ELASTIC ANALYSIS OF STEEL FRAMES SUBJECTED TO DYNAMIC LOADING

FINAL YEAR PROJECT UG2016

By

(Leader - 132848 Ibad-ur-Rehman Siddiqui)
(Member 1 – 177027 Danyal Hameed)
(Member 2 - 193349 Raja Muhammad Ibrahim Mushtaq)

NUST Institute of Civil Engineering
School of Civil and Environmental Engineering
National University of Sciences and Technology, Islamabad, Pakistan

2020

This is to certify that the

Final Year Project Titled

ELASTIC ANALYSIS OF STEEL FRAMES SUBJECTED TO DYNAMIC LOADING

submitted by

(Leader - 132848 Ibad-ur-Rehman Siddiqui)
(Member 1 – 177027 Danyal Hameed)
(Member 2 - 193349 Raja Muhammad Ibrahim Mushtaq)

has been accepted towards the requirements for the undergraduate degree

in

CIVIL ENGINEERING

_____

Dr. Azam Khan
Assistant Professor
NUST Institute of Civil Engineering
School of Civil and Environmental Engineering
National University of Sciences and Technology, Islamabad, Pakistan

ABSTRACT

Safety of a structure is the first and foremost priority of a structural engineer. That can be achieved only when the different properties and behavior of the various materials used for construction are known. Initially we set out on this project with the objective of analyzing the behavior of steel frames subjected to dynamic loading. But unfortunately, just like the rest of the world, our project was also affected by the current global pandemic. This led to a change of scope and restricted us to the analysis of steel beams subjected to static loads.

The method consists of the following: (1) developing a code using MATLAB that is able to carry out the elastic analysis of beams, (2) verification of results on ABAQUS.

The code developed is able to carry out the linear analysis of determinate and indeterminate 2D beams subjected to dynamic loading and the comparison with results from ABAQUS verifies the code. Our work has laid down the basis for further studies in this regard and this project shall be continued by the upcoming batch

DEDICATION

This research is wholeheartedly dedicated to our beloved parents, who have been our source of inspiration and gave us strength when we thought of giving up, who continually provide their moral, spiritual, emotional, and financial support.

To our mentor, Sir Azam Khan, who guided us throughout the whole project and kept on pushing us which helped us achieve our goals.

To our beloved country Pakistan, we set out on this project with a goal to do something for the betterment of the construction industry and we hope it will greatly benefit from this research.

ACKNOWLEDGEMENTS

First of all, we would like to thank the supreme power, The Almighty Allah Who is obviously the one Who guided us to work on the right path of life. Without His grace, this project could not become a reality.

Next to Him are our parents to whom we are greatly indebted for bringing us up with love and encouragement to this stage. Their support is, was, and would always be the foundation of any little or great achievements that have and may come our way.

We are highly obliged to thank our mentor, Sir Azam Khan, whose teachings and guidance have been integral to this project. His work ethic inspired and motivated us to push ourselves towards our goal.

At last but not the least we are thankful to all our teachers and friends who have always helped and encouraged us throughout our 4 years here at NICE. We have no valuable words to express our thanks, but our hearts are full of the favors received from every person.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 General

Steel is widely used in construction all over Pakistan because of its various beneficial properties. Therefore, a better understanding of its behavior is essential to benefit the industry. Currently, majority of the research being carried on in the industry in Pakistan is material based. There is a lack of numerical modelling. There are no locally made software available in the country that can be used for quick and efficient analysis of the steel structures. Software that are available in the market are very expensive and hence legal software-aided design practices are negligible. In such a scenario, the use of cracked software is increasing, which is an illegal act and should be discouraged.

Initially, we took on this project to carry a detailed analysis on the behavior of steel frames subjected to dynamic loading. However, the current global pandemic affected our project as well and we were forced to reduce our scope to the elastic analysis of beams subjected to static loading. Our findings are going to be the steppingstone in the process of better understanding the behavior of steel frames and in the development of a software that can be used for the quick and efficient analysis of steel frames subjected to dynamic loading.

1.2 Finite Element Method

1.2.1 Introduction

The finite element method (FEM) is a computational technique used to obtain approximate solutions of boundary value problems in engineering, sometimes also referred to as finite element analysis (FEA). A boundary value problem, simply stated, is a mathematical problem in which one or more dependent variables must satisfy a differential equation everywhere within a known domain of independent variables and satisfy specific conditions on the boundary of the domain. Boundary value problems are also sometimes called field problems. The field is the domain of interest and most often represents a physical structure. The field variables are the dependent variables of interest governed by the differential equation. The boundary conditions are the specified values of the field variables or related variables such as derivatives, on the boundaries of the field. Depending on the type of physical problem being analyzed, the field variables may include physical displacement, temperature, fluid velocity and heat flux to name a few.

1.2.2 Brief History

The mathematical roots of FEM date back to a little over half a century. In the late 1940s, aircraft engineers were dealing with the invention of the jet engine and the need for more sophisticated analysis of airframe structures to withstand greater loads associated with higher speeds. These engineers developed the matrix method of force analysis known as the flexibility method in which the unknowns are the forces and the knowns are displacements and they did so without the benefit of modern computers. In FEM, the term displacement is quite general and can represent physical displacement, temperature, or fluid velocity for example. The term finite element was first used by Dr. Ray Clough in 1960 in context of plane stress analysis and has been in common usage since that time. During 1960s and 1970s, the applications of FEM were extended to plate and shell bending, pressure vessels, and general 3D problems in elastic structural analysis. FEM is computationally intensive and in the early years, calculations were performed using mainframe computers only. During 1960s, the first major finite element software code, NASTRAN, was developed in conjunction with the space exploration program of the United States. Since then many commercial software have been introduced that can be used on desktop computers and engineering workstations to obtain solutions to large problems in static and dynamic structural analysis, heat transfer, fluid low, electromagnetics, and seismic response.

