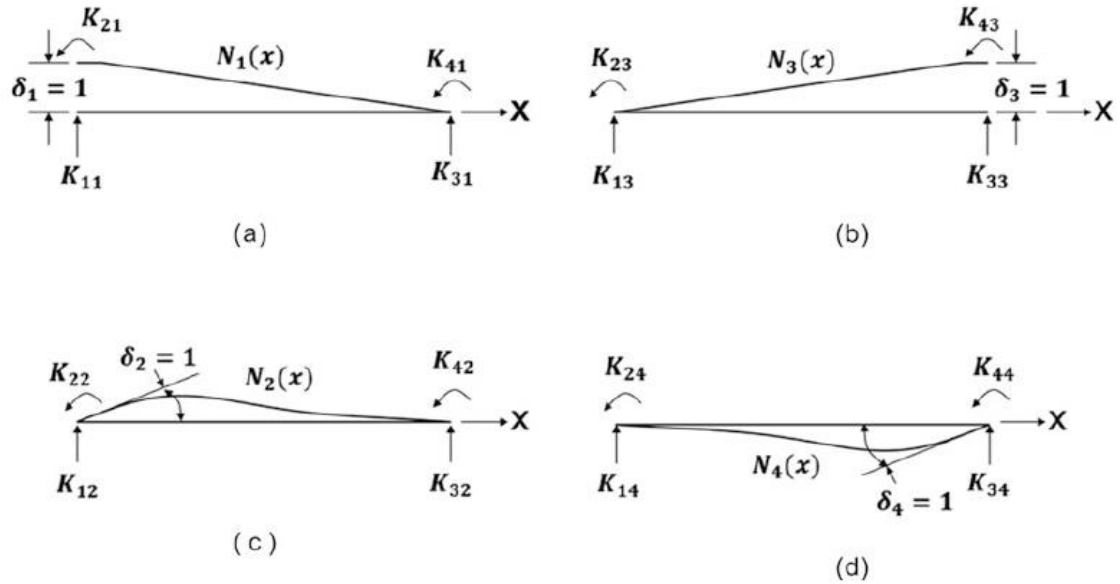


Figure 2.2 Beam element displaying static deflection curves because of a unit displacement at one of the nodal coordinates

$$\frac{du}{dx} = \frac{1}{2}C_1x^2 + C_2x + C_3 \quad (2.5)$$

$$u = \frac{1}{6}C_1x^3 + C_2x^2 + C_3x + C_4 \quad (2.6)$$

where C_1 , C_2 , C_3 and C_4 are constants of integration to be found by the usage of boundary conditions. As an example, for the determination, the function $N_1(x)$ for the curve displayed in Fig. 2.2a, we use these boundary conditions:

$$\text{at } x = 0 \quad u(0) = 1 \text{ and } \frac{du(0)}{dx} = 0 \quad (2.7)$$

$$\text{at } x = L \quad u(L) = 0 \text{ and } \frac{du(L)}{dx} = 0 \quad (2.8)$$

By using the above conditions in Eqs. (2.5) and (2.6), gives a system of four algebraic equations which can be used to find the constants C_1 , C_2 , C_3 and C_4 .

The further substitutions of above-mentioned constants into Eq. (2.6) give us the equation for the deflected curve for the beam element in Fig. (2.2a) as

$$N_1(x) = 1 - 3\left(\frac{x}{L}\right)^2 + 2\left(\frac{x}{L}\right)^3 \quad (2.9a)$$

where $N_1(x)$ is used in the place of $u(x)$ to correspond to condition $\delta_1 = 1$ applied on

the beam element. Going on in similar way, we get the following equations for the equations of the deflected curves:

$$N_2(x) = x\left(1 - \frac{x}{L}\right)^2 \quad (2.9b)$$

$$N_3(x) = 3\left(\frac{x}{L}\right)^2 + 2\left(\frac{x}{L}\right)^3 \quad (2.9c)$$

$$N_4(x) = \frac{x^2}{L} + \left(\frac{x}{L} - 1\right) \quad (2.9d)$$

As we know that $N_i(x)$ is the deflection equivalent to a unit displacement $\delta_i = 1$, the displacement obtained from a random displacement δ_i , is $N_i(x) \delta_i$. “In the same way, the deflection obtained from nodal displacements δ_2 , δ_3 and δ_4 are $N_2(x) \delta_2$, $N_3(x) \delta_3$ and $N_4(x) \delta_4$. Hence, the combined deflection $u(x)$ at coordinate x because of random (arbitrary) displacements at nodal coordinates of beam element is produced by principle of superposition as

$$u(x) = N_1(x) \delta_1 + N_2(x) \delta_2 + N_3(x) \delta_3 + N_4(x) \delta_4 \quad (2.10)$$

The equations of shape, given by Eqs. (2.9a, b, c and d) and which are related to unit displacements at nodal coordinates of a beam element, can be utilized to find expressions of stiffness coefficients (Paz & Kim, 2019). For instance, take into account the beam in Fig. 2. For this beam in the balanced position, we say that a virtual displacement which is equal to the deflection curve displayed in Fig. 2.2a happens. After that we apply the principle of virtual work, which tells us that, for an elastic system, work performed by external forces is the same as the work of internal forces during the virtual displacement. To apply this principle, we notice that external work W_E is equal to the multiplication product of the force k_{12} displaced by $\delta_1 = 1$, which is

$$W_E = k_{12} \delta_1 \quad (2.11)$$

This work is the same as the work done by elastic forces during the virtual displacement” (Paz & Kim, 2019). Taking into account the work done by the bending-moment, we get for the internal work this equation

$$W_1 = \int_0^L M(x) d\theta \quad (2.12)$$

in which $d\theta$ is the incremental angular displacement of this section of the element and

$M(x)$ is the bending moment at section x of the beam.

For the virtual displacement, the transverse deflection of the beam is found by Eq. (2.9b), which is connected to the bending moment by the differential Eq. (2.1).

Substitution of the second derivative $N''_2(x)$ of Eq. (2.9b) into Eq. (2.1) gives us

$$EIN''_2(x) = M(x) \quad (2.13)$$

The angular deflections $d\theta$ formed during virtual displacement is connected to the transverse deflection of the beam $N_1(x)$ by

$$\frac{d\theta}{dx} = \frac{d^2N_1(x)}{dx^2} = N''_1(x)$$

or

$$d\theta = N''_1(x) dx \quad (2.14)$$

Making equal the external virtual work, W_E from Eq. (2.11) with the internal virtual work W_I from

Eq. (10.12) after using $M(x)$ and $d\theta$, respectively, from Eqs. (2.13) and (2.14) ultimately results in stiffness coefficient:

$$k_{12} = \int_0^L E IN''_1(x)N''_2(x)dx \quad (2.15)$$

Generally, any stiffness coefficient k_{ij} in relation with beam flexure, can be expressed as:

$$k_{ij} = \int_0^L E IN''_i(x)N''_j(x) dx \quad (2.16)$$

As observed from Eq. (2.16) that $k_{ij} = k_{ji}$, because the swapping of indices needs only an interchange of 2 factors, $N''_i(x)$ and $N''_j(x)$ in Eq. (2.16). This equivalence of $k_{ij} = k_{ji}$ is a special case of Betti's theorem; however, it is more appropriately known as *Maxwell's reciprocal theorem*.

It is important to notice that even though the “shape functions, Eqs. (2.9a, b, c and d), were found for a uniform beam, in practical applications they are also used in finding out the stiffness coefficients for non-uniform beams (Paz & Kim, 2019).

If we consider the case of a uniform beam element of length L and cross-sectional moment of inertia I , we may calculate any stiffness coefficient from Eqs. (2.16) and

the use of Eqs. (29a, b, c and d).” (Paz & Kim, 2019)

Specifically, the stiffness coefficient k_{12} is found as below:

From Eq.(2.9a), get

$$N''_1(x) = -\frac{6}{L^2} + \frac{12x}{L^3}$$

from Eq. (2.9b)

$$N''_2(x) = \frac{4}{L} + \frac{6x}{L^2}$$

Substitution in Eq. (10.15) gives us

$$k_{12} = EI \int_0^L \left(\frac{-6}{L^2} + \frac{12x}{L^3} \right) \left(\frac{-4}{L} + \frac{6x}{L^2} \right) dx$$

doing integration gives us

$$k_{12} = \frac{6EI}{L^2}$$

“Because the stiffness coefficient k_{ij} is described as the force at the nodal coordinate 1 because of unit displacement at the coordinate j , the forces at coordinate 1 due to displacements $\delta_1, \delta_2, \delta_2$ and δ_4 at the 4 nodal coordinates are given as: $k_{11} \delta_1, k_{12} \delta_2, k_{13} \delta_3$ and $k_{14} \delta_4$. So, the total force P_1 at coordinate is found by the superposition of the resulting forces:

$$P_1 = k_{11} \delta_1 + k_{12} \delta_2 + k_{13} \delta_3 + k_{14} \delta_4$$

In the same manner, the forces at the other nodal coordinates obtained from the nodal displacement $\delta_1, \delta_2, \delta_3, \delta_4$ are: Eq. (2.17)

$$P_2 = k_{12} \delta_1 + k_{22} \delta_2 + k_{23} \delta_3 + k_{24} \delta_4$$

$$P_3 = k_{31} \delta_1 + k_{32} \delta_2 + k_{33} \delta_3 + k_{34} \delta_4$$

$$P_4 = k_{41} \delta_1 + k_{42} \delta_2 + k_{43} \delta_3 + k_{44} \delta_4$$

The above equations are better written in matrix notation as below:

$$\begin{Bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{Bmatrix} = \begin{Bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{Bmatrix} \begin{Bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{Bmatrix} \quad (2.18)$$

or summarized:

$$\{P\} = [k]\{\delta\} \quad (2.19)$$

The use of Eq. (2.16) in the manner displayed above to determine the coefficient k_{12} gives us the way to calculate all the coefficients” (Paz & Kim, 2019). For a uniform beam element:

$$\begin{Bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{Bmatrix} = \frac{E_1}{L^3} \begin{pmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{pmatrix} \begin{Bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{Bmatrix} \quad (2.20)$$

2.1.1 System Stiffness Matrix

So far, we have a relation between nodal forces (forces and moments) and nodal displacements (linear and angular). Now, we have to find a similar type of connection for the nodal forces and the nodal displacements; but for the whole structure, which is the ‘system stiffness equation’.

2.2 Geometric Stiffness

When there is an addition of an axial force on top of flexural force, the stiffness coefficients are altered by the existence of the axial force. The alteration is known as “the geometric stiffness coefficient k_{Gij} , defined as the force corresponding to the nodal coordinate I due to a unit displacement at coordinate j and resulting from the axial forces in the structure (Paz & Kim, 2019). Coefficients can be calculated by the principle of virtual work. If we consider a beam element subjected to a spread axial force per unit of length $N(x)$, as portrayed in Fig. 2.9a. In the drawing in Fig. 2.9b, the beam section is subjected to a unit rotation of the left end, $\delta_2 = 1$. The nodal forces due to this displacement are the corresponding geometric stiffness coefficients.” (Paz & Kim, 2019) Now if we suppose for this deformed beam a unit displacement $\delta_I = 1$, the external work is

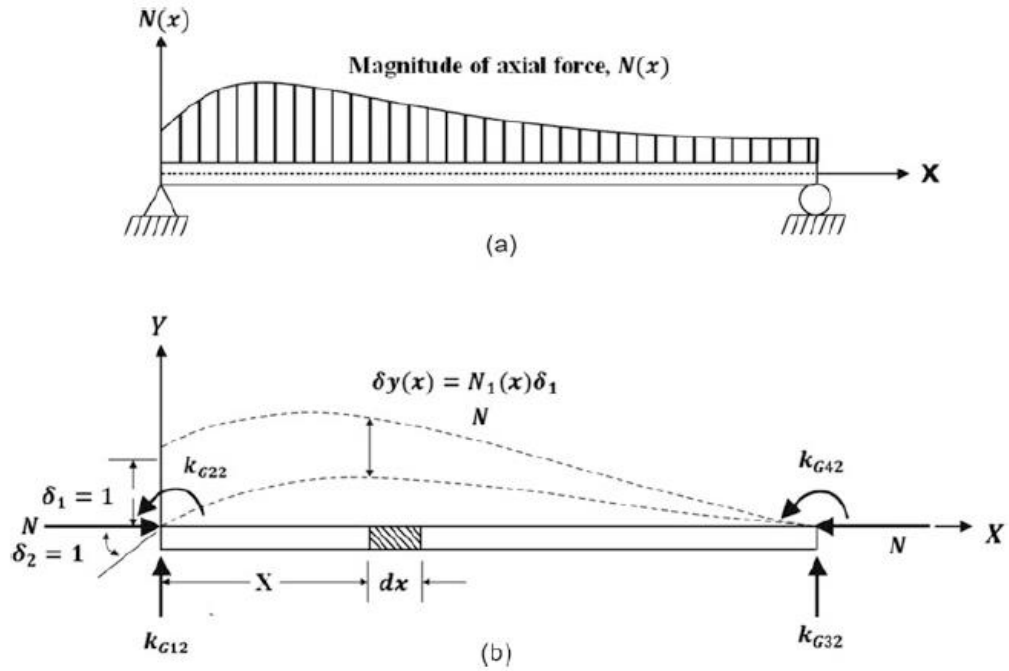


Figure 2.3 (a) “Beam element loaded with arbitrary distributed axial force, (b) Beam element acted on by nodal forces resulting for, displacement $\delta_2 = 1$ undergoing a virtual displacement $\delta_1 = 1$ ”

$$W_e = k_{G12}\delta_1$$

or

$$W_e = k_{G12} \quad (2.21)$$

because $\delta_1 = 1$.

The internal work is found by taking into account a differential element of length dx which we take from the beam in Fig. 2.3b and displayed in a larger size in Fig. 2.4.

Work done by the axial force $N(x)$ during the virtual displacement:

$$dW_j = N(x)\delta_e \quad (2.22)$$

where δ_e denotes the relative displacement taking place by the normal force $N(x)$ acting on the differential element during the virtual displacement. From Fig. 2.4, by similar triangles (triangles 1 and 11), we get

$$\frac{\delta_c}{dN_1(x)} = \frac{dN_2(x)}{dx}$$

or

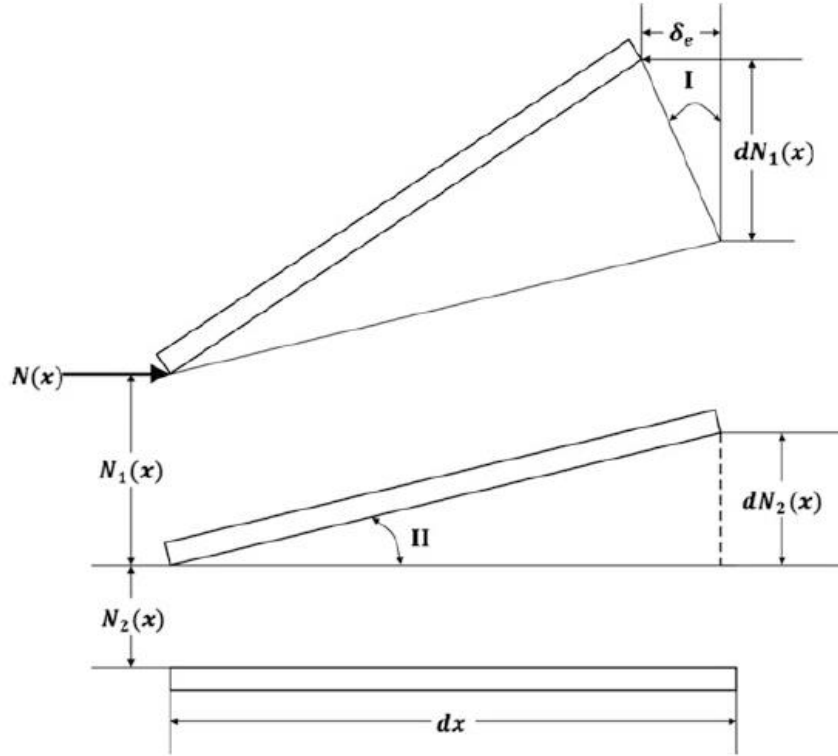


Figure 2.4 Differential segment of deflected beam in Fig. 2.3

$$\delta_e = \frac{dN_1(x)}{dx} \cdot \frac{dN_2(x)}{dx} dx$$

$$\delta_e = N'_1(x)N'_2(x)dx$$

in which $N'_1(x)$ and $N'_2(x)$ are the derivatives.

Substituting δ_e in Eq. (2.22), gives us

$$dW_I = N(x)N'_1(x)N'_2(x)dx \quad (2.23)$$

After that by integrating the expression and equalizing the result to the external work, Eq. (2.21), eventually gets us

$$k_{G12} = \int_0^L N(x)N'_1(x)N'_2(x)dx \quad (2.24)$$

Generally, any geometric stiffness expression:

$$k_{Gij} = \int_0^L N(x)N'_i(x)N'_j(x)dx \quad (2.25)$$

Normal force $N(x)$ is assumed to be independent of time. When the displacement equations, Eqs. (2.9a, b, c and d), are referenced in Eq. (2.25) to find out the

geometric stiffness coefficients, the result is known as the consistent geometric stiffness matrix. When the axial force is uniform along the length, use of Eqs. (2.25) and (2.9a, b, c and d) provides us the geometric stiffness matrix equation:

$$\begin{Bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{Bmatrix} = \frac{N}{30L} \begin{pmatrix} 36 & 3L & -36 & 3L \\ 3L & 4L^2 & -3L & -L^2 \\ -36 & -3L & 36 & -3L \\ 3L & -L^2 & -3L & 4L^2 \end{pmatrix} \begin{Bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{Bmatrix} \quad (2.26)$$

The combined stiffness matrix $[K_c]$ for the structure is given by

$$[K_c] = [K] - [K_G] \quad (2.27)$$

here $[K]$ is the combined elastic stiffness matrix for the structure and $[K_G]$ the geometric stiffness matrix.

