DEVELOPING GENERALIZED PYTHON CODE FOR STRUCTURAL AND MODAL ANALYSIS OF CONTINUOUS BEAM

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
MECHANICAL ENGINEERING

Submitted by

I Purna Adhitya Varma	**(318126520083)**
D Dhruva Teja	**(318126520075)**
K Chandu Reddy	**(318126520071)**
M Avinash	**(318126520099)**
Y Hemanth	**(318126520115)**

Under the guidance of
Mr.R.Vara Prasad (PhD)
Assistant Professor

DEPARTMENT OF MECHANICAL ENGINEERING
ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCES (Autonomous)

(Affiliated to A.U, Approved by AICTE, Accredited by NBA & NAAC)

Sangivalasa, Bheemunipatnam Mandal, Visakhapatnam - 531163
(2018-2022)

DEPARTMENT OF MECHANICAL ENGINEERING
ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCES (Autonomous)

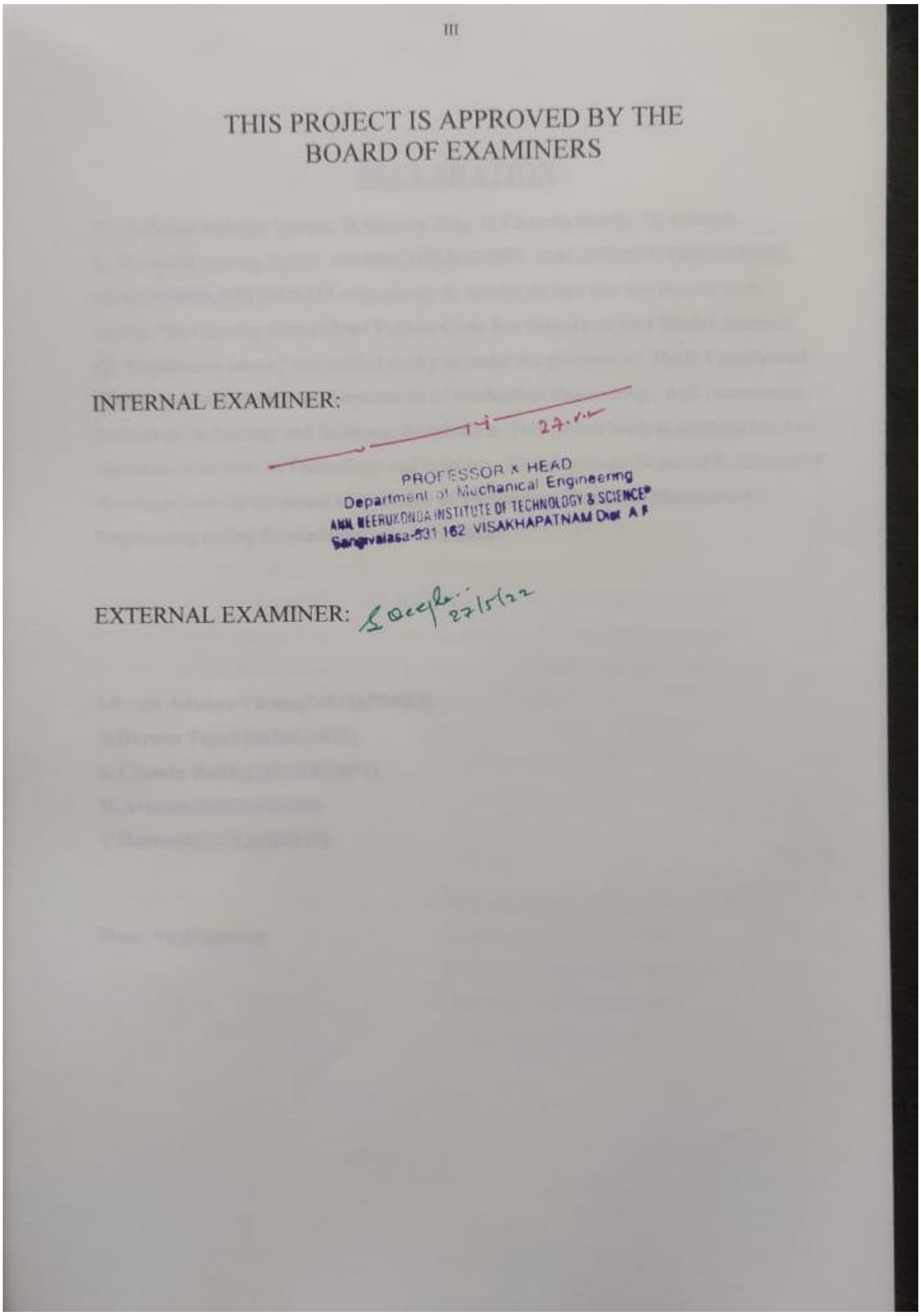
(Affiliated to A.U, Approved by AICTE, Accredited by NBA & NAAC)

Sangivalasa, Bheemunipatnam Mandal, Visakhapatnam - 531163
(2018-2022)

CERTIFICATE

This is to certify that the project report entitled **"Developing Generalized Python Code For Structural and Modal Analysis Of Continuous Beam"**submitted by **Indukuri Purna Adhitya Varma (318126520083),Dannana Dhruva Teja (318126520075), Chandu Reddy Kolli (318126520071), Mutti Avinash (318126520099), Yarlagadda Hemanth (318126520115)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Mechanical Engineering** of Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

Project Guide	**Head Of the Department**
26/5/22	26.5
Mr.R.Vara Prasad	**Dr.B.Naga Raju**
Assistant Professor	Professor
Dept. of Mechanical Engineering	Dept. of Mechanical Engineering
ANITS	ANITS



THIS PROJECT IS APPROVED BY THE BOARD OF EXAMINERS

INTERNAL EXAMINER: 27.5.22

PROFESSOR & HEAD
Department of Mechanical Engineering
ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY & SCIENCE
Sangivalasa-531 162 VISAKHAPATNAM Dist A P

EXTERNAL EXAMINER: 27/5/22

DECLARATION

We, **I.Purna Adhitya Varma, D.Dhruva Teja, K.Chandu Reddy, M.Avinash, Y. Hemanth** bearing register numbers **318126520083, 318126520075, 318126520071, 318126520099,318126520115** respectively do hereby declare that this project work entitled **"Developing Generalized Python Code For Structural and Modal Analysis Of Continuous Beam"** was carried out by us under the guidance of **Mr.R.VaraPrasad (PhD), Assistant Professor** , Department of Mechanical Engineering , Anil Neerukonda Institute of Technology and Sciences , Sangivalasa. This project work is submitted to Anil Neerukonda Institute of Technology and Sciences , Visakhapatnam in partial fulfillment of the requirement for the award of **Bachelor of Technology Degree** in **Mechanical Engineering** during the academic year **2021-2022**

I.Purna Adhitya Varma(318126520083)
D.Dhruva Teja(318126520075)
K.Chandu Reddy(318126520071)
M.Avinash(318126520099)
Y.Hemanth(318126520115)

Place : Visakhapatnam

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr.R.VaraPrasad (PhD), Assistant Professor** , Department of Mechanical Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement.

We are grateful to **Dr. B.Naga Raju , Professor**, Head of the Department, Mechanical Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa,** for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of Mechanical Engineering, whose suggestions during reviews helped using accomplishment of our project.

We would like to thank **all non teaching staff** of the Department of Mechanical Engineering, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

I.Purna Adhitya Varma(318126520083)
D.Dhruva Teja(318126520075)
K.Chandu Reddy(318126520071)
M.Avinash(318126520099)
Y.Hemanth(318126520115)

ABSTRACT

Python is a high-sophisticated, general- purposefulness, object-oriented, enhanced programming language that includes several general features like clean, easy and simple language, indicative language, dynamically typed, automatic memory management and interpreted and very best tool when comparing with MATLAB because of various rich libraries. In various fields of science, computational work and numerical calculations forms a bridge between theory and experimentation which leads to developing automation and simulation. Developing of automation or simulation has created several benefits like quality, high production rate, efficient use of materials, increased safety and decreases industry lead time. Therefore, the present work have been investigated to develop Python code for automation in structural and Modal analysis of any beam structure . In order to develop python code Continuous beam structures are to be considered and the data was taken from introduction to finite elements by Chandrapatla. Fully working python code was developed using jupyter notebook and python 3.9.10 and investigated deflections,slope,shear force,bending moment,element stresses, support reactions, free natural frequencies and mode shapes. These results are compared using ANSYS APDL R21. Therefore, using this developed python code, many researchers can do automation for analysis of any beam at rocket speed.

Keywords — ANSYS APDL, MATLAB, Automation, Jupyter Notebook, Python, Beams,Modal analysis.

LIST OF CONTENTS

List of figures……………………………………………….. 1-4

List of tables…………………………………………………. 5-6

List of Abbreviations……………………………………….. 7

CHAPTER 1 : INTRODUCTION…………………… 8

1.1 What is Structural analysis?……………………. 9

1.2 What is Modal analysis?………………………… 9

1.3 Introduction to python………………………….. 10

1.3.1 Python libraries…………………………………………… 12

1.3.2 Python libraries used in code……………………………… 12

1.3.3 Anaconda………………………………………………….. 13

1.3.4 Jupyter Notebook…………………………………………. 14

1.4 Introduction to Finite Element Analysis……….. 14

1.5 Introduction to beams……………………………. 15

1.5.1 Types of beams…………………………………………... 15

1.5.2 Types of loads acting on beams………………………….. 22

1.6 Introduction to ANSYS……………………………. 23

1.7 Problem Statement…………………………………. 24

CHAPTER 2 : LITERATURE SURVEY………….. 25

CHAPTER 3 : NUMERICAL ANALYSIS OF A CONTINUOUS BEAM ………….. 29

3.0 Methodology……………………………………… 30

3.1 Step-wise procedure for analyzing a beam in ANSYS APDL…………………………………….. 31

3.1.1 Preprocessor…………………………………………….. 32

3.1.2 Solution…………………………………………………. 41

3.1.3 Postprocessor……………………………………………. 42

3.2 Python code for analyzing the beams……………. 51

3.2.1 Input given for Python code……………………………….. 72

3.2.2 Output from Python code………………………………….. 72

CHAPTER 4 : EXPERIMENTAL RESULTS…….. 77

4.1 Structural analysis results using Python……….. 78

4.2 Modal analysis results from Python…………… 80

CHAPTER 5 : DISCUSSION OF RESULTS……… 89

5.1 Comparing ANSYS APDL results with Python results……………………………………. 90

CHAPTER 6 : CONCLUSION…………………….. 94

CHAPTER 7 : REFERENCES…………………….. 96

LIST OF FIGURES

Fig.1.1 Simply supported beam	15
Fig.1.2 Continuous beam	16
Fig.1.3 Over hanging beam	16
Fig.1.4 Cantilever beam	16
Fig.1.5 Fixed beam	17
Fig.1.6 I-Section beam	17
Fig.1.7 Rectangular beam	18
Fig.1.8 T-Section beam	18
Fig.1.9 L-Section beam	19
Fig.1.10 Straight beam	19
Fig.1.11 Curved beam	20
Fig.1.12 Tapered beam	20
Fig.1.13 Statically determinate beam	21
Fig.1.14 Statically indeterminate beam	21
Fig.1.15 Point load or Concentrated load	22

Fig.1.16 Uniformly distributed load	22
Fig.1.17 Uniformly varying load	23
Fig.3.1 Example beam for analysis in ANSYS APDL	31
Fig.3.2 Preprocessor step1	32
Fig.3.3 Preprocessor step2	33
Fig.3.4 Preprocessor step3	34
Fig.3.5 Preprocessor step4	34
Fig.3.6 Preprocessor step5	35
Fig.3.7 Preprocessor step6	36
Fig.3.8 Preprocessor step7	37
Fig.3.9 Preprocessor step8	38
Fig.3.10 Preprocessor Step9	39
Fig.3.11 Preprocessor step10	40
Fig.3.12 Preprocessor step11	41
Fig.3.13 Solution step	42

Fig.3.14 Postprocessor step1	43
Fig.3.15 Postprocessor step2	44
Fig.3.16 Shear force diagram from ANSYS APDL	45
Fig.3.17 Bending moment diagram from ANSYS APDL	46
Fig.3.18 Deformed shape of beam from ANSYS APDL	46
Fig.3.19 Nodal deflection diagram from ANSYS APDL	47
Fig.3.20 Nodal slope diagram from ANSYS APDL	47
Fig.3.21 Von Mises stress diagram from ANSY APDL	48
Fig.3.22 Beam to be analyzed using Python code	71
Fig.4.1 Graph plotted between Length of the beam-Nodal Slopes from Python results	79
Fig.4.2 Graph plotted between Length of the beam-Nodal deflections from Python results	79
Fig.4.3 Mode Shape 1 for corresponding frequency 1.	82
Fig.4.4 Mode Shape 2 for corresponding frequency 2.	82
Fig.4.5 Mode Shape 3 for corresponding frequency 3.	83

Fig.4.6 Mode Shape 4 for corresponding frequency 4.	83
Fig.4.7 Mode Shape 5 for corresponding frequency 5.	84
Fig.4.8 Mode Shape 6 for corresponding frequency 6.	84
Fig.4.9 Mode Shape 7 for corresponding frequency 7.	85
Fig.4.10 Mode Shape 8 for corresponding frequency 8.	85
Fig.4.11 Mode Shape 9 for corresponding frequency 9.	86
Fig.4.12 Mode Shape 10 for corresponding frequency 10.	86
Fig.4.13 Mode Shape 11 for corresponding frequency 11.	87
Fig.4.14 Mode Shape 12 for corresponding frequency 12.	87
Fig.4.15 Mode Shape 13 for corresponding frequency 13.	88
Fig.4.16 Mode Shape 14 for corresponding frequency 14.	88

LIST OF TABLES

Table.3.1 Element connectivity table for example beam	31
Table.3.2 Nodal deflections from ANSYS APDL	44
Table.3.3 Nodal slopes from ANSYS APDL	49
Table.3.4 Reaction forces from ANSYS APDL	49
Table.3.5 Reaction moment from ANSYS APDL	49
Table 3.6 Natural frequencies from ANSYS APDL	50
Table 3.7 Element connectivity table for beam analyzed using Python	71
Table.4.1 Nodal deflections and nodal displacements from Python code	78
Table.4.2 Reaction forces from Python code	78
Table.4.3 Reaction moment from Python code	78
Table.4.4 Natural frequency from Python code	80
Table.4.5 Eigen values from Python code	81
Table.5.1 Comparing results of Nodal deflections from Python code to ANSYS APDL	90
Table.5.2 Comparing results of Nodal slopes from Python code to ANSYS APDL	91

Table.5.3 Comparing results of Reaction forces from Python code to ANSYS APDL	92
Table.5.4 Comparing result of Reaction moment from Python code to ANSYS APDL	92
Table.5.5 Comparing Natural frequencies from Python code to ANSYS APDL	93

LIST OF ABBREVIATIONS

S.NO	ABBRIVATION	FULL FORM
1.	ANSYS	Analysis of Systems
2.	APDL	ANSYS Parametric Design Language
3.	MATLAB	Matrix Laboratory
4.	CFD	Computational Fluid Dynamics
5.	FEA	Finite Element Analysis
6.	KP	Key Points
7.	FEM	Finite Element Method
8.	SciPy	Scientific Python
9.	NumPy	Numerical Python

CHAPTER-1
INTRODUCTION

1.1 What is Structural Analysis?

A structure is a system of interconnected members used to support external loads.

Structural analysis is the prediction of the response of structures to specified arbitrary external loads. During the preliminary structural design stage, a structure's potential external load is estimated, and the size of the structure's interconnected members are determined based on the estimated loads.

Structural analysis establishes the relationship between a structural member's expected external load and the structure's corresponding developed internal stresses and displacements that occur within the member when in service.

This is necessary to ensure that the structural members satisfy the safety and the serviceability requirements of the local building code and specifications of the area where the structure is located.

1.2 What is Modal Analysis?

Modal analysis is an indispensable tool in understanding the structural dynamics of objects - how structures and objects vibrate and how resistant they are to applied forces. The modal analysis allows machines and structures to be tested, optimized, and validated.

Natural resonance frequencies of the objects and damping parameters can be calculated, and mode shapes can be visualized on an animated geometry of the measured objects.

1.3 Introduction to python

The common myth is that Mechanical Engineering is not connected with any coding type of platform. Generally, Mechanical Engineers tend to have an aversion to computer programming and end up not understanding the opportunities that they miss out on. As we move into a future that is tied up intrinsically with electric cars, autonomous transportation, and automation, the next era of mechanical, aerospace & automotive engineers need to understand how they can integrate mechanical engineering concepts with a computer language in order to simulate concepts or automate them at a faster pace. Python for that matter is an extremely easy and efficient programming language. It can solve complex problems in a matter of seconds. Even if we are a mechanical/automobile engineer, Python can still be handy for us on many occasions.

Areas, where Python is used in the mechanical engineering industry, includes but are not limited to,

Numerical analysis:

The most popular application of python is to perform numerical analysis. When problems linear equations and ODE/PDE are involved, it would take a long time to solve the problems analytically. In terms of mechanical engineering, there are usually boundary conditions present which make it twice harder to solve numerical analysis problems.

Let us assume we are trying to find the pressure difference across a pipe when there is a liquid flowing through it. Not only will the problem take forever to solve, but it will also be extremely hard to obtain accurate values or plot the differences in a graph.
With programming languages, we can solve such problems in a matter of seconds and obtain graphical simulations at the same instant.

Learning numerical analysis and coding opens up a plethora of opportunities in areas like manufacturing, automotive, energy, and even mechanical jobs in software companies (like thermal engineers). Software companies like Google and Facebook hire Mechanical/Thermal engineers to ensure efficient and safe thermal management of their database and cluster computers in their respective companies. These engineers use programming languages like MATLAB/Python to write scripts and then import them to CFD software to test numerous designs. It is common knowledge that Google has the following engineering motto “Python where we can, C++ where we must” because Python is less complex to use than C++.

Thermodynamics:

Python can be used to solve classical thermodynamics problems. Whether our problem involves chemical kinetics or fluid dynamics, we can write a code to solve the problem and save our time. In the real world, industries do not pay attention to how we solve our problems or complete our tasks. We can spend 60% of your time solving mathematical/ thermodynamics problems and rush through the other 40% of the task or you can spend 20% of your time on these problems by solving them in Python and focusing on the real troubles at hand. The only aspect of our outcome that matters is our efficiency. Python has a huge library and a simple syntax that can help us to solve complex problems easily.

CFD:

In the field of computational fluid dynamics, Python has a massive application. In order to simulate problems in CFD software, we will be required to write our scripts in programming languages like MATLAB/Python.

Python is also used in other areas of mechanical engineering like vibrations and dynamic motion, simulation and modelling engineering etc. Mechanical and automobile industries use python to automate tasks. Even when the script is written in another programming language, it is rewritten in Python before automation since it is the most common language and hence the interface between industries and codes.

A look at the careers page of companies like Tesla, Mercedes Benz and Boeing, etc. reveal that they employ and prefer mechanical engineers who can code.

1.3.1 Python Libraries

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs.More than 200 core modules sit at the heart of the standard library.

1.3.2 Python Libraries used in code

1.Matplotlib

Matplotlib helps with data analyzing, and is a numerical plotting library.

2.SciPy

One of the libraries we have been talking so much about. It has a number of user-friendly and efficient numerical routines.

These include routines for optimization and numerical integration.

3.NumPy

It is a python library used for working with arrays.It has functions for working in domain of linear algebra,fourier transform ,and matrices.

4.Math

The python Math library provides us access to some common math functions and constants in python,which we can use throughout our code for more complex mathematical computations.

1.3.3 Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012.

Package versions in Anaconda are managed by the package management system conda.

Anaconda distribution comes with over 250 packages automatically installed, and over 7,500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator,as a graphical alternative to the command-line interface (CLI).

Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

●JupyterLab

●Jupyter Notebook

●QtConsole

●Spyder

1.3.4 Jupyter Notebook

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

The best method of installing the Jupyter Notebooks is by the installation of the Anaconda package. The Jupyter Notebook and the Jupyter Lab comes pre-installed in the Anaconda package.

1.4 Introduction to Finite Element Analysis

Finite Element Analysis (FEA) is method used for the evaluation of structures, providing an accurate prediction of components response subjected to various structural loads.

It is a computerized method for predicting how a product reacts to real world forces, vibration, heat, fluid flow and other physical effects.