1.2.3 General Procedure for Finite Element Analysis

Certain steps in formulating a finite element analysis of a physical problem are common to all such analyses. These steps are described as follows.

1.2.3.1 Preprocessing

The preprocessing step is described as defining the model and includes defining the:

- geometric domain of the problem
- element type(s) to be used.
- material properties of the elements.
- geometric properties of the elements (length, area etc.).
- element connectivities.
- boundary conditions.
- loadings.

The preprocessing step is critical because a perfectly computed finite element solution is of no use if it corresponds to the wrong problem.

1.2.3.2 Solution

In this step, the software assembles the governing algebraic equations in matrix form and computes the unknown values of the primary field variables. The computed values are then used by back substitution to compute additional, derived variables, such as reaction forces, heat flow, and element stresses.

1.2.3.3 Postprocessing

Analysis and evaluation of the results is referred to as postprocessing. Postprocessing software contains sophisticated routines used for sorting, printing, and plotting the desired results from a finite element solution. Examples of operations that can be accomplished include:

- Calculate factors of safety.
- Plot deformed structural shape.
- Produce color-coded temperature plots.
- Check equilibrium.

The solution data can be manipulated many ways in postprocessing, the most important objective is to apply sound engineering judgement in determining whether the solution results are physically reasonable.

1.3 Literature Review

Since the inception of the concept of the finite element method, it has been extensively used for modelling in different engineering fields. In the early stages, much of the modelling was carried out using the mainframe computer and due to their limited availability, the method was not widespread. However, the advancements in computers in the last two to three decades has enabled the common desktop user to be able to use software that can help do extensive numerical modelling. Talking about structural analysis and more specifically about steel structures, lots of research has been carried out around the world to understand the behavior of steel structures subjected to different conditions such as the effects of blast loading, dynamic wind loading, cyclic lateral loads, the behavior of steel frame models with bracing, shear wall and base isolation arrangements compared to wooden frame models with bracing arrangement, and the dynamic response analysis of steel frames with semi rigid joints to name a few. Majority of these studies involve numerical modelling using the finite element method and the results of the analysis, which are duly supported by the experimental results, have allowed us to know the behavior of steel structures better which has led to better and safer design practices. This review shows that numerical modelling techniques are the right way to move forward.

In Pakistan, the emphasis is mainly on material-based research. There is a lack of numerical modelling and such studies which employ the use of finite element analysis to analyze steel structures are rare and virtually non-existent. On the other hand, the use of steel structures in construction has increased, mainly in the northern parts of the country, which were most affected by the devastating earthquake in 2005. Studies like these are therefore needed to ensure safe construction practices.

1.4 Methodology

The basic principle behind the finite element analysis is the energy principle. The law of conservation of energy holds here which states that the work done on a system is equal to the energy stored in it. The method consists of the following steps:

- Defining element properties.
- Discretization of beam and a.
- Derivation of shape functions.
- Formation of gradient matrix.
- Numerical integration using gaussian quadrature.
- Assembling the entries to form K-matrix.
- Formation of force vector.
- Application of boundary conditions.
- Plotting of results.

All these steps are further explained in detail in Chapter 2.

CHAPTER 2

METHODOLOGY

This chapter briefly explains the basics of direct stiffness method and the finite element method and highlights how both the methods differ in their approach to formulate the global stiffness matrix. Furthermore, the stepwise procedure adopted by the code for the analysis is explained for understanding.

2.1 Direct Stiffness Method

The direct stiffness method is the most common implementation of the finite element method and it is most effective for solving fairly simple problems. Before talking about the procedure, we define the sign conventions which are going to be followed throughout the project because it is essential to have uniform sign conventions to avoid any complications and/or confusions. Following sign conventions have been adopted:

- Forces/Displacements to the right are positive.
- Forces/Displacements in the upward direction are considered positive.
- Moments/Rotations in the counterclockwise direction are taken as positive.

The procedure for the direct stiffness is similar to the finite element method, the only difference is the approach for the formulation of the stiffness matrices. For the analysis of a beam, initially the beam is discretized, and the elements and nodes are numbered. Then the local stiffness matrices are obtained through hand calculations and evaluation of matrices. These local stiffness matrices are then assembled into a global stiffness matrix. The method is not suitable for real life problems because the global stiffness matrix for such cases can be as big as a 20 x 20 matrix, the calculations for which get too complex to follow by hand. This is where finite element method works better.

2.2 Finite Element Method

The general procedure used to solve a system using finite element method is already discussed in chapter 1, which is similar to the direct stiffness method. The only difference is how we establish the stiffness matrix. We are basically utilizing the energy approach known as the principle of virtual work. Recall the law of conservation of energy, which states that energy can neither be created nor destroyed but can be converted from one form to another. The two energies that we are concerned with in case of a static system are the external work done and the stored strain energy. External work can either be the

product of force and displacement or moment and rotation. While the strain energy is the response to the applied axial load or bending moment. Using the shape functions, we make a guess of how the displaced shape of an element would look like under the influence of external load. Based on this assumption, we compute the energies of the system. A matrix for shape functions is formed whose double derivative gives us the gradient matrix. The gradient matrix is then numerically integrated using the gaussian quadrature to give the local stiffness matrices which are then assembled into the global stiffness matrix.