2.3 Element Stiffness Matrix for Axial Effects

The presence of axial forces in the stiffness matrix of a flexural beam element needs the stiffness coefficients for axial loads. For finding out the stiffness matrix, look at Fig. 2.5. For a uniform and prismatic beam segment of cross sectional A and length L , it is easy to get the stiffness relation for axial effects by using the Hooke's law (Paz & Kim, 2019). The displacements δ_1 made by P_1 at node 1 while node 2 is fixed ($\delta_2 = 0$) is:

$$\delta_1 = \frac{P_1 L}{AE} \quad (2.28)$$

From Eq. (2.28) and k_{11} , we get

$$k_{11} = \frac{P_1}{\delta_1} = \frac{AE}{L} \quad (2.29a)$$

The balance of the beam segment by the force k_{11} needs a force k_{21} :

$$k_{21} = -k_{11} = -\frac{AE}{L} \quad (2.29b)$$

The other stiffness because of displacement at node 2 ($\delta_2 = 1$):

$$k_{22} = \frac{AE}{L} \quad (2.29c)$$

More:

$$k_{12} = \frac{AE}{L} \quad (2.29d)$$

Contents of Eq. (2.29a) are a part of the stiffness matrix relating displacement and

axial forces for a uniform beam segment:

$$\begin{Bmatrix} P_1 \\ P_2 \end{Bmatrix} = \frac{AE}{L} \begin{Bmatrix} 1 & -1 \\ -1 & 1 \end{Bmatrix} \begin{Bmatrix} \delta_1 \\ \delta_2 \end{Bmatrix} \quad (2.30)$$

The stiffness matrix shown in Fig. 2.6 is found by the combination in one matrix the stiffness matrix for flexural effects and the stiffness matrix for axial effects, Eq. (2.30), Eq. (2.20) (Paz & Kim, 2019). The matrix obtained from this relates the displacements δ_1 and the forces P_i at the coordinates shown in Fig. 2.6:

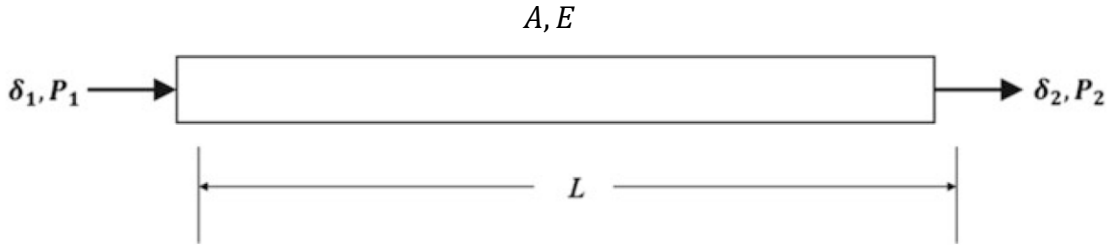


Figure 2.5 Beam element showing nodal axial loads P_1, P_2 , and corresponding nodal displacements δ_1, δ_2

$$\begin{Bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{Bmatrix} = \frac{EI}{L^3} \begin{Bmatrix} \frac{AL^2}{I} & & & & & \\ & 12 & & & & \\ & 0 & 6L & & & \\ & 0 & 6L & 4L^2 & & \\ & -\frac{AL^2}{I} & 0 & 0 & \frac{AL^2}{I} & \\ & 0 & -12 & -6L & 0 & 12 \\ & 0 & 6L & 2L^2 & 0 & -6L & 4L^2 \end{Bmatrix} \begin{Bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \delta_5 \\ \delta_6 \end{Bmatrix} \quad (2.31)$$

or, in concise notation,

$$\{P\} = [K]\{\delta\} \quad (2.32)$$

2.4 Coordinate Transformation

The stiffness matrix for any element of a plane frame in Eq. (2.31) is defined by coordinate axes fixed on beam. “These axes are known as *local* or *element coordinate axes*; the coordinate axes for the complete structure are called *global* or *system coordinate axes*. Figure 2.6 displays a beam element containing nodal forces P_1, P_2, \dots, P_6 called the local coordinate axes x, y, z , and $\bar{P}_1, \bar{P}_2, \dots, \bar{P}_6$ referred to global coordinate set of axes X, Y, Z . The goal: transform the element matrices from local to

global. This transformation is needed because the matrices for all the elements relate to the identical coordinates. We start by stating the forces (P_1, P_2, P_3) as forces $(\bar{P}_1, \bar{P}_2, \bar{P}_3)$. Because these 2 sets of forces are equal, we get from Fig. 2.6 these relationships.” (Paz & Kim, 2019)

For node one: Eq. (2.33)

$$P_1 = \bar{P}_1 \cos\theta + \bar{P}_2 \sin\theta$$

$$P_2 = -\bar{P}_1 \sin\theta + \bar{P}_2 \cos\theta$$

$$P_3 = \bar{P}_3$$

The equations of Eq. (2.33) can be written as:

$$\begin{Bmatrix} P_1 \\ P_2 \\ P_3 \end{Bmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{Bmatrix} \bar{P}_1 \\ \bar{P}_2 \\ \bar{P}_3 \end{Bmatrix} \quad (2.34)$$

Similarly, we get for the forces on node two: Eq. (2.35)

$$P_4 = \bar{P}_4 \cos\theta + \bar{P}_5 \sin\theta$$

$$P_5 = -\bar{P}_4 \sin\theta + \bar{P}_5 \cos\theta$$

$$P_6 = \bar{P}_6$$

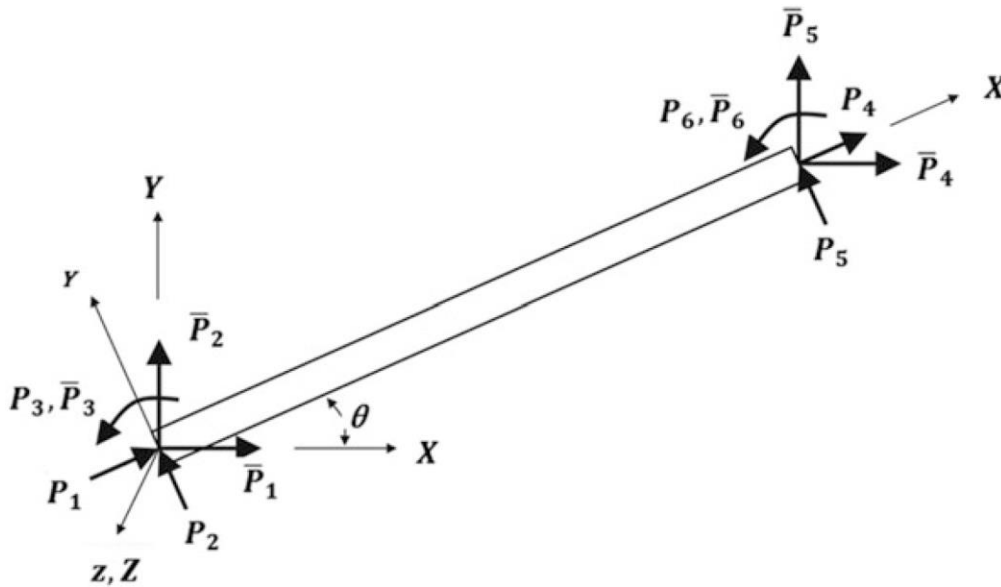


Figure 2.6 Beam element showing nodal forces P_i in local (x, y, z) and nodal forces \bar{P}_i in global coordinate axes (X, Y, Z)

Equations (2.33) and (2.35) in matrix form:

$$\begin{Bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 0 & 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \bar{P}_1 \\ \bar{P}_2 \\ \bar{P}_3 \\ \bar{P}_4 \\ \bar{P}_5 \\ \bar{P}_6 \end{Bmatrix} \quad (2.36)$$

Condensing:

$$\{P\} = [T]\{\bar{P}\} \quad (2.37)$$

Where $\{P\}$ and $\{\bar{P}\}$ are vectors of the element nodal forces in local coordinates and global coordinates and $[T]$ is the transformation. Going over the same process, we get a relation: nodal displacements $(\delta_1, \delta_2, \dots, \delta_6)$ in local, and nodal displacements in global $(\bar{\delta}_1, \bar{\delta}_2, \dots, \bar{\delta}_6)$:

$$\begin{Bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \delta_5 \\ \delta_6 \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 0 & 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \bar{\delta}_1 \\ \bar{\delta}_2 \\ \bar{\delta}_3 \\ \bar{\delta}_4 \\ \bar{\delta}_5 \\ \bar{\delta}_6 \end{Bmatrix} \quad (2.38)$$

or

$$\{\delta\} = [T]\{\bar{\delta}\} \quad (2.39)$$

The substitution of $\{P\}$ from Eq. (2.37) and $\{\delta\}$ from Eq. (2.39) in stiffness equation $\{P\} = [K]\{\delta\}$ gives us:

$$[T]\{P\} = [K][T]\{\delta\}$$

or

$$\{\bar{P}\} = [T]^{-1}[K][T]\{\bar{\delta}\} \quad (2.40)$$

where $[T]^{-1}$ is the inverse of matrix $[T]$. However, the transformation matrix $[T]$ in Eq. (2.36) is an orthogonal matrix, $[T]^{-1} = [T]^T$. Thus

$$\{\bar{P}\} = [T]^T [K] [T] \{\bar{\delta}\} \quad (2.41)$$

Or more conveniently,

$$\{\bar{P}\} = [\bar{K}] \{\bar{\delta}\} \quad (2.42)$$

where

$$\{\bar{K}\} = [T]^T [K] [T] \quad (2.43)$$

is the stiffness matrix.

2.5 Local and Global Coordinate Systems

“For a grid frame element, the local orthogonal axes are maintained in a way that the x - y plane will coincide with the plane of the structural system and x defines the longitudinal centroidal axis of the member. The z axis defines the minor-principal axis of the x -section, the y axis defines the bigger axis of the x -section. The grid member may have either a constant cross section along its length or variable. In Fig. 2.7, the possible nodal displacements with respect to the local or to the global systems of coordinates are identified. It is observable that the linear displacements along the Z direction for the global system and along the z direction for local axes are similar because the two axes match. Generally, rotational components at the nodal coordinates differ for coordinate systems. Thus, a transformation of coordinates is needed to transform from the local to the global coordinates.” (Paz & Kim, 2019)

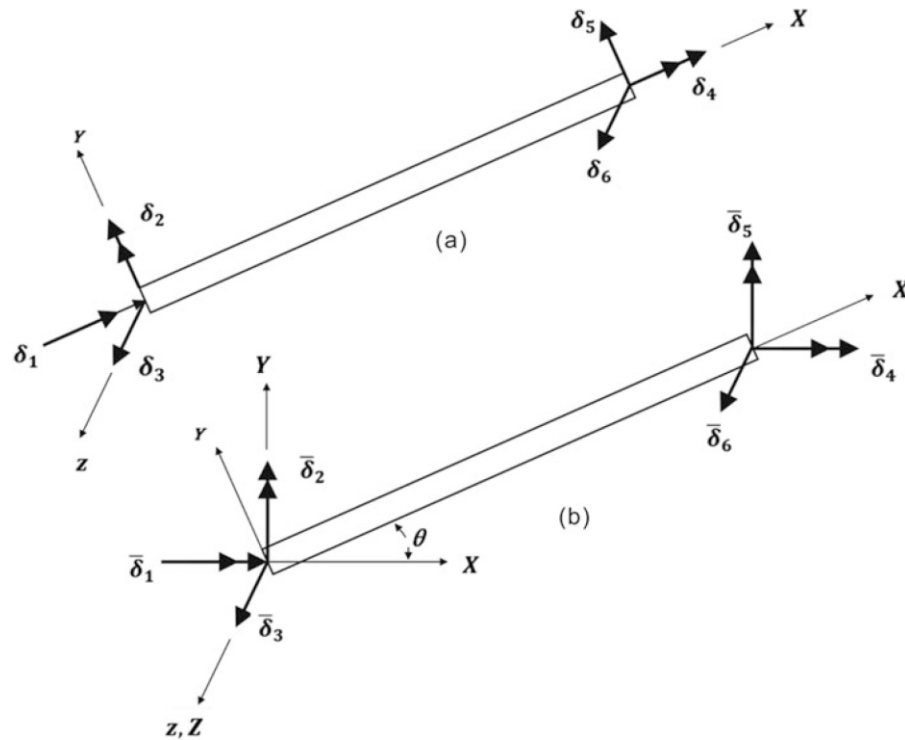


Figure 2.7 Components of nodal displacements for a grid member. (a) Local coordinate system. (b) Global coordinate system

2.6 Torsional Effects

The determination of the torsional stiffness and mass coefficients is needed. For the axial problem, the differential equation for the displacement function is given by

$$\frac{du}{dx} = \frac{P}{AE} \quad (2.44)$$

Similarly, the DE for torsional displacement:

$$\frac{d\theta}{dx} = \frac{T}{JG} \quad (2.45)$$

Comparing equations (2.44) and (2.45), we write these results obtained already for axial effects. The displacement functions:

$$\theta_1(x) = \left(1 - \frac{x}{L}\right) \quad (2.46)$$

$$\theta_2(x) = \frac{x}{L} \quad (2.47)$$

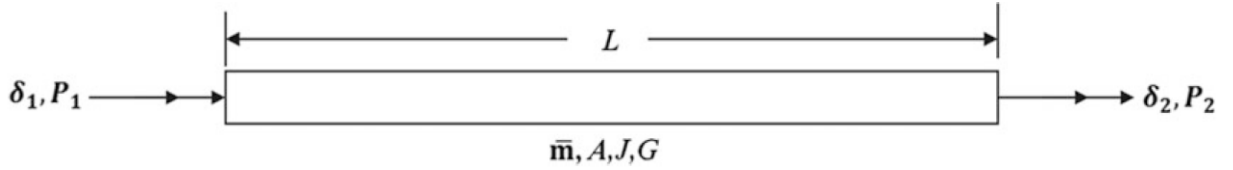


Figure 2.8 Nodal torsional coordinates for a beam element

The stiffness coefficients for torsional effects can be found by

$$k_{ij} = \int_0^L J G \theta'_i(x) \theta'_j(x) dx \quad (2.48)$$

The consistent mass matrix coefficients for torsional effects:

$$m_{ij} = \int_0^L I_{\bar{m}} \theta_i(x) \theta'_j(x) dx \quad (2.48.1)$$

This moment of inertia may be expressed as the product of the mass \bar{m} per unit length times the radius of gyration squared, k^2 . The mass polar moment of inertia per unit length I_m is:

$$I_{\bar{m}} = \bar{m} \frac{I_0}{A} \quad (2.48.2)$$

Eqs. (2.48) and (2.48.1) gives the stiffness and mass matrices for torsional effects:

$$\begin{Bmatrix} T_1 \\ T_2 \end{Bmatrix} = \frac{JG}{L} \begin{Bmatrix} 1 & -1 \\ -1 & 1 \end{Bmatrix} \begin{Bmatrix} \delta_1 \\ \delta_2 \end{Bmatrix} \quad (2.49)$$

and

$$\begin{Bmatrix} T_1 \\ T_2 \end{Bmatrix} = \frac{I_{\bar{m}} L}{6} \begin{Bmatrix} 2 & 1 \\ 1 & 2 \end{Bmatrix} \begin{Bmatrix} \dot{\delta}_1 \\ \dot{\delta}_2 \end{Bmatrix} \quad (2.50)$$

in which I_m is found by Eq. (2.48.2), and T_1, T_2 are torsional moments.

2.7 Stiffness Matrix: Grid Element

The flexural stiffness matrix, Eq. (2.20) is combined with the torsional stiffness matrix, Eq. (2.49), for finding the stiffness matrix for an element of a grid frame. Relating to the local coordinate system shown in Fig. 2.7a, the stiffness equation for a uniform element:

$$\begin{Bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{Bmatrix} = \frac{EI}{L^3} \begin{Bmatrix} JGL^2/EI & & & & & \\ 0 & 4L^2 & & & & \\ 0 & -6L & 12 & & & \\ -JGL^2/EI & 0 & 0 & JGL^2/EI & & \\ 0 & 2L^2 & -6L & 0 & 4L^2 & \\ 0 & 6L & -12 & 0 & 6L & 12 \end{Bmatrix} \begin{Bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \delta_5 \\ \delta_6 \end{Bmatrix} \quad (2.51)$$

Condensing:

$$\{P\} = [K]\{\delta\} \quad (2.52)$$

2.8 Element Stiffness Matrix

“The stiffness matrix for a 3-D beam segment is found by the superposition of the torsional stiffness matrix from Eq. (2.48.2), the axial stiffness matrix from Eq. (2.30), and the flexural stiffness matrix in Eq. (2.20). Combining these matrices in an appropriate manner, we get in Eq. (2.53) the stiffness equation:”

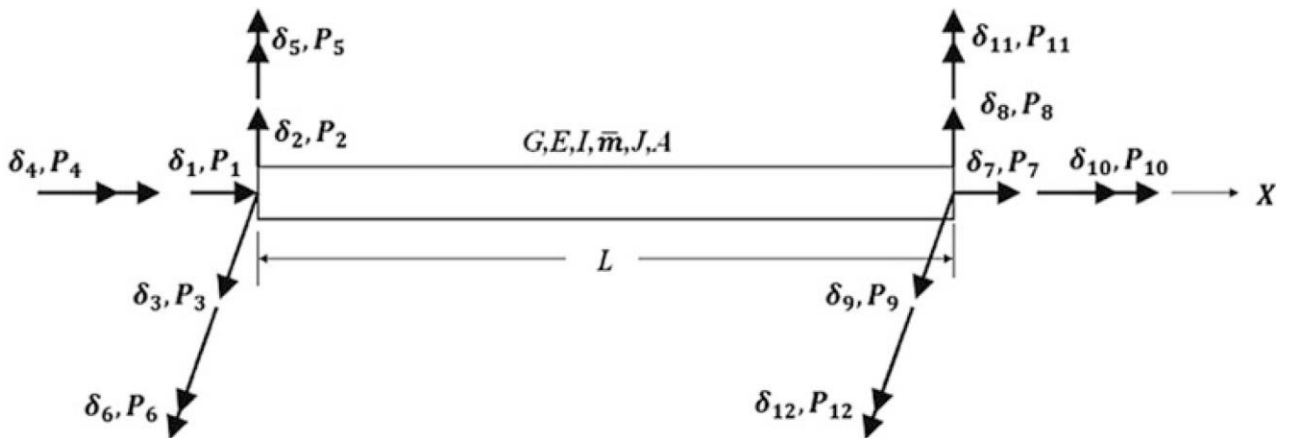


Figure 2.9 Beam segment of a space frame showing forces and displacements at the nodal coordinates

$$y = X \cos yX + Y \cos yY + Z \cos yZ \quad (2.55b)$$

$$z = X \cos zX + Y \cos zY + Z \cos zZ \quad (2.55c)$$

Equations (13.6a, 13.6b, and 13.6c) are conveniently written in matrix notation as

$$\begin{Bmatrix} \lambda \\ y \\ Z \end{Bmatrix} = \begin{Bmatrix} \cos xX & \cos xY & \cos xZ \\ \cos yX & \cos yY & \cos yZ \\ \cos zX & \cos zY & \cos zZ \end{Bmatrix} \begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} \quad (2.56)$$

or in short notation

$$\{A\} = \{T_1\}\{\bar{A}\} \quad (2.57)$$

in which $\{A\}$ and $\{\bar{A}\}$ are the components in the local systems and global systems of the general vector A and $[T_1]$ the transformation matrix found by:

$$[T_1] = \begin{Bmatrix} \cos xX & \cos xY & \cos xZ \\ \cos yX & \cos yY & \cos yZ \\ \cos zX & \cos zY & \cos zZ \end{Bmatrix} \quad (2.58)$$

The cosines needed in the transformation matrix $[T_1]$ are normally found in computer codes from the global coordinates of 3 points. The 2 points defining the 2 ends of the beam element along the local x axis and the 3rd point located in xy local plane where y is the principal axes of the x-sectional area. The input data including the global coordinates of these 3 points are enough for the evaluation of all cosine terms in Eq. (2.58).” (Paz & Kim, 2019) To show

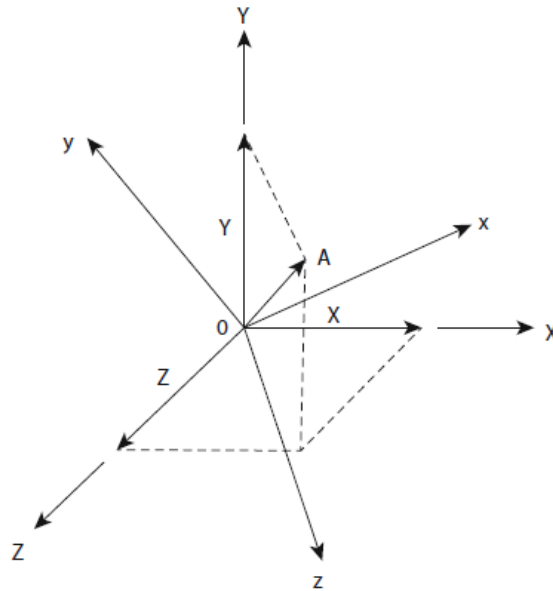


Figure 2.10 Components of a general vector A in local and global coordinates

this: “designate x_j, y_j, z_j and x_i, y_i, z_i the coordinates of point I and J at the two ends of a beam element and by x_p, y_p, z_p , the coordinates of a point P placed on the local xy plane. The direction cosines of local axis x along the beam element are found by

$$\cos xX = \frac{x_j - x_i}{L}, \quad \cos xY = \frac{y_j - y_i}{L}, \quad \cos xZ = \frac{z_j - z_i}{L} \quad (2.59)$$

where L is the length of the beam element given by

$$L = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2} \quad (2.60)$$

The direction cosines of the z axis can be calculated from the condition that any vector Z along the z axis must be perpendicular to the plane formed by any two vectors in the local x - y plane. These two vectors could simply be the vector X from point I to point J along the x axis and the vector P from point I to point P . The orthogonality condition is then expressed by the cross product between vectors X and P as

$$\mathbf{Z} = \mathbf{X} \times \mathbf{P} \quad (2.61)$$

or substituting the components of these vectors as

$$z_x \hat{i} + z_y \hat{j} + z_z \hat{k} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_j - x_i & y_j - y_i & z_j - z_i \\ x_p - x_i & y_p - y_i & z_p - z_i \end{vmatrix} \quad (2.62)$$

where \hat{i}, \hat{j} , and \hat{k} are unit vectors along the global coordinate axes X, Y , and Z , respectively.