FEA works by breaking down a real object into a mesh of large number of finite elements, that combine to create shape of structure that is being assessed. Each of these small elements are subjected to calculations, with these mesh refinements combining to produce the final result of the whole structure.

1.5 Introduction to Beams

Beam:

Beam is a horizontal structure member used to carry vertical load, shear load and sometime horizontal load. It is the major component of building structures. It mainly use in construction of bridges, trusses, and other structures which carry vertical load.

1.5.1 Types of Beams

1. According to end support:

Simply supported beams:

Simply supported beam is supported at both end. One end of the beam is supported by hinge support and other one by roller support. This support allow to horizontal movement of beam. It beam type undergoes both shear stress and bending moment.

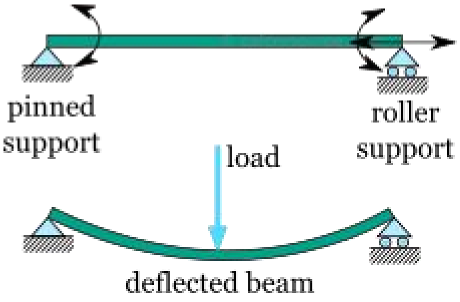


Fig.1.1 Simply supported beam

Continuous beams:

This beam is similar to simply supported beam except more than two support are used on it. One end of it is supported by hinged support and other one is roller support. One or more supports are use between these beams. It is used in long concrete bridges where length of bridge is too large.

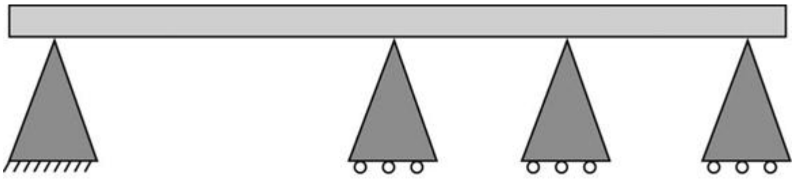


Fig.1.2 Continuous beam

Over hanging beams:

Overhanging beam is combination of simply supported beam and cantilever beam. One or both of end overhang of this beam. This beam is supported by roller support between two ends.

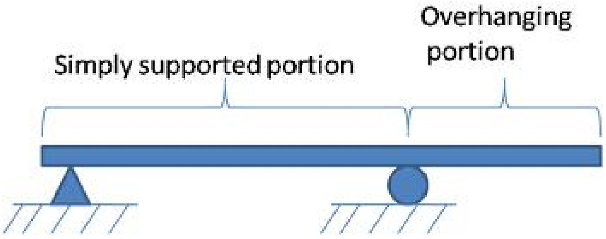


Fig.1.3 Over hanging beam

`Cantilever beams:

Cantilever beams a structure member of which one end is fixed and other is free. This is one of the famous type of beam use in trusses, bridges and other structure member.

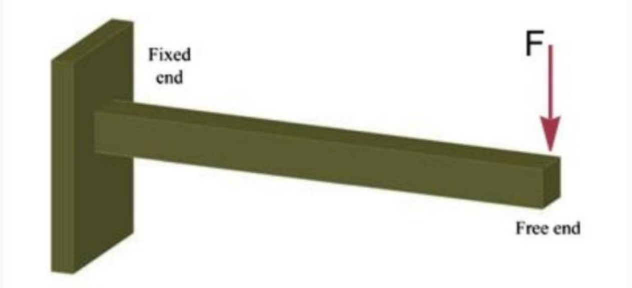


Fig.1.4 Cantilever beam

Fixed beams:

This beam is fixed from both ends. It does not allow vertical movement and rotation of the beam. It is only under shear stress and no moment produces in this beams.

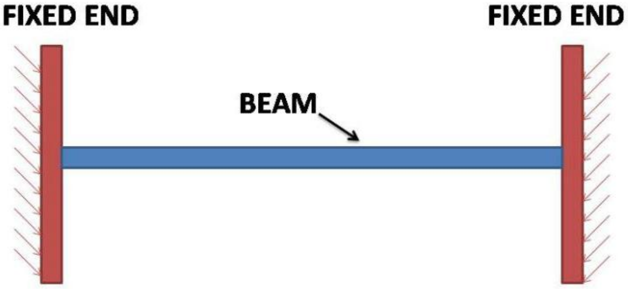


Fig.1.5 Fixed beam

2. According to cross section:

I-section beam:

Has high resistance to bending.

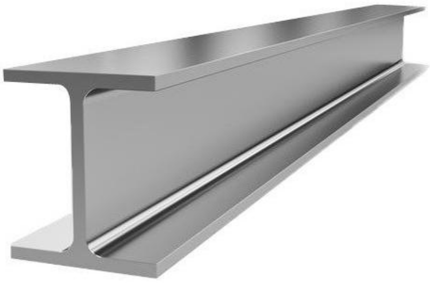


Fig.1.6 I-Section beam

Rectangular beam:

This type of beam is widely used in the construction of reinforced concrete buildings and other structures.

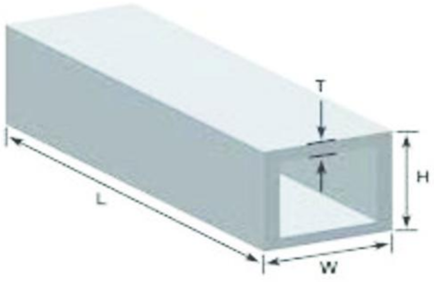


Fig.1.7 Rectangular beam

T-section beam:

Mostly constructed with a reinforced concrete slab.Isolated T-beam is built to increase the compression strength of concrete.

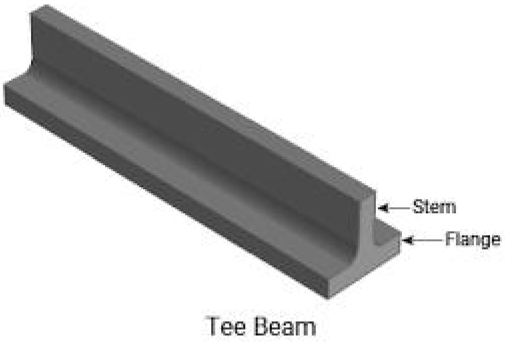


Fig.1.8 T-Section beam

L-section beam:

This type of beam is constructed monolithically with a reinforced concrete slab at the perimeter of the structure.

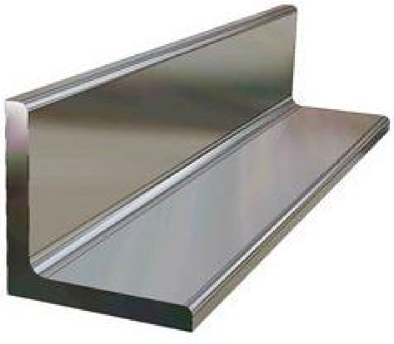


Fig.1.9 L-Section beam

Square, circular, H-shaped, C-shaped, and tubular are some of the other examples of beam cross-sectional shapes constructed from steel.

3. Based on geometry:

Straight beam:

Beam with a straight profile and the majority of beams in structures are straight beams.

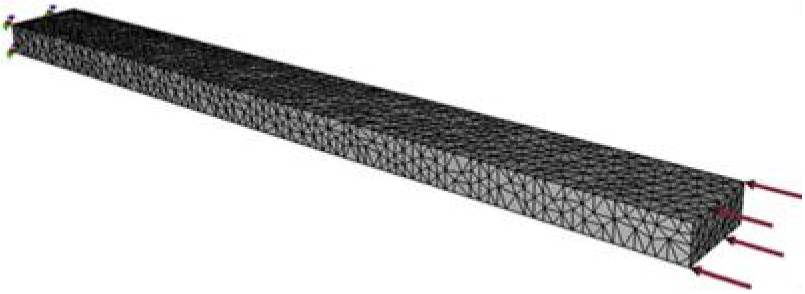


Fig.1.10 Straight beam

Curved beam:

Beam with curved profile, such as in the case of circular buildings.

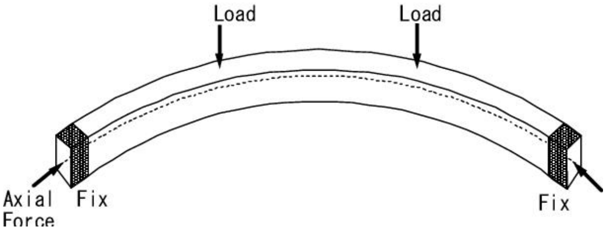


Fig.1.11 Curved beam

Tapered beam:

Beam with tapered cross section.

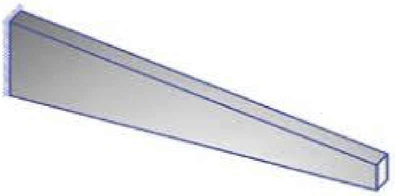
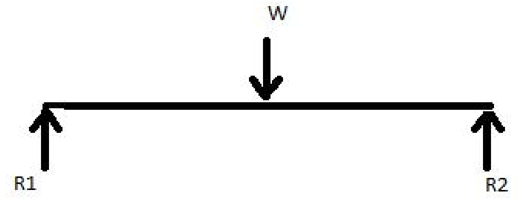


Fig.1.12 Tapered beam

4. Based on equilibrium condition:

Statically determinate beam:

For a statically determinate beam, equilibrium conditions alone can be used to solve reactions. The number of unknown reactions is equal to the number of equations.



Static equations are

1. R1+R2=W where L = span of the beam
2. R2=(W*x)/L x=distance of load W from left support

Number of unknowns (a)=2
Number of static equations (b)=2
As, a=b the beam is determinate beam

Fig.1.13 Statically determinate beam

Statically indeterminate beam:

For a statically indeterminate beam, equilibrium conditions are not enough to solve reactions. So, the analysis of this type of beam is more complicated than that of statically determinate beams.

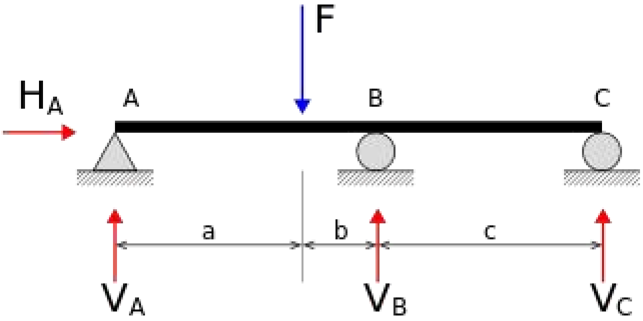


Fig.1.14 Statically indeterminate beam

1.5.2 Types of load acting on beam:

A beam is usually horizontal member and load which will be acting over the beam will be usually vertical loads. There are following types of loads:

1.Point load or concentrated load:

Point load or concentrated load, as name suggest, acts at a point on the beam.

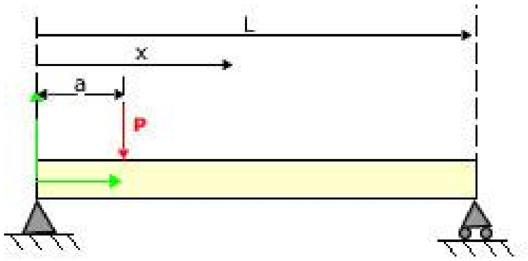


Fig.1.15 Point load or Concentrated load

2.Uniformly distributed load:

Uniformly distributed load is the load which will be distributed over the length of the beam in such a way that rate of loading will be uniform throughout the distribution length of the beam.

Uniformly distributed load is also expressed as U.D.L and with value as w N/m.

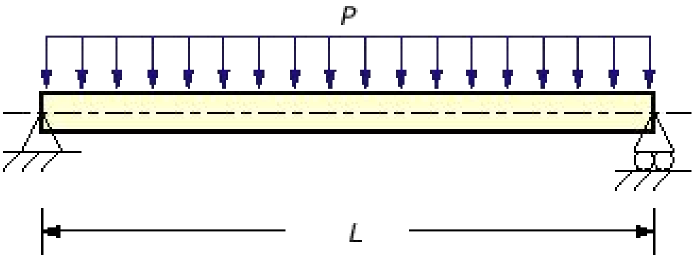


Fig.1.16 Uniformly distributed load

3. Uniformly varying load:

Uniformly varying load is the load which will be distributed over the length of the beam in such a way that rate of loading will not be uniform but also vary from point to point throughout the distribution length of the beam.Uniformly varying load is also termed as triangular load.

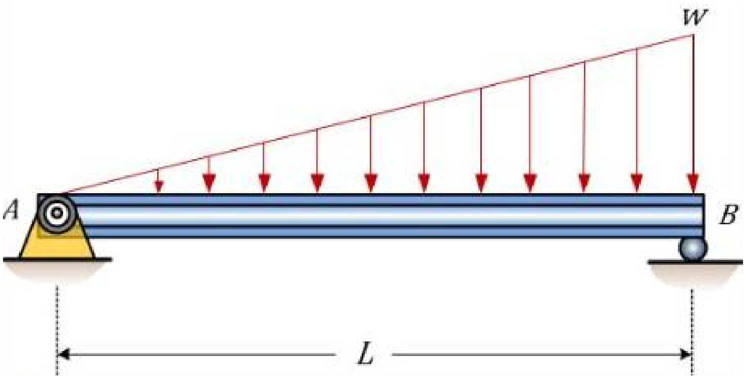


Fig.1.17 Uniformly varying load

1.6 Introduction to ANSYS

ANSYS(Analysis of Systems) mechanical APDL(ANSYS Parametric Design Language) has become a powerful tool for finite element analysis to solve enormous range of mechanical engineering applications. It has capabilities to solve problems ranging from simple, linear, static analysis to a complex, nonlinear, transient dynamic analysis, in the fields of stress analysis, fluid and heat transfer, etc.

1.7 PROBLEM STATEMENT

1.Finding Shear stress and Bending moment and do Structural analysis of a continuous beam using ANSYS(Analysis of Systems) software.

2.Developing a python program that gives the structural analysis of a continuous beam as output.

3.Verifying and compare the results obtained using ANSYS software with the results obtained using python program.

4.By comparing both the results, a final python program is developed that gives accurate results.

5.Extend Python code for modal analysis.

This python code can be used for analysis of any type of beam.

CHAPTER-2
LITERATURE SURVEY

Literature Survey

Dorival et.al.,[1] investigated that the Generalized Finite Element Method (GFEM) is a numerical method based on the Finite Element Method (FEM), presenting as its main feature the possibility of improving the solution by means of local enrichment functions. In spite of its advantages, the method demands a complex data structure, which can be especially benefited by the Object-Oriented Programming (OOP). Even though the OOP for the traditional FEM has been extensively described in the technical literature, specific design issues related to the GFEM are yet little discussed and not clearly defined. In the present article it is described an Object-Oriented (OO) class design for the GFEM, aiming to achieve a computational code that presents a flexible class structure, circumventing the difficulties associated to the method characteristics. The proposed design is evaluated by means of some numerical examples, computed using a code implemented in Python programming language.

Aleves et.al.,[2] The objective computing project was created with the intention of writing a fully interactive-adaptive finite element program using the object oriented programming philosophy. In a first phase of the project, a graphical environment was developed to simplify and generalize the interactive programming concept. The resulting interface library gives the programmer easy access to graphical user interface tools such as windows, menus and dialogs. In order to improve programmer efficiency, the same interface library is being implemented to run under various existing toolboxes such as Macintosh, MS-Windows, OSF/Motif and others. During the second phase of the project, an innovative finite element data structure is developed which will be used as a finite element research platform.

Bordas et.al.,[3] This paper presents and exercises a general structure for an object-oriented-enriched finite element code. The programming environment provides a robust tool for extended finite element (XFEM) computations and a modular and extensible system. The programme structure has been designed to meet all natural requirements for modularity, extensibility, and robustness. To facilitate mesh–geometry interactions with hundreds of enrichment items, a mesh generator and mesh database are included. The salient features of the programme are: flexibility in the integration schemes (subtriangles, subquadrilaterals, independent near-tip, and discontinuous quadrature rules); domain integral methods for homogeneous and bi-material interface cracks arbitrarily oriented with respect to the mesh; geometry is described and updated by level sets, vector level sets or a standard method; standard and enriched approximations are independent; enrichment detection schemes: topological, geometrical, narrow-band, etc.; multi-material problem with an arbitrary number of interfaces and slip-interfaces; non-linear material models such as J2 plasticity with linear, isotropic and kinematic hardening. To illustrate the possible applications of our paradigm, we present 2D linear elastic fracture mechanics for hundreds of cracks with local near-tip refinement, and crack propagation in two dimensions as well as complex 3D industrial problems.

Duarte et.al.,[4] A new methodology to build discrete models of boundary-value problems is presented. The h-pcloud method is applicable to arbitrary domains and employs only a scattered set of nodes to build approximate solutions to BVPs. This new method uses radial basis functions of varying size of supports and with polynomial reproducing properties of arbitrary order. The approximating properties of the h-p cloud functions are investigated in this article and a several theorems concerning these properties are presented. Moving least squares interpolants are used to build a partition of unity on the domain of interest. These functions are then used to construct, at a very low cost, trial and test functions for Galerkin approximations. The method exhibits a very high rate of convergence and has a greater -exibility than traditional h-p finite element methods. Several numerical experiments in I-D and 2-D are also presented.

Tom et.al.,[5] Partial differential equations (PDEs)—such as the Navier–Stokes equations in fluid mechanics, the Maxwell equations in electromagnetism, and the Schrödinger equation in quantum mechanics—are the basic building blocks of modern physics and engineering. The finite element method (FEM) is a flexible computational technique for the discretization and solution of PDEs, especially in the case of complex spatial domains. Conceptually, the FEM transforms a time-independent (or temporally discretized) PDE into a system of linear equations Ax = b. scikit-fem is a lightweight Python library for the creation, or assembly, of the finite element matrix A and vector b. The user loads a computational mesh, picks suitable basis functions, and provides the PDE's weak formulation (Logg, Mardal, Wells, & others, 2012). This results in sparse matrices and vectors compatible with the SciPy (Virtanen et al., 2020) ecosystem.

Cimrman et.al.,[6] SfePy (simple finite elements in Python) is a software for solving various kinds of problems described by partial differential equations in one, two, or three spatial dimensions by the finite element method. Its source code is mostly (85%) Python and relies on fast vectorized operations provided by the NumPy package. For a particular problem, two interfaces can be used: a declarative application programming interface (API), where problem description/definition files (Python modules) are used to define a calculation, and an imperative API, that can be used for interactive commands, or in scripts and libraries. After outlining the SfePy package development, the paper introduces its implementation, structure, and general features. The components for defining a partial differential equation are described using an example of a simple heat conduction problem. Specifically, the declarative API of SfePy is presented in the example. To illustrate one of SfePy's main assets, the framework for implementing complex multiscale models based on the theory of homogenization, an example of a two-scale piezoelastic model is presented, showing both the mathematical description of the problem and the corresponding code.

Hunter et.al.,[7] Matplotlib is a 2D graphics package for Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems.

CHAPTER-3
NUMERICAL ANALYSIS OF A CONTINUOUS BEAM

3.0 METHODOLOGY

The present work consists of two parts. In the first part a fully working python code was developed using finite element procedure in the jupyter notebook(Jupyter Notebook (open source code), which began as the iPython Notebook project, is development environment for writing and executing Python code. Jupyter Notebook is often used for exploratory data analysis and visualization.

Project Jupyter is the top-level project name for all of the sub projects under development, which includes Jupyter Notebook. Jupyter Notebooks can also run code for other programming languages such as Julia and R.) which is accompanied by python 3.9.10.in the second part, the results obtained using python are compared with the results obtained from ANSYS APDL 2021 R21 software. The python developed in this current work is applicable for any type of beam structure of isotropic material and different loading and support conditions. This python code is used to investigate both static and vibrational analysis (A technique used to determine a structure's vibration characteristics: – Natural frequencies – Mode shapes – Mode participation factors (how much a given mode participates in a given direction). It is most fundamental of all the dynamic analysis types. Modal analysis allows the design to avoid resonant vibrations or to vibrate at a specified frequency (speakers, for example). Gives engineers an idea of how the design will respond to different types of dynamic loads. Helps in calculating solution controls (time steps, etc.) for other dynamic analysis.

Because a structure's vibration characteristics determine how it responds to any type of dynamic load, always perform a modal analysis first before trying any other dynamic analysis) of any beam structure. To check whether the code developed was giving exact results, the present work considered any type of beam problem and the data was taken from the textbook of introduction to finite elements by Chandrapatla and Daryl L. Logan.