2.3 Methodology Adopted by the Code

Initially, the member properties (see figure 1 and figure 2) and support constraints are identified and then the structure is discretized into finite elements. Coordinates are assigned to nodes and the nodal connectivity is established for each element (see figure 3). The unknown nodal displacements are identified and numbered.

```
global Elem Stiffmat
%
%  Input Data
%
Len = 45;               %meter
nel = 45;               % number of elements
nnel=2;                 % number of nodes per element
ndof = 2;
E = 200e+9;              %N/m^2
I = 1/12;           %m^4
area=1;             % cross-sectional area of the beam
rho=1;              % mass density ( not used for static probelms)
ngl = 5;                %input('integration points')
```

Figure 1: Input for member properties.

```
%---------------------
% Initialise matrices
%---------------------
gcoord = zeros (nel+1,1); %initialize the nodal coordinate array
nodes = zeros (nel,2);  %initialize the nodal connectivity array
%
nnode = (nnel-1)*nel+1; % total number of nodes in system
sdof = nnode*ndof; % total system dofs
kk=zeros(sdof,sdof);
index=zeros(nnel*ndof,1); %initialization of index vector
ff = zeros(sdof,1);     %initialize nodal force vector
fix = zeros(sdof,1);     %initialize nodal force vector
% Element degree of freedom- for member loads
%for i = 1:1:nel
 %   Eldof(i)=zeros(nnel*nnel);
%end
 Eldof=cell(nnel*nnel,1);
```

Figure 2: Initialization of matrices in MATLAB.

```
%---------------------------------
% Data for nodal coordinate values
%---------------------------------
%
j = 0;
i = 1;
%
while j <= Len
            gcoord(i) = j;
            i = i+1;
            j = j+ Len/nel;
end
%-----------------------------------------
% Nodal connectivity for each element
%-----------------------------------------
%
for i = 1:1:nel
     nodes(i,1) = i;
end
for i = 1:1:nel
     nodes(i,2) =1+i;
end
```

Figure 3: Nodal coordinate values and connectivities.

Following this, the local stiffness matrices for the elements are required which are then to be assembled into the global stiffness matrix. For this purpose, we use shape functions. A shape function helps to approximate the solution for the points present between two nodes through interpolation. Using boundary conditions, shape function for beam elements are found and assembled into a matrix of shape functions which is denoted by [N]. The double derivative of shape functions for a beam element gives us the gradient matrix [B]. Using the following formula (equation 1), the local stiffness matrices are obtained using the gradient matrix.

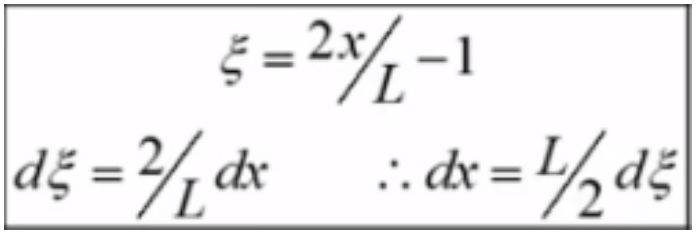
$$\mathbf{K} = \iiint_V \left(\mathbf{B}^T \mathbf{C} \mathbf{B}\right) dV$$

Equation 1:

In the next step, numerical integration using gaussian quadrature is done which is the fastest and most accurate method since all our elements are based on polynomials. For that purpose, first we use the substitution of ξ (Zhi) to change the limits of "*x = 0*" to "*ξ = -1*" and "*x = L*" to "*ξ = +1*" in order to make it possible for the computer to give a solution, where "*ξ = (2x/L)*-1". This substitution is shown through figure 4 and 5. What it does is that it approximates the integral by a weighted sum. A general formula for it is given by figure 6 where:

- n = no. of weighting points
- w_i = weighting factors
- x_i = sampling points

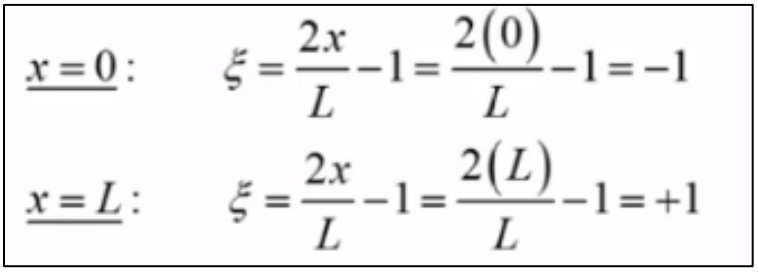
The sampling points and the weights used for five-point gaussian quadrature are given in figure 7.



$$\xi = 2x/L - 1$$

$$d\xi = 2/L\, dx \qquad \therefore dx = L/2\, d\xi$$

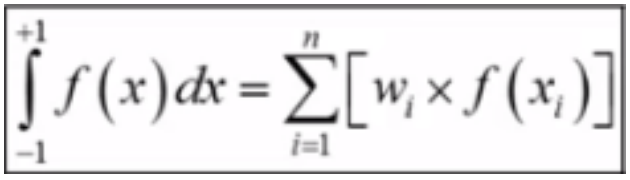
Figure 4: Substitution of Zhi.



$$\underline{x = 0}: \qquad \xi = \frac{2x}{L} - 1 = \frac{2(0)}{L} - 1 = -1$$

$$\underline{x = L}: \qquad \xi = \frac{2x}{L} - 1 = \frac{2(L)}{L} - 1 = +1$$

Figure 5: Substitution of Zhi.



$$\int_{-1}^{+1} f(x)\,dx = \sum_{i=1}^{n} \left[w_i \times f(x_i) \right]$$

Figure 6: General formula for gaussian quadrature.

n = 5 jump to n = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,

i	weight - w_i	abscissa - x_i
1	0.5688888888888889	0.0000000000000000
2	0.4786286704993665	-0.5384693101056831
3	0.4786286704993665	0.5384693101056831
4	0.2369268850561891	-0.9061798459386640
5	0.2369268850561891	0.9061798459386640

Figure 7: Weights and sampling points.