Then, the direction cosines of axis z are given by

$$\cos zX = \frac{z_x}{|Z|}, \quad \cos zY = \frac{z_y}{|Z|}, \quad \cos zZ = \frac{z_z}{|Z|} \quad (2.65)$$

in which: Eq. 2.66

$$z_x = (y_j - y_i)(z_p - z_i) - (z_j - z_i)(y_p - y_i)$$

$$z_y = (z_j - z_i)(x_p - x_i) - (x_j - x_i)(z_p - z_i)$$

$$z_z = (x_j - x_i)(y_p - y_i) - (y_j - y_i)(x_p - x_i)$$

and

$$|Z| = \sqrt{z_x^2 + z_y^2 + z_z^2} \quad (2.67)$$

The direction cosines of the local axis y are calculated from the condition of orthogonality between a vector \mathbf{Y} along the y axis and the unit vectors \mathbf{X}_1 and \mathbf{Z}_1 along the x and z axes, respectively. Hence,

$$\mathbf{Y} = \mathbf{X}_1 \times \mathbf{Z}_1$$

or in expanded notation

$$y_x \hat{i} + y_y \hat{j} + y_z \hat{k} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ \cos xX & \cos xY & \cos xZ \\ \cos zX & \cos zY & \cos zZ \end{vmatrix} \quad (2.68)$$

Therefore,

$$\cos yX = \frac{y_x}{|Y|}, \quad \cos yY = \frac{y_y}{|Y|}, \quad \cos yZ = \frac{y_z}{|Y|}$$

where: Eq. 2.69

$$y_x = \cos xY \cos zZ - \cos xZ \cos zY$$

$$y_y = \cos zX \cos zX - \cos xX \cos zZ$$

$$y_z = \cos xX \cos zY - \cos xY \cos zX$$

and

$$|Y| = \sqrt{y_x^2 + y_y^2 + y_z^2}$$

It is shown that knowing the coordinates of points at the 2 ends of an element of a point P on the local plane $x - y$ are enough to find the direction cosines.

In another way, the direction cosines can be found by the nodal coordinates $(x_i, y_i, z_i,)$ and (x_j, y_j, z_j) at the two ends of the beam element and if we know the angle of rolling.

For a 3-dimensional figure, the transformation is required at every joint of the segment. Then, a beam element of a space frame needs, for 2, the transformation of 4 displacement vectors. Transformation of the 12 nodal displacements $\{\bar{\delta}\}$ global coordinates to the displacement $\{\delta\}$ in local coordinates can be abbreviated as:

$$\{\delta\} = [T]\{\bar{\delta}\} \quad (2.70)$$

where

$$[T] = \begin{bmatrix} [T_1] & & & \\ & [T_1] & & \\ & & [T_1] & \\ & & & [T_1] \end{bmatrix}$$

Similarly, the transformation from nodal forces $\{\bar{P}\}$ in global coordinates to nodal forces $\{P\}$ in local coordinates is:

$$\{P\} = [T]\{\bar{P}\} \quad (2.71)$$

Eventually, to get the stiffness matrix $[\bar{K}]$ in reference to the global system of coordinates, substitute, into Eq. (2.54), $\{\delta\}$ from Eq. (2.70) and $[P]$ from Eq. (2.72) to obtain

$$[T]\{\bar{P}\} = [K][T]\{\bar{\delta}\}$$

or

$$\{\bar{P}\} = [T]^T [K] [T] \{\bar{\delta}\} \quad (2.72)$$

since $[T]$ is an orthogonal matrix. From Eq. (13.21), we may write

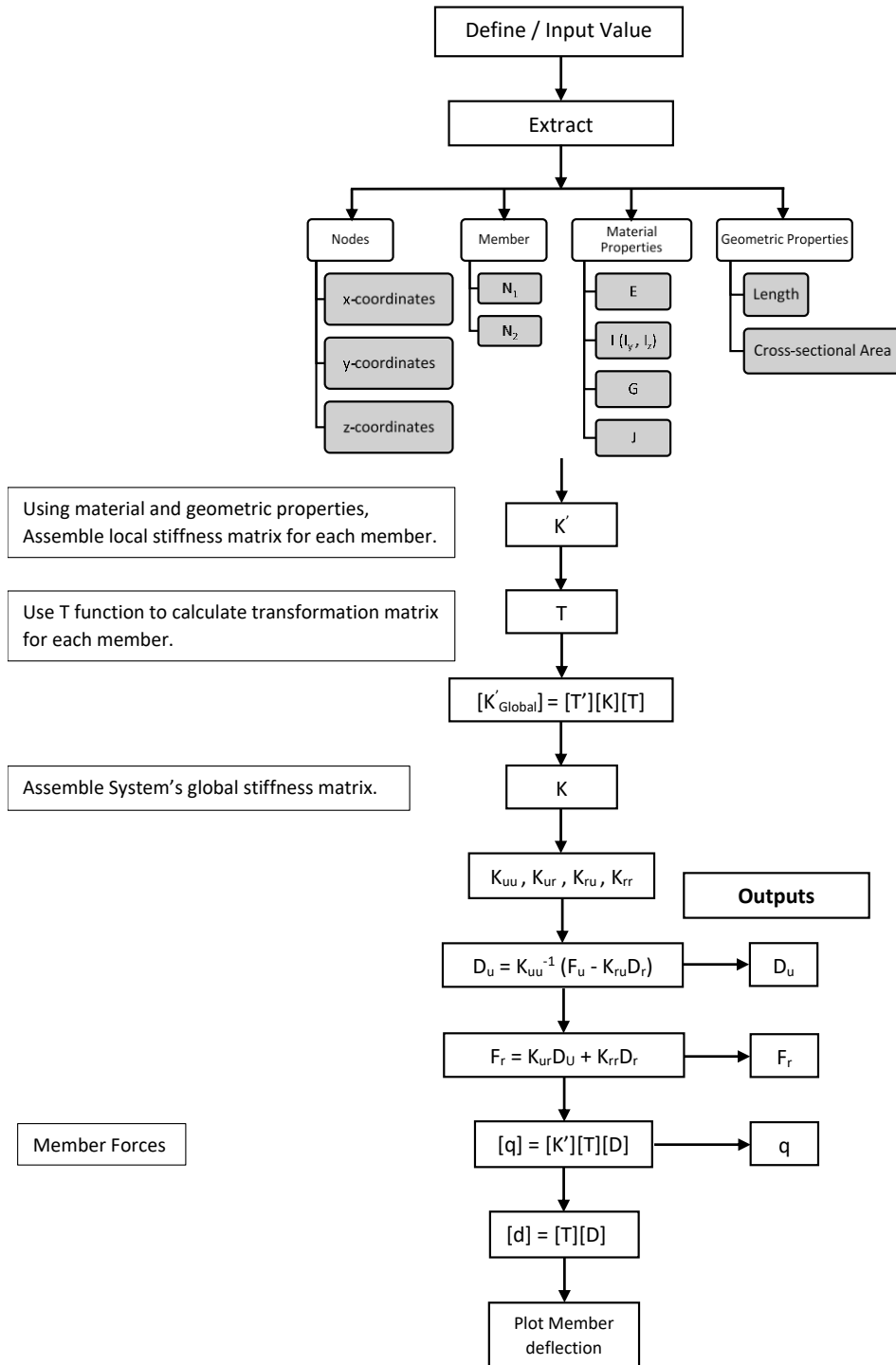
$$\{\bar{P}\} = [\bar{K}]\{\bar{\delta}\} \quad (2.73)$$

in which $[\bar{K}]$ is defined as

$$[\bar{K}] = [T]^T [K] [T] \quad (2.74)''$$

(Paz & Kim, 2019)

3.0 METHODOLOGY



In the first step of the program is to get input from the user. The input can be provided in one of the two following ways:

- From an excel file.
- Through Graphical User Interface (GUI).

The program stores the provide input in arrays named according to their types i.e.

- Nodal data is extracted in array “nodal_data”.
- Nodal data is further categorized into different arrays like:
 - Number of nodes are stored in array “Nnodes”.
 - x, y and z-coordinates are stored into arrays “X1”, “X2”, “Y1”, “Y2”, “Z1”, and “Z2”, respectively. Where 1 and 2 signifies starting and ending coordinates of the members.
- Member data is stored in array “member_data”.
 - Length of members is stored in array “l”.
 - Area of the member is stored in array “Area”.
 - Moment of inertia for y and z are stored in arrays “iy” and “iz” respectively.
 - Modulus of Elasticity is stored in array “E”.
 - Shear Modulus is stored in array “g”.
 - Torsional constant is stored in array “j”.
 - Mass per unit length is stored in array “m”.

```

% Connectivity

n1 = member_data(:,2); % 1st Node of member n
n2 = member_data(:,3); % 2nd Node of member n

X1 = nodal_data(n1,2);
Y1 = nodal_data(n1,3);
Z1 = nodal_data(n1,4);
X2 = nodal_data(n2,2);
Y2 = nodal_data(n2,3);
Z2 = nodal_data(n2,4);

% Material and Geometric properties

l = sqrt((X2-X1).^2 + (Y2-Y1).^2 + (Z2-Z1).^2);
Area = member_data(:,4);
iz = member_data(:,5);
iy = member_data(:,6);
E = member_data(:,7);
g = member_data(:,8);
j = member_data(:,9);
k = member_data(:,10); %Shear effect on beams
m = member_data(:,23);

```

Figure 3.1: MATLAB Code for getting data from input file

3.1 Initialize Matrices and Degree of Freedom:

Depending on the number of nodes and number of matrices, following matrices are defined in this section and are modified later in the program:

- dof: each member will have 12 degrees of freedom. Therefore, we define zero matrix of order 12×1 for each member.
- T: each member will have its own transformation matrix of 12×12 depending upon its orientation in space.
- u_global: this would be a column matrix of order $6n$, where n is the number of nodes.
- q_local: each member will have 6 reactions for each end. Hence, we have to define a zero matrix of order 12×12 for each member.
- Q_global: zero matrix of order $6n \times 1$, where n is number of nodes.

3.2 Transformation Matrix:

The Program calculates the transformation matrix for each member by calling

function “T_matrix”. The process is iterated for each member by using **for loop**.

```

% Decide Degrees of Freedom + Initialize Matrices

dof = zeros(12,1,Nmembers); % 12 Degrees of Freedom for every member
T = zeros(12,12,Nmembers); % Transformation Matrix currently set to zeros
u_global = zeros(6*Nnodes,1);
q_local = zeros(12,1,Nmembers);
Q_global = zeros(6*Nnodes,1);

for k = 1:Nmembers
%Note: Degrees of Freedom corresponding to node i are
    %[6i-5    6i-4    6i-3    6i-2    6i-1    6i]

    dof(:, :, k) = [6*n1(k)-5; 6*n1(k)-4; 6*n1(k)-3; 6*n1(k)-2; 6*n1(k)-1; 6*n1(k);
                    6*n2(k)-5; 6*n2(k)-4; 6*n2(k)-3; 6*n2(k)-2; 6*n2(k)-1; 6*n2(k)];
end

for k = 1:Nmembers
    T(:, :, k) = T_matrix(X1(k), Y1(k), Z1(k), X2(k), Y2(k), Z2(k));
end

all_dof = (1:6*Nnodes)';
restraints = nodal_data(:,17:22)'; % Restrained Supports, column 17 to 22 of Excel File
restrained_dof = find(restraints==1);
unrestrained_dof = setdiff(all_dof, restrained_dof);

```

Figure 3.2: MATLAB Code for making Transformation matrix

3.3 Stiffness Matrix:

Stiffness matrix is assembled using the extracted material and geometric properties as explained above. The problem calls the external function “klocal_combined” and provides with the necessary arguments to calculate the stiffness matrix. Analogously, the process is iterated for each member by using **for loop**. The local combined stiffness matrix is then transformed into global stiffness matrix by using the following relation:

$$[k'_{global}] = [T^T] \times [k'_{local}] \times [T]$$

The global stiffness matrices are then combined to form a system global matrix KC .

```

% Local Combined Stiffness Matrix

KLocal = zeros(12,12,Nmembers);

for k=1:Nmembers
    L = l(k);
    e = E(k);
    Iz = iz(k);
    Iy = iy(k);
    A = Area(k);
    J = j(k);
    G = g(k);

    KLocal(:, :, k) = kLocal_combined(e,A,L,Iz,Iy,G,J); % Calling function of kLocal_combined
end

% Global Combined Stiffness Matrices

% Member Global
KG_Member = zeros(12,12,Nmembers);

for k=1:Nmembers
    KG_Member(:, :, k) = (T(:, :, k))'*KLocal(:, :, k)*T(:, :, k); % Calling Transformation Matrix Function
end

% System Global
KGlobal = zeros(6*Nnodes);

```

Figure 3.3: MATLAB Code for making Local Stiffness matrix

```

for k=1:Nmembers
    KGlobal(dof(:, :, k),dof(:, :, k)) = KGlobal(dof(:, :, k),dof(:, :, k)) + KG_Member(:, :, k);
end

% Combined Global Stiffness Matrix
KC = KGlobal;

```

Figure 3.4: MATLAB Code for making Global Stiffness matrix

3.4 Compartmentalizing the KC matrix:

The system global matrix is compartmentalized into:

- K_{uu} (unrestrained, unrestrained).
- K_{ru} (restrained, unrestrained).
- K_{ur} (unrestrained, restrained).
- K_{rr} (restrained, restrained).

```

% Compartmentalizing the K Matrix and the Force Vector
Kuu = KC(unrestrained_dof, unrestrained_dof);
Kru = KC(restrained_dof, unrestrained_dof);
Kur = KC(unrestrained_dof, restrained_dof);
Krr = KC(restrained_dof, restrained_dof);

% Considering Support Settlement & Nodal load
for k = 1:Nnodes
    support_settlement(6*k-5:6*k,1) = nodal_data(k,5:10)';
    F_nodal(6*k-5:6*k,1) = nodal_data(k,11:16)';
    restraints(6*k-5:6*k,1) = nodal_data(k,17:22)';
end

P = F_nodal;

F_active = P(unrestrained_dof);
u_restrained = support_settlement(restrained_dof);

```

Figure 3.5: MATLAB Code for finding restrained and unrestrained nodal co-ordinates

3.5 Finding Displacements:

Global displacements are found out by following relation:

$$[D_u] = [K_{uu}^{-1}] \times ([F_u] - [K_{ru}][D_r])$$

```

% Finding Displacement

u_global = zeros(6*Nnodes,1);
u_unrestrained = Kuu\F_active - Kur*u_restrained;
u_global(restrained_dof,1) = u_restrained;
u_global(unrestrained_dof,1) = u_unrestrained;
u_global
u_local = zeros(12,1,Nmembers);

for k = 1:Nmembers
    u_local(:, :, k) = u_local(:, :, k) + T(:, :, k)*u_global(dof(:, :, k),1);
end

```

Figure 3.6: MATLAB Code for finding deflections

3.6 Finding Support Reactions:

The support reactions will be found out by following relation:

$$[F_r] = [K_{ur}][D_u] + [K_{rr}][D_r]$$


```
% Support Reactions
Q_global = Kru*u_unrestrained
```

Figure 3.7 MATLAB Code for support reactions

3.7 Finding Member Forces:

The internal reactions for the member are found out by following relation:

$$[q] = [K'] [T] [D]$$

```
% Member Forces
for j = 1:Nmembers
    q_local(:, :, j) = (KLocal(:, :, j)*T(:, :, j)*u_global(dof(:, :, j)));
end
```

Figure 3.8: MATLAB Code for finding member forces

3.8 Plotting the frame:

- Plotting original frame:

```

%% Plotting the Frame

%-----
% Plotting the Original Frame
original_coordinates = [X1 Y1 Z1 X2 Y2 Z2];
original_coordinates_X = [X1 X2];
original_coordinates_Y = [Y1 Y2];
original_coordinates_Z = [Z1 Z2];

for k = 1:Nmembers

figure(1)
plot3(original_coordinates_X(k,:),original_coordinates_Z(k,:), original_coordinates_Y(k,:), 'k')
title ('Displacement')
xlabel ('X coordinate (in)')
ylabel ('Z coordinate (in)')
zlabel ('Y coordinate (in)')
hold on; axis equal

% camup([0 1 0])

end