3.1 Step-wise procedure for Numerical analysis (Structural and Modal) of a beam in ANSYS APDL

The following beam has been taken as an example for analyzing and later the results will be compared.

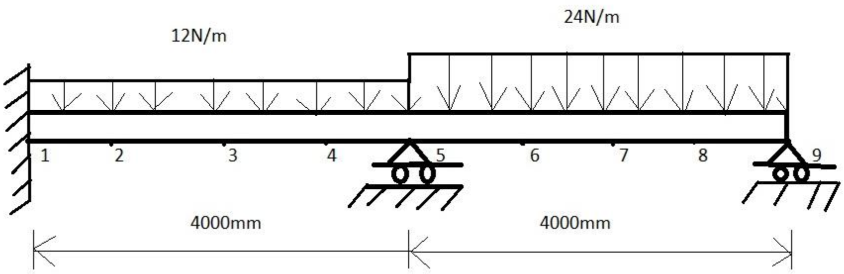


Fig.3.1 Example beam for analysis in ANSYS APDL

ELEMENT CONNECTIVITY TABLE

ELEMENT NO.	NODE 1	NODE 2
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	7	8
8	8	9

Table.3.1 Element connectivity table for example beam

3.1.1 Preprocessor

To begin the analysis, a preference needs to be set. Preferences allow you to apply filtering to the menu choices; ANSYS will remove or gray out functions that are not needed. A structural analysis, for example,will not need all the options available for a thermal, electromagnetic, or fluid dynamic analysis.

> Main Menu > Preferences

Place a check mark next to the Structural box.

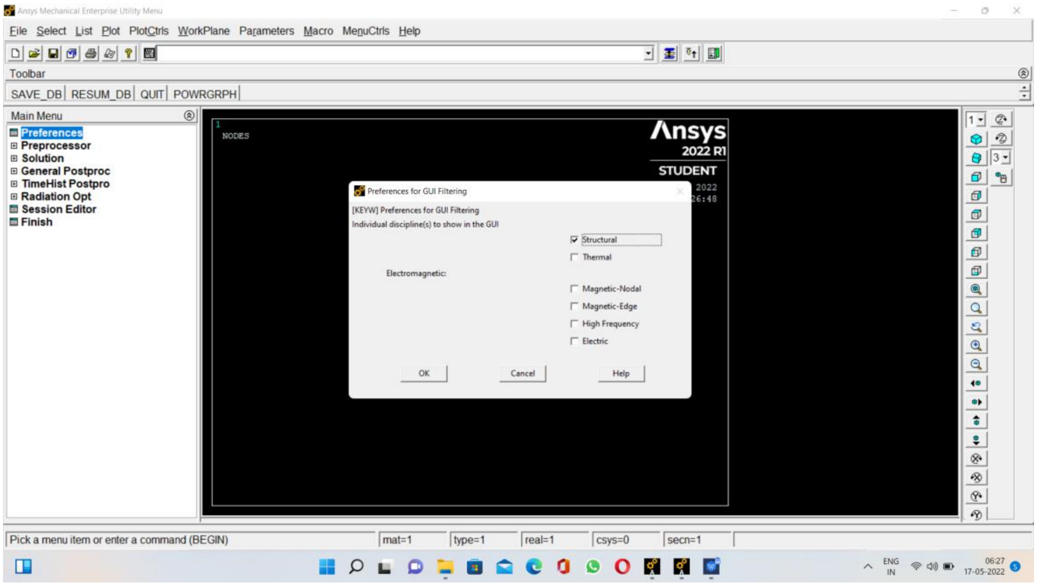
> OK

Look at the ANSYS Main Menu.

Click once on the “+” sign next to Preprocessor.

> Main Menu > Preprocessor

The Preprocessor options currently available are displayed in the expansion of the Main Menu tree as shown to the right. The most important preprocessing functions are defining the element type, defining real constraints and material properties, and modeling and meshing the geometry.



Preferences for GUI Filtering

[KEYW] Preferences for GUI Filtering

Individual discipline(s) to show in the GUI

☑ Structural

□ Thermal

Electromagnetic:

□ Magnetic-Nodal

□ Magnetic-Edge

□ High Frequency

□ Electric

Pick a menu item or enter a command (BEGIN) | mat=1 | type=1 | real=1 | csys=0 | secn=1

Fig.3.2 Preprocessor step1

The ANSYS Main Menu is designed in such a way that you should start at the beginning and work towards the bottom of the menu in preparing, solving, and analyzing your model.

Note: This procedure will be shown throughout the tutorial.Select the "+" next to Element Type or click on Element Type. The extension of the menu is shown to the right.

> Element Type

Select Add/Edit/Delete and the Element Type window appears.

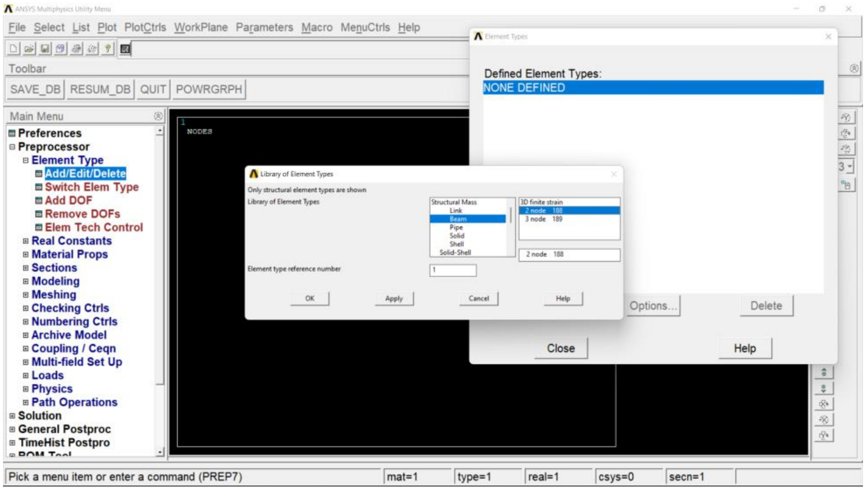
Select add and the Library of Element Types window appears.

> Add/Edit/Delete > Add

In this window, select the types of elements to be defined and used for the problem. For a pictorial description of what each element can be used for, click on the Help button.

For this model 2D Elastic Beam elements will be used.

> Beam > 2 node 188> OK



Defined Element Types:
NONE DEFINED

Only structural element types are shown

Library of Element Types: Structural Mass, Link, Beam, Pipe, Solid, Shell, Solid-Shell — Beam; 3D finite strain, 2 node 188, 3 node 189 — 2 node 188

Element type reference number: 1

Pick a menu item or enter a command (PREP7) mat=1 type=1 real=1 csys=0 secn=1

Fig.3.3 Preprocessor step2

The material properties for the Beam element need to be defined.

> Preprocessor > Material Props > Material Models

The Define Material Models Behavior window should now be open. We will use isotropic, linearly, elastic, structural properties.

Select the following from the Material Models Available window:

> Structural > Linear > Elastic > Isotropic

The window titled Linear Isotropic Properties for Material Number 1 now appears.

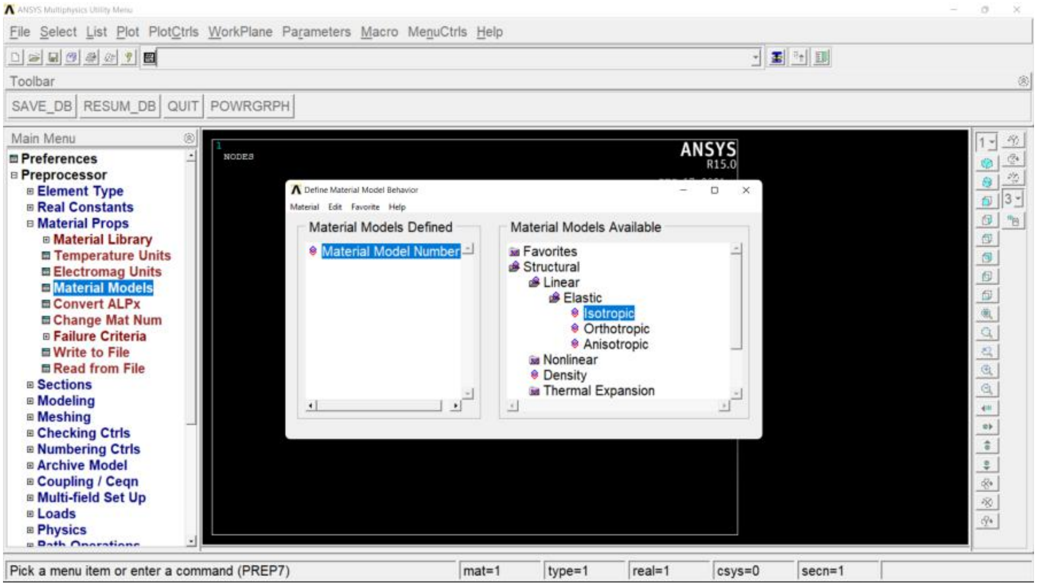


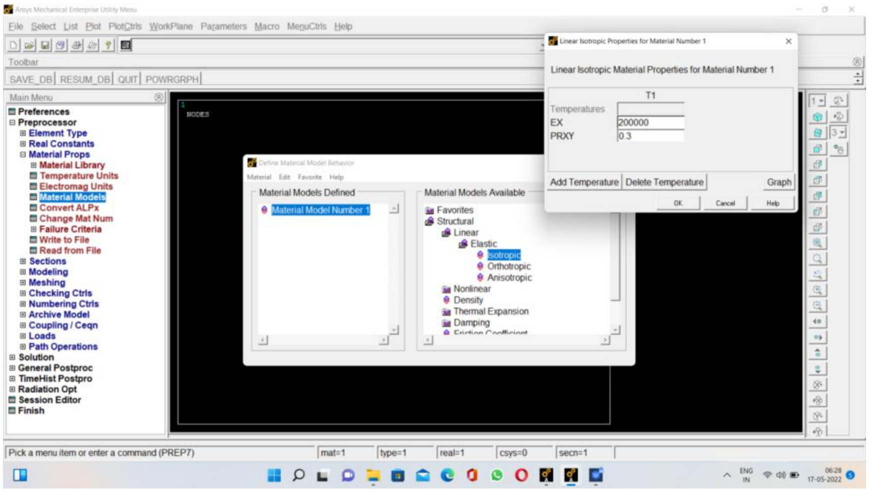
Fig.3.4 Preprocessor step3

Enter 2e5 for EX (Young's Modulus) and 0.3 for PRXY (Poission’s Ratio).

> OK

Close the Define Material Model Behavior window.

> Material > Exit


Linear Isotropic Material Properties for Material Number 1

	T1
Temperatures	
EX	200000
PRXY	0.3

Fig.3.5 Preprocessor step4

The next step is to define the keypoints(KP's) that will help you build the rest of your model:

> Preprocessing > Modeling > Create > Keypoints > In Active CS

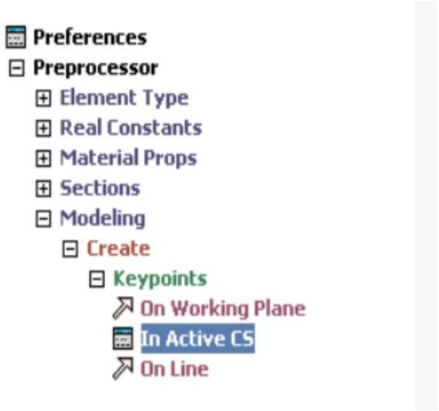


Fig.3.6 Preprocessor step5

The Create Keypoints in Active CS window will now appear. Here the KP's will be given numbers and their respective (XYZ)coordinates. Enter the KP numbers and coordinates that will correctly define the beam.

Select Apply after each KP has been defined.

KP # 1:	X=0, Y=0, Z=0
KP # 2:	X=1000, Y=0, Z=0
KP # 3:	X=2000, Y=0, Z=0
KP # 4:	X=3000, Y=0, Z=0
KP # 5:	X=4000, Y=0, Z=0
KP # 6:	X=5000, Y=0, Z=0
KP # 7:	X=6000, Y=0, Z=0
KP # 8:	X=7000, Y=0, Z=0
KP # 9:	X=8000, Y=0, Z=0

Select OK when complete. In case you make a mistake in creating the keypoints, select:

> Preprocessing > Modeling > Delete > Keypoints

Select the incorrect KP’s and select OK.

The next step is to create lines between theKP’s.

> Preprocessing > Modeling > Create > Lines > Straight Lines

The Create Straight Lines window should appear. You will create 8 lines. Create line 1between the first two keypoints.The other lines will be created in a similar manner.

Verify that each line only goes between the specified keypoints. When you are done creating the lines click ok in the Create Straight Lines window.

> OK

If you make a mistake, use the followingsteps to delete the lines:

> Preprocessing > Modeling > Delete > Lines

Only You should now have something similar to the image shown :

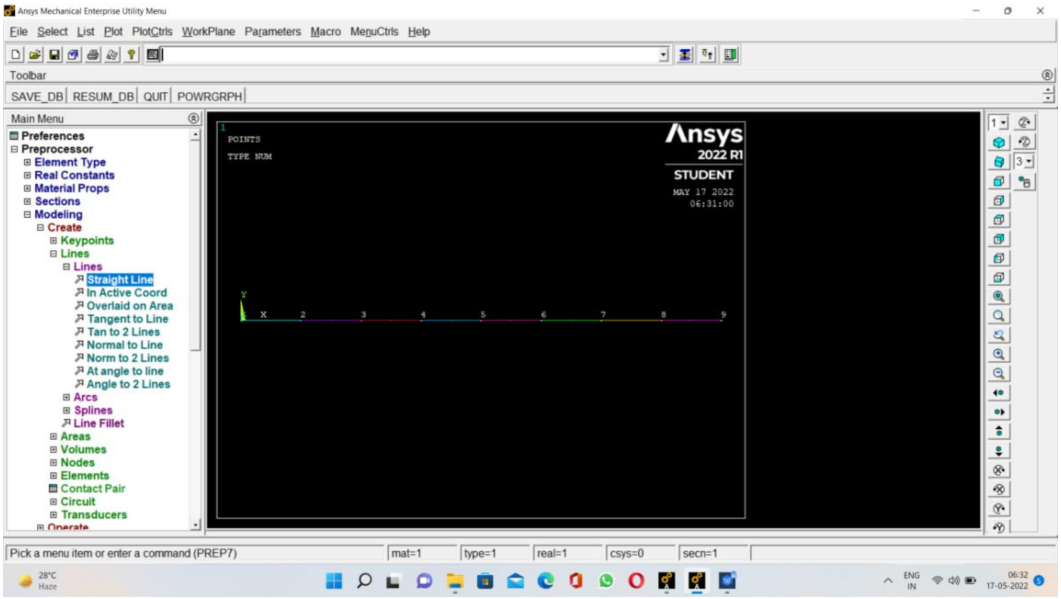


Fig.3.7 Preprocessor step6

Now that the model has been created, it needs to be meshed. Models must be meshed before they can be solved. Models are meshed with elements.First, the element size needs to be specified.

> Preprocessing > Meshing > Size Cntrls > Manual Size > Lines >

All Lines The Element Sizes on All Selected Lines window should appear. From this window,the number of divisions per element can be defined and also the element edge length.

Here we are giving element division as 2.

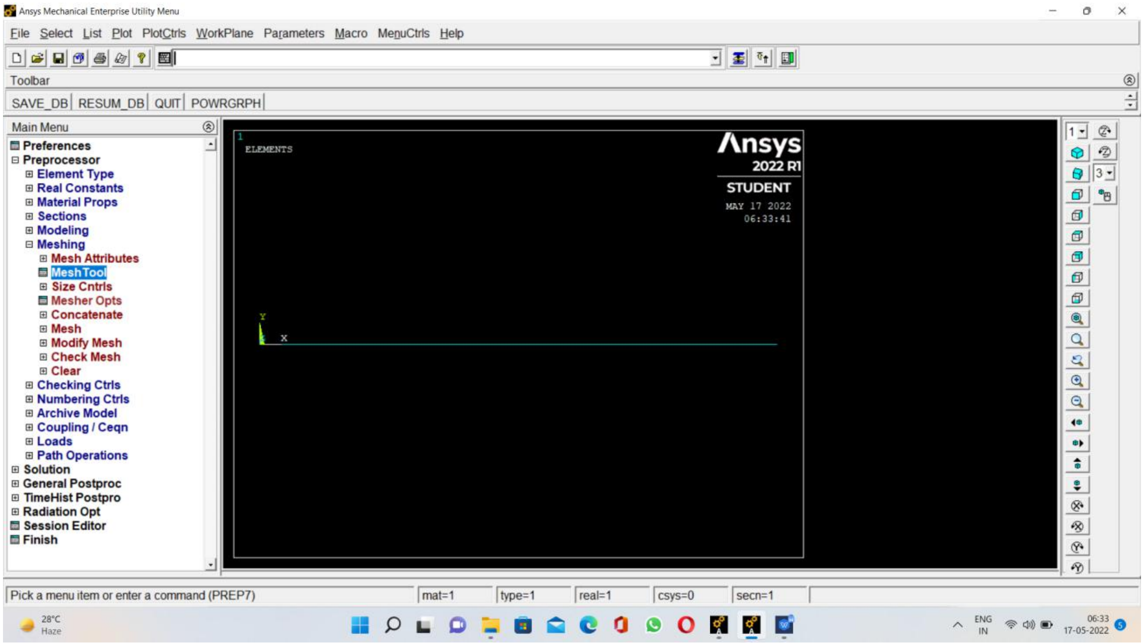
With the mesh parameters complete, the lines representing the beam can now be meshed. Select:

> Preprocessing > Meshing > Mesh > Lines

From the Mesh Lines window select PickAll.

> Pick all

Selecting Pick all will mesh all of the line segments that have been created.The meshed line should appear similar to the one shown . This completes the preprocessing stage.



Pick a menu item or enter a command (PREP7) mat=1 type=1 real=1 csys=0 secn=1

Fig.3.8 Preprocessor step7

However, the current view probably shows just the elements and not the keypoints. You can see both the elements and the keypoints on the screen by selecting:

> Plot > Multiplots

To see just the keypoints;

> Plot > Keypoints

Use the plot menu to view your model in the way that will make it easier to complete each step.

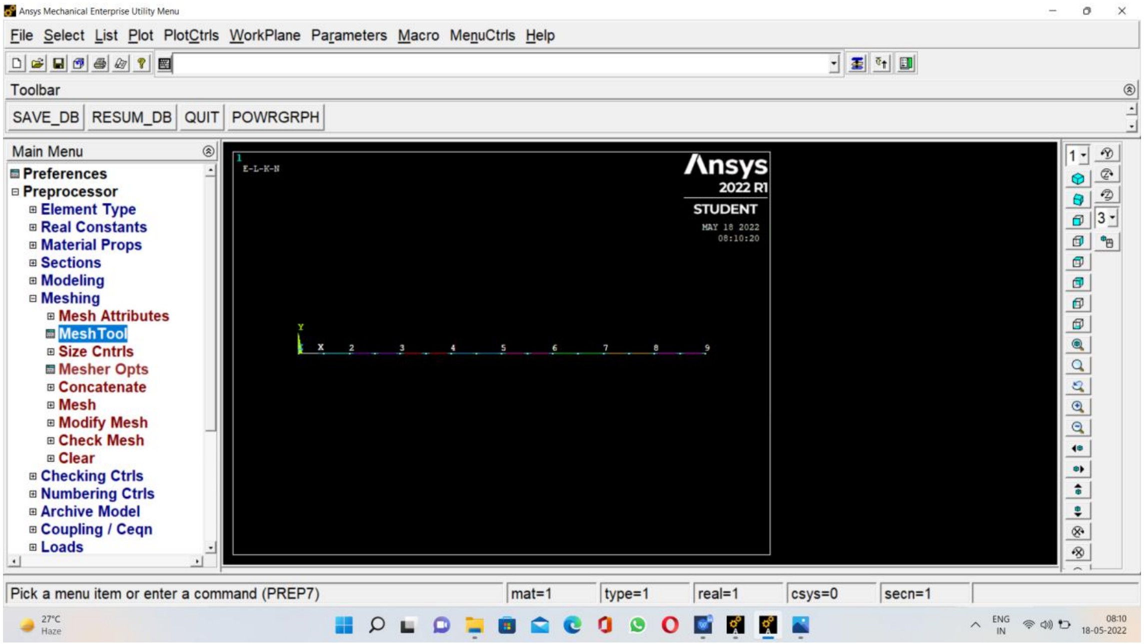


Fig.3.9 Preprocessor step8

The constraints and degrees of freedom should be applied at respective keypoints.