The function for the sampling points and weights in the code is shown in figure 8.

```
function [point1,weight1]= feglqd1(ngl)
% Variable Description:
%ngl - number of integration points
% point1 - vector containing integration points
% weight1 - vector containing weighting coefficient
% -----------------------------------------------------------
% initialization
point1 = zeros (ngl,1);
weight1 = zeros (ngl,1);
% find corresponding integration points and weights
%
if ngl == 1    % 1-point quadrature rule
point1(1) = 0.0;
weight1(1) = 2.0;
%
elseif ngl == 2 % 2-point quadrature rule
point1(1) = -0.577350269189626;
point1(2) = -0.577350269189626;
weight1(1) = 1.0;
weight1(2) = weight1(1);
%
```

Figure 8: Function for the sampling points and weights.

The local stiffness matrices thus obtained are then assembled into a global stiffness matrix denoted by [K] and a function that specifies the nodal connectivities of different elements is used for this purpose (see figure 9).

```
%-------------------------------------------
% Computation of element matrices and vectors and their assembly
%-------------------------------------------
%
for iel = 1:nel   % loop for the total number of elements
nl=nodes(iel,1); nr = nodes(iel,2); % extract nodes for (iel)-th element
xl = gcoord(nl); xr=gcoord(nr); % extract nodal coord values
eleng = xr-xl;  % element length
%
index = sysdof(iel,nnel,ndof); % extract system dofs associated
%
Eldof(iel)={index};
%
[pointl, weightl]=feglqdl(ngl); % extract sampling points and weights
%
k=feode2l(pointl,weightl,ngl,E(iel),I(iel),eleng); % compute element matrix
%
[kk]=feasmbl2a(kk,k,index); %assemble element matrices
%
%kk
end
```

Figure 9: Assembly of global stiffness matrix.

Moving on the applied loads are specified and the code forms a force vector using them, denoted by [F]. Boundary conditions are then applied (see figure 10) in order to substructure out the degrees of freedom which are constrained so that the matrix for the unknown displacements only is formed, denoted by [U]. Following equation, rearranged as shown, is used to calculate the unknown displacement vector:

$$[F] = [K][U]$$

Where,

$$[U] = [K]^{-1}[F]$$

```
%-------------------------------------------------
% apply boundary conditions
%-------------------------------------------------
[kk,ff]=feaplyc2(kk,ff,bcdof,bcval);
%
u=inv(kk)*(ff-fix); % displacements
%
A=cell2mat(Eldof);
%
kt=1
```

Figure 10: Function for boundary conditions.

The unknown displacements thus found are back substituted to solve for the reactions and internal forces. The code also plots the shear force and bending moment values as well (see figure 11 and figure 12).

```
% Print the title of the table.
fprintf(' Element Forces\n\n');
fprintf(' Element Number     F1(V1)                  F2(M1)                  F3(V2)                  F4(M2)\n');
fprintf(' =============     =======                 ======                  ======                  ======\n');
%
j=1;
%
for i = 1:1:nel
B=[fix(A(i,1)) fix(A(i,2)) fix(A(i,3)) fix(A(i,4))]'+...
    k*[u(A(i,1)) u(A(i,2)) u(A(i,3)) u(A(i,4))]';
B';
out = [i' B(j)' B(j+1)' B(j+2)' B(j+3)'];
fprintf (' %4d %33.4f %23.4f %23.4f %22.4f\n',out)
end
```

Figure 11: Function for SFD and BMD plots.

```
% Print the title of the table.
fprintf(' \n');
fprintf(' Nodal displacements\n\n');
fprintf(' Node Number     Displacements         rotation\n');
fprintf(' ===========     =============   ==========\n');
%
%
j=1;
for i = 1:1:nnode
out = [i' u(j)' u(j+1)'];
fprintf (' %4d %33.8f  %22.8f\n',out)
j=j+2;
end
%
```

Figure 12: Function for SFD and BMD plots.

CHAPTER 3

RESULTS, CONCLUSIONS AND RECOMMENDATIONS

3.1 Results

Using the basic principles of finite element method and MATLAB, we have developed a code that is able to carry out the linear elastic analysis of two-dimensional determinate and indeterminate beams subjected to static loads. The code gives the results in the form of nodal displacements, shear forces, and bending moments. It is also able to plot the shear force diagrams and bending moment diagrams to give a graphical representation of the results. Figure 13 shows the results for shear forces and moments while figure 14 shows the displacements and rotations at each node.