```

Figure 3.9: MATLAB Code for plotting actual frame

- Plotting deformed frame:

```

%-----
% General Displacement Functions N1,N2,..., N12

% D = [ (1 - x/L);
%       (2*x^3)/L^3 - (3*x^2)/L^2 + 1;
%       (2*x^3)/L^3 - (3*x^2)/L^2 + 1;
%       (1 - x/L);
%       (x*(x/L - 1)^2);
%       (x*(x/L - 1)^2);
%       (x/L);
%       (3*x^2)/L^2 - (2*x^3)/L^3;
%       (3*x^2)/L^2 - (2*x^3)/L^3;
%       (x/L);
%       (x^2*(x/L - 1))/L;
%       (x^2*(x/L - 1))/L];

syms x

```

Figure 3.10: General deformation functions for 3D frame element in x-direction

```

% Displacement Functions for Y-Direction
Dy = [0;
      (2*x^3)/L^3 - (3*x^2)/L^2 + 1;
      0;
      0;
      0;
      (x*(x/L - 1)^2);
      0;
      (3*x^2)/L^2 - (2*x^3)/L^3;
      0;
      0;
      0;
      (x^2*(x/L - 1))/L];

% Displacement Functions for X-Direction
Dx = [(1 - x/L);
      0;
      0;
      0;
      0;
      0;
      (x/L);
      0;
      0;
      0;
      0;
      0];

```

Figure 3.11: General deformation functions for 3D frame element in y-direction

```

% Displacement Functions for Z-Direction
Dz = [0;
      0;
      (2*x^3)/L^3 - (3*x^2)/L^2 + 1;
      0;
      (x*(x/L - 1)^2);
      0;
      0;
      0;
      (3*x^2)/L^2 - (2*x^3)/L^3;
      0;
      (x^2*(x/L - 1))/L;
      0];

% Displacement Functions for Torsion
Dt = [0;
      0;
      0;
      (1 - x/L);
      0;
      0;
      0;
      0;
      0;
      0;
      (x/L);
      0;
      0];

```

Figure 3.12: General deformation functions for 3D frame element in z-direction

```

for k = 1:Nmembers

L = l(k);
syms x

magnification = 50;

u_global_k = u_global(dof(:, :, k));
a = magnification*u_global_k;

Xo = Dx.*a;
ux = sum(Xo);    % ux1 + ux2 +ux3 + ..... +uxn
Yo = Dy.*a;
uy = sum(Yo);    % uy1 + uy2 +uy3 + ..... +uyn
Zo = Dz.*a;
uz = sum(Zo);    % uz1 + uz2 +uz3 + ..... +uzn

ndivs = 100;    % Number of Divisions for member 'k'

subs_ux = subs(ux,x,0:L/ndivs:L);
subs_ux = subs_ux(:);
subs_uy = subs(uy,x,0:L/ndivs:L);
subs_uy = subs_uy(:);
subs_uz = subs(uz,x,0:L/ndivs:L);
subs_uz = subs_uz(:);

```

Figure 3.13: Deformed co-ordinates of frame elements

```

%-----
% Plotting Member Deformations
for ii = 1:ndivs+1
format shorteng
xyz(ii,1) = X1(k) + (X2(k)-X1(k))*((ii-1)/ndivs);
xyz(ii,2) = Y1(k) + (Y2(k)-Y1(k))*((ii-1)/ndivs);
xyz(ii,3) = Z1(k) + (Z2(k)-Z1(k))*((ii-1)/ndivs);

% xyz = xyz^T(:, :, k)
end

x = xyz(:,1) + subs_ux;
y = xyz(:,2) + subs_uy;
z = xyz(:,3) + subs_uz;

plot3(x,z,y, 'k--')
axis equal ; hold on

% Plotting New Nodal Coordinantes with use of Marker
deformed_coordinates_X = [x(1), x(end)];
deformed_coordinates_Y = [y(1), y(end)];
deformed_coordinates_Z = [z(1), z(end)];
plot3(deformed_coordinates_X, deformed_coordinates_Z, deformed_coordinates_Y, 'o'...
, 'MarkerFaceColor', 'r', 'MarkerEdgeColor', 'k')
end

```

Figure 3.14: MATLAB Code for plotting defelected shape of frame

3.9 Guide To Make Application Using App Designer

3.9.1 Creating a Main GUI

Now we must make a Main GUI which can run 3D Linear Analysis of steel structure, its detailed steps are given below:

3.9.1.1 MATLAB code

Make the simple code on MATLAB which can run the 3D Analysis of steel structures which we will use later in Main GUI for Analysis, for making this code its detailed explanation is given in above paras.

3.9.1.2 APP designer and MATLAB code

Now we have to embed that code into app designer, for it we have to understand the use of MATLAB APP DESIGNER, it consists of different tools which can easily be used to make a user-friendly interface.

The figure below shows the component library ,component browser ,design view and

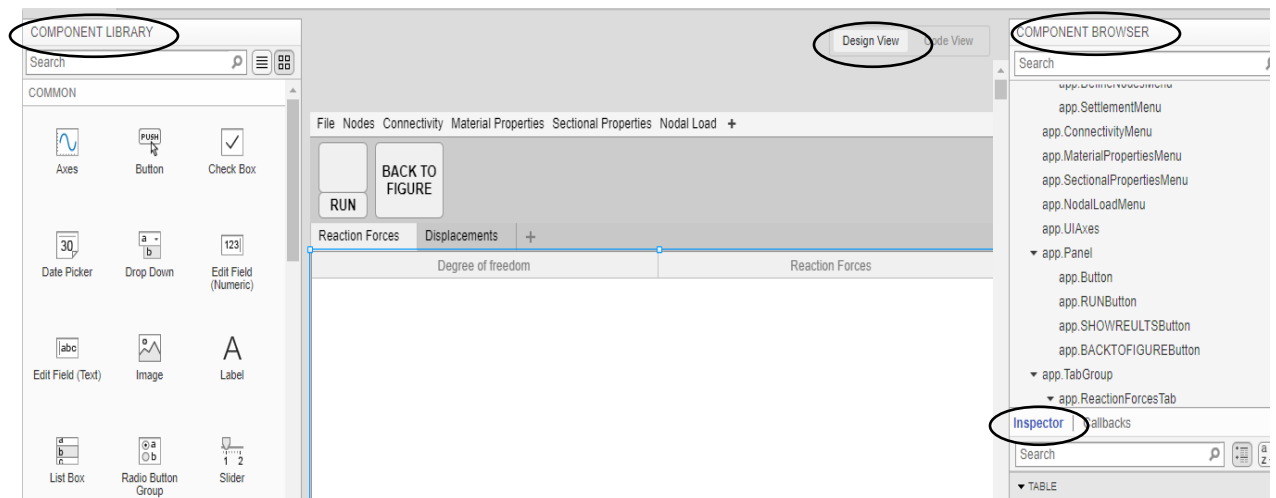


Figure 3.15: Interface of MATLAB APP designer

inspector.

From component library we can drag and drop any component in our GUI according

to requirement of our software and can easily edit using Inspector Menu, Component browser tells the all components being used to make a user friendly GUI and code view is used to define the call back behind every component of app designer.

Call back is the background working which should be done when we access that APP DESIGNER component, for it we make some lines of code which is executed when we press that button, we can write callback by right click on that APP DESIGNER component then go to callbacks option then click on callback it will move towards the code view and callback behind that APP DESIGNER component is made. As shown in figure:

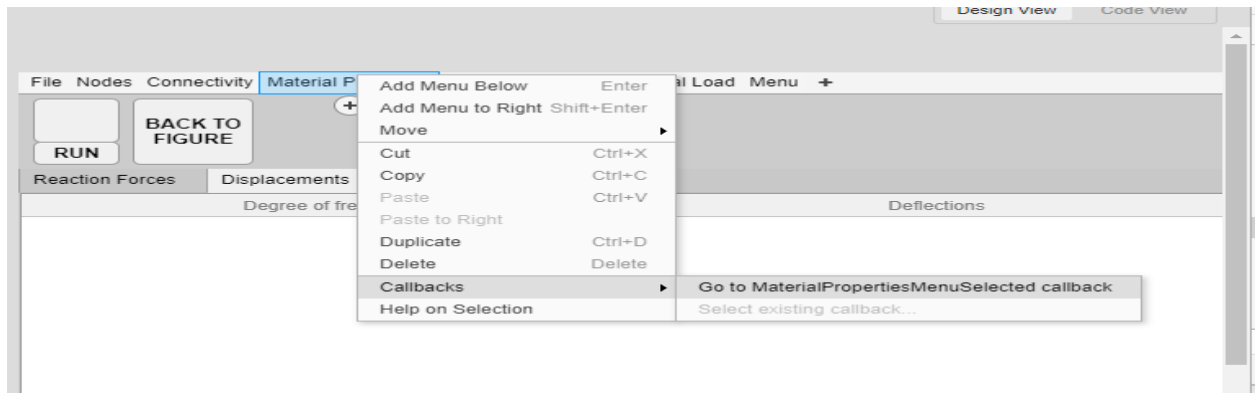


Figure 3.16: Call back option in MATLAB APP designer

3.9.1.3 Design view of main GUI

First, we make the main GUI which contain different buttons depends on the need of software, our software consists of multiple buttons and graph and tables which is shown below:

- a) Run
- b) Back to figure
- c) Show Results
- d) Define nodes
- e) Table
- f) Settlement
- g) Connectivity

- h) Material properties
- i) Sectional properties
- j) Nodal load
- k) Import data
- l) Directory
- m) Quit
- n) Plot

We can easily drag and drop these buttons and plot from component library into design view of APP designer and easily adjust their positions by using mouse and edit their appearance and properties by using inspector of each button and plot.

- a. First, we drop down the menu bar from component library and make different menus like file, directory, nodal load etc.
- b. We drop down the panel just because of the contrast of color that color contains three buttons
 - i. RUN button
 - ii. SHOW REULTS button
 - iii. Back to figure button
- c. We drop down the Axes from component Library so that we can draw the shape of the frame and its deflected shape.
- d. Then we drop down the Tab Group from component library and make two tabs, In first tab group we drop table which shows reaction forces and In 2nd Tab group we also drop the Table which the deflections of each node of frame after analysis.

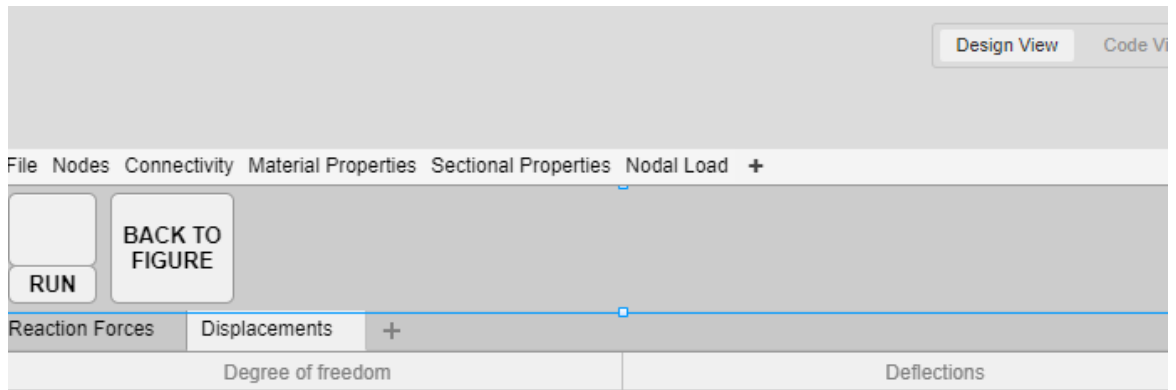


Figure 3.17: Components of our GUI

This is the final Layout of our Main GUI:

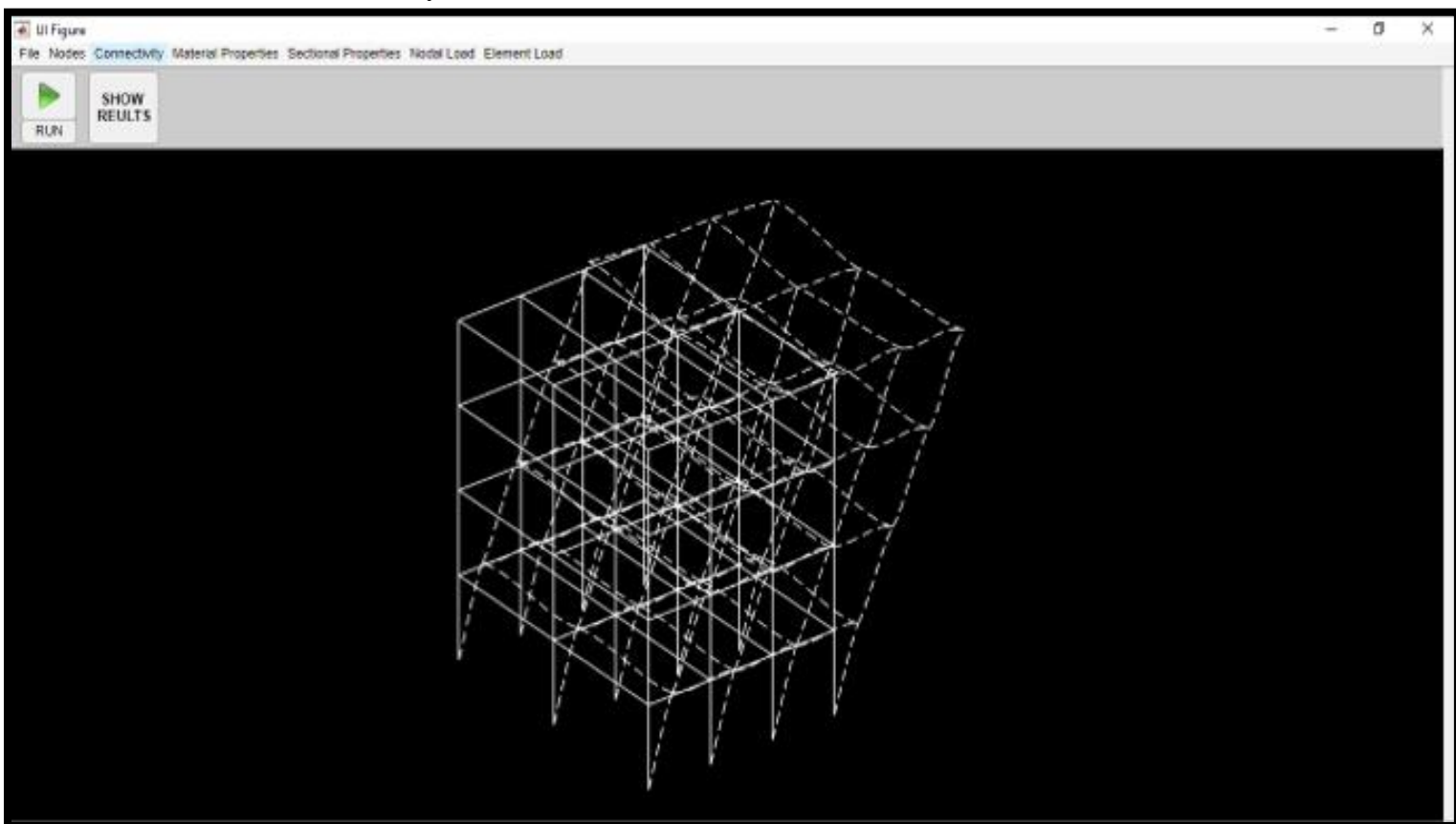


Figure 3.18: Layout of Main GUI

Now we made the design view of our software now we have to set callback behind every button, call back is the background working which should be done when we

access that button , for it we make some lines of code which is executed when we press that button, now we write callback function for every button in code view, these callbacks are explained in detail in coming steps.

3.9.1.4 Properties

Properties is the built up function in APP DESIGNER in it we can assign any variable and can be used as a global variable with `app.variable_name` syntax for calling variable actually variables assigned in the properties function is stored in the backend directory of APP DESIGNER. We also define some variables in properties according to need which is shown below:

```
properties (Access = private)
    input_data % Description
    check =0
    table_u
    table_q
    selpath
end
```

Figure 3.19: Property Function in MATLAB APP designer

3.9.1.5 Start Up Function

It is the built-in function in APP DESIGNER, when we start the program then code inside this function is evaluated on start of app, our startup function is shown below:

```

function startupFcn(app)
drawnow;
app.UIFigure.WindowState = 'maximized';
app.TabGroup.Visible = 'off';
app.BACKTOFIGUREButton.Visible = 'off';
app.UIAxes.Color = [0.00,0.00,0.00]
plot3(app.UIAxes,[0 0 0],[0 0 0],[0 0 0]);

end

```

Figure 3.20: Start up function in MATLAB APP designer

In this startup function 1st two lines are used to maximize the window in start of our app, as default position is not set as maximized so for this, we must use this command. 3rd and 4th line is used to invisible backtofigure and tab group options in design view for start of application because they are overlapped with another button and graph respectively.

5th line is used to change the color of graph from default(white) to black

6th line is used to show the plot in 3-D in start of the application.

3.9.1.6 Directory Function in File Menu:

This function is used to get the directory/Folder from user where user give data for analysis and can save data in that folder.

```

% Menu selected function: DirectoryMenu
function DirectoryMenuSelected(app, event)
f=figure();
drawnow;
app.selpath = uigetdir;
delete(f)

end
end

```

Figure 3.21: Directory Function

In this code 1st two lines are used because of when we use uigetdir function command

window of MATLAB appear and application window is minimized in APP DESIGNER so whenever we set directory we have to make the figure and delete it after getting path of directory, we save directory path in selpath variable as it is described in properties so we used “app.” To access the variable “uigetdir” is command which is used to select the path for directory.

3.9.1.7 Import Data Function in File Menu:

This function is used to get the data from the user from the selected directory in above function.

```
% Menu selected function: ImportDataMenu
function ImportDataMenuSelected(app, event)
% Giving input as excel file
f=figure();
drawnow;
app.input_data = uigetfile('*.xlsx','Select an input file',app.selpath);
app.check =1;
delete(f);
end
```

Figure 3.22:Import Data

We take input as a excel file, In this code 1st two lines are used because when we use uigetfile function command window of MATLAB appear and application window is minimized in APP DESIGNER so whenever we get data from user we have to make the figure and delete it after getting data. We save input data in “input_data” variable which is defined in properties so it can be accessed by using “app.” Infront of variable, uigetfile is the command which is used to get data from user and selpath is the directory from where it get that data which is defined by user in above function and check variable is used because we have both options either give data manually or by excel file so check variable shows that data is given by excel file or manually and helps in calculation.