>Preprocessor>Loads>Define Loads>Apply>Structural>Displacement>On Keypoints

Select Keypoint 1 and select All DOF in DOFs to be constrained box.

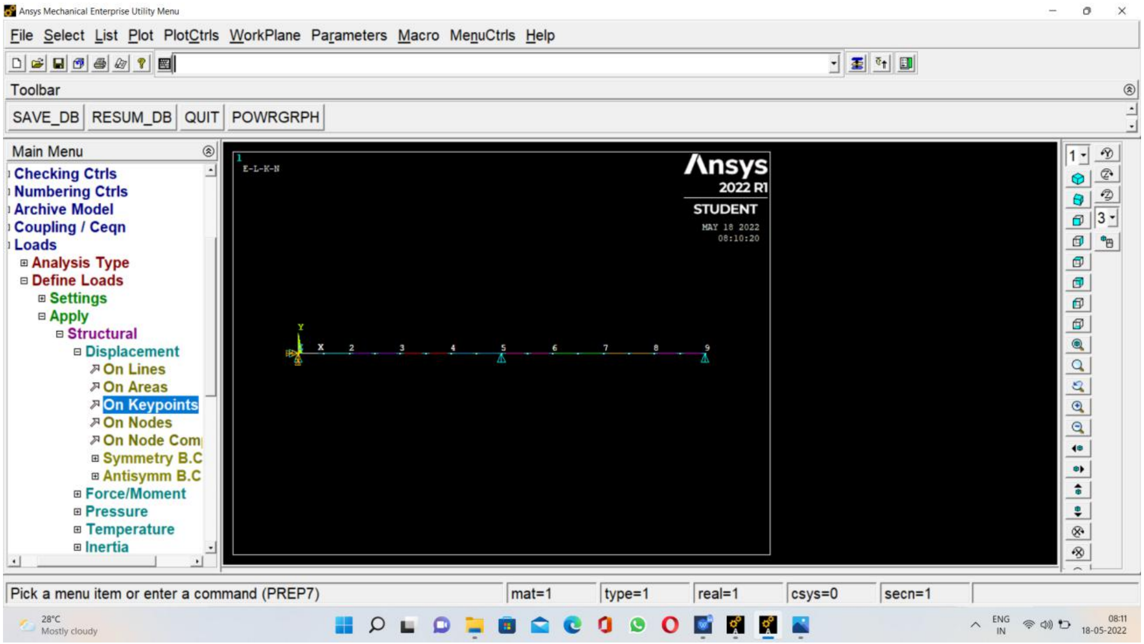
>Apply

Select Keypoint 5 and select UY for roller support.

>Apply

Similarly apply UY as constraints on Keypoint 9 as it is roller supported.

>Ok



Pick a menu item or enter a command (PREP7) | mat=1 | type=1 | real=1 | csys=0 | secn=1

Fig.3.10 Preprocessor Step9

The distributed loads will now be applied to the beam.

> Preprocessor >Loads> Define Loads > Apply> Structural > Pressure > On Beams

Select all the elements between keypoints 1 and 5.

> Apply

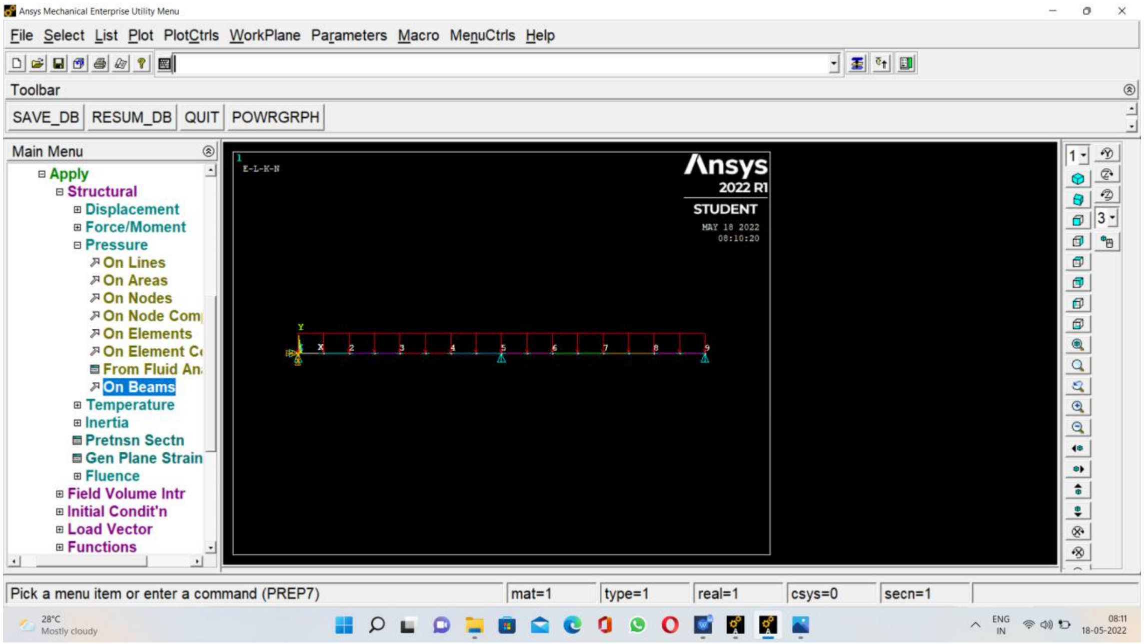
The expanded Apply Pressure on Beams window should appear. From this window the direction of the pressure and its magnitude can be specified. Enter 12 in the Pressure at Node i and 12 in the pressure node at Node j value field which will apply the pressure over the beam from keypoints 1 to 2. A positive entry in this field is defined as a downward pressure.

>Ok

Add the other two distributed load in a similar manner. Use the same commands as shown.For the second distributed load select all of the elements between KP5 and KP9. Set the value at node i and node j to be 24.

> OK

The model is now completed.



Pick a menu item or enter a command (PREP7) mat=1 type=1 real=1 csys=0 secn=1

Fig.3.11 Preprocessor step10

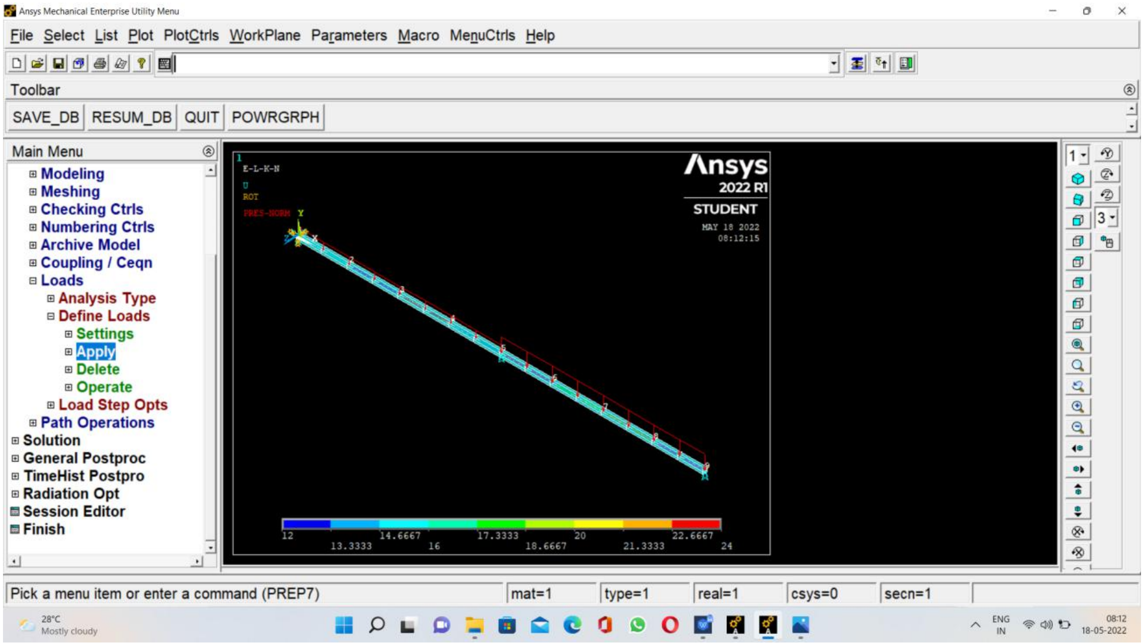
If you wish to view a 3D picture of your model select

> Plot Controls > Style > Size and Shape

The Size and Shape window opens. Click the check box next to Display of Element to turn on the 3D image.

> OK

Now when you rotate your model using CTRL + MB3 , the model should appear to be 3D.



Pick a menu item or enter a command (PREP7)

Fig.3.12 Preprocessor step11

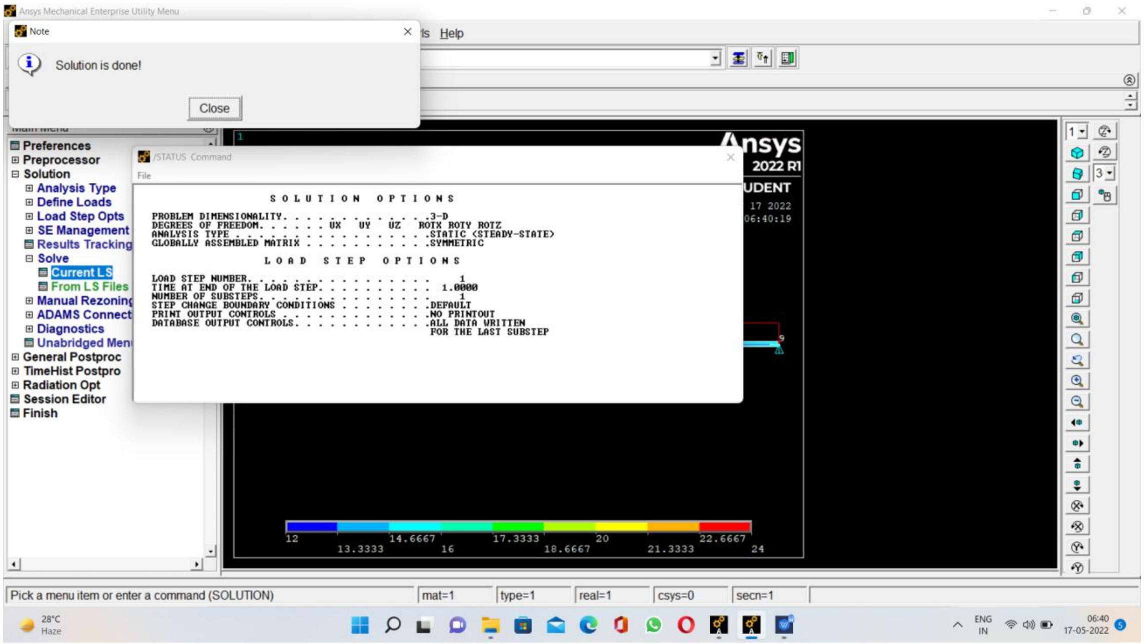
3.1.2 Solution

The next step is to solve the current load step that has been created. Select:

Solution > Solve > Current LS

A note will be appeared as solution is done.

>Close



Solution is done!

```
                    S O L U T I O N   O P T I O N S

   PROBLEM DIMENSIONALITY. . . . . . . . . . . . .3-D
   DEGREES OF FREEDOM. . . . . . UX   UY   UZ   ROTX ROTY ROTZ
   ANALYSIS TYPE . . . . . . . . . . . . . . . . .STATIC (STEADY-STATE)
   GLOBALLY ASSEMBLED MATRIX . . . . . . . . . . .SYMMETRIC

                    L O A D   S T E P   O P T I O N S

   LOAD STEP NUMBER. . . . . . . . . . . . . . . .    1
   TIME AT END OF THE LOAD STEP. . . . . . . . . .  1.0000
   NUMBER OF SUBSTEPS. . . . . . . . . . . . . . .    1
   STEP CHANGE BOUNDARY CONDITIONS . . . . . . . .DEFAULT
   PRINT OUTPUT CONTROLS . . . . . . . . . . . . .NO PRINTOUT
   DATABASE OUTPUT CONTROLS. . . . . . . . . . . .ALL DATA WRITTEN
                                                   FOR THE LAST SUBSTEP
```

Fig.3.13 Solution step

3.1.3 Postprocessor

There are several different ways to view the results of a solution. To find the shear and bending moment diagrams we define what is called an element table and then plot a contour plot.Defining an element table is nothing more than a way of telling ANSYS which solution items you want to see. To define an element table, select the following:

> General Postproc > Element Table> Define Table

The Element Table Data window now appears. Select Add..

> Add...

We will define the element table items by using the "By sequence num" option. For the Beam element, the sequence numbers for the I moment (at left end of beam) and the J moment (at right end of beam) are 3 and 16. The sequence numbers for the forces in the Y direction are 6 and 19. The sequence numbers can be found for any element in the help documentation. Simply do a search in help for the element that you are using, and then scroll down in the text to find the table that lists the sequence numbers.

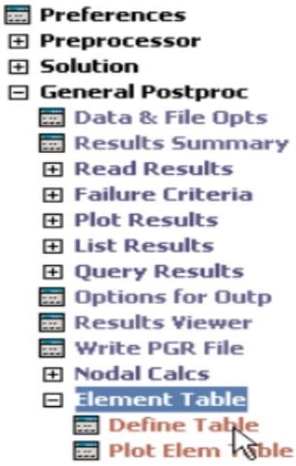


Fig.3.14 Postprocessor step1

Give the first item a label name of I moment, select By sequence number, select SMISC, and type in the number 3.

> Apply

Give the second item a label name of J moment, select By sequence number, select SMISC, and type in the number 16.

> Apply

Give the third item a label name of I force, select By sequence number, select SMISC, and type in the number 6 as shown to the right.

> Apply

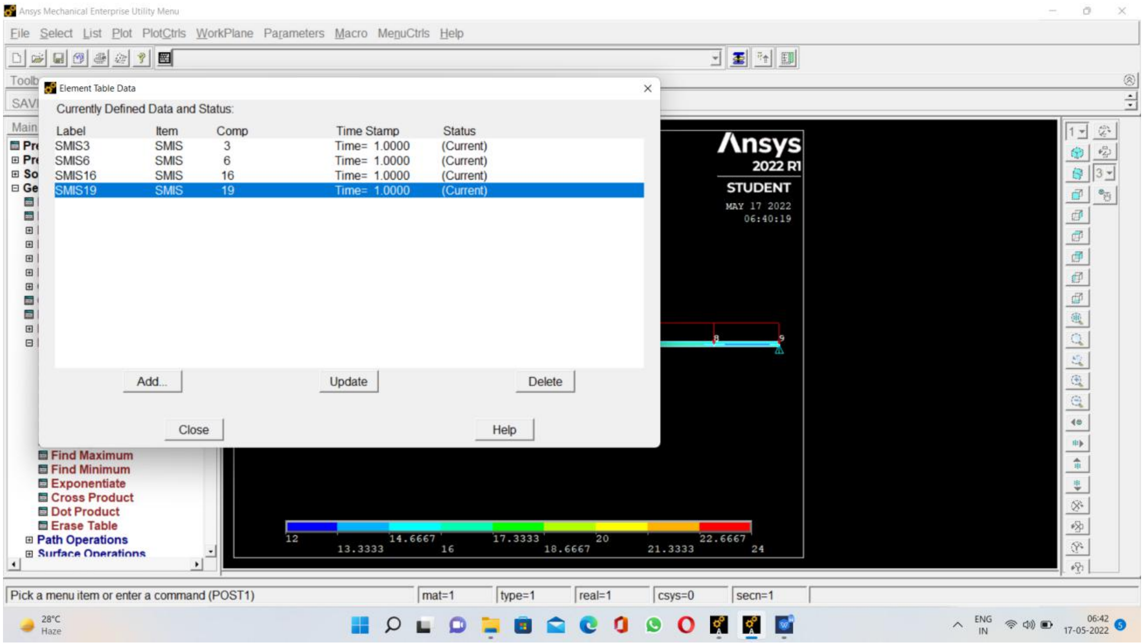
Give the fourth item a label name of J force, select By sequence number, select SMISC, and type in the number 19 as shown to the right.

> OK

When you are done you should have four items in
the Element Table Data window.

Close the Element Table Data window.

> Close



Currently Defined Data and Status:

Label	Item	Comp	Time Stamp	Status
SMIS3	SMIS	3	Time= 1.0000	(Current)
SMIS6	SMIS	6	Time= 1.0000	(Current)
SMIS16	SMIS	16	Time= 1.0000	(Current)
SMIS19	SMIS	19	Time= 1.0000	(Current)

Pick a menu item or enter a command (POST1) mat=1 type=1 real=1 csys=0 secn=1

Fig.3.15 Postprocessor step2

The shear force diagram will now be plotted.

> General Postproc > Plot Results > Contour Plot > Line Elem Res

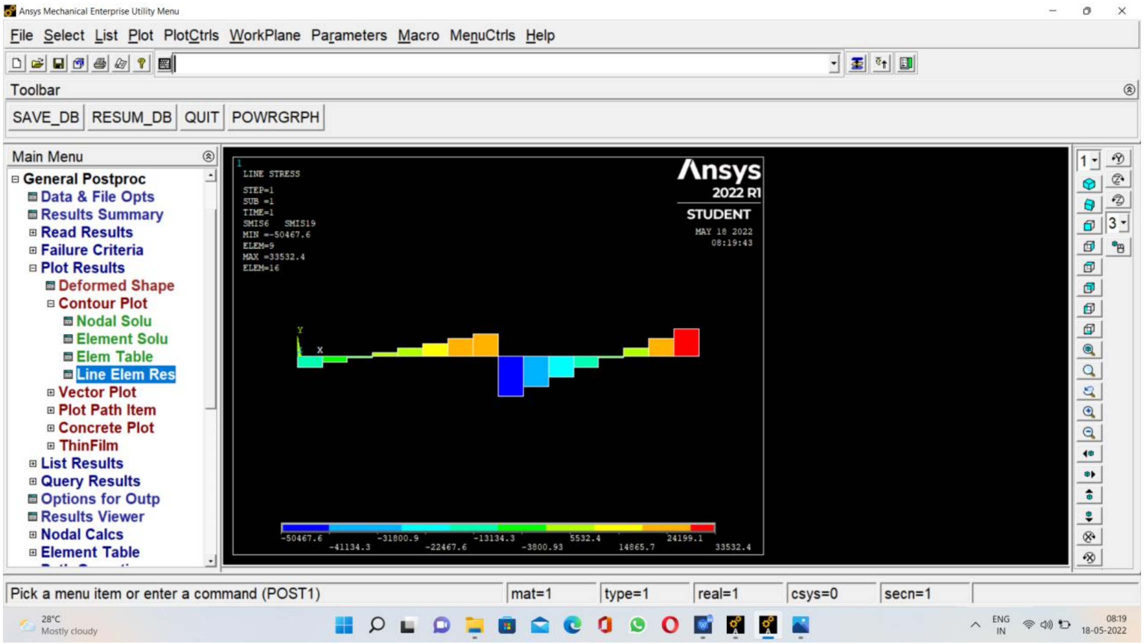
The Plot Line-Element Results window now appears.

Select SMIS6 the table item at node I and SMIS19 as the table item at node J.

> OK

The shear force diagram is plotted on the screen.

From the diagram, the max and min shear force can easily be seen.



```
LINE STRESS
STEP=1
SUB =1
TIME=1
SMIS6   SMIS19
MIN =-50467.6
ELEM=9
MAX =33532.4
ELEM=16
```

Pick a menu item or enter a command (POST1) | mat=1 | type=1 | real=1 | csys=0 | secn=1

Fig.3.16 Shear force diagram from ANSYS APDL

The bending moment diagram will now be plotted.