Element Number	F1(V1)	F2(M1)	F3(V2)	F4(M2)
1	208333.3443	0.0055	-208333.3443	208333.3388
2	208333.3327	-208333.3336	-208333.3327	416666.6663
3	208333.3450	-416666.6608	-208333.3450	625000.0059
4	208333.3170	-625000.0081	-208333.3170	833333.3251
5	208333.3396	-833333.3303	-208333.3396	1041666.6699
6	208333.3294	-1041666.6686	-208333.3294	1249999.9980
7	208333.3242	-1250000.0046	-208333.3242	1458333.3288
8	208333.3454	-1458333.3273	-208333.3454	1666666.6727

Figure 13: Results for shear forces and moments.

Node Number	Displacements	Rotation
1	0.00000000	-0.00058333
2	-0.00058125	-0.00057708
3	-0.00115000	-0.00055833
4	-0.00169375	-0.00052708
5	-0.00220000	-0.00048333
6	-0.00265625	-0.00042708
7	-0.00305000	-0.00035833
8	-0.00336875	-0.00027708
9	-0.00360000	-0.00018333
10	-0.00373125	-0.00007708
11	-0.00375000	0.00004167
12	-0.00364875	0.00015792
13	-0.00344000	0.00025667
14	-0.00314125	0.00033792
15	-0.00277000	0.00040167
16	-0.00234375	0.00044792
17	-0.00188000	0.00047667
18	-0.00139625	0.00048792
19	-0.00091000	0.00048167
20	-0.00043875	0.00045792
21	0.00000000	0.00041667

Figure 14: Results for displacements and rotations.

3.2 Conclusions

The comparison of the MATLAB results with the results from ABAQUS show that the developed code is correct and efficient for the analysis of beams subjected to static loads. The code is able to carry out the complex analysis within seconds with great accuracy. The plots from ABAQUS also validate the plots given by the code on MATLAB. Figure 15 and 17 show the SFD and BMD respectively from MATLAB and the figures 16 and 18 show SFD and BMD respectively from ABAQUS. Figure 19 shows the comparison of results.

There is an average difference of just 0.05%, which is very small and negligible.

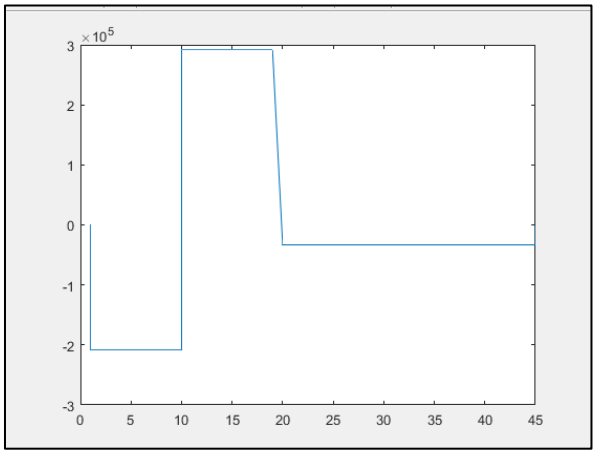


Figure 15: SFD plot from MATLAB.

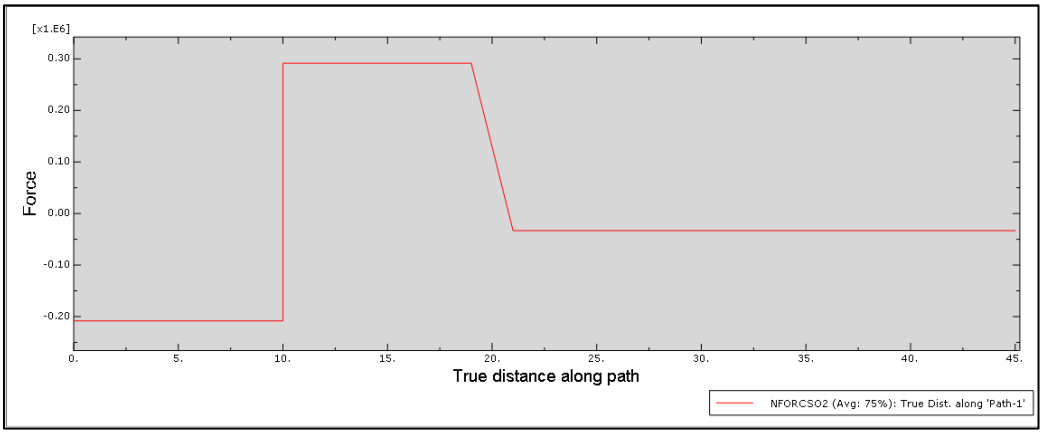


Figure 16: SFD plot from ABAQUS.

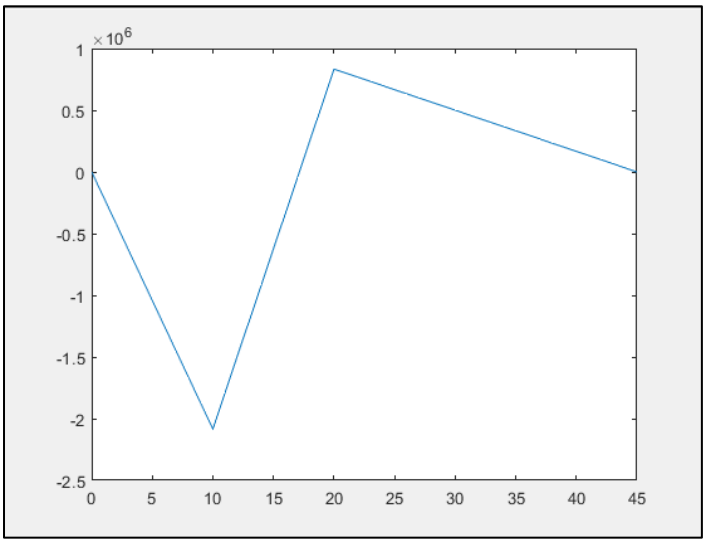


Figure 17: BMD plot from MATLAB.