3.9.1.8 Quit Function in File Menu:

This function is used to quit the application

```
% Menu selected function: QuitMenu
function QuitMenuSelected(app, event)
    delete(app);
end
```

Figure 3.23:Quit Function

3.9.1.9 Define Nodes in Node Menu:

We call the GUI which we built to make data of different nodes, we make call of GUI by its name:

```
% Menu selected function: DefineNodesMenu
function DefineNodesMenuSelected(app, event)
    Nodes;
end
```

Figure 3.24Calling Nodes GUI

“Nodes” is the name of already built GUI which is used to save data for different nodes of structure.

3.9.1.10 Settlement in Node Menu:

We call the GUI which can take the data of settlement of every node for this we made GUI with name “Settlement” which we call upon using this button which is show below:

```

% Menu selected function: SettlementMenu
function SettlementMenuSelected(app, event)
    settlement;
end

```

Figure 3.25: Calling Settlement GUI

3.9.1.11 Connectivity Menu:

We made the GUI which can take data of connection between different nodes, so for this we made GUI with name “connection” , This GUI is called using connectivity button, whose command is show below:

```

% Menu selected function: ConnectivityMenu
function ConnectivityMenuSelected(app, event)
    |connection;
end

```

Figure 3.26: Calling Connectivity GUI

3.9.1.12 Material Properties Menu:

We made the GUI which can take data of different materials being used to build the structure, so for this we made GUI with name “material” , This GUI is called using Material properties button, whose command is show below:

```

% Menu selected function: MaterialPropertiesMenu
function MaterialPropertiesMenuSelected(app, event)
    |material;
end

```

Figure 3.27: Calling Material properties GUI

3.9.1.13 Sectional Properties Menu:

We made the GUI which can take data of different sections being used to build the

structure, so for this we made GUI with name “Sectional_Properties” , This GUI is called using Sectional properties button, whose command is show below:

```
% Menu selected function: SectionalPropertiesMenu
function SectionalPropertiesMenuSelected(app, event)
    Sectional_Properties;
end
```

Figure 3.28: Calling sectional properties GUI

3.9.1.14 Nodal Load Menu:

We made the GUI which can take data of different Loads being applied to nodes , so for this we made GUI with name “Nodal_Load” , This GUI is called using Nodal Load button, whose command is show below:

```
% Menu selected function: NodalLoadMenu
function NodalLoadMenuSelected(app, event)
    Nodal_Load;
end
```

Figure 3.29: Calling Nodal load GUI

3.9.1.15 Run Button:

It is the most important button because it contains all the code which run analysis of structure, first we copy all the code which we made to run 3D analysis of steel structure in matlab, As the syntax of MATLAB and app designer is little bit different so we have to do some changes in this code which we already made in MATLAB, specially plotting is doing different in app designer now detailed changes is explained step by step:

```

% Button pushed function: Button, RUNButton
function ButtonPushed(app, event)

% clearing data in plot if already plotted
cla(app.UIAxes)

```

Figure 3.30: Clearing all data plotted in our APP

In the start of Run button we use this command so that if we want to do multiple analysis so first we have to clear the already plotted structure for it “cla” command is being used and “app.UIAxes” is name of the plot on which we are plotting original and deflected shape of structure.

```

%check that input data is in excel file or manually
if app.check == 0
    app.input_data = 'C:\Users\Mr.Smart\Desktop\fyp material\code\Direct Matrix\1'
end

```

Figure 3.31: APP designer code for Saving data from user in variable

After clearing plot, we check that the data given by user is either in excel file or manually filled by user by different options (i.e. Define Nodes, Settlement, etc.). this check is done by using the property variable which is also used in Import Data function. We have built a default excel file with name “1.xlsx” in it all the data is saved which user put manually through different options(i.e. Define Nodes, Settlement, etc.) and we get input in “input_data” property variable which is used further to extract data from it for analysis of steel structure.

```

%pop-up window which shows analysis has been started
f = msgbox({'Congratulations:"Analysis has been started"'; 'Wait for few seconds'}, 'Busy');

```

Figure 3.32: APP designer code for pop-up window

Above code is used to create a pop-up window to ensure that the Analysis has been started, “msgbox()” is the command used to create a pop-up window, As shown we can split our message in different lines using semicolon and can also put heading of pop-up window after comma as we can see Pop up window

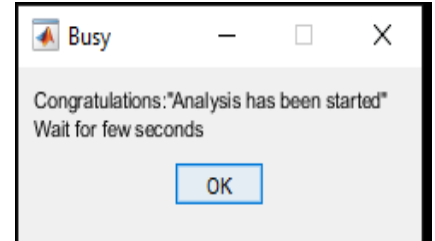


Figure 3.33: APP designer code for pop-up window

All the MATLAB code which run 3D Analysis that remains same, but some changes is done which is shown below:

```
% Support Reactions  
  
Q_global = Kru*u_unrestrained  
  
app.table_q = [restrained_dof,Q_global];
```

Figure 3.34: Saving support reactions in property variable

```
varnames1 = ['Dofs','Reaction forces'];  
app.table_u = [all_dof,u_global]
```

Figure 3.35: Saving deformations in property variable

In our MATLAB code we just done some minor changes so that we access its data for showing our results for it we made two property variables which “table_u” and “table_q” which is used to show the deflections of each Node and support reactions respectively in show results option of GUI.

After it the only changes is done in plotting of original and deflected shape of the structure which is shown below:


```

% Plotting the Original Frame
original_coordinates = [X1 Y1 Z1 X2 Y2 Z2];
original_coordinates_X = [X1 X2];
original_coordinates_Y = [Y1 Y2];
original_coordinates_Z = [Z1 Z2];

for k = 1:Nmembers

plot3(app.UIAxes,original_coordinates_X(k,:),original_coordinates_Z(k,:), original_coordinates_Y(k,:), 'k', "Color",[1,1,1]);
hold(app.UIAxes,"on");

% camup([0 1 0])

end

```

Figure 3.36: APP designer code to plotting frame

Above code shows how the original code is being plotted in APP DESIGNER , it is just similar to MATLAB code but you have to specify the Axes on which you want to plot and then define co-ordinates and other properties for plotting , it's color for plotting is kept white by using “[1,1,1]” in plot3 command as “[1,1,1]” in APP DESIGNER denotes white color, hold command is kept on for Axes on which we want to plot, this hold command is mostly used when we have to plot multiple data on one plot so we have to also plot the deflected shape so let's see the command for plotting of deflected shape:

```

plot3(app.UIAxes,x,z,y, 'k--', "Color",[1,1,1]);

% Plotting New Nodal Coordinates with use of Marker

deformed_coordinates_X = x(end);
deformed_coordinates_Y = y(end);
deformed_coordinates_Z = z(end);

plot3(app.UIAxes,deformed_coordinates_X, deformed_coordinates_Z, deformed_coordinates_Y,"Color",[1,1,1]);
end

hold(app.UIAxes,"off");

```

Figure 3.37: APP designer code for plotting deflected shape of frame

As we can see the code for plotting the deflected shape of structure using Analysis in it the same procedure is being used for plotting as described above, but we can see we did not use hold command on because it is already on and after doing all plotting we make hold command off so that no more plotting is being done on our desired axes.

These all are the minor changes which we done in our MATLAB code to Run in APP

designer.

3.9.1.16 Show Results Button:

In this button we show the results of the Analysis that we have done by using Run button for it we show the deflections on each node and end reaction by using tables, tables are being placed at same position on the graph so position off graphs and plot is same so we use visible command as shown below:

```
% Button pushed function: SHOWRESULTSButton
function SHOWRESULTSButtonPushed(app, event)
    app.TabGroup.Visible = 'on';
    app.BACKTOFIGUREButton.Visible = 'on';

    app.UITable.Data = [app.table_q];
    app.UITable_2.Data=app.table_u;
end
```

Figure 3.38: Show results function

As we used our two tables in tab group so to show tables we keep visibility of tab group on by Visible command and we also built the button “BACKTOFIGURE” which used to show the plot instead of results so we also kept on visibility of BACKTOFIGURE button, we already dropped two tables in tab group in 1st table we saved data of end reactions and in 2nd table we saved the deflections along each node.

3.9.1.17 Back to Figure Button:

This button is visible when show results buttons is being used, function of this button is when user want to go back to show the shape of structure, so this button is being used its command is shown below:

```
% Button pushed function: BACKTOFIGUREButton
function BACKTOFIGUREButtonPushed(app, event)
    app.TabGroup.Visible = 'off';
    app.BACKTOFIGUREButton.Visible = 'off';
end
```

Figure 3.39: Back to figure button

In this simply tab group visibility is kept off because the results is saved in tab group and back to figure button visibility button is kept off because it is in similar position to the show results button and is not in being used further.

Now we have completed the Main GUI now we have to make a GUIs which we use in callback function of different buttons

3.9.2 CREATING A SUB GUI

Nodes GUI:

Now we have to make the GUIs which is being used in our main GUI to get the data from user for this we open new GUI in APP DESIGNER, we make the GUI with name “Nodes” which is being used in callback function of define nodes in main GUI, step by step guide is shown below to make “Nodes” GUI:

a. Design view of GUI:

For the sake of convenience first we make the design view of our GUI and drop all the options in Design view which are required, we first drop Panel, numeric edit fields, buttons, and table according to need as shown below:

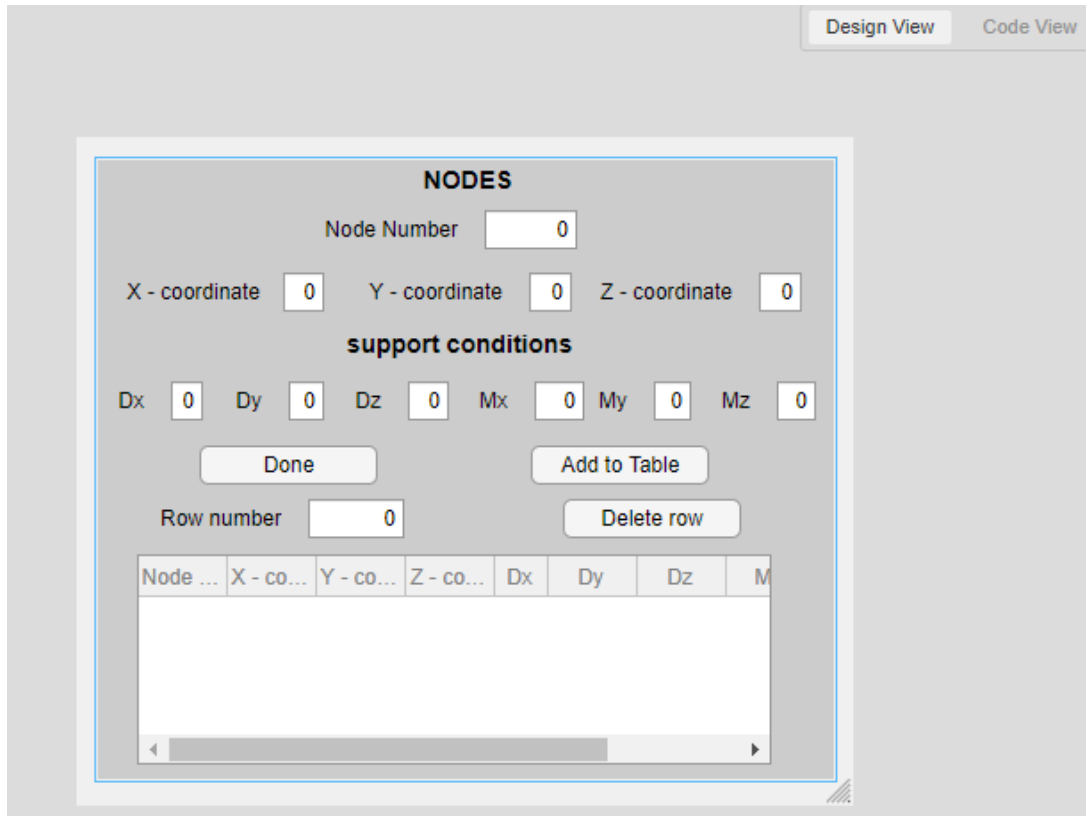


Figure 3.40: Design view of nodes GUI

So we make the panel name NODES and it contains x, y and z coordinates corresponding to every node for support conditions we define two standards 0 means free and 1 means fixed so along every axes user can specify support conditions as default condition is set free for every node. For support conditions is either 1 or 0 so we set limit on all 6 numeric edit field between 0 and 1 and its input is always integer which is done using inspector window as shown below:

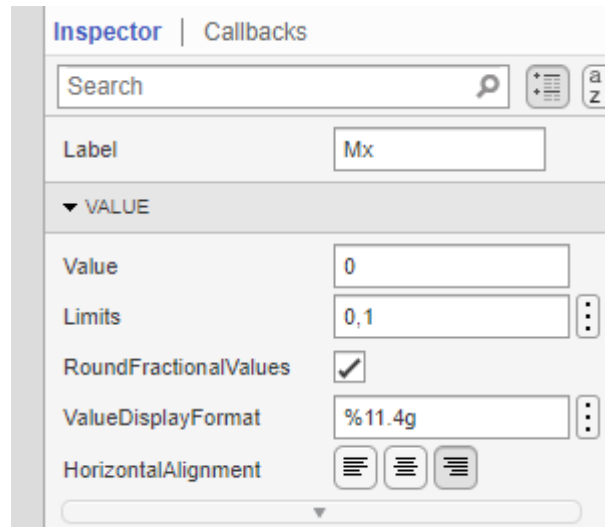


Figure 3.41: Inspector window in APP designer

After this we used some button according to need and delete row option if user want to edit any entered data and table is also being edited in inspector window like shown above.

b. Properties:

We used one property variable in which we save the data given by user in form of table as shown below:

```

properties (Access = private)
    Node % Description
end

```

Figure 3.42: Property Function

c. Add to Table Button:

This button saves the data from user in table and show the data in the table dropped in design view let's see the callback for this function:

```
% Button pushed function: AddtoTableButton
function AddtoTableButtonPushed(app, event)
    NodeNumber = app.NodeNumberEditField.Value;
    Xcoordinate = app.XcoordinateEditField.Value;
    Ycoordinate = app.YcoordinateEditField.Value;
    Zcoordinate = app.ZcoordinateEditField.Value;
    Dx=app.DxEditField.Value;
    Dy=app.DyEditField.Value;
    Dz=app.DyEditField.Value;
    Mx=app.MxEditField.Value;
    My=app.MyEditField.Value;
    Mz =app.MzEditField.Value;
```

Figure 3.43: Add to table function

Above code shows that we save the data given by user in certain variables and to access that data from numeric edit field we used “.Value” corresponding to every numeric edit field, this is the syntax of APP DESIGNER.

```
nr = {NodeNumber Xcoordinate Ycoordinate Zcoordinate Dx Dy Dz Mx My Mz};
app.UITable.Data = [app.Node;nr]; %New row added
app.Node=app.UITable.Data; %table is modified
```

After getting input from user and saving it in variable, we make a cell array of all the data being given by user it is added as a new row in the table which is dropped in GUI and “app.Node” is the empty variable then we save the new data of our table in this variable for further use and to save the previous data.

```

% NOW restting all the values
app.NodeNumberEditField.Value = 0;
app.XcoordinateEditField.Value= 0;
app.YcoordinateEditField.Value=0;
app.ZcoordinateEditField.Value=0;
app.DxEditField.Value=0;|
app.DyEditField.Value=0;
app.DzEditField.Value=0;
app.MxEditField.Value=0;
app.MyEditField.Value=0;
app.MzEditField.Value=0;
end

```

Figure 3.44: Resetting values in add to table function

After saving data in table and variable we reset the values of every numeric edit field to 0 so that user can easily re-input the data.

d. Delete row Button:

This button is used because if user wants to edit any given data after putting it then we can easily delete it using this button and its callback is shown below:

```

% Button pushed function: DeleterowButton
function DeleterowButtonPushed(app, event)
    rn = app.RownumberEditField.Value;
    app.UITable.Data(rn,:) = [];
    app.Node=app.UITable.Data;
    app.RownumberEditField.Value = 0;
end

```

Figure 3.45: Delete row function

In 1st line we get input from user and save it in variable “rn”, in 2nd line we take rnth row and all columns from table and equal it to empty matrix by using empty square brackets, in 3rd row we modify our edited data by saving it in variable “app.Node” and in 4th row reset the numeric edit field to zero so that user can easily edit data again.

e. Done Button

This button saves the all the data given by user and quit the GUI and its callback is shown below:

```

% Button pushed function: DoneButton
function DoneButtonPushed(app, event)
    % Enable the Plot Options button in main app
    folder = 'C:\Users\Mr.Smart\Desktop\fyp material\code\Direct Matrix\1';
    SheetNum="nodes";
    [N, T, Raw]=xlsread(folder, SheetNum);
    [Raw{:}, :]= deal(NaN);
    xlswrite(folder,Raw,SheetNum, 'B2');
    xlswrite(folder,app.UITable.Data,SheetNum, 'B2');

% Delete the dialog box
    delete(app)
end

```

Figure 3.46: Done Button function

We save the data given by user in excel file which is already built in our directory with name “1.xlsx” and all sheet names is already made so first two lines is used to save the path and sheet name in which we save the data, 3rd and 4th is used make an empty matrix of same size as of the data present in specified excel sheet and 5th row is used to delete all the data if already present in specified sheet name because this is the only excel file which is used multiple times and 6th row is used to save the new data in excel file which is given by user in specified excel sheet, we start writing our data from range “B2” in excel sheet and in last after saving data in excel, software is being closed using delete(app) command.