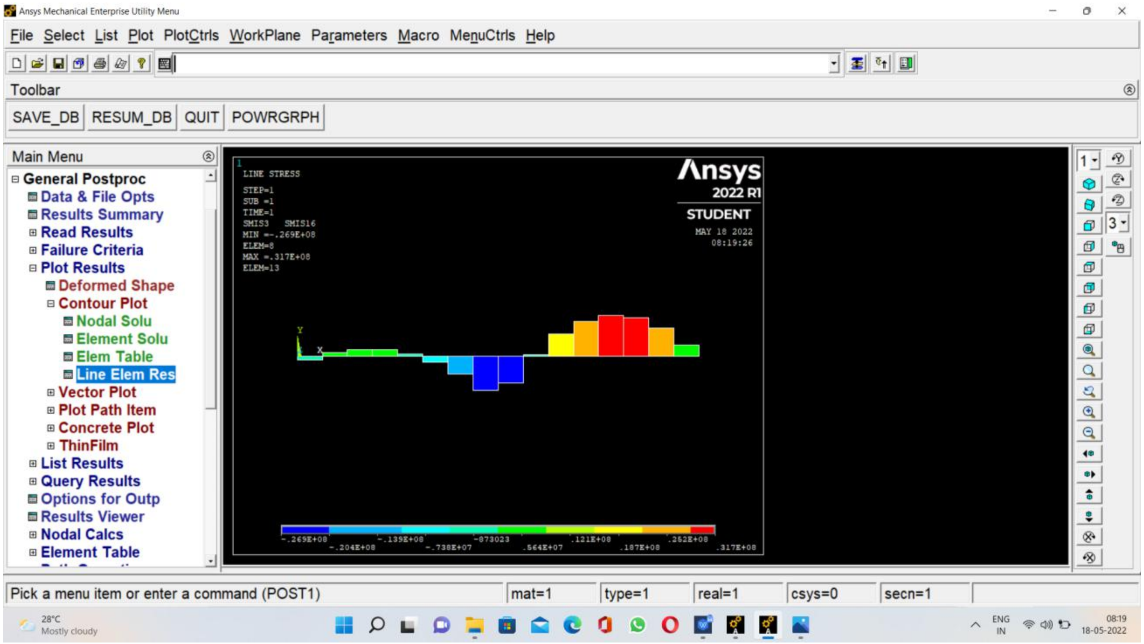
> General Postproc > Plot Results > Contour Plot > Line Elem Res

The Plot Line-Element Results window now appears.Select SMIS3 as the table item at node I and SMIS16 as the table item at node J.

> OK

The bending moment diagram is plotted on the screen.

From the diagram, the max and min bending moment can easily be seen.

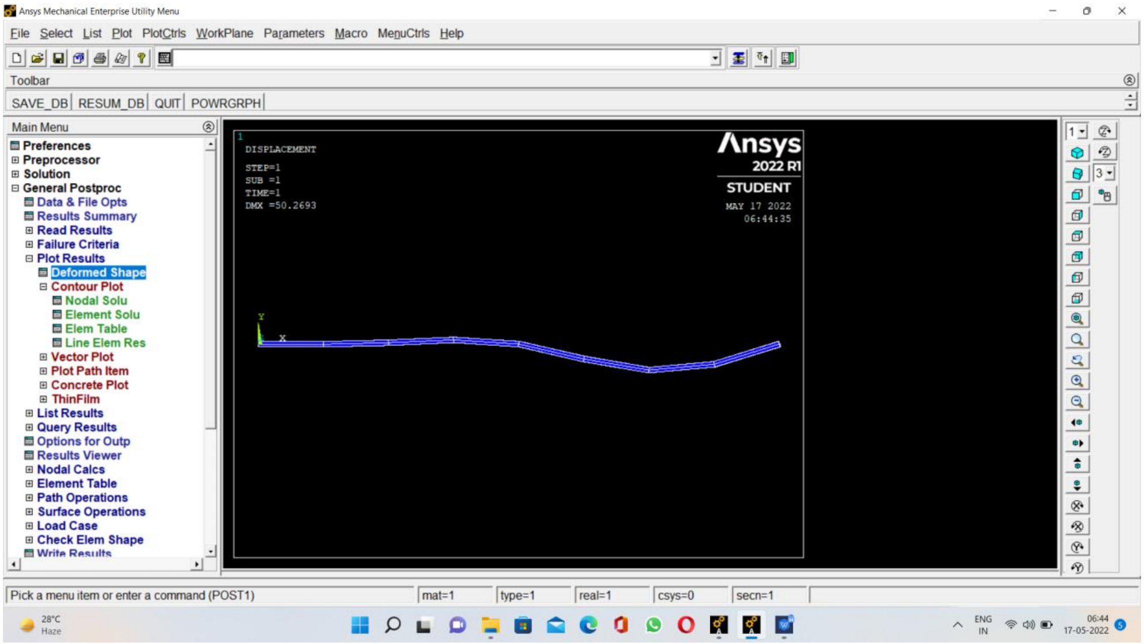


```
1
LINE STRESS
STEP=1
SUB =1
TIME=1
SMIS3   SMIS16
MIN =-.269E+08
ELEM=8
MAX =.317E+08
ELEM=13

MAY 18 2022
08:19:26

-.269E+08      -.139E+08      -873023      .121E+08      .252E+08
      -.204E+08      -.738E+07      .564E+07      .187E+08      .317E+08
```

Pick a menu item or enter a command (POST1) | mat=1 | type=1 | real=1 | csys=0 | secn=1

Fig.3.17 Bending moment diagram from ANSYS APDL

Deformed shape

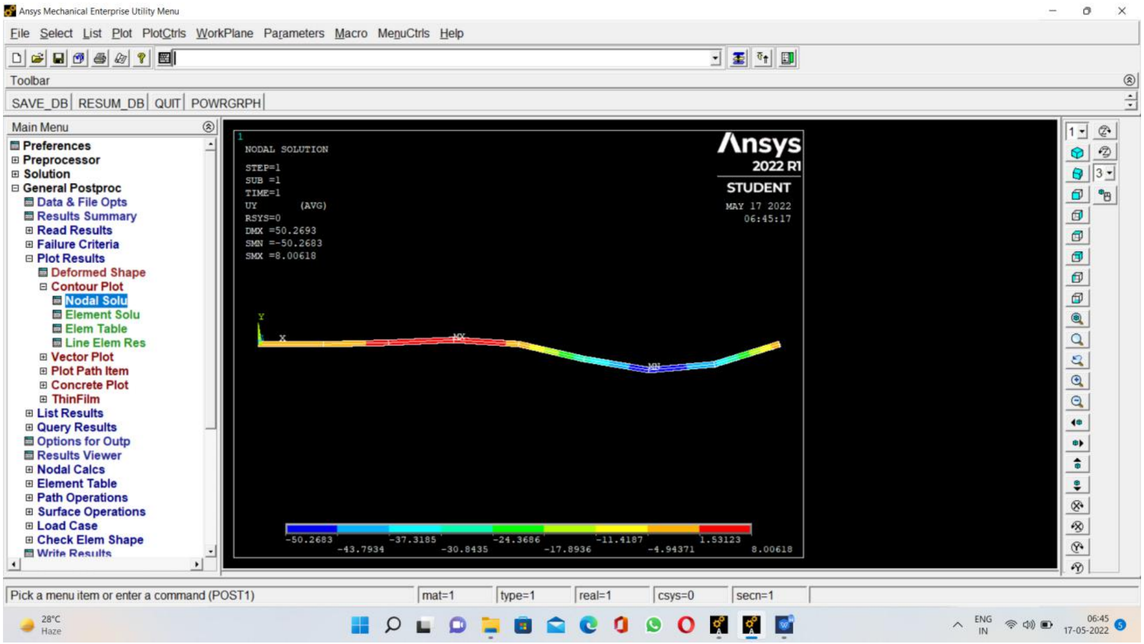


```
1
DISPLACEMENT
STEP=1
SUB =1
TIME=1
DMX =50.2693

MAY 17 2022
06:44:35
```

Pick a menu item or enter a command (POST1) | mat=1 | type=1 | real=1 | csys=0 | secn=1

Fig.3.18 Deformed shape of beam from ANSYS APDL

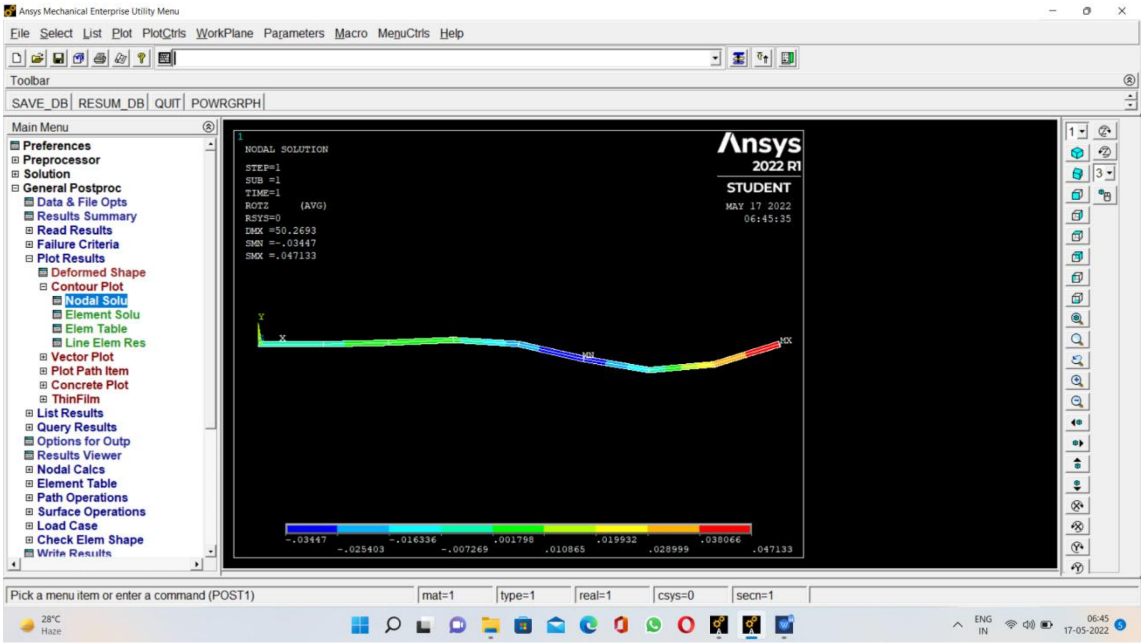
Nodal deflection



```
1
NODAL SOLUTION

STEP=1
SUB =1
TIME=1
UY       (AVG)
RSYS=0
DMX =50.2693
SMN =-50.2683
SMX =8.00618
                                  MAY 17 2022
                                     06:45:17
```

Fig.3.19 Nodal deflection diagram from ANSYS APDL

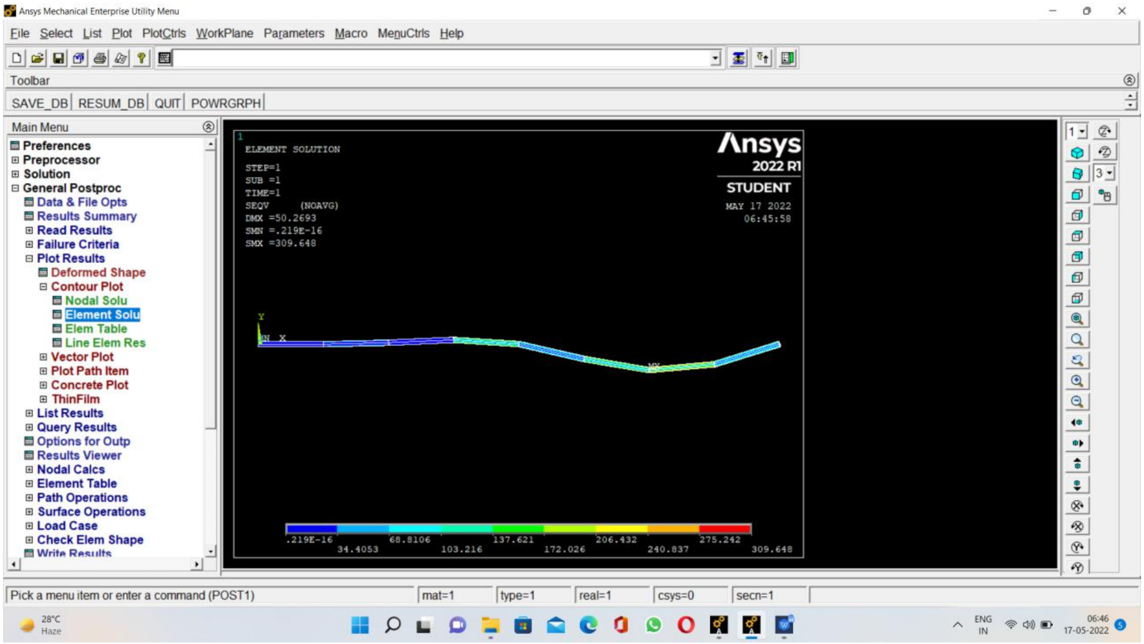
Nodal slopes



```
1
NODAL SOLUTION

STEP=1
SUB =1
TIME=1
ROTZ     (AVG)
RSYS=0
DMX =50.2693
SMN =-.03447
SMX =.047133
                                  MAY 17 2022
                                     06:45:35
```

Fig.3.20 Nodal slope diagram from ANSYS APDL

Von Mises stress



```
1
ELEMENT SOLUTION
STEP=1
SUB =1
TIME=1
SEQV      (NOAVG)
DMX =50.2693
SMN =.219E-16
SMX =309.648
```

Fig.3.21 Von Mises stress diagram from ANSY APDL

Nodal deflections(Uy)

in mm

NODE 1	0.00
NODE 2	-1.1242
NODE 3	1.6806
NODE 4	7.3962
NODE 5	0.00
NODE 6	-32.608
NODE 7	-55.526
NODE 8	-43.187
NODE 9	0.00

Table.3.2 Nodal deflections

from ANSYS APDL

Nodal slopes(Mz)

in rad.

NODE 1	0.00
NODE 2	-0.0003378
NODE 3	0.005808
NODE 4	0.003424
NODE 5	-0.022465
NODE 6	-0.035129
NODE 7	-0.0072174
NODE 8	0.031292
NODE 9	0.05838

Table.3.3 Nodal slopes

from ANSYS APDL

Reaction forces(in N)

NODE 1	17186
NODE 2	87282
NODE 3	39532

Table.3.4 Reaction forces

from ANSYS APDL

Reaction moment(in N-mm)

NODE 1	6613000

Table.3.5 Reaction moment

from ANSYS APDL

Natural frequencies(in Hertz)