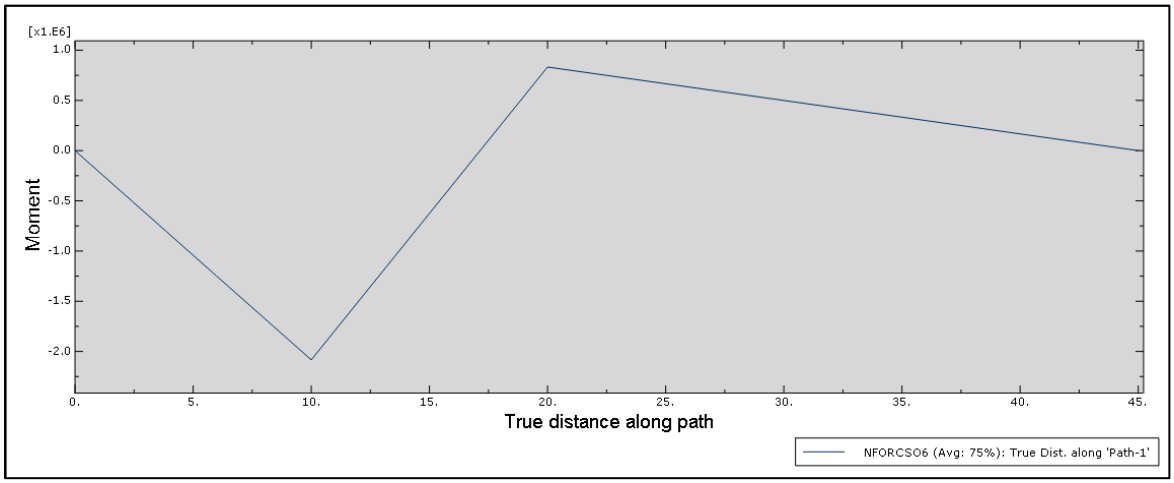


Figure 18: BMD plot from ABAQUS.

Reactions from MATLAB	Reactions from ABAQUS	Percentage change
208333	208374	0.02
324999	324927	0.02
-33333	-33301	0.1
		Avg. change = 0.05%

Figure 19: Comparison of results.

3.3 Recommendation

In its current form, the code is not suitable for the analysis of steel frames subjected to dynamic loading. However, with this research, a solid foundation has been laid for the aforementioned. With some minor changes this code can be used for the analysis of beams subjected to dynamic loading and going a step further, it can be improved for the analysis of steel frames subjected to dynamic loading. Therefore, this research should be carried further to achieve this goal.

REFERENCES

- Fundamentals of Finite Element Analysis by David V. Hutton-1st Edition.
- SHOBHA P, G RAVI and GOURAV K, 2018. DYNAMIC ANALYSIS AND TESTING OF STEEL AND WOODEN FRAMES. International Journal of Civil Engineering and Technology (IJCIET). Volume:9, Issue:6, Pages:264-272.
- Member Discrete Element Method (MDEM) for Static and Dynamic Responses, Analysis of Steel Frames with Semi-Rigid Joints by Jihong Ye and Lingling Xu. (Article published on 11th July 2017).
- Sampling points and weights for Gaussian Quadrature by Mike "Pomax" Kamermans, uploaded on June 5th, 2011- (https://pomax.github.io/bezierinfo/legendre-gauss.html).
- Introduction to Finite Element Analysis Using MATLAB and ABAQUS by Amar Khennane.
- “Improving the Response of Steel Structures Subjected to Blast Loading” by “Omid Khandel”, submitted to the graduate faculty of “The University of Toledo” (August 2016).
- “SAFETY ANALYSIS OF STEEL BUILDING FRAMES UNDER DYNAMIC WIND LOADING” by “NIRMAL K. DAS”. A dissertation in civil engineering submitted to “The Graduate Faculty of Texas Tech University”.

APPENDIX

MATLAB CODE

This appendix includes the main MATLAB code along with the different functions necessary for the two-dimensional analysis of determinate and indeterminate beams subjected to static loading.