3.9.3 Settlement GUI:

Now we make the GUI with name “Settlement” which is being used in callback function of Settlement in main GUI, step by step guide is shown below to make “Settlement” GUI:

a. Design view of GUI:

For the sake of convenience first we make the design view of our GUI and drop all the

options in Design view which are required, we first drop Panel, numeric edit fields, buttons and table according to need as shown below:

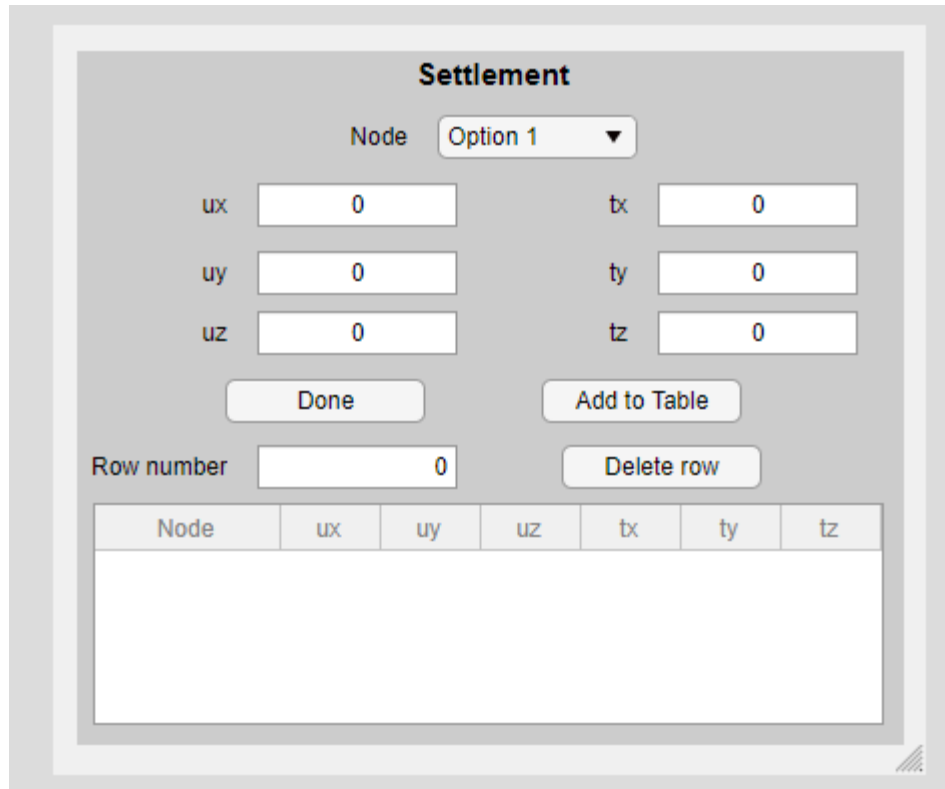


Figure 3.47: Design view of Settlement GUI

So we make the panel name Settlement and it contains deflection and rotations along x, y and z Axes corresponding to every node, we also used some buttons according to need and delete row option if user want to edit any data.

b. Properties:

We used one property variable in which we save the data given by user in form of table as shown below:

```
properties (Access = private)
    load % Description
end
```

Figure 3.48: Property Function in Settlement GUI

c. Startup Function:

This function is performed when this GUI is called let's see the call back for this function:

```
% Code that executes after component creation
function startupFcn(app)
    sheetnum="nodes";
    data = xlsread("1.xls", sheetnum);
    x = data(:,1);
    x=x.';
    y=length(x);
    for i=1:y
        z{i}= num2str(x(i));
    end
    app.NodeDropDown.Items = z;
end
```

Figure 3.49: Start up function for Settlement GUI

Above code is used to take the data of nodes being given by user in the define node option of main GUI, As Nodes data is being saved in excel file "1.xlsx" and "nodes" sheet so first three lines is used to extract data from excel using "xlsread" command and saved in data variable as 1st column contains nodes number in data variable which is being saved in x variable later on, Now we have to show the number of nodes in drop down menu so that user can easily select it, as data is saved in array but we have to save each node number into corresponding cell array because drop down menu consider matrix members as one component of drop down menu so we are converting into cell array for this for loop is being used as shown above and data of matrix is being saved in corresponding data of cell array after it using command ".Items" after drop down we can assign data of nodes by cell array which is being saved in variable z.

d. Add to Table Button:

This button saves the data from user in table and show the data in the table dropped in design view let's see the callback for this function:

```

% Button pushed function: AddtoTableButton
function AddtoTableButtonPushed(app, event)
    Node = app.NodeDropDown.Value;
    fx=app.uxEtField.Value;
    fy=app.uYEtField.Value;
    fz=app.uZEtField.Value;
    mx=app.txEtField.Value;
    my=app.tYEtField.Value;
    mz=app.tZEtField.Value;

```

Figure 3.50: Add to table function for Settlement GUI

Above code shows that we save the data given by user in certain variables and to access that data from numeric edit field and drop down we used “Value” corresponding to every numeric edit field and drop down, this is the syntax of APP DESIGNER.

```

nr = {Node fx fy fz mx my mz};
app.UITable.Data = [app.load;nr]; %New row added
app.load=app.UITable.Data; %table is modified

```

Figure 3.51: adding input data in variable

After getting input from user and saving it in variable, we make a cell array of all the data being given by user and it is added as a new row in the table which is dropped in GUI and “app.load” is the empty variable then we save the new data of our table in this variable for further use and to save the previous data.

```

% NOW restting all the values
app.uxEtField.Value=0;
app.uYEtField.Value=0;
app.uZEtField.Value=0;
app.txEtField.Value=0;
app.tYEtField.Value=0;
app.tZEtField.Value=0;

```

Figure 3.52:Resetting variable in add to table function

After saving data in table and variable we reset the values of every numeric edit field to 0 so that user can easily re-input the data.

e. Delete row Button:

This button is used because if user wants to edit any given data after putting it then we can easily delete it using this button and its callback is shown below:

```
% Button pushed function: DeleterowButton
function DeleterowButtonPushed(app, event)
    rn = app.RownumberEditField.Value;
    app.UITable.Data(rn,:) = [];
    app.Node=app.UITable.Data;
    app.RownumberEditField.Value = 0;
end
```

Figure 3.53: Delete row function for Settlement GUI

In 1st line we get input from user and save it in variable “rn”, in 2nd line we take rnth row and all columns from table and equal it to empty matrix by using empty square brackets, in 3rd row we modify our edited data by saving it in variable “app.load” and in 4th row we reset the numeric edit field to zero so that user can easily edit data again.

f. Done Button:

This button saves the all the data given by user and quit the GUI and its callback is shown below:

```
% Button pushed function: DoneButton
function DoneButtonPushed(app, event)
    % Enable the Plot Opions button in main app
    folder = 'C:\Users\Mr.Smart\Desktop\fyp material\code\Direct Matrix\1';
    SheetNum="settlement";
    [N, I, Raw]=xlsread(folder, SheetNum);
    [Raw{:}, :]= deal(NaN);
    xlswrite(folder,Raw,SheetNum,'B2');
    xlswrite(folder,app.UITable.Data,SheetNum,'B2');

    % Delete the dialog box
    delete(app)
end
```

Figure 3.54: Done function for Settlement GUI

We save the data given by user in excel file which is already built in our directory with name “1.xlsx” and all sheet names is already made so first two lines is used to save the path and sheet name in which we save the data, 3rd and 4th is used make an empty matrix of same size as of the data present in specified excel sheet and 5th row is used to delete all the data if already present in specified sheet name because this is the only excel file which is used multiple times and 6th row is used to save the new data in excel file which is given by user in specified excel sheet, we start writing our data from range “B2” in excel sheet and in last after saving data in excel, software is being closed using delete(app) command.

3.9.4 Connection GUI:

Now we make the GUI with name “connection” which is being used in callback function of Connectivity in main GUI, step by step guide is shown below to make “connection” GUI:

a) Design view of GUI:

For the sake of convenience first we make the design view of our GUI and drop all the options in Design view which are required, we first drop Panel, numeric edit fields, buttons, and table according to need as shown below:

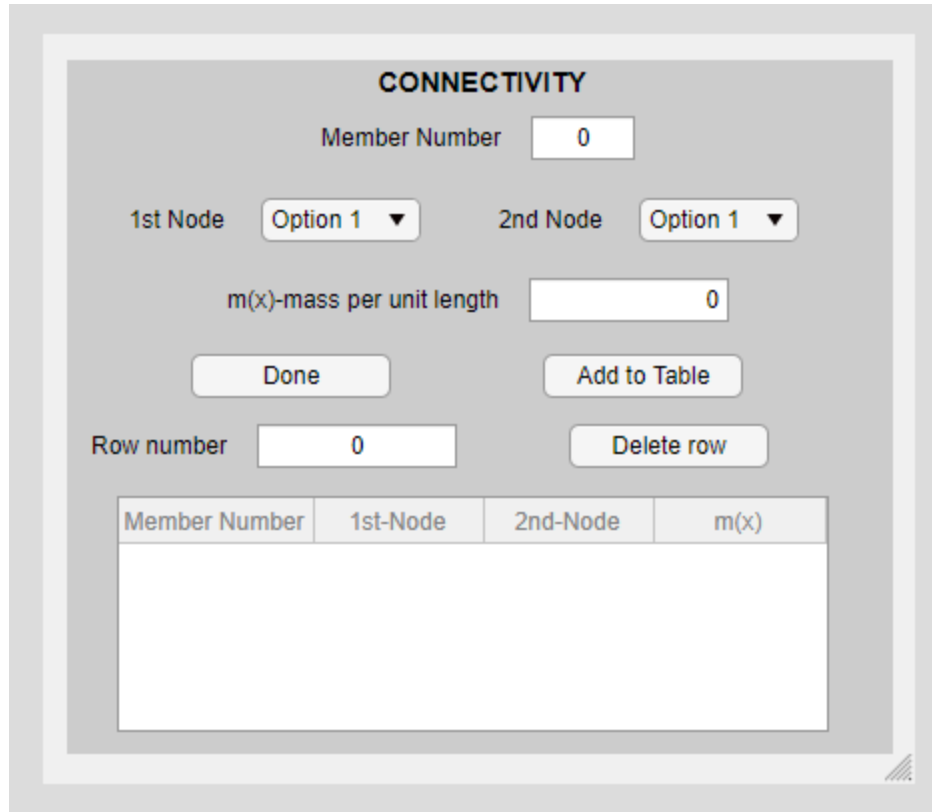


Figure 3.55: Design view of Connectivity GUI

So we make the panel name Connectivity and it is used to define the connection of nodes which make member and $m(x)$ for corresponding to every member, we also used some buttons according to need and delete row option if user want to edit any data.

b) Properties:

We used one property variable in which we save the data given by user in form of table as shown below:

```
properties (Access = private)
    connect % Description
end
```

c) Startup Function:

This function is performed when this GUI is called let's see the call back for this function:

```

% Code that executes after component creation
function startupFcn(app)
sheetnum="nodes";
data = xlsread("1.xlsx", sheetnum);
x = data(:,1);
x=x.';
y=length(x);
for i=1:y
    z{i}= num2str(x(i));
end
app.stNodeDropDown.Items = z;
app.ndNodeDropDown.Items = z;
end

```

Figure 3.56: : Start up function for Connectivity GUI

Above code is used to take the data of nodes being given by user in the define node option of main GUI, As Nodes data is being saved in excel file "1.xlsx" and "nodes" sheet so first three lines is used to extract data from excel using "xlsread" command and saved in data variable as 1st column contains nodes number in data variable which is being saved in x variable later on, Now we have to show the number of nodes in both drop down menus so that user can easily select it, as data is saved in array but we have to save each node number into corresponding cell array because drop down menu consider matrix members as one component of drop down menu so we are converting into cell array for this for loop is being used as shown above and data of matrix is being saved in corresponding data of cell array as shown above cell array is being saved in variable z, after using command ".Items" after drop down we can assign data of nodes by variable z to both drop down items.

d) Add to Table Button:

This button saves the data from user in table and show the data in the table dropped in design view let's see the callback for this function:

```

% Button pushed function: AddtoTableButton
function AddtoTableButtonPushed(app, event)
    MemberNumber = app.MemberNumberEditField.Value;
    stnode = app.stNodeDropDown.Value;
    ndnode = app.ndNodeDropDown.Value;
    mx=app.mxmassperunitlengthEditField.Value;

```

Figure 3.57: : Add to table function for Connectivity GUI

Above code shows that we save the data given by user in certain variables and to access that data from numeric edit field and drop down we used “.Value” corresponding to every numeric edit field and drop down, this is the syntax of APP DESIGNER.

```

nr = {MemberNumber stnode ndnode mx};
app.UITable.Data = [app.connect;nr]; %New row added
app.connect=app.UITable.Data; %table is modified

```

Figure 3.58: adding input data in variable

After getting input from user and saving it in variable, we make a cell array of all the data being given by user and it is added as a new row in the table which is dropped in GUI and “app.connect” is the empty variable then we save the new data of our table in this variable for further use and to save the previous data.

```

% NOW resetting all the values
app.MemberNumberEditField.Value = 0;
app.mxmassperunitlengthEditField.Value=0;

```

Figure 3.59: Resetting all variables

After saving data in table and variable we reset the values of every numeric edit field to 0 so that user can easily re-input the data.

e) Delete row Button:

This button is used because if user wants to edit any given data after putting it then we can easily delete it using this button and its callback is shown below:


```

% Button pushed function: DeleterowButton
function DeleterowButtonPushed(app, event)
    rn = app.RownumberEditField.Value;
    app.UITable.Data(rn,:) = [];
    app.Node=app.UITable.Data;
    app.RownumberEditField.Value = 0;

end

```

Figure 3.60: Delete Row function for Connectivity GUI

In 1st line we get input from user and save it in variable “rn”, in 2nd line we take rnth row and all columns from table and equal it to empty matrix by using empty square brackets, in 3rd row we modify our edited data by saving it in variable “app.load” and in 4th row we reset the numeric edit field to zero so that user can easily edit data again.

f) Done Button:

This button saves the all the data given by user and quit the GUI and its callback is shown below:

```

% Button pushed function: DoneButton
function DoneButtonPushed(app, event)
    % Enable the Plot Options button in main app
    folder = 'C:\Users\Mr.Smart\Desktop\fyp material\code\Direct Matrix\1';
    SheetNum="connectivity";
    [N, T, Row]=xlsread(folder, SheetNum);
    [Row{:, :}] = deal(NaN);
    xlswrite(folder, Row, SheetNum, 'B2');
    xlswrite(folder, app.UITable.Data, SheetNum, 'B2');

    % Delete the dialog box
    delete(app)

end

```

Figure 3.61: : Done function for Connectivity GUI

We save the data given by user in excel file which is already built in our directory with name “1.xlsx” and it’s all sheet names is already made so first two lines is used to save the path and sheet name in which we save the data, 3rd and 4th is used make an empty matrix of same size as of the data present in specified excel sheet and 5th row is used

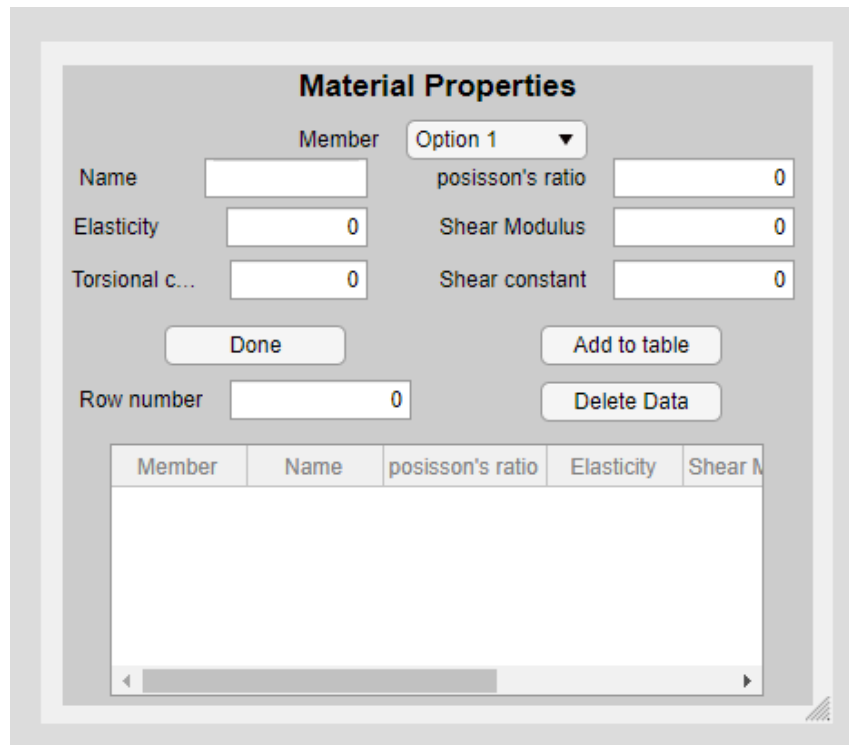
to delete all the data if already present in specified sheet name because this is the only excel file which is used multiple times and 6th row is used to save the new data in excel file which is given by user in specified excel sheet, we start writing our data from range “B2” in excel sheet and in last after saving data in excel, software is being closed using delete(app) command.

3.9.5 Material GUI:

Now we make the GUI with name “material” which is being used in callback function of Material properties in main GUI, step by step guide is shown below to make “material” GUI:

a. Design view of GUI:

For the sake of convenience first we make the design view of our GUI and drop all the options in Design view which are required, we first drop Panel, numeric edit fields, buttons, and table according to need as shown below:



The image shows a design view of a GUI titled "Material Properties". It features a central panel with several input fields and buttons. At the top, there is a "Member" dropdown menu set to "Option 1". Below this, there are five numeric input fields: "Name", "Elasticity", "Torsional c...", "posisson's ratio", "Shear Modulus", and "Shear constant". The "Elasticity", "Torsional c...", and "Shear constant" fields have a value of "0". There are three buttons: "Done", "Add to table", and "Delete Data". A "Row number" input field is set to "0". At the bottom, there is a table with five columns: "Member", "Name", "posisson's ratio", "Elasticity", and "Shear M". The table is currently empty.