NODE	FREQUENCY
1	0.0010517
2	0.0068072
3	0.015030
4	0.020348
5	0.027770
6	0.044255
7	0.071635
8	0.084450
9	0.089786

Table.3.6 Natural frequencies from ANSYS APDL

3.2 Python code for analyzing the beams

```
# importing libraries
import math
import matplotlib.pyplot as plot
import numpy as np
file = open(r"D:\Remo desktop\PhD work\project 2022\out.txt","w+")
noofelements=int(input("enter the total number of elements"))
file.write("the total number of elements= "+str(noofelements)+ '\n')
noofnodes=noofelements+1
file.write("the total number of nodes= "+str(noofnodes)+ '\n')

length_of_beam=float(input("enter total length of the beam in mm"))

length_of_element=length_of_beam/noofelements

coordinates={}
a_val=0
b_val=0
y_diff=float(input("enter 0 or enter y value difference for coordinates:"))
for i in range(1,noofnodes+1):
   coordinates[i]=[a_val,b_val]
   a_val+=length_of_element
   b_val+=y_diff
   file.write('coordinates for node' + str(i)+ 'in mm : '+ str(coordinates[i])+ '\n')
   print('coordinates for node' + str(i)+ 'in mm : '+ str(coordinates[i])+ '\n')
elasticity=float(input("enter the modulus of elasticity in N/mm-square"))
density=float(input("enter density in gm/mm-cube "))
file.write("modulus of elasticity in N/mm-square= "+str(elasticity)+ '\n')
startend={}
for i in range(1,noofelements+1):
   startend[i]=[i,i+1]
   file.write('the start node and end node for element '+ str(i)+ ' :'+str(startend[i])+ '\n')
```

```
    print('the start node and end node for element '+ str(i)+ ' :'+str(startend[i])+ '\n')
print("enter the shape of cross section")
print("1 for rectangle")
print("2 for hollow rectangle")
print("3 for triangle")
print("4 for circle")
print("5 for hollow circle")
print("6 for I section")
print("7 for T section")
print("8 for C section")
print("9 for L section")
print("10 enter moment of inertia value in mm power 4")

cross_section=int(input())
cs=cross_section
if(1==cs):
    b=float(input("enter width in mm"))
    d=float(input("enter depth in mm"))
    I=(b*(d**3))/12
    print("area moment of inertia of rectangle section in mm power 4:",I)
    area=b*d
elif(2==cs):
    b=float(input("enter width in mm"))
    d=float(input("enter depth in mm"))
    b1=float(input("enter inner width in mm"))
    d1=float(input("enter inner depth in mm"))
    I=((b*(d**3))/12)-((b1*(d1**3))/12)
    print("area moment of inertia of hollow rectangle section in mm power 4:",I)
    area=(b*d)-(b1*d1)
elif(3==cs):
    b=float(input("enter base in mm"))
    d=float(input("enter height in mm"))
    I=(b*(d**3))/36
    print("area moment of inertia of triangle section in mm power 4:",I)
    area=0.5*b*d
elif(4==cs):

    d=float(input("enter diameter of circle in mm"))
    I=(math.pi*(d**4))/64
    print("area moment of inertia of circle section in mm power 4:",I)
    area=(math.pi/4)*(d**2)
elif(5==cs):
    d=float(input("enter outer diameter of circle in mm"))
    d1==float(input("enter inner diameter of circle in mm"))
    I=((math.pi*(d**4))/64)-((math.pi*(d1**4))/64)
    print("area moment of inertia of circle section in mm power 4:",I)
    area=((math.pi/4)*(d**2 -  d1**2))
elif(6==cs):
    b1=float(input("enter top flamge width in mm"))
    d1=float(input("enter top flamge thickness in mm"))
    b2=float(input("enter web thickness in mm"))
    d2=float(input("enter web depth in mm"))
    b3=float(input("enter bottom flamge width in mm"))
    d3=float(input("enter bottom flamge thickness in mm"))
    a1=b1*d1
    a2=b2*d2
    a3=b3*d3
    y1=(d2+(d1/2))+d3
    y2=(d2/2)+d3
    y3=d3/2

    yc=(a1*y1+a2*y2+a3*y3)/(a1+a2+a3)
    I=((b1*(d1**3))/12)+(a1*((yc-y1)**2))+((b2*(d2**3))/12)+(a2*((yc-y2)**2))+((b3*(d3**3))/12)+(a3*((yc-y3)**2))
    print("area moment of inertia of I section in mm power 4:",I)
    area=a1+a2+a3
elif(7==cs):
    b1=float(input("enter flamge width in mm"))
    d1=float(input("enter flamge thickness in mm"))
    b2=float(input("enter web thickness in mm"))
    d2=float(input("enter web depth in mm"))
    a1=b1*d1
    a2=b2*d2
    y1=(d2+(d1/2))
    y2=d2/2
    yc=(a1*y1+a2*y2)/(a1+a2)
    I=((b1*(d1**3))/12)+(a1*((yc-y1)**2))+((b2*(d2**3))/12)+(a2*((yc-y2)**2))

    print("area moment of inertia of T section in mm power 4:",I)
    area=a1+a2
elif(8==cs):
    b1=float(input("enter top flamge width in mm"))
    d1=float(input("enter top flamge thickness in mm"))
    b2=float(input("enter web thickness in mm"))
    d2=float(input("enter web depth in mm"))
    b3=float(input("enter bottom flamge width in mm"))
    d3=float(input("enter bottom flamge thickness in mm"))
    a1=b1*d1
    a2=b2*d2
    a3=b3*d3
    y1=(d2+(d1/2))+d3
    y2=(d2/2)
    y3=d3/2

    yc=(a1*y1+a2*y2+a3*y3)/(a1+a2+a3)
    I=((b1*(d1**3))/12)+(a1*((yc-y1)**2))+((b2*(d2**3))/12)+(a2*((yc-
y2)**2))+((b3*(d3**3))/12)+(a3*((yc-y3)**2))

    print("area moment of inertia of C section in mm power 4:",I)
    area=a1+a2+a3
elif(9==cs):
    b1=float(input("enter flamge width in mm"))
    d1=float(input("enter flamge thickness in mm"))
    b2=float(input("enter web thickness in mm"))
    d2=float(input("enter web depth in mm"))
    a1=b1*d1
    a2=b2*d2
    y1=(d1/2)
    y2=d2/2+d1
    yc=(a1*y1+a2*y2)/(a1+a2)
    I=((b1*(d1**3))/12)+(a1*((yc-y1)**2))+((b2*(d2**3))/12)+(a2*((yc-y2)**2))

    print("area moment of inertia of L section in mm power 4:",I)
    area=a1+a2
else:
    I=float(input())
    print("moment of inertia in mm power 4",I)
    area=float(input("enter area in mm-square"))
print("area=",area)
stiffness={}
L=length_of_element
E=elasticity

for i in range(1,noofelements+1):

    mat=np.array([[12*E*I/L**3, (6*E*I)/(L**2), -12*E*I/L**3, 6*E*I/L**2],
                  [6*E*I/L**2, 4*E*I/L, -6*E*I/L**2, 2*E*I/L],
                      [-12*E*I/L**3, -6*E*I/L**2, 12*E*I/L**3, -6*E*I/L**2],
                      [6*E*I/L**2, 2*E*I/L, -6*E*I/L**2, 4*E*I/L]])

    for j in range(4):
        for k in range(4):
            mat[j][k]=mat[j][k]
    print("stiffness matrix of element "+str(i))
    for j in range(4):
        for k in range(4):
            print(mat[j][k],end='    ')
        print()
    print()
    print()
    stiffness[i]=mat
mass={}
L=length_of_element
for i in range(1,noofelements+1):

    mat=np.array([[156,22*L,54,-1*34*L],[22*L,4*L**2,13*L,-1*3*L**2],[54,13*L,156,-
1*22*L],[-1*13*L,-1*3*L**2,-1*22*L,4*L**2]])
    a=(area*density*L)/420
    for j in range(4):
        for k in range(4):
            mat[j][k]=a*mat[j][k]
    print("mass matrix of element "+str(i))
    for j in range(4):
        for k in range(4):
            print(mat[j][k],end='    ')
        print()
    print()
    print()
    mass[i]=mat
```

```
ndof = noofnodes * 2
print(ndof)
globstifmat=[]

for i in range(ndof):
    globstifmat.append([0]*ndof)




x=0
asinode={}
for i in range(1,noofnodes+1):
    asinode[i]=[x,x+1]
    x+=2
e=startend
n=asinode
d={}
for i in range(1,noofelements+1):
    x=[]
    p=e[i]
    q=p[0]
    r=p[1]
    x.extend([n[q][0],n[q][1],n[r][0],n[r][1]])
    d[i]=x
for i in d:
    print(d[i])
for i in range(ndof):
    for j in range(ndof):
        s=0
        for k in d:
            if(i in d[k] and j in d[k]):
                zzz=stiffness[k]
                s+=zzz[d[k].index(i)][d[k].index(j)]

        globstifmat[i][j]=s
print("global stiffness matrix ")
for j in range(ndof):
    for k in range(ndof):
        print(globstifmat[j][k],end='    ')
    print()
print()
print(len(globstifmat))
print(len(globstifmat[0]))
ndof = noofnodes * 2
print(ndof)
globmassmat=[]
for i in range(ndof):
    globmassmat.append([0]*ndof)
for i in range(ndof):
    for j in range(ndof):
        s=0
        for k in d:
            if(i in d[k] and j in d[k]):
                zzz=mass[k]
                s+=zzz[d[k].index(i)][d[k].index(j)]

        globmassmat[i][j]=s
print("global mass matrix ")
for j in range(ndof):
    for k in range(ndof):
        print(globmassmat[j][k],end='    ')
    print()
print()
```

```
#load vector and boundary conditions
noofsuppnodes=int(input("enter the total number of nodes having supports"))
file.write('the total number of nodes having supports '+str(noofsuppnodes)+ '\n')
nodesandtypesupport=[]
for i in range(noofsuppnodes):
   x=int(input("enter the node number having support"))
   print("enter the type of support")
   print("h for hinged")
   print("f for fixed")
   print("r for roller/pin support")
   t=input()
   file.write('the node number having support is '+str(x)+' type of support is '+t+ '\n')
   nodesandtypesupport.append([x,t.lower()])
print(nodesandtypesupport)
print("enter total number of point load nodes")
pln1=int(input())
y_dir1=[0]*ndof
rot1=[0]*ndof
for i in range(pln1):
   print("enter node number")
   nn1=int(input())
   print("enter y direction i Newton and rotation in N-mm ( comma seperated)")
   print("hint: upward is +ve and anticlock wise is +ve")
   tp1=input().split(',')
   a1=tp1[0]
   b1=tp1[1]
   y_dir1[nn1]=float(a1)
   rot1[nn1]=float(b1)
print("load vector due to point load")
print(y_dir1)
print(rot1)
comb1=[]
```

```
for i in range(len(y_dir1)):
   comb1.append(y_dir1[i]+rot1[i])
print(comb1)
print("enter total number of point couple or moment load nodes")
pln2=int(input())
y_dir2=[0]*ndof
rot2=[0]*ndof
for i in range(pln2):
   print("enter node number")
   nn2=int(input())
   print("enter y direction i Newton and rotation in N-mm ( comma seperated)")
   print("hint: upward is +ve and anticlock wise is +ve")
   tp2=input().split(',')
   a2=tp2[0]
   b2=tp2[1]
   y_dir2[nn2]=float(a2)
   rot2[nn2]=float(b2)
print("load vector due to point couple or moment load")
print(y_dir2)
print(rot2)
comb2=[]
for i in range(len(y_dir2)):
   comb2.append(y_dir2[i]+rot2[i])
print(comb2)
L=length_of_element
print("enter total number of UDL load elements")
udl_ele=int(input())
load_li=[0]*ndof
for i in range(udl_ele):
   print("enter element number")
   ele=int(input())
   print("enter  UDL load value in N/mm")
```

```
    udl_load=float(input())
    form=[(udl_load*L)/2,(udl_load*(L**2))/12,(udl_load*L)/2,-1*(udl_load*(L**2))/12]
    print(form)
    node1=ele
    node2=ele+1
    dof1=2*node1-1
    dof2=2*node1
    dof3=2*node2-1
    dof4=2*node2
    load_li[dof1-1]+=form[0]
    load_li[dof2-1]+=form[1]
    load_li[dof3-1]+=form[2]
    load_li[dof4-1]+=form[3]
print(load_li)
#uvl




uvl_li=[0]*ndof

print("enter number of starting uvl loads")
num=int(input())
for j in range(num):
    print("enter total number of uvl load elements for ",j+1,"load")
    uvl=int(input())

    print("left side uvl load in N/mm")
    left=float(input())
    print("right side uvl load in N/mm")
    right=float(input())
    print("enter uvl load length in mm")
    loadlen=float(int(input()))

    rate=((right-left)*L)/(loadlen)
    print(rate)

    def fun(left):
        a=[(left*L)/2,(left*(L**2))/12,(left*L)/2,-1*(left*(L**2))/12]
        b=[(3*rate*L)/20,(rate*(L**2))/30,(7*rate*L)/20,(-1*rate*(L**2))/20]
        zz=[]
        for i in range(4):
            zz.append(a[i]+b[i])
        return zz
    print("enter starting uvl element number")
    ele=int(input())

    for i in range(uvl):
        form=fun(left+rate*(ele+i-1))

        print("for element",ele+i,"uvl load is",form)
        node1=ele+i
        node2=ele+i+1
        dof1=2*node1-1
        dof2=2*node1
        dof3=2*node2-1
        dof4=2*node2
        uvl_li[dof1-1]+=form[0]
        uvl_li[dof2-1]+=form[1]
        uvl_li[dof3-1]+=form[2]
        uvl_li[dof4-1]+=form[3]
print(uvl_li)
#final load vector
```

```
final=[]
for i in range(len(uvl_li)):
   final.append(uvl_li[i]+load_li[i]+comb2[i]+comb1[i])
print(final)
coltorem=[]
rowtorem=[]
for i in nodesandtypesupport:
   if(i[1]=='f'):
      coltorem.extend(asinode[i[0]])
      rowtorem.extend(asinode[i[0]])
   elif(i[1]=='h'):
      coltorem.append(asinode[i[0]][0])
      rowtorem.append(asinode[i[0]][0])
   elif(i[1]=='r'):
      coltorem.append(asinode[i[0]][0])
      rowtorem.append(asinode[i[0]][0])



print(coltorem)
print(rowtorem)
aftuse=rowtorem
npglobstifmat=np.array(globstifmat)
onpglobstifmat=npglobstifmat
print(npglobstifmat)
print(npglobstifmat.shape)
#removing row
npglobstifmat=np.delete(npglobstifmat, rowtorem, 0)
print(npglobstifmat)
print(npglobstifmat.shape)
#removing column
npglobstifmat=np.delete(npglobstifmat, rowtorem, 1)
print(npglobstifmat)
```

```
rednpglobstifmat=npglobstifmat
print(npglobstifmat.shape)
#mass matrix removing rows nd columns
npglobmassmat=np.array(globmassmat)
onpglobmassmat=npglobmassmat
print(npglobmassmat)
npglobmassmat=np.delete(npglobmassmat, rowtorem, 0)
print(npglobmassmat)
npglobmassmat=np.delete(npglobmassmat, rowtorem, 1)
print(npglobmassmat)
rednpglobmassmat=npglobmassmat
print(npglobmassmat.shape)
globloadvector=final

print(globloadvector)
#coverting to column vector
npglobloadvector=np.array(globloadvector)
orignpglobloadvector=npglobloadvector
print(npglobloadvector.shape)
#removing rows
npglobloadvector=np.delete(npglobloadvector, rowtorem, 0)
print(npglobloadvector)
#linear eqn solving
a = npglobstifmat
b = npglobloadvector
nodaldeflectionslope = np.linalg.solve(a, b)
print("nodal deflection in mm and slope in radians")
print(nodaldeflectionslope)
#changing dimension
xx=[]
zz=list(nodaldeflectionslope)
i=0
```

```
s=set(rowtorem)
for j in range(noofnodes*2):
   if(j not in s):
      xx.append(nodaldeflectionslope[i])
      i+=1
   else:
      xx.append(0)
print(xx)
file.write('nodal deflection in mm and slope in radians are ' + str(xx)+'\n')
#changing to col matrix
npnodaldeflectionslope=np.array(xx)
print(npnodaldeflectionslope.shape)
print('nodal deflection in mm and slope in radians')
print(npnodaldeflectionslope)
npglobstifmat=np.array(globstifmat)
npglobstifmat
x=np.dot(npglobstifmat,npnodaldeflectionslope)
print(x)
t1=list(x)
t2=list(orignpglobloadvector)
res=[]
for i in range(len(t1)):
   res.append(t1[i]-t2[i])
print(res)
#reaction of support




y=x-orignpglobloadvector
reactionmat=y
print("shear force in N and Bending moment in N-mm")
print(reactionmat)
j=0
aftuse=rowtorem
ind=0
for i in nodesandtypesupport:
    if(i[1]=='f'):
        print('reaction node at R'+str(i[0])+"X in N "+str(reactionmat[aftuse[ind]]))
        file.write('reaction node at R'+str(i[0])+"X in N "+str(reactionmat[aftuse[ind]])+ '\n')
        ind+=1;
        print('reaction node at R'+str(i[0])+"Y in N "+str(reactionmat[aftuse[ind]]))
        file.write('reaction node at R'+str(i[0])+"Y in N "+str(reactionmat[aftuse[ind]])+ '\n')
        ind+=1



    elif(i[1]=='r'):
        print('reaction node at R'+str(i[0])+"Y in N "+str(reactionmat[aftuse[ind]]))
        file.write('reaction node at R'+str(i[0])+"Y in N "+str(reactionmat[aftuse[ind]])+ '\n')
        ind+=1
    elif(i[1]=='h'):
        print('reaction node at R'+str(i[0])+"Y in N "+str(reactionmat[aftuse[ind]]))
        file.write('reaction node at R'+str(i[0])+"Y in N "+str(reactionmat[aftuse[ind]])+ '\n')
        ind+=1
```

Modal Analysis of Beam: Calculating Natural frequencies and Mode Shapes

```
    class Eigen(object):

    def _init_(self, *args, **kwargs):
        return super()._init_(*args, **kwargs)

    def Rescale(self, x):
        max = x.max()
        min = x.min()
        s = x.shape
        n = s[0]
        amax = max
        if abs(min) > max: amax = abs(min)
        for i in range(n):
            x[i] = x[i] / max
        return x

    def RescaleEigenVectors(self, evec):
        dims = evec.shape
        ndofs = dims[0]
        for i in range(ndofs):
            evec[:,i] = self.Rescale(evec[:,i])
        return evec

    def GetOrthogonalVector(self, ndofs, trial, mg, ev,evec):
        const = 0

        s = [ndofs]
        sumcu= np.zeros(s)
        for e in range(ev  ):
            U = evec[:,e]
            const += trial @ mg @ U
            cu = [x * const for x in U]
            sumcu += cu
        trial = trial - sumcu
        return trial
    def Solve(self,kg, mg, tolerance = 0.00001 ):
        dims = kg.shape
        ndofs = dims[0]
        s = (ndofs,ndofs)
        evec = np.zeros(s)
        s = (ndofs)
```

```
eval = np.zeros(ndofs)
trial = np.ones(ndofs)
eigenvalue0 = 0
for ev in range(ndofs):
    print("Computing eigenvalue and eigen vector " + str(ev) + "... " , end="")

    converged = False
    uk_1 = trial
    k = 0
    while converged == False:
        k += 1

        if ev > 0:
            uk_1 = self.GetOrthogonalVector(ndofs,uk_1,mg,ev,evec)
        vk_1 = mg @ uk_1
        uhatk = np.linalg.solve(kg,vk_1)
        vhatk = mg @ uhatk
        uhatkt = np.transpose(uhatk)
        eigenvalue = (uhatkt @ vk_1)/(uhatkt @ vhatk)
        denominator = math.sqrt(uhatkt @ vhatk)
        uk = uhatk/denominator
        tol = abs((eigenvalue - eigenvalue0) / eigenvalue)
        if tol <= tolerance:
            converged = True
            evec[:,ev] = uk
            eval[ev] = eigenvalue
            print("Eigenvalue = " + str(eigenvalue))
            print('no of iterations= ',k)
        else:
            eigenvalue0 = eigenvalue
            uk_1 = uk
```

```
                    if k > 1000:
                        evec[:,ev] = uk
                        eval[ev] = eigenvalue
                        print ("could not converge. Tolerance = " + str(tol))
                        break

        self.eigenvalues = eval
        return evec
# compute eigenvalues and eigen vectors
e = Eigen()
evec = e.Solve(rednpglobstifmat,rednpglobmassmat)
evect = e.RescaleEigenVectors(evec)

eval = e.eigenvalues
neval = len(eval)
print("eigen values")
eigvalues=e.eigenvalues
print(e.eigenvalues)
file.write("eigen values are "+str(eigvalues)+'\n')
print(len(e.eigenvalues))
print("eigen vectors")
print(len(evect))
print(evect)
for i in evect:
    file.write("eigen vectors are "+str(list(i))+'\n')
from scipy.linalg import eigvalsh
ab=eigvalsh(rednpglobstifmat,rednpglobmassmat)
print(ab[0])
f1=np.sqrt(ab[0])
f1
f2=np.sqrt(ab[0])
f2=f2/(2*3.14)
```

```
f2
#frequency
freq=[]
for i in eigvalues:
   freq.append(math.sqrt(i)/(2*3.14))
print("natural frequency in hertz",freq)
file.write("natural frequency in hertz "+str(freq)+ '\n')
import matplotlib.pyplot as plt
z=1
for i in evect:
   time        = np.array(i)
   amplitude   = np.sin(time)
   plot.plot(time, amplitude)
   plot.title('mode'+str(z))
   plot.xlabel('vector')
   plot.ylabel('sin(vector)')
   plot.grid(True, which='both')
   plot.axhline(y=0, color='k')
   plot.savefig(r"D:\Remo desktop\PhD work\project 2022\mode"+str(z)+'.png',
bbox_inches='tight')
   plot.show()
   plot.show()

   z+=1
```

Beam to be executed in Python

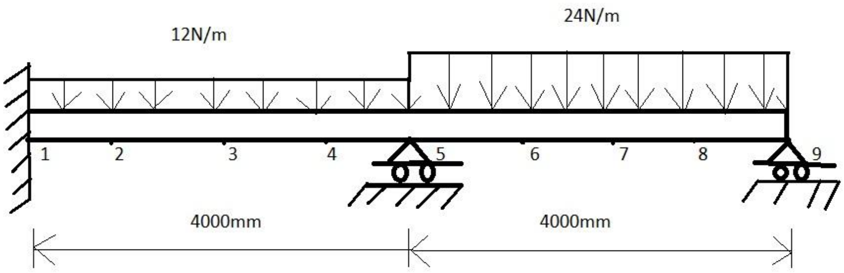


Fig.3.22 Beam to be analyzed using Python code

ELEMENT CONNECTIVITY TABLE

ELEMENT NO.	NODE 1	NODE 2
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	7	8
8	8	9