Main Code:

```
% Script File: Find The stiffness matrix
%
%Purpose: To carry out dynamic analysis of a beam
clear
clc
close all
%
%  Global Variable
%
global Elem Stiffmat
%
%  Input Data
%
Len = 45;              %meter
nel = 45;              % number of elements
nnel=2;                % number of nodes per element
ndof = 2;
E = 200e+9;            %N/m^2
I = 1/12;          %m^4
area=1;            % cross-sectional area of the beam
```

```
rho=1;            % mass density ( not used for static probelms)
ngl = 5;              %input('integration points')
%----------------------
% Initialise matrices
%----------------------
gcoord = zeros (nel+1,1); %initialize the nodal coordinate array
nodes = zeros (nel,2);  %initialize the nodal connectivity array
%
nnode = (nnel-1)*nel+1; % total number of nodes in system
sdof = nnode*ndof; % total system dofs
kk=zeros(sdof,sdof);
index=zeros(nnel*ndof,1); %initialization of index vector
ff = zeros(sdof,1);     %initialize nodal force vector
fix = zeros(sdof,1);     %initialize nodal force vector
% Element degree of freedom- for member loads
%for i = 1:1:nel
 %   Eldof(i)=zeros(nnel*nnel);
%end
 Eldof=cell(nnel*nnel,1);
%
%----------------------
% Material and Geometrical Properties
%----------------------
%
for i = 1:1:nel
    E(i) = 200e+9;
    I (i)= 1/12;
end
```

```
%E(1) = 200e+9;    I (1)= 1/12;        %N/m^2  m^4
%E(2) = 200e+9;    I (2)= 1/12;        %N/m^2  m^4
%E(3) = 200e+9;    I (3)= 1/12;        %N/m^2  m^4
%E(4) = 200e+9;    I (4)= 1/12;        %N/m^2  m^4
%E(5) = 200e+9;    I (5)= 1/12;        %N/m^2  m^4
%---------------------------------
% Data for nodal force values
%---------------------------------
%ff(21)=-500000;  % applied force
ff(21)=-500000;
%---------------------------------
% Data for fix end forces
%---------------------------------
fix(1)=0;  % applied force
%
%---------------------------------
% Data for nodal coordinate values
%---------------------------------
%
j = 0;
i = 1;
%
while j <= Len
        gcoord(i) = j;
        i = i+1;
        j = j+ Len/nel;
end
%----------------------------------------
```

```
% Nodal connectivity for each element
%----------------------------------------
%
for i = 1:1:nel
    nodes(i,1) = i;
end
for i = 1:1:nel
    nodes(i,2) =1+i;
end
%----------------------------------------
%
%----------------------------------------
% Input Data for Boundary Conditions
%----------------------------------------
%
bcdof(1) = 1; % first dof
bcval(1) = 0; % value of the constrained node
bcdof(2) = 41; % second constrained node
bcval(2) = 0; % value of constrained node
bcdof(3) = 91; % second constrained node
bcval(3) = 0; % value of constrained node
%
%----------------------------------------
% Computation of element matrices and vectors and their assembly
%----------------------------------------
%
for iel = 1:nel   % loop for the total number of elements
nl=nodes(iel,1); nr = nodes(iel,2); % extract nodes for (iel)-th element
```

```
xl = gcoord(nl); xr=gcoord(nr); % extract nodal coord values
eleng = xr-xl;  % element length
%
index = sysdof(iel,nnel,ndof); % extract system dofs associated
%
Eldof(iel)={index};
%
[point1, weight1]=feglqd1(ngl); % extract sampling points and weights
%
k=feode2l(point1,weight1,ngl,E(iel),I(iel),eleng); % compute element matrix
%
[kk]=feasmbl2a(kk,k,index); %assemble element matrices
%
%kk
end
%-----------------------------------------------
% apply boundary conditions
%-----------------------------------------------
[kk,ff]=feaplyc2(kk,ff,bcdof,bcval);
%
u=inv(kk)*(ff-fix); % displacements
%
A=cell2mat(Eldof);
%
kt=1
```

```
% Print the title of the table.
fprintf(' Element Forces\n\n');

fprintf(' Element Number     F1(V1)              F2(M1)            F3(V2)              F4(M2)\n');
fprintf('  ==============       =======                    ======                    ======
======\n');
%
j=1;
%
for i = 1:1:nel
B=[fix(A(i,1)) fix(A(i,2)) fix(A(i,3)) fix(A(i,4))]'+...
    k*[u(A(i,1)) u(A(i,2)) u(A(i,3)) u(A(i,4))]';
B';
out = [i' B(j)' B(j+1)' B(j+2)' B(j+3)'];
fprintf (' %4d %33.4f %23.4f %23.4f %22.4f\n',out)
end
%
% Print the title of the table.
fprintf(' \n');
fprintf(' Nodal displacements\n\n');
fprintf(' Node Number     Displacements        rotation\n');
fprintf(' ============     =============   ==========\n');
%
```

```
%
j=1;
for i = 1:1:nnode
out = [i' u(j)' u(j+1)'];
fprintf (' %4d %33.8f  %22.8f\n',out)
j=j+2;
end
%
```

Function of System Degrees of Freedom:

```
function[index]=sysdof(iel,nnel,ndof)
%----------------------------------------
% Purpose:
%Compte system dofs associated with each element in
% once dimensional problem
%
% Synopsis
% [index] = feeldof1 (iel,nnel,ndof)
%
% variable Description:
% index - system dof vector associated with element iel
% iel - element number whose system dofs are to be determined
% nnel - number of nodes per element
```

```
% ndof - number of dofs per node
%
%-------------------------------------------
%
edof=nnel*ndof;
start=(iel-1)*(nnel-1)*ndof;
for i=1:edof
   index(i)=start+i;
end
```