Figure 3.62: Design view of Material properties

So, we make the panel name Material Property and it is used to define the property for every member, we also used some buttons according to need and delete row option if user want to edit any data.

b. Properties:

We used one property variable in which we save the data given by user in form of table as shown below:

```
properties (Access = private)
    Material_property % Description
end
```

Figure 3.63: Property Function

c. Startup Function:

This function is performed when this GUI is called let's see the call back for this function:

```
% Code that executes after component creation
function startupFcn(app)
    sheetnum="connectivity";
    data = xlsread("1.xls", sheetnum);
    x = data(:,1);
    x=x.';
    y=length(x);
    for i=1:y
        z{i}= num2str(x(i));
    end
    app.MemberDropDown.Items = z;
end
```

Figure 3.64: Startup function For Material properties GUI

Above code is used to take the data of members being given by user in the connectivity option of main GUI, As Members data is being saves in excel file “1.xlsx” and “connectivity” sheet so first three lines is used to extract data from excel using “xlsread” command and saved in data variable as 1st column contains nodes number

in data variable which is being saved in x variable later on, Now we have to show the Members in drop down menu so that user can easily select it, as data is saved in array but we have to save each Member into corresponding cell array because drop down menu consider matrix members as one component of drop down menu so we are converting into cell array for this for loop is being used as shown above and data of matrix is being saved in corresponding data of cell array after it using command ".Items" after drop down we can assign data of Members by cell array which is being saved in variable z.

d. Add to Table Button:

This button saves the data from user in table and show the data in the table dropped in design view let's see the callback for this function:

```
% Button pushed function: AddtotableButton
function AddtotableButtonPushed(app, event)
    member=app.MemberDropDown.ItemsData;
    Name = app.NameEditField.Value;
    Elasticity = app.ElasticityEditField.Value;
    posissonsratio = app.posissonsratioEditField.Value;
    ShearModulus = app.ShearModulusEditField.Value;
    torsion =app.TorsionalconstantEditField.Value;
    shear=app.ShearconstantEditField.Value;
```

Figure 3.65: Add to table function for Material properties GUI

Above code shows that we save the data given by user in certain variables and to access that data from numeric edit field, text edit field and drop down we used ".Value" corresponding to every numeric edit field and drop down, this is the syntax of APP DESIGNER.

```
nr = {member Name posissonsratio Elasticity ShearModulus torsion shear};
app.UITable.Data = [app.Material_property;nr];%New row added
app.Material_property=app.UITable.Data;%table is modified
```

Figure 3.66: : adding input data in variable

After getting input from user and saving it in variable, we make a cell array of all the

data being given by user and it is added as a new row in the table which is dropped in GUI and “app.Material_property” is the empty variable then we save the new data of our table in this variable for further use and to save the previous data.

```
% NOW resetting all the values
app.NameEditField.Value = "";
app.ElasticityEditField.Value = 0;
app.poissonsratioEditField.Value = 0;
app.ShearModulusEditField.Value = 0;
app.TorsionalconstantEditField.Value = 0;
app.ShearconstantEditField.Value = 0;
```

Figure 3.67: Resetting all values

After saving data in table and variables we reset the values of every numeric and text edit field to 0 and empty respectively so that user can easily re-input the data.

e. Delete row Button:

This button is used because if user wants to edit any given data after putting it then we can easily delete it using this button and its callback is shown below:

```
% Button pushed function: DeleterowButton
function DeleterowButtonPushed(app, event)
    rn = app.RownumberEditField.Value;
    app.UITable.Data(rn,:) = [];
    app.Node=app.UITable.Data;
    app.RownumberEditField.Value = 0;
end
```

Figure 3.68: Delete row function in Material properties function

In 1st line we get input from user and save it in variable “rn”, in 2nd line we take rnth row and all columns from table and equal it to empty matrix by using empty square brackets, in 3rd row we modify our edited data by saving it in variable “app.load” and in 4th row we reset the numeric edit field to zero so that user can easily edit data again.

f. Done Button:

This button saves the all the data given by user and quit the GUI and its callback is

shown below:

```
% Button pushed function: DoneButton
function DoneButtonPushed(app, event)
% Enable the Plot Opions button in main app
folder = 'C:\Users\Mr.Smart\Desktop\fyp material\code\Direct Matrix\1';
SheetNum="material";
[N, T, Row]=xlsread(folder, SheetNum);
[Row{:, :}] = deal(NaN);
xlswrite(folder, Row, SheetNum, 'B2');
xlswrite(folder, app.UITable.Data, SheetNum, 'B2');

% Delete the dialog box
delete(app)
end
```

Figure 3.69: Done function in Material properties GUI

We save the data given by user in excel file which is already built in our directory with name “1.xlsx” and it’s all sheet names is already made so first two lines is used to save the path and sheet name in which we save the data, 3rd and 4th is used make an empty matrix of same size as of the data present in specified excel sheet and 5th row is used to delete all the data if already present in specified sheet name because this is the only excel file which is used multiple times and 6th row is used to save the new data in excel file which is given by user in specified excel sheet, we start writing our data from range “B2” in excel sheet and in last after saving data in excel, software is being closed using delete(app) command.

3.9.6 Sectional Properties GUI:

Now we make the GUI with name “Sectional_Properties” which is being used in callback function of Sectional Properties in main GUI, step by step guide is shown below to make “Sectional_Properties” GUI:

a. Design view of GUI:

For the sake of convenience first we make the design view of our GUI and drop all the options in Design view which are required, we first drop Panel, numeric edit fields,

buttons, and table according to need as shown below:

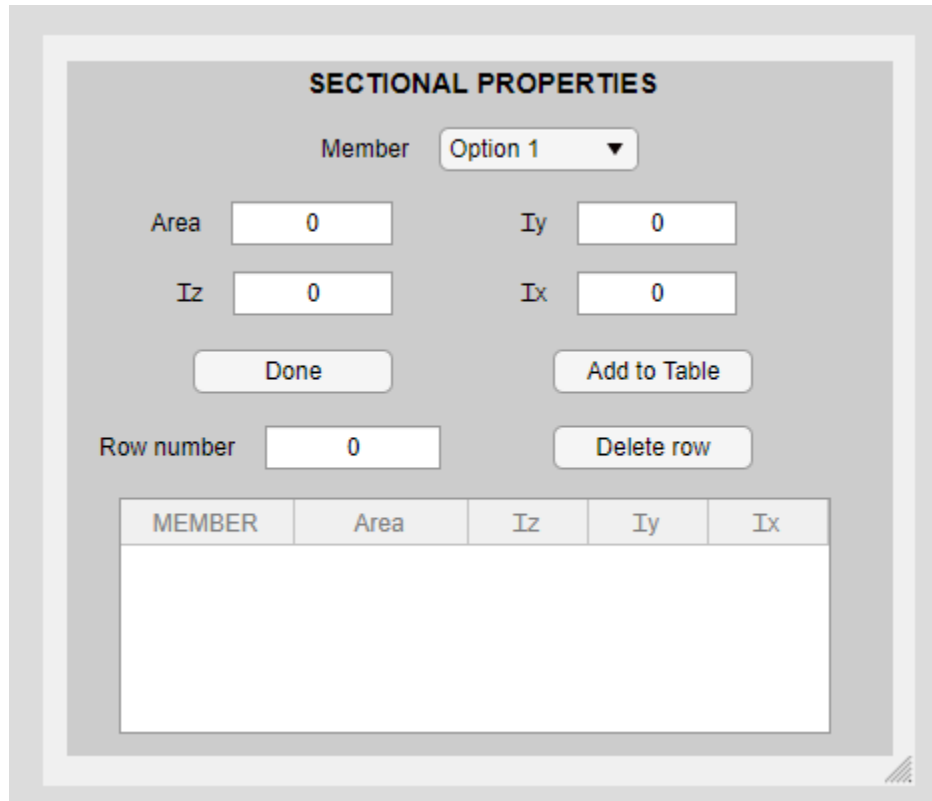


Figure 3.70: Design view of Sectional Properties GUI

So, we make the panel name Sectional Property and it is used to define the sectional property for every member, we also used some buttons according to need and delete row option if user want to edit any data.

b. Properties:

We used one property variable in which we save the data given by user in form of table as shown below:

```
properties (Access = private)
    Property % Description
end
```

Figure 3.71: Property Function for Sectional Properties GUI

c. Startup Function:

This function is performed when this GUI is called let's see the call back for this function:

```
% Code that executes after component creation
function startupFcn(app)
    sheetnum="connectivity";
    data = xlsread("1.xls", sheetnum);
    x = data(:,1);
    x=x.';
    y=length(x);
    for i=1:y
        z{i}= num2str(x(i));
    end
    app.MemberDropDown.Items = z;
end
```

Figure 3.72: Startup function for Sectional Properties GUI

Above code is used to take the data of members being given by user in the connectivity option of main GUI, As Members data is being saved in excel file "1.xlsx" and "connectivity" sheet so first three lines is used to extract data from excel using "xlsread" command and saved in data variable as 1st column contains nodes number in data variable which is being saved in x variable later on, Now we have to show the Members in drop down menu so that user can easily select it, as data is saved in array but we have to save each Member into corresponding cell array because drop down menu consider matrix members as one component of drop down menu so we are converting into cell array for this for loop is being used as shown above and data of matrix is being saved in corresponding data of cell array after it using command ".Items" after drop down we can assign data of Members by cell array which is being saved in variable z.

d. Add to Table Button:

This button saves the data from user in table and show the data in the table dropped in design view let's see the callback for this function:


```

% Button pushed function: AddtoTableButton
function AddtoTableButtonPushed(app, event)
    Member= app.MemberDropDown.Value;
    Area=app.AreaEditField.Value;
    ix=app.xEditField.Value;
    iy=app.yEditField.Value;
    iz=app.zEditField.Value;

```

Figure 3.73: Add to table function for Sectional Properties GUI

Above code shows that we save the data given by user in certain variables and to access that data from numeric edit field and drop down we used “.Value” corresponding to every numeric edit field and drop down, this is the syntax of APP DESIGNER.

```

nr = {Member Area iz iy ix};
app.UITable.Data = [app.Property;nr]; %New row added
app.Property=app.UITable.Data; %table is modified

```

Figure 3.74: Adding input data in variables

After getting input from user and saving it in variable, we make a cell array of all the data being given by user and it is added as a new row in the table which is dropped in GUI and “app.Property” is the empty variable then we save the new data of our table in this variable for further use and to save the previous data.

```

app.Property=app.UITable.Data; %table is modified
% NOW resetting all the values
app.AreaEditField.Value =0;
app.xEditField.Value=0;
app.yEditField.Value =0;
app.zEditField.Value =0;

```

Figure 3.75: Resetting all values

After saving data in table and variables we reset the values of every numeric edit field to 0 so that user can easily re-input the data.

e. Delete row Button:

This button is used because if user wants to edit any given data after putting it then we can easily delete it using this button and its callback is shown below:

```

% Button pushed function: DeleterowButton
function DeleterowButtonPushed(app, event)
    rn = app.RownumberEditField.Value;
    app.UITable.Data(rn,:) = [];
    app.Node=app.UITable.Data;
    app.RownumberEditField.Value = 0;

end

```

Figure 3.76: Delete row function for Sectional Properties GUI

In 1st line we get input from user and save it in variable “rn”, in 2nd line we take rnth row and all columns from table and equal it to empty matrix by using empty square brackets, in 3rd row we modify our edited data by saving it in variable “app.load” and in 4th row we reset the numeric edit field to zero so that user can easily edit data again.

f. Done Button:

This button saves the all the data given by user and quit the GUI and its callback is shown below:

```

% Button pushed function: DoneButton
function DoneButtonPushed(app, event)
    % Enable the Plot Opions button in main app
    folder = 'C:\Users\Mr.Smart\Desktop\fyp material\code\Direct Matrix\1';
    SheetNum="sectional";
    [N, T, Raw]=xlsread(folder, SheetNum);
    [Raw{:}, :]= deal(NaN);
    xlswrite(folder,Raw,SheetNum,'B2');
    xlswrite(folder,app.UITable.Data,SheetNum,'B2');

    % Delete the dialog box
    delete(app)

end

```

Figure 3.77: Done function for Sectional Properties GUI

We save the data given by user in excel file which is already built in our directory with name “1.xlsx” and it’s all sheet names is already made so first two lines is used to save the path and sheet name in which we save the data, 3rd and 4th is used make an empty matrix of same size as of the data present in specified excel sheet and 5th row is used

to delete all the data if already present in specified sheet name because this is the only excel file which is used multiple times and 6th row is used to save the new data in excel file which is given by user in specified excel sheet, we start writing our data from range “B2” in excel sheet and in last after saving data in excel, software is being closed using delete(app) command.

3.9.7 Nodal Load GUI:

Now we make the GUI with name “Nodal_Load” which is being used in callback function of Nodal Load in main GUI, step by step guide is shown below to make “Nodal_Load” GUI:

a. Design view of GUI:

For the sake of convenience first we make the design view of our GUI and drop all the options in Design view which are required, we first drop Panel, numeric edit fields, buttons, and table according to need as shown below:

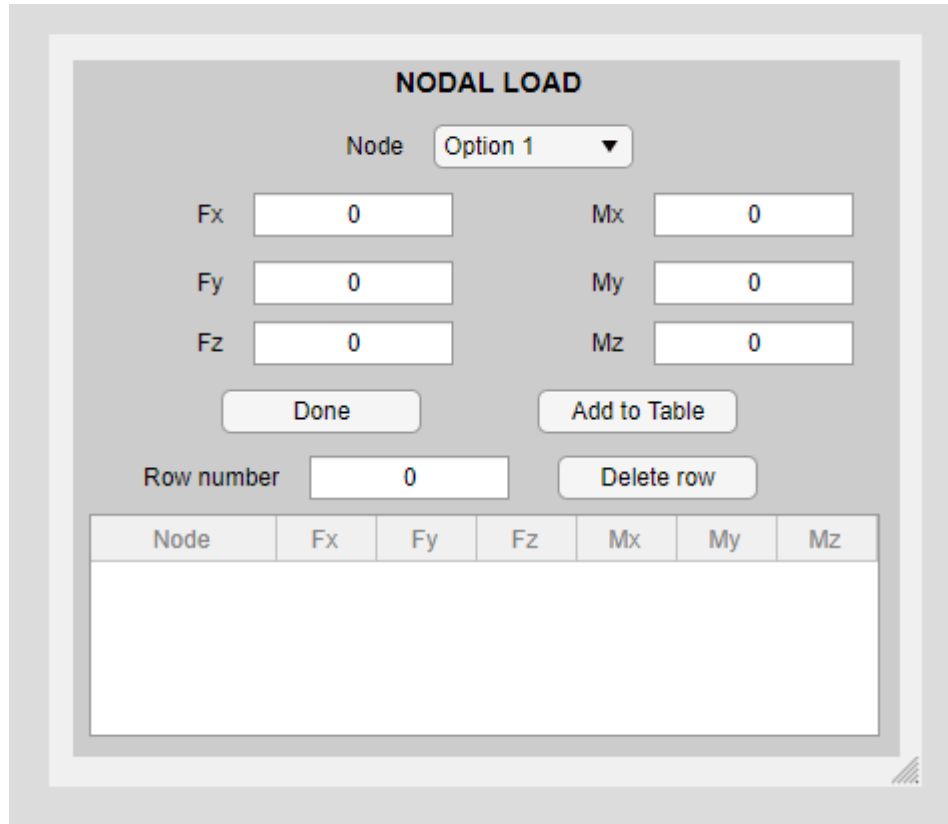


Figure 3.78: Design view of Nodal Load GUI

So we make the panel name Nodal Load and it is used to define the forces and moments in x, y and z Axes corresponding to every node, we also used some buttons according to need and delete row option if user want to edit any data.

b. Properties:

We used one property variable in which we save the data given by user in form of table as shown below:

```

properties (Access = private)
    load % Description
end

```

Figure 3.79: Properties for Nodal Load GUI

c. Startup Function:

This function is performed when this GUI is called let's see the call back for this function:

```
% Code that executes after component creation
function startupFcn(app)
    sheetnum="nodes";
    data = xlsread("1.xlsx", sheetnum);
    x = data(:,1);
    x=x.';
    y=length(x);
    for i=1:y
        z{i}= num2str(x(i));
    end
    app.NodeDropDown.Items = z;
end
```

Figure 3.80: Startup function for Nodal Load GUI

Above code is used to take the data of nodes being given by user in the define node option of main GUI, As Nodes data is being saves in excel file “1.xlsx” and “nodes” sheet so first three lines is used to extract data from excel using “xlsread” command and saved in data variable as 1st column contains nodes number in data variable which is being saved in x variable later on, Now we have to show the number of nodes in drop down menu so that user can easily select it, as data is saved in array but we have to save each node number into corresponding cell array because drop down menu consider matrix members as one component of drop down menu so we are converting into cell array for this for loop is being used as shown above and data of matrix is being saved in corresponding data of cell array after it using command “.Items” after drop down we can assign data of nodes by cell array which is being saved in variable z.

d. Add to Table Button:

This button saves the data from user in table and show the data in the table dropped in design view let's see the callback for this function:

```

% Button pushed function: AddtoTableButton
function AddtoTableButtonPushed(app, event)
    Node = app.NodeDropDown.Value;
    fx=app.FxEditField.Value;
    fy=app.FyEditField.Value;
    fz=app.FzEditField.Value;
    mx=app.MxEditField.Value;
    my=app.MyEditField.Value;
    mz=app.MzEditField.Value;

```

Figure 3.81: Add to table function for Nodal Load GUI

Above code shows that we save the data given by user in certain variables and to access that data from numeric edit field and drop down we used “.Value” corresponding to every numeric edit field and drop down, this is the syntax of APP DESIGNER.

```

nr = {Node fx fy fz mx my mz};
app.UITable.Data = [app.load;nr]; %New row added
app.load=app.UITable.Data; %table is modified

```

Figure 3.82: Adding input data in variables

After getting input from user and saving it in variable, we make a cell array of all the data being given by user and it is added as a new row in the table which is dropped in GUI and “app. Load” is the empty variable then we save the new data of our table in this variable for further use and to save the previous data.

```

% NOW resetting all the values
app.FxEditField.Value=0;
app.FyEditField.Value=0;
app.FzEditField.Value=0;
app.MxEditField.Value=0;
app.MyEditField.Value=0;
app.MzEditField.Value=0;

```

Figure 3.83: Resetting all values

After saving data in table and variable we reset the values of every numeric edit field to 0 so that user can easily re-input the data.

e. Delete row Button:

This button is used because if user wants to edit any given data after putting it then we can easily delete it using this button and its callback is shown below:

```
% Button pushed function: DeleterowButton
function DeleterowButtonPushed(app, event)
    rn = app.RownumberEditField.Value;
    app.UITable.Data(rn,:) = [];
    app.Node=app.UITable.Data;
    app.RownumberEditField.Value = 0;
end
```

Figure 3.84: Delete row function for Nodal Load GUI

In 1st line we get input from user and save it in variable “rn”, in 2nd line we take rnth row and all columns from table and equal it to empty matrix by using empty square brackets, in 3rd row we modify our edited data by saving it in variable “app.load” and in 4th row we reset the numeric edit field to zero so that user can easily edit data again.

f. Done Button:

This button saves the all the data given by user and quit the GUI and its callback is shown below:

```
% Button pushed function: DoneButton
function DoneButtonPushed(app, event)
    % Enable the Plot Opions button in main app
    folder = 'C:\Users\Mr.Smart\Desktop\fyp material\code\Direct Matrix\1';
    SheetNum="Nodal";
    [N, T, Raw]=xlsread(folder, SheetNum);
    [Raw{:, :}] = deal(NaN);
    xlswrite(folder,Raw,SheetNum,'B2');
    xlswrite(folder,app.UITable.Data,SheetNum,'B2');

    % Delete the dialog box
    delete(app)
end
```

Figure 3.85: Done button function for Nodal Load GUI

We save the data given by user in excel file which is already built in our directory with name "1.xlsx" and it's all sheet names is already made so first two lines is used to save the path and sheet name in which we save the data, 3rd and 4th is used make an empty matrix of same size as of the data present in specified excel sheet and 5th row is used to delete all the data if already present in specified sheet name because this is the only excel file which is used multiple times and 6th row is used to save the new data in excel file which is given by user in specified excel sheet, we start writing our data from range "B2" in excel sheet and in last after saving data in excel, software is being closed using delete(app) command.

3.10 Ways to Share App

3.10.1 Share MATLAB Files Directly:

MATLAB gives app details to an operating-systems for providing display in file browsers. App details further improve the process of packaging/compiling apps. The “.mlapp file” gives details to interfaces by itself.