Table 3.7 Element connectivity table for beam analyzed using Python

3.2.1 Input given for python code

Length of the beam : 8000 mm

Young’s modulus : 200000 N/mm3

Inertia : 4000000 mm4

Area of cross-section : 6927.23 mm2

Density of element : 0.008 g/mm3

No. Of elements : 8

UDL value for 1-4 elements : 12 N/mm

UDL value for 5-8 elements : 24 N/mm

No. Of supports : 3

Type of support at node 1 : Fixed

Type of support at node 5: Roller

Type of support at node 9: Roller

3.2.2 Output from python code

```
the total number of elements= 8
the total number of nodes= 9
coordinates for node1in mm : [0, 0]
coordinates for node2in mm : [1000.0, 0.0]
coordinates for node3in mm : [2000.0, 0.0]
coordinates for node4in mm : [3000.0, 0.0]
coordinates for node5in mm : [4000.0, 0.0]
coordinates for node6in mm : [5000.0, 0.0]
coordinates for node7in mm : [6000.0, 0.0]
coordinates for node8in mm : [7000.0, 0.0]
coordinates for node9in mm : [8000.0, 0.0]
modulus of elasticity in N/mm-square= 200000.0
the start node and end node for element 1 :[1, 2]
the start node and end node for element 2 :[2, 3]
the start node and end node for element 3 :[3, 4]
the start node and end node for element 4 :[4, 5]
the start node and end node for element 5 :[5, 6]
the start node and end node for element 6 :[6, 7]
the start node and end node for element 7 :[7, 8]
the start node and end node for element 8 :[8, 9]
the total number of nodes having supports 3
the node number having support is 1 type of support is f
the node number having support is 5 type of support is r
the node number having support is 9 type of support is r
```

```
nodal deflection in mm and slope in radians are [0, 0, -1.3392857142857035, -
0.0003571428571428396, 1.4285714285714557, 0.0057142857142857256, 7.232142857142884,
0.003214285714285701, 0, -0.022857142857142895, -33.75000000000004, -
0.0353571428571429, -57.14285714285722, -0.00714285714285715, -44.464285714285765,
0.03178571428571433, 0, 0.05142857142857148]
```

```
reaction node at R1X in N 17142.857142857123
reaction node at R1Y in N 6857142.857142834
reaction node at R5Y in N 87428.57142857145
reaction node at R9Y in N 39428.57142857148
```

```
eigen values are [0.00725779 0.02178356 0.0685719  0.01522055 0.01029292 0.00836923
 0.00757259 0.00787322 0.00751117 0.00738306 0.00732841 0.00736993
 0.00732448 0.007359  ]
```

```
eigen vectors are [0.41445573593735274, -727.5601283713471, -0.02173815532887317,
0.06830778956473842, -2.918701741735896, 0.05523551305301973, -0.08193926493635977,
0.18976122940042878, 0.29984473778387644, 0.34622500643726845, -0.12777378883341506,
0.3519255393812662, -0.12899995555355395, -0.1194393684022216]
```

```
eigen vectors are [0.0006585308894527483, -1.0565433709564918, 0.000168043220985051l,
1.81420567550139e-05, -0.0036082484736192854, 0.00022752977364182863, -
0.000145976281234507, 0.0003884031698490004, 0.0005220609260106166,
0.0005787256590650591, -0.00020990701591274696, 0.0005856490590289405, -
0.00021159845885560067, -0.000198036620888728]

eigen vectors are [1.0, -1447.2001688630303, 0.4666129676553466, -0.08180028194500041, -
3.8234860493556306, 0.5361907289397004, -0.24385377332910455, 0.7109269980571082,
0.8574961052713916, 0.9198747364074458, -0.32872097235084746, 0.9274683722413606, -
0.33094321166919327, -0.31266181442697777]

eigen vectors are [0.0003761121905018182, -0.16349776967416552, 0.0006820955254143652, -
0.0002939416745326104, 0.002574604444149564, 0.0006620407872419246, -
0.0001453164177559864, 0.0005599832860413701, 0.0004789454574453561,
0.0004452358373665161, -0.00014774301709449765, 0.00044104198629622046, -
0.00014774304576565115, -0.0001464562828335143]

eigen vectors are [0.9699998561923173, -961.9142164924119, 0.8448784305507451, -
0.32446538738113445, 1.0, 1.0, -0.2937752894095101, 1.0, 1.0, 1.0, -0.34519170760076306, 1.0,
-0.34648463690672654, -0.33476647730137626]

eigen vectors are [-0.0004905290098510929, 1.0, -0.00018066572807887238, -
5.631360722150974e-05, 0.005023490397584205, -6.057897116548884e-06,
8.734440725345193e-05, -0.00017609463222872796, -0.00032439529544711735, -
0.0003884788830530746, 0.00014571409538450395, -0.0003961680146981308,
0.00014725584461919574, 0.0001348428581632132]

eigen vectors are [-0.0014222253857468724, 0.5511543886508126, -0.0013205593581225275,
0.0007425566966816885, -0.01080604698367122, -0.002210309938758033 7,
0.0005247733466420562, -0.0019684267983010512, -0.0017470771240982114, -
0.0016497232372897873, 0.0005516170814733838, -0.0016382364556770342,
0.0005522084887537334, 0.0005450847747384799]
```

```
eigen vectors are [-1.5574730312674263, -65.43935245638593, -0.9274373697524942, 0.841484061269176, -19.262784752853047, -2.8381780417329687, 0.6331329173009832, -2.459961341892664, -2.0923427608900074, -1.9272931653249363, 0.6346111134633659, -1.9081957640077036, 0.6345623072866547, 0.6331911186246816]

eigen vectors are [-0.0014260102405674487, -0.4892168664288267, -5.315082103204132e-05, 0.0006459203391735085, -0.02252886926693689, -0.00273804253658901, 0.0006054615652517517, -0.002378675217238815, -0.0019992140448686843, -0.0018244061002301206, 0.0005967564820816925, -0.0018046669500307213, 0.0005965186621537283, 0.0005982750078710261]

eigen vectors are [-2.4214053665139152, -590.7957835610205, 0.004924703112589671, 1.0, -35.666261499987826, -4.4214208977453735, 1.0, -3.888060350722453, -3.3113840007875357, -3.0438952880966474, 1.0, -3.013886295243407, 1.0, 1.0]

eigen vectors are [-0.00016078644828104678, -0.3884783976545204, 0.001523249806475169, -0.00030462124462214495, -0.006583846138730444, -0.00016397803613634704, 6.51760032307528e-05, -0.00023066234896169214, -0.0002283305156601891, -0.00021739663489323105, 7.184212962894666e-05, -0.00021721642187399359, 7.21646544395875e-05, 7.250878990162124e-05]

eigen vectors are [-1.7942177180776593, -622.9969887498738, 1.0, 0.48607893421830334, -28.714615722174237, -3.155071604561017, 0.7355492429103335, -2.838307914035749, -2.445340117899427, -2.256382088932153, 0.742284888206237, -2.2358929683483657, 0.7425593136183439, 0.7422879443638698]

eigen vectors are [0.0013564264001733326, 0.35262377123593436, 1.1890394057169294e-05, -0.0005707893235929866, 0.02021176117191715, 0.0024990604204100718, -0.0005628723418596965, 0.002192559905170224, 0.0018629150202141974, 0.0017102420374484502, -0.0005614344569881734, 0.001693089410969928, -0.0005613946630236216, -0.0005616786196880239]
```

eigen vectors are [0.002021299847320939, 0.7672876343606069, -0.0015745260854522088, -0.000427977336201311, 0.03318252379908634, 0.0034849551502229934, -0.0008242910554695968, 0.003168173603910115, 0.0027453672530228686, 0.0025386216777152372, -0.000835886521552246, 0.002516556238690057, -0.0008363452087774558, -0.0008357127744434184]

natural frequency in hertz [0.013565710622506125, 0.023502000070651712, 0.04169783077574668, 0.019645159201469753, 0.016155098416523073, 0.014567439477518181, 0.01385678379202456, 0.014129169116553817, 0.013800478303801923, 0.013682279157822436, 0.013631549666467846, 0.013670108787534494, 0.01362789077396322, 0.01365996653517018]

CHAPTER-4

EXPERIMENTAL RESULTS

4.1 STRUCTURAL ANALYSIS RESULTS FROM PYTHON

Distance (in mm)	Node Number	Deflection (in mm)	Slope (in Radians)
0	1	0	0
1000	2	-1.339285714	-0.000357143
2000	3	1.428571429	0.005714286
3000	4	7.232142857	0.003214286
4000	5	0	-0.022857143
5000	6	-33.75	-0.035357143
6000	7	-57.14285714	-0.007142857
7000	8	-44.46428571	0.031785714
8000	9	0	0.051428571

Table.4.1 Nodal deflections and nodal displacements from Python code

REACTION FORCES(in N)

Node	Force
1	17142
2	87428
3	39428

Table.4.2 Reaction forces from Python code

REACTION MOMENT (in N-mm)

Node	Moment
1	6851720

Table.4.3 Reaction moment from Python code

Length of the beam(in mm)

Vs

Nodal slopes(in N-mm)

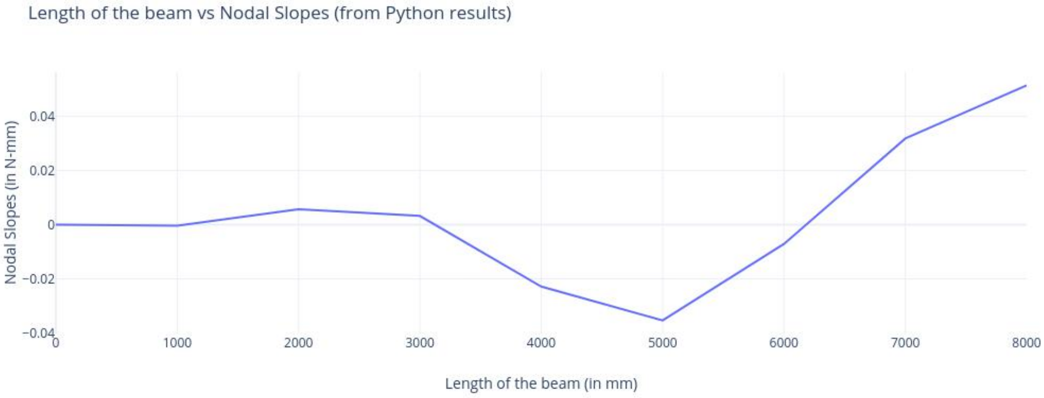


Fig.4.1 Graph plotted between Length of the beam-Nodal Slopes from Python results

Length of the beam(in mm)

Vs

Nodal deflections(in mm)

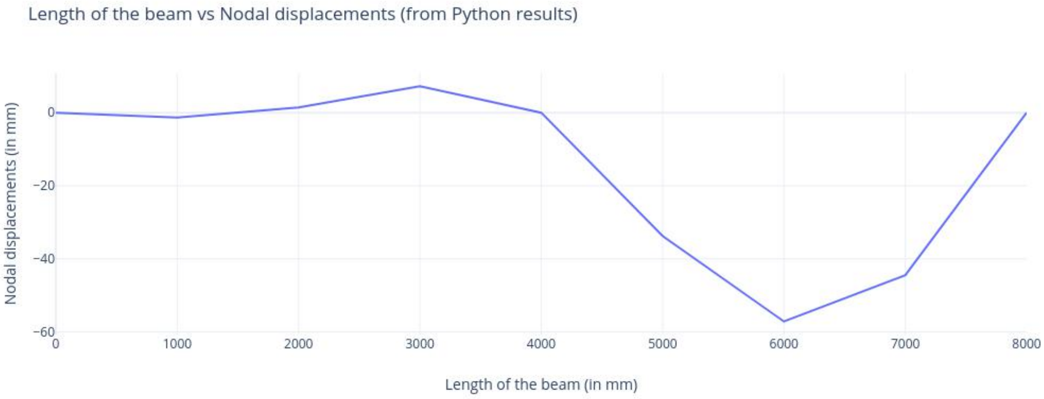


Fig.4.2 Graph plotted between Length of the beam-Nodal deflections from Python results

4.2 MODAL ANALYSIS RESULTS FROM PYTHON

NATURAL FREQUENCY

(in Hertz)

Node	Frequency(Hz)
1	0.013565710622506125
2	0.023502000070651712
3	0.04169783077574668
4	0.019645159201469753
5	0.016155098416523073
6	0.014567439477518181
7	0.01385678379202456
8	0.014129169116553817
9	0.013800478303801923
10	0.013682279157822436
11	0.013631549666467846
12	0.013670108787534494
13	0.01362789077396322
14	0.01365996653517018

Table.4.4 Natural frequency from Python code

EIGEN VALES

Node	Eigen value
1	0.00725779
2	0.02178356
3	0.0685719
4	0.01522055
5	0.01029292
6	0.00836923
7	0.00757259
8	0.00787322
9	0.00751117
10	0.00738306
11	0.00732841
12	0.00736993
13	0.00732448
14	0.007359

Table.4.5 Eigen values from Python code

MODE SHAPES

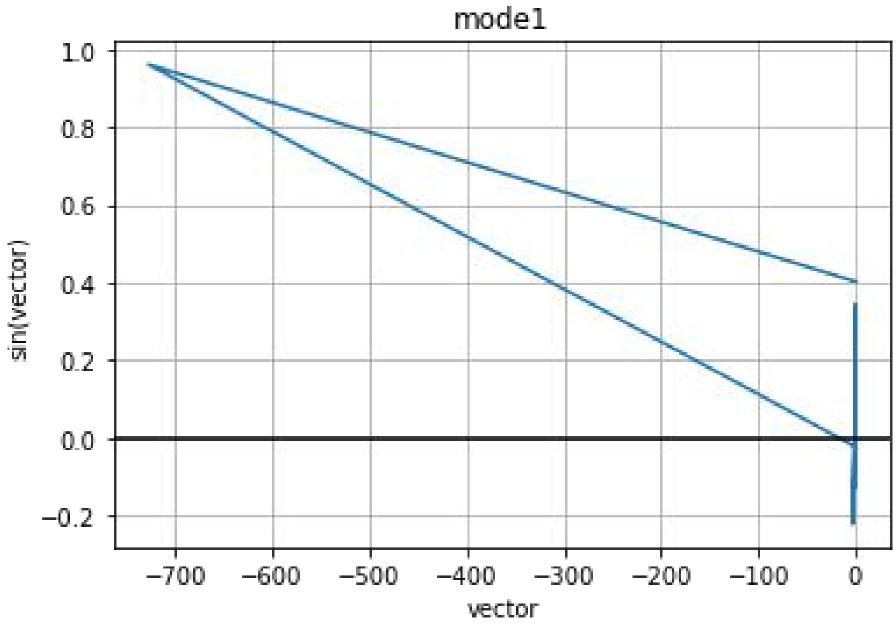


Fig.4.3 Mode Shape 1 for corresponding frequency 1.

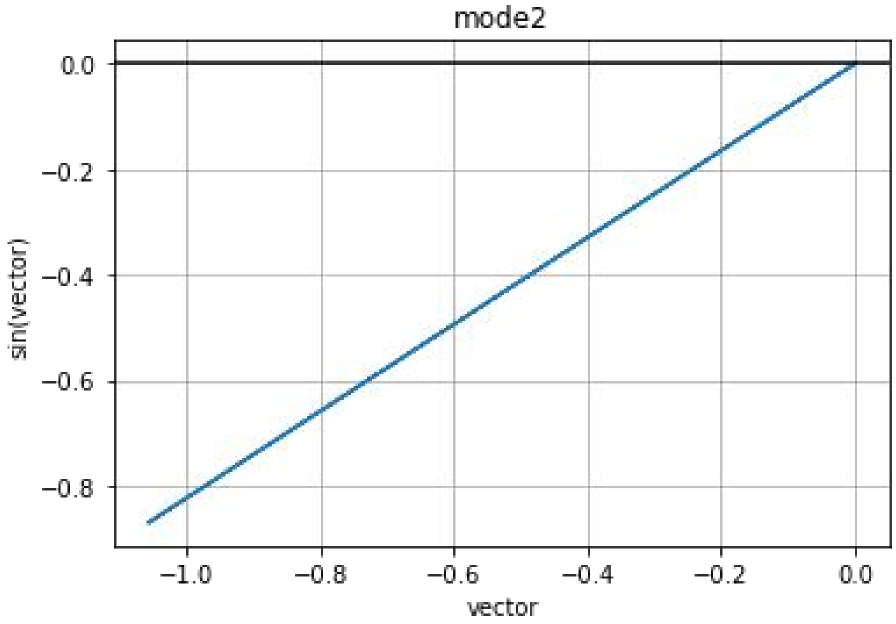


Fig.4.4 Mode Shape 2 for corresponding frequency 2.

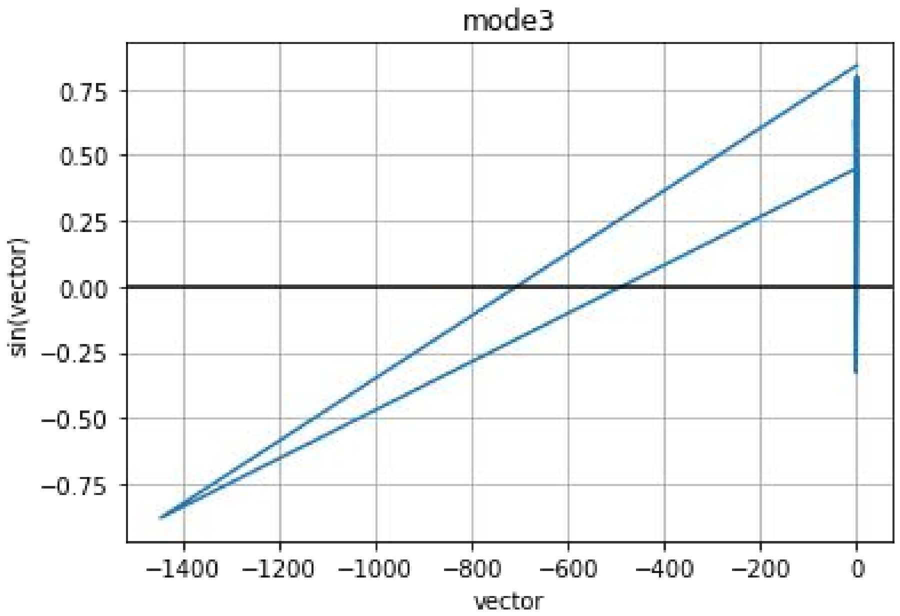


Fig.4.5 Mode Shape 3 for corresponding frequency 3.

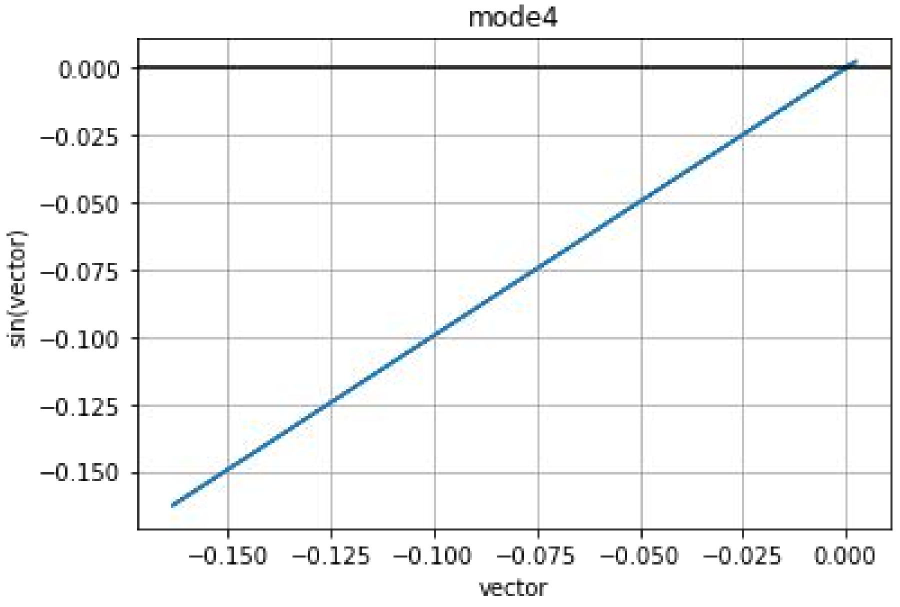


Fig.4.6 Mode Shape 4 for corresponding frequency 4.

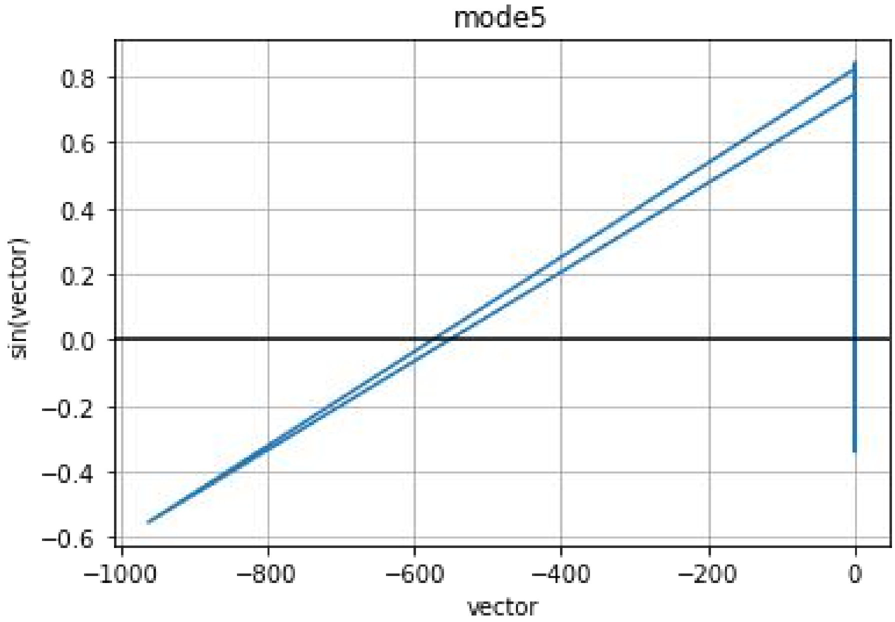


Fig.4.7 Mode Shape 5 for corresponding frequency 5.

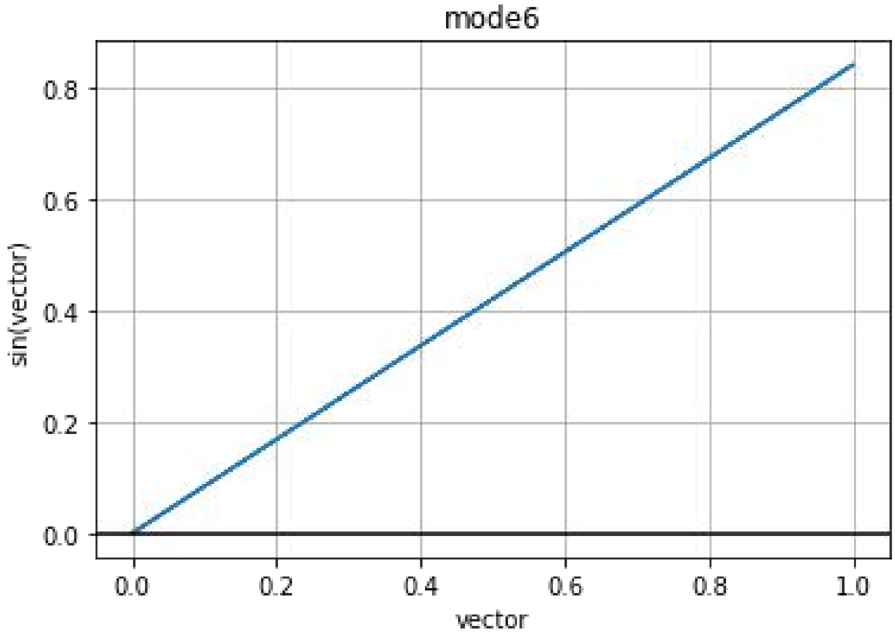


Fig.4.8 Mode Shape 6 for corresponding frequency 6.