Function for Sampling Points and Weights for Gaussian Quadrature:

```
function [point1,weight1]= feglqd1(ngl)
% Purpose:
% determine the integration points and weighting coefficients
% of Gauss-Legendre quadrature for one-dimensional integration
%
%Synopsis:
%[point1,weight1]=feglqd1(ngl)
%
% Variable Description:
%ngl - number of integration points
% point1 - vector containing integration points
% weight1 - vector containing weighting coefficient
% ---------------------------------------------------------
%
% initialization
%
```

```
point1 = zeros (ngl,1);
weight1 = zeros (ngl,1);
%
% find corresponding integration points and weights
%
if ngl == 1    % 1-point quadrature rule
point1(1) = 0.0;
weight1(1) = 2.0;
%
elseif ngl == 2 % 2-point quadrature rule
point1(1) = -0.577350269189626;
point1(2) = -0.577350269189626;
weight1(1) = 1.0;
weight1(2) = weight1(1);
%
elseif ngl == 3 % 3-point quadrature rule
point1(1) = -0.774596669241483;
point1(2) = 0.0;
point1(3) = point1(1);
weight1(1) = 0.555555555555556;
weight1(2) = 0.888888888888889;
weight1(3) = weight1(1);
%
elseif ngl == 4 % 4-point quadrature rule
point1(1) = -0.861136311594053;
point1(2) = -0.33981043584856;
point1(3) = -point1(2);
point1(4) = -point1(1);
```

```
weight1(1) = 0.347854845137454;
weight1(2) = 0.652145154862546;
weight1(3) = weight1(2);
weight1(4) = weight1(1);
%
else        % 5-point quadrature rule
point1(1) = -0.906179845938664;
point1(2) = -0.538469310105683;
point1(3) = 0.0;
point1(4) = -point1(2);
point1(5) = -point1(1);
weight1(1) = 0.236926885056189;
weight1(2) = 0.478628670499366;
weight1(3) = 0.568888888888889;
weight1(4) = weight1(2);
weight1(5) = weight1(1);
%
end
%
%-----------------------------------------
```

Function for Integration of Entries of K-Matrix:

```
function [k]=feode2l(point1, weight1,ngl,E,I,eleng)
%---------------------------------
% Purpose:
% element matrix for stiffness using
```

```
% Euler Bernouli Beam element
%
% Synopsis:
%
% Gauss-Legendre quadrature of a function in 1-dimension
%
%Variable descriptions
% point1 = integration (or sampling) points
% weight1 = weighting coefficients
% ngl = number rof integration points
%ngl = 5;   % (2*ngl-1)=5
%[point1, weight1]=feglqd1(ngl); % extract sampling points and weights
%E=1;
%I=1;
%eleng=1;
%
% ----------------------------
% summation for numerical integration
% ----------------------------
%
value=0.0;
%
for int=1:ngl
   x=point1(int);
   wt=weight1(int);
   k11=18*(1+x)^2 - 36*(1+x)+18;
   value=value+k11*wt;
end
```

```
%
k11=value;
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k22=18/4*(1+x)^2 - 24/2*(1+x)+8;
    value=value+k22*wt;
end
%
k22=value*eleng*eleng;
%
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k33=72/4*(1+x)^2 - 72/2*(1+x)+18;
    value=value+k33*wt;
end
%
k33=value;
%
%
value=0.0;
```

```
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k44=9/2*(1+x)^2 - 12/2*(1+x)+4/2;
    value=value+k44*wt;
end
%
k44=value*eleng*eleng;
%
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k12=72/8*(1+x)^2 -84/4*(1+x)+24/2;
    value=value+k12*wt;
end
%
k12=value*eleng;
k21=k12;
%
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
```

```
    wt=weight1(int);
    k13=-144/8*(1+x)^2 +144/4*(1+x)-36/2;
    value=value+k13*wt;
end
%
k13=value;
k31=k13;
%
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k14=72/8*(1+x)^2 -60/4*(1+x)+12/2;
    value=value+k14*wt;
end
%
k14=value*eleng;
k41=k14;
%
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k23=-72/8*(1+x)^2 +84/4*(1+x)-24/2;
```

```
    value=value+k23*wt;
end
%
k23=value*eleng;
k32=k23;
%
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k24=36/8*(1+x)^2 -36/4*(1+x)+8/2;
    value=value+k24*wt;
end
%
k24=value*eleng*eleng;
k42=k24;
%
%
value=0.0;
%
for int=1:ngl
    x=point1(int);
    wt=weight1(int);
    k34=-72/8*(1+x)^2 +60/4*(1+x)-12/2;
    value=value+k34*wt;
end
```

```
%
k34=value*eleng;
k43=k34;
%
k=(E*I/eleng^3)*[k11 k12 k13 k14;k21 k22 k23 k24;
    k31 k32 k33 k34;k41 k42 k43 k44];
```

Function for Assembling Local Element Matrices to Form a Global K-Matrix:

```
function[kk]=feasmbl2(kk,k,index)
%--------------------------------------
% Assembly of element matrixes into the system matrix and
% Assembly of element vectors into the system vector
%
% Synopsis:
%[kk]=feasmbl2(kk,k,index)
%
%Variable Description:
%kk,mm- system matrices
%k,m - element matrices
%
%index - d.o.f. vector associated with an element
%--------------------------------------
%
edof=length(index); % total length of index array
```

```
for i = 1:edof
   ii=index(i);
   for j = 1:edof
      jj = index(j);
      kk(ii,jj)=kk(ii,jj)+k(i,j);
   end
end
```

Function for the Application of Boundary Conditions:

```
function [kk,ff]=feaplyc2(kk,ff,bcdof,bcval)
%--------------------------------------------
% Purpose
% Apply constraints to matrix equation
%
%Synopsis:
%[kk]=feaplybc(kk,bcdof,bcval)
%kk- system matrix before applying constrains
% bcdof - a vector containing constrained d.o.f
% bcval - a vector containing constrained value
%
%--------------------------------------------
%
n=length(bcdof);
sdof=size(kk);
%
```

```
for i=1:n
    c=bcdof(i);
    for j=1:sdof
        kk(c,j)=0;
    end
    %
     kk(c,c)=1;
    ff(c)=bcval(i);
end
```

Function for Plotting of Results:

```
clear all;
close all;
clc

A=dlmread('file_having_values_of_forces.txt');
x=A(:,1);
y=A(:,2);
plot(x,y)
```