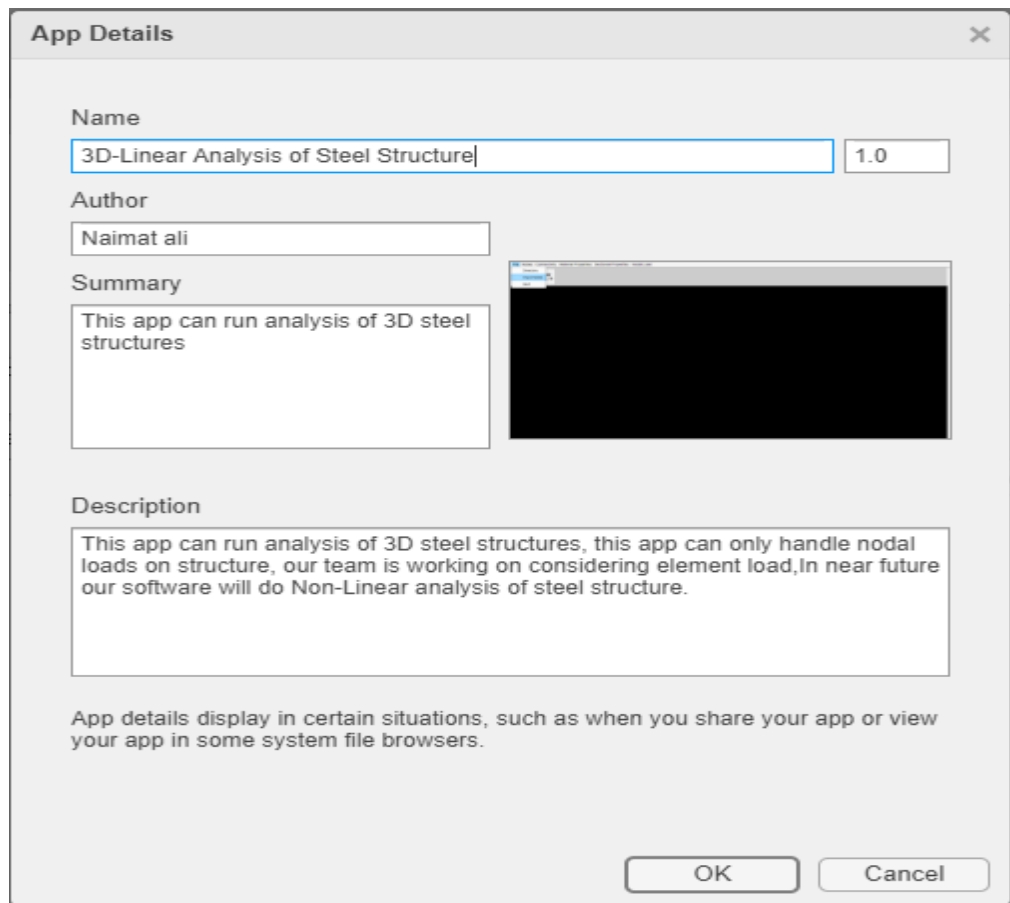


Figure 3.86: Saving application Data in MATLAB APP designer

3.10.2 PACKAGING APPS IN APP-DESIGNER:

The functionality for packaging in App-Designer is similar to the procedure explains the Add-Ons>PackageApp option.

1. In App Designer > Designer tab. Then select **Share > MATLAB App**.



Figure 3.87: Ways to share APP

Package App dialog box is opened by MATLAB.

2. The Package App dialog box has these items.
 - The name of the application matches the name given to the figure in App Designer.
 - The Main file is the MLAPP file you presently have designated for editing.
 - The installation file will be saved in The Output folder.

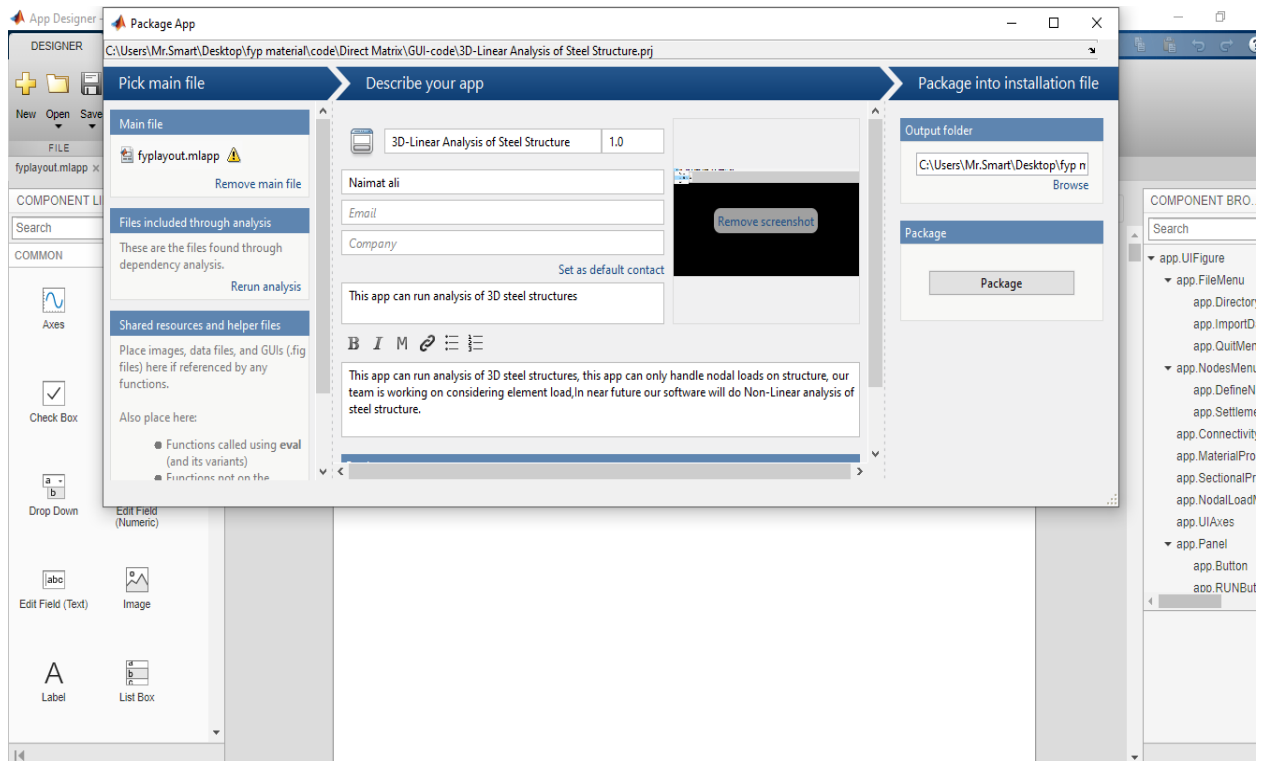


Figure 3.88: Window for creating MATLAB APP

3. Stipulate particulars to show in the app's gallery. Input suitable info in required fields: Company, Author Name, Description, Email and Summary.
4. Products part: choice the products which are needed to operate app.
5. Click Select screenshot: Specify an icon to display gallery of the app.
6. Click Package: create the ".mlappinstall file" to share with users. Click: Package App button in the App Designer Toolstrip again. The Package App dialog box gives you the newly altered .prj file for the MLAPP file.

3.10.3 Creating a Web App:

Apps which can run in the browser are called Web Apps.

Creation of deployed web apps needs MATLAB Compiler apps is deployed as web

apps. Further, some functionality is not maintained in deployed apps. Open the **Web App Compiler** from inside App Designer by clicking **Share** in the **Designer** tab and choosing **Web App** when you have MATLAB Compiler on system,

3.10.4 Creating a Standalone Desktop Application:

You get the ability to share the app with users who do not have MATLAB on their systems by the creation of a standalone desktop application.

It is important to have the MATLAB compiler installed on the system for the creation of a standalone app. Also, MATLAB runtime systems should be installed on the users' computer to run the app. The Application Compiler can be opened once you have the MATLAB compiler. This is done by clicking Designer tab > Share > Standalone Desktop App.

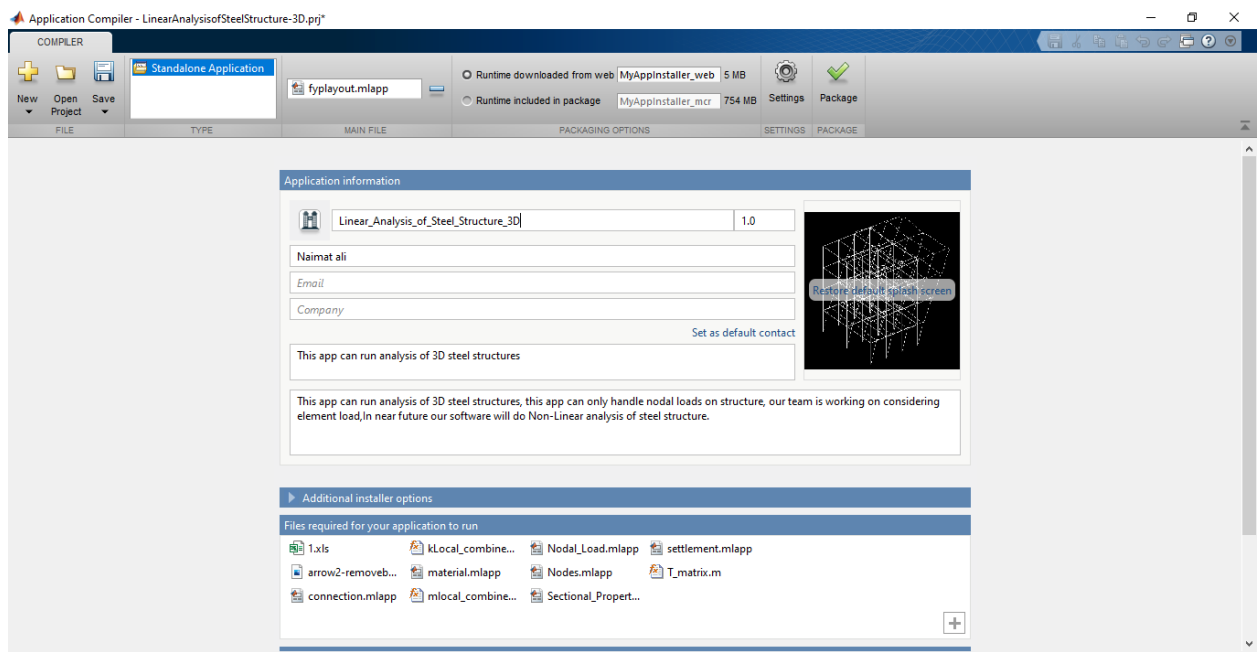


Figure 3.89: Window for creating stand alone software

CHAPTER 4

4.0 RESULTS AND DISCUSSION

4.1 Comparison with MASTAN2:

The comparison of results with MASTAN2 show minimum difference as the results of reaction forces and deflections are overlapping in the *Figure 2.14* and *Figure 2.15*, respectively.

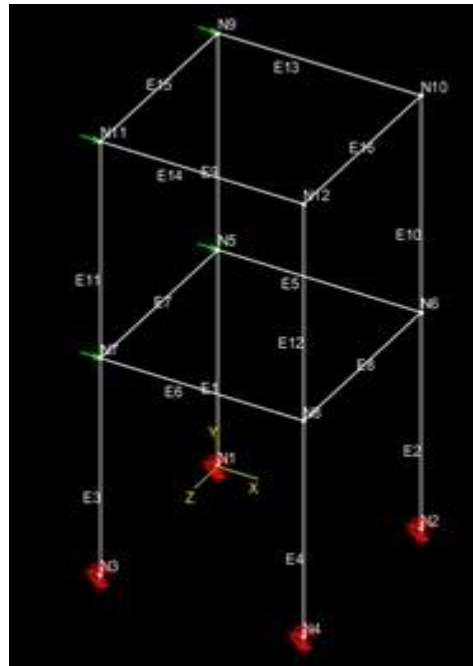


Figure 4.1 3D fixed jointed frame in MASTAN2

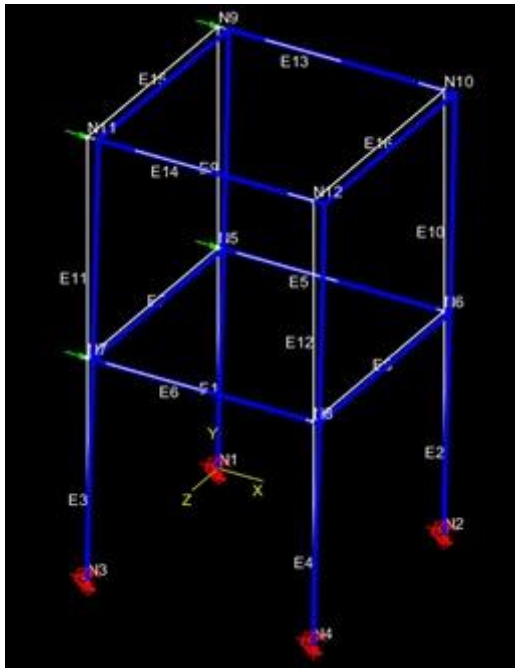


Figure 4.2 3D fixed jointed frame deflection in MASTAN2

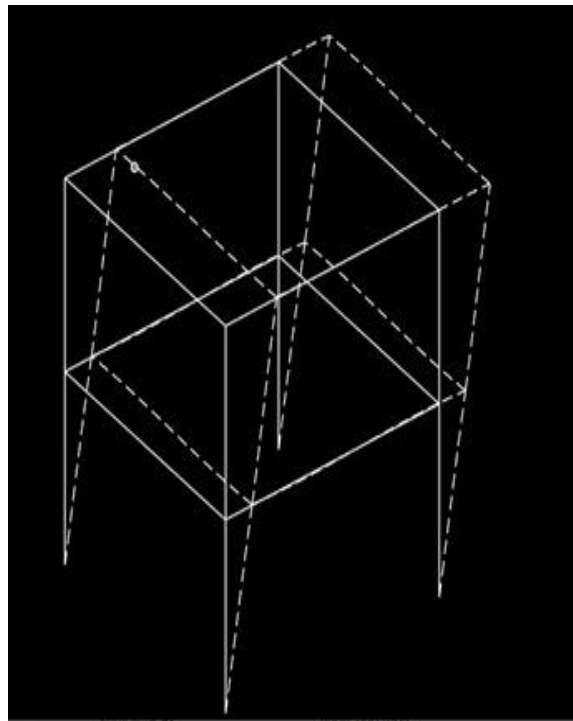


Figure 4.3 Linear Elastic Frame Solver (LEFS) Deflection

4.2 Graphs:

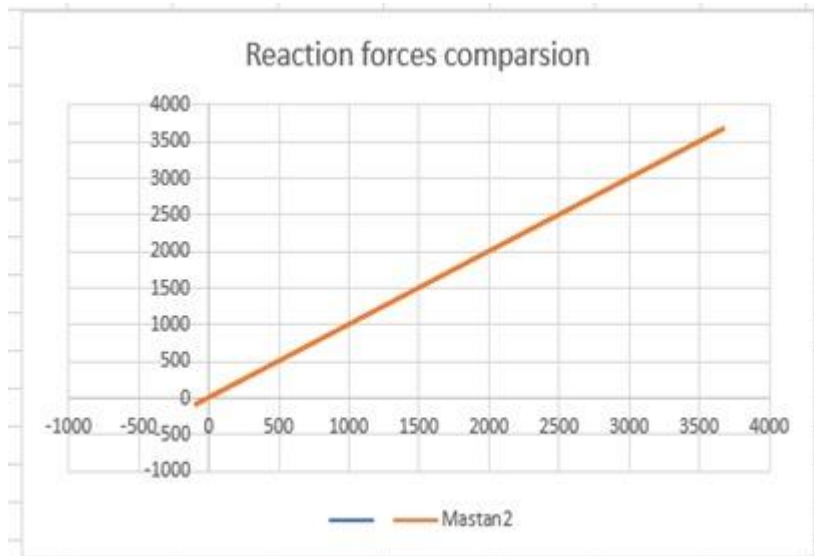


Figure 4.4 Reaction forces comparison of LEFS with MASTAN2

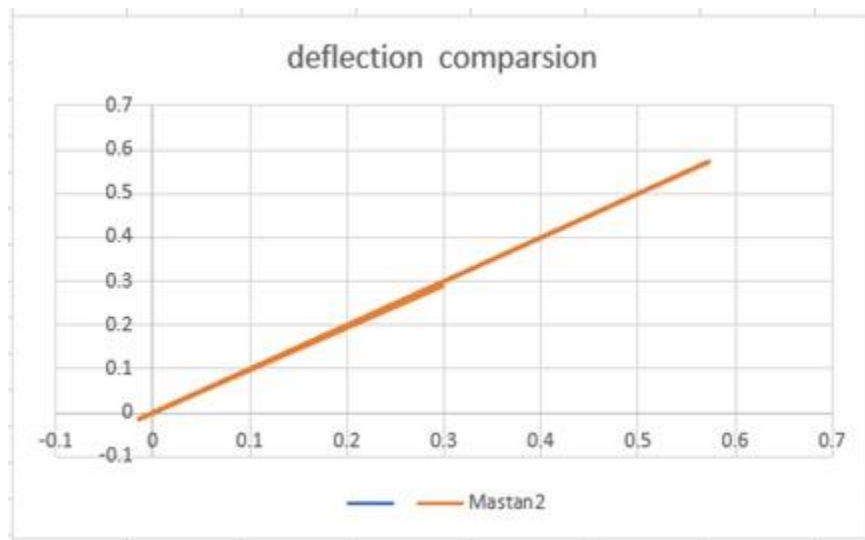


Figure 4.5 Deflection comparison of LEFS with MASTAN2

CHAPTER 5

5.0 CONCLUSION

Linear Elastic Frame Solver (LEFS) serves as an effective means to solve complex engineering structures by following simple instruction provided in the “Instruction Manual”. MATLAB app designer is a very powerful tool to build and develop apps. It can also be manipulated and modified since it's an open source.

LEFS gives accurate results in Linear Elastic range as the comparison with commercial softwares validate the said claim. It is developed specially for university undergraduate students who want to learn structural analysis in MATLAB language. The students can be able to modify the software freely according to their needs.

6.0 References

Maria Paz, Y. H. (2019). Element Stiffness Matrix for Axial Effects. In Y. H. Maria Paz, *Structural Dynamics* (pp. 291-307). Springer Nature Switzerland AG.

Mario Paz, Y. H. (2019). *Structural Dynamics*. Retrieved from [openlibrary.telkomuniversity.ac.id](https://openlibrary.telkomuniversity.ac.id/pustaka/158883/structural-dynamics.html): <https://openlibrary.telkomuniversity.ac.id/pustaka/158883/structural-dynamics.html>

- Dynamics of Structures by Anil K. Chopra (4th Ed)
- Structural Dynamics by Mario Paz & Young Hoon Kim
- Introduction to Finite Element Vibration Analysis (2010) by Maurice Petyt
- Dynamics of Structures - CSI (Computers and Structures, Inc.) (2003) by Ray Clough, Joseph Penzien
- Structural Analysis - Hibbeler-Pearson (2014) - Russell C. Hibbeler
- Matrix Structural Analysis, Second Edition (1999) - William McGuire, Richard H. Gallagher, Ronald D. Ziemian
- <https://openlibrary.telkomuniversity.ac.id/pustaka/158883/structural-dynamics.html>