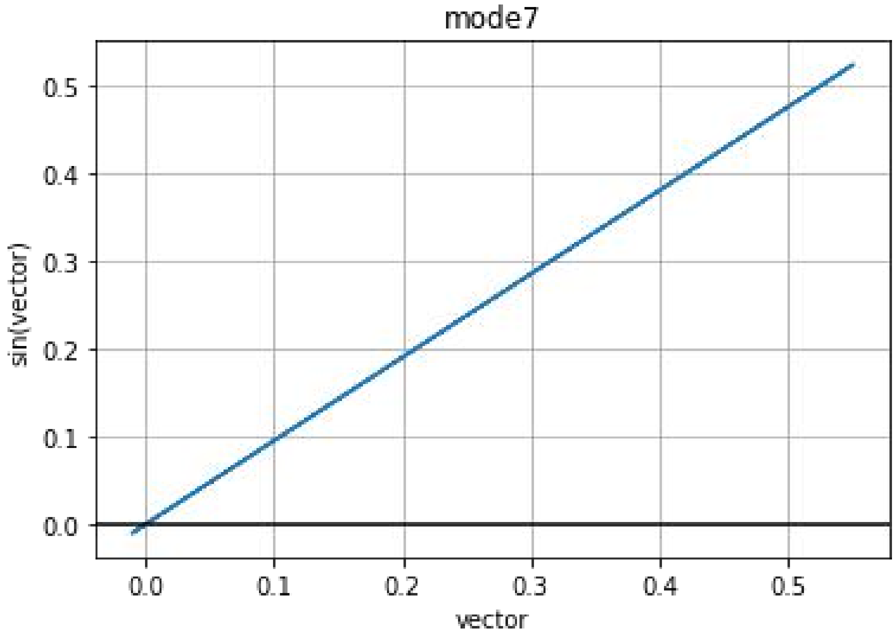


Fig.4.9 Mode Shape 7 for corresponding frequency 7.

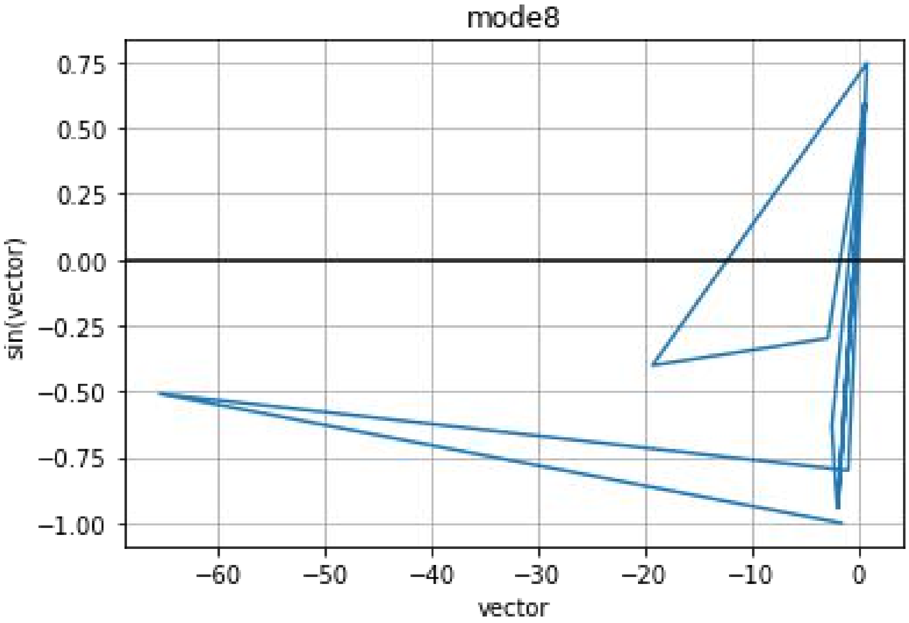


Fig.4.10 Mode Shape 8 for corresponding frequency 8.

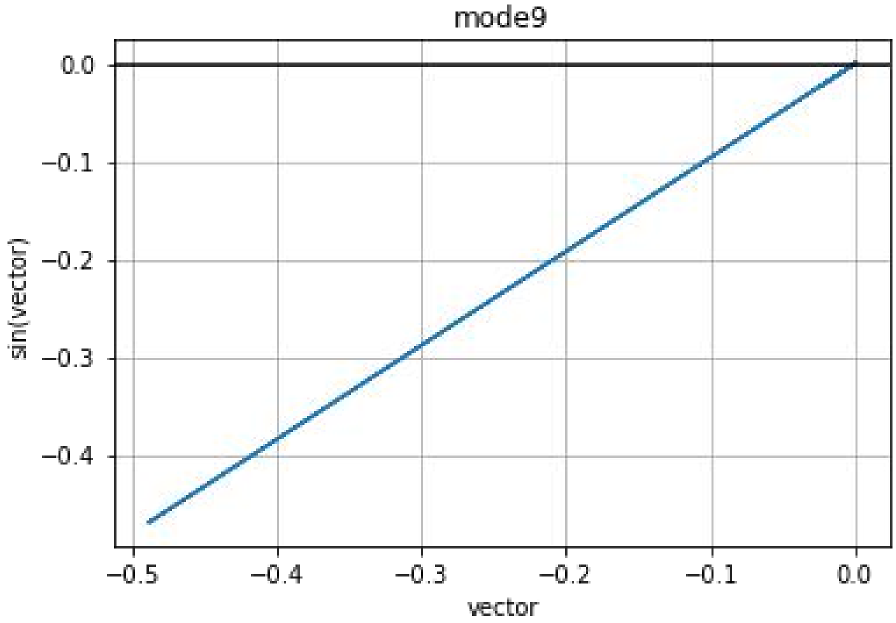


Fig.4.11 Mode Shape 9 for corresponding frequency 9.

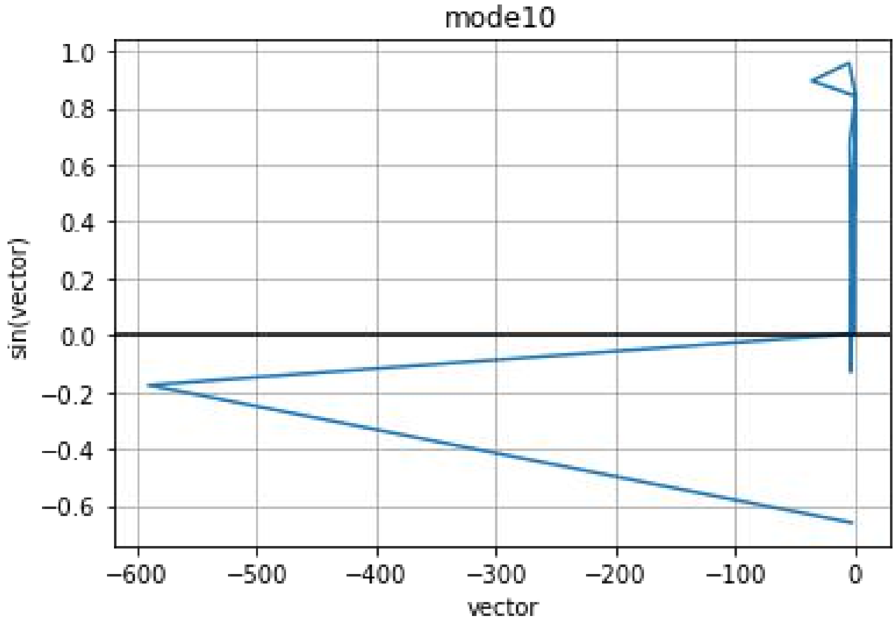


Fig.4.12 Mode Shape 10 for corresponding frequency 10.

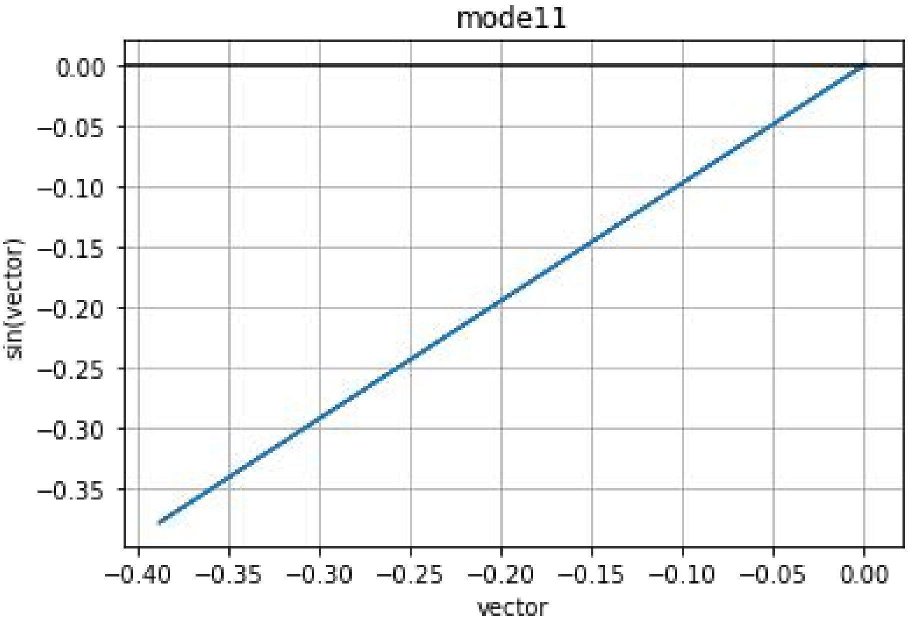


Fig.4.13 Mode Shape 11 for corresponding frequency 11.

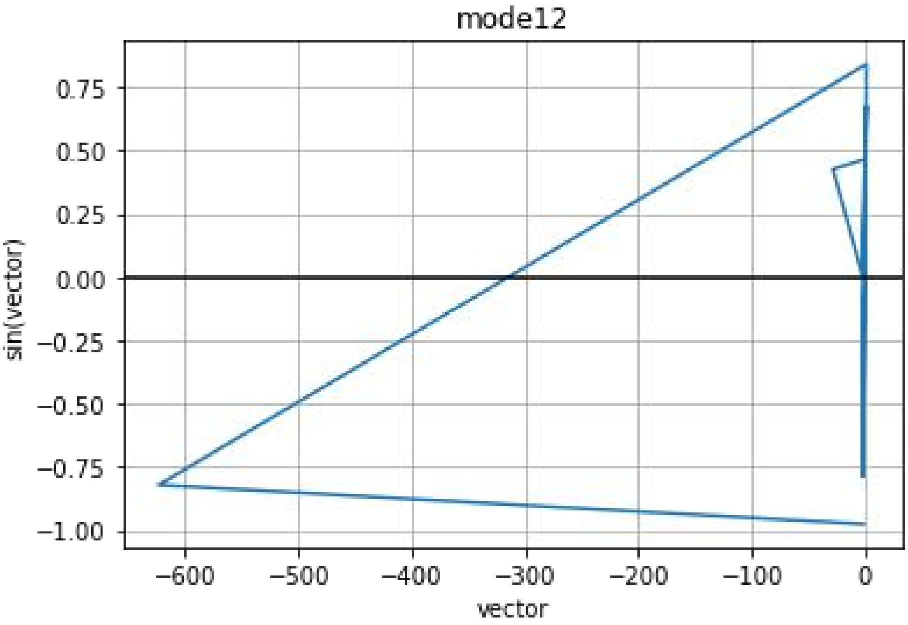


Fig.4.14 Mode Shape 12 for corresponding frequency 12.

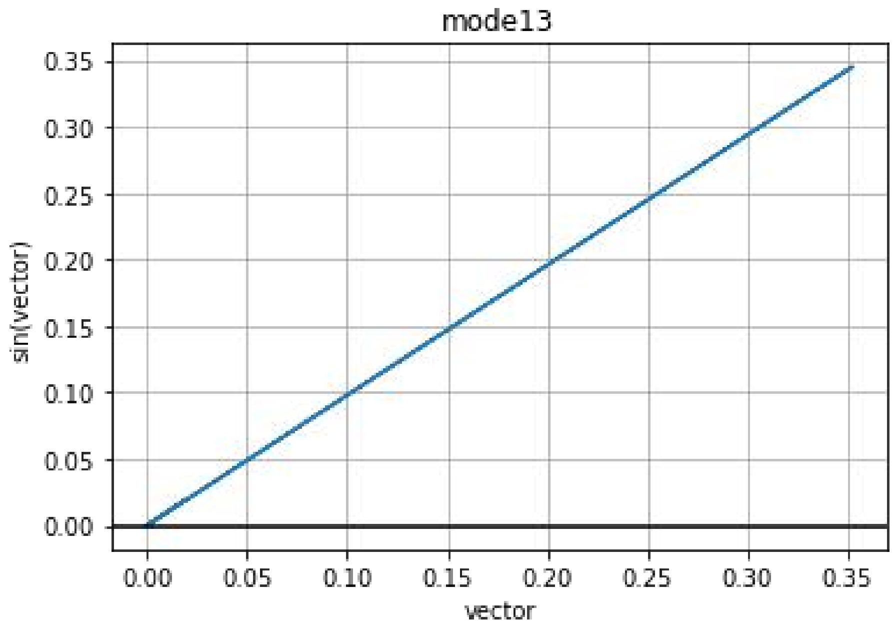


Fig.4.15 Mode Shape 13 for corresponding frequency 13.

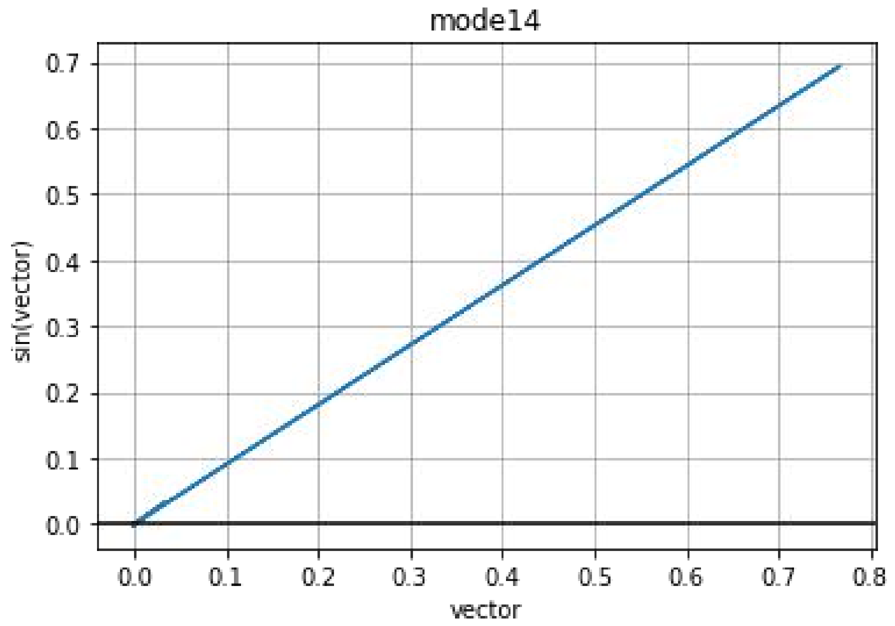


Fig.4.16 Mode Shape 14 for corresponding frequency 14.

CHAPTER-5

DISCUSSION OF RESULTS

5.1 COMPARING ANSYS APDL RESULTS WITH PYTHON RESULTS

NODAL DEFLECTIONS(in mm)

NODE	ANSYS APDL	PYTHON
1	0.00	0
2	-1.1242	-1.339285714
3	1.6806	1.428571429
4	7.3962	7.232142857
5	0.00	0
6	-32.608	-33.75
7	-55.526	-57.14285714
8	-43.187	-44.46428571
9	0.00	0

Table.5.1 Comparing results of Nodal deflections from Python code to ANSYS APDL

Nodal Slopes(in N-mm)

NODE	ANSYS APDL	PYTHON
1	0.00	0
2	-0.0003378	-0.000357143
3	0.005808	0.005714286
4	0.003424	0.003214286
5	-0.022465	-0.022857143
6	-0.035129	-0.035357143
7	-0.0072174	-0.007142857
8	0.031292	0.031785714
9	0.05838	0.051428571

Table.5.2 Comparing results of Nodal slopes from Python code to ANSYS APDL

Reaction forces(in N)

NODE	ANSYS APDL	PYTHON
1	17186	17142
2	87282	87428
3	39532	39428

Table.5.3 Comparing results of Reaction forces from Python code to ANSYS APDL

Reaction moment (in mm)

NODE	ANSYS APDL	PYTHON
1	6613000	6851720

Table.5.4 Comparing result of Reaction moment from Python code to ANSYS APDL

NATURAL FREQUENCIES (in Hertz)

NODE	ANSYS APDL	PYTHON
1	0.015030	0.013565710622506125
2	0.027770	0.023502000070651712
3	0.044255	0.04169783077574668
4	0.020348	0.019645159201469753
5	0.0068072	0.016155098416523073
6	0.0010517	0.014567439477518181
7	0.071635	0.01385678379202456
8	0.084450	0.014129169116553817
9	0.089786	0.013800478303801923

Table.5.5 Comparing Natural frequencies from Python code to ANSYS APDL

Here, in ANSYS the constant value C(eg. WLe3/3EI, 5WLe4/384EI …) varies according to the type of beam and type of supports.

Where as in our Python code the value of C will be taken automatically according to the material property of the beam.

So, few natural frequencies values will vary while comparing.

CHAPTER-6

CONCLUSION

CONCLUSION

From the present investigation, it can be concluded that the results obtained by running python code were almost matched with the results obtained from ANSYS APDL software. The current project investigated structural analysis of a continuous beam using Python code and results obtained are exceptional.

The python code which we developed is suitable for all types of beams includes cantilever beam,simply supported beam,Continuous beam,Fixed beam,Over hanging beam, with any material, for any type of loads including Point load,Point couple / Moment load,Uniformly varying load (Triangle,Trapezium),Uniformly distributed load .

Any type of cross section like Square,Rectangle,T- Section, L - Section,

C- Section,Straight beam, Curved, Tapered,Pipe sections,according to moment of inertia.

The code which we developed is used for automation for conducting structural analysis.

If for different materials this same experiment has to run, then the entire preprocessing process has to be changed and re-run the analysis. If in another case the truss structure was different or number of elements used are different, then the entire modeling work and then the pre & post works has to be carried out newly while using ANSYS APDL Software. With this Python code, the mentioned challenges can be solved very easily and at high speeds.

In future it can be upgraded to work for all structural members like trusses, columns etc. . Also we can implement in composite materials also.

CHAPTER-7

REFERENCES

REFERENCES

[1] An Object-Oriented class design for the Generalized Finite Element Method programming Dorival Piedade Neto* Manoel Dênis Costa Ferreira Sergio Persival Baroncini Proença, latin American journal of solids and structures, 10(2013) 1267 – 1291

[2] Alves Filho, J. S. R., Devloo, P. R. B. (1991). Object Oriented programming in Scientific computations: the beginning of a new era. Engineering Computations, Vol. 8, Issue 1, pp. 81-87.

[3] Bordas, S. P. A., Nguyens, P. V., Dunant, C., Guidoum, A., Nguens-Dand, H. (2007). An extended finite element library, International Journal for Numerical Methods in Engineering, Vol. 71, pp. 703-732

[4] Duarte, C. A. and Oden, J. T. (1996b).Hpclouds – an hpmeshless method. Numerical Methods for Partial Differential Equations, Vol. 12, pp. 673–705.

[5] scikit-fem: A Python package for finite element assembly
Tom Gustafsson1 and G. D. McBain DOI: 10.21105/joss.02369

[6] Cimrman, R., Lukeš, V., & Rohan, E. (2019). Multiscale finite element calculations in Python using SfePy. Advances in Computational Mathematics. doi:10.1007/s10444-019 09666-0

[7] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science
&
Engineering, 9(3), 90–95. doi:10.1109/MCSE.2007.55

[8] Bathe, K. J. (1996). Finite Element Procedures, Prentice-Hall, Inc. Englewood Cliffs, New Jersey. http://matplotlib.sf.net/Matplotlib.pdf, (acessed April 2011).

[9] https://chart-studio.plotly.com/create/#/

[10] https://www.mech4study.com/2016/04/what-is-beam-types-of-beams.html

[11] https://www.hkdivedi.com/2017/03/different-types-of-load-acting-on-beam.html

[12] https://theconstructor.org/structural-engg/types-beams-construction/24684/

[13] https://semesters.in/definition-and-types-of-a-beam-notes-pdf-ppt/