IIT GANDHINAGAR

MECHANICS OF DEFORMABLE BODIES

ME 321

Deflection of Beams

Authors:
Mihir SALOT *15110072*
Rushali SAXENA *15110108*
Saurav NAGAR *15110117*
Shashimohan SINGH *15110120*

Contents

1 Introduction

A beam is one of the structural element which resists loads applied laterally to the beam's axis.The deformation of a beam is usually expressed in terms of its deflection from its original unloaded position.Deflection is defined as the vertical displacement of a point on a loaded beam.

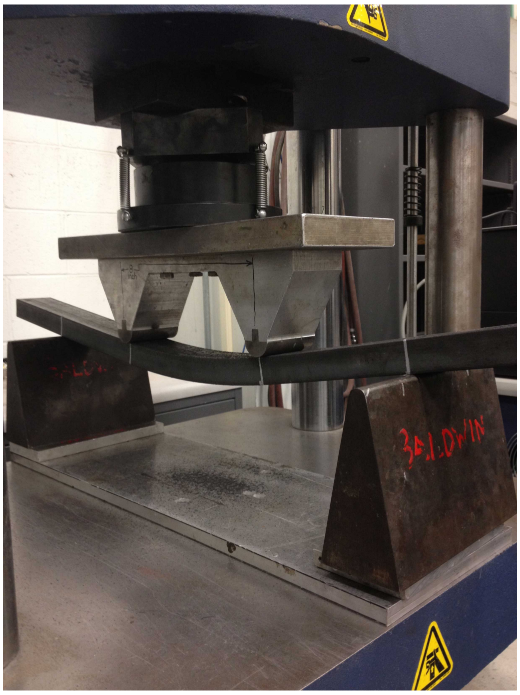

Figure 1: Deflection in a Beam

2 Objective

Our objective is to design a basic graphic user interface (GUI) using Matlab which take parameters such as dimensions, elasticity, joints, loads and moments on a beam and provides bending moment diagram, shear force diagram, deflection function along length of beam and comparison between deflections at different boundary conditions.

2.1 Assumptions and Limitations

During the validation process, we tested the test cases only for discretization(ϵ =0.05. Therefore, any increment greater than that holds validity can't be said forth. The users do not get very adventurous and follow the user guide written. The coordinates are expected to be written in ascending order of their appearance. The origin is the start of beam.

2.2 Basic Approach

We applied the finite element method to obtain the algorithms and superposition method for handling beam subjected to uniformly distributed loads. Using one finite element, we can get exact solution. The problem with the finite element formulation that we developed is – it gives exact solutions for deflections.

3 Algorithm and Design

- **For all pinned supports:** We have n unknown forces at supports and moment equilibrium equations at n supports.These leaves us with determinate system of equations where forces can be solved using matrices.
- For n pinned supports and clamped support at the end: we have n+2 unknowns and $(n+1)$ moment equilibrium equations.This is indeterminate system.We assume moment at clamps to be known (M_0), and solve equations to get support forces in terms of M_0.
- For n pinned supports and clamped supports at either ends: we have $(n+4) unknowns and$(n+2) moment equilibrium equations.This is indeterminate system.We assume moment at clamps to be known (M_0,M_1), and solve equations to get support forces in terms of M_0 and M_1. We use the forces to get a bending moment diagram,calculate deflections across beams.This gives us deflection function in case of pinned joints.However in case of clamps our deflection function is in terms of moments we assumed as constants.

  To solve indeterminate systems of equations we use the fact that slopes at clamped ends must be zero.
  For case 2:(1 clamp) This gives us one boundary condition to solve for M_0 .
  For case 3:(2 clamps) This gives us two boundary conditions to solve for M_0 and M_1.
  Once we have values we normalize the deflections graph with respect to maximum deflection and plot it, to get an idea of coordinates at which beam suffers maximum deflections.

4 User Manual

4.1 Prerequisites

Need symbolic toolbox to run the code. Preferably run it in R-2017 version of MATLAB. Ensure that the cases you enter are for fixed ends.

4.2 Steps

- Open the program Beams.m and run it. A Window pops up will open up.
- The GUI takes in input in the specified dimensions. Principal axis 1 denotes dimensions of the main axis which represents the dominating dimension for area of inertia. The elasticity modulus is to be taken in GPa.
- The input is expected in an increasing order of loading from left to right. Clamped joints can be put only at the ends. Pinned joints are represented by ‘P’ while clamped are to be denoted by ‘C’. The applied force is taken positive in upper direction,and bending moment is positive in clockwise direction .
- I cross-section moment of inertia can also be found out by using Inertia of I beam function. For circular and rectangular cross-sections, you can use Inertia of Circle and Inertia of Rectangle function respectively. Provide the relevant parameters as explained.
- The window consists of following elements:
  Geometric properties:
  The Dropdown menu is to select the type of cross sections:
  Square, Rectangle, Circle. I cross-section moment of inertia can also be found out by using the Inertia of I beam function. For circular and rectangular cross-sections, you can use Inertia of Circle and Inertia of Rectangle function respectively. Provide the relevant parameters as explained.
- Principal value 1 depends on the cross section selected. In case of square, it is the edge length. For rectangular, it is the breadth (b) of the cross section. For circle, it is the radius. Principal value 2 is to be filled when there is rectangular cross section. Modulus of elasticity and Length of beam are to be entered as required. The xcoordinates MUST be entered from the left end of the beam.
- The code involving fixed ends will work only if the coordinate system is fixed such that a fixed end is the origin.
- Number of supports must be greater than 2 while entering.
- Variables V-func, M-func, slope-func, def-func provides an array of expressions in terms for shear force, bending moment, slope and deflection in terms of t. These can be accessed in the command window of Matlab.
- Variables xchanges-V, xchanges-M, xchanges-slope, xchanges-def refer to the points where the shear force, bending moment, slope and deflections functions change their values respectively. How to use these functions?
- Suppose you access ith element in the array of expressions. Say call it g. Then if you wish to compute value at x, the particular expression is used if xchanges(i) <= g(x) < xchanges(i+1).
  In these expressions, ‘t’ is the xcoordinate of the point to be considered. The inputs must be given in the order of their xcoordinates

- Obtaining plots from these functions: sfd-from-func, bmd-from-func, slope-d-from-func, def-d-from-func are the functions used to obtain plots of shear forces, bending moment, slope and deflections from the expressions form of these. Required parameters must be provided.
  Accuracy of plots can be improved by decreasing the value of e used in these functions.
- Alternative method to run the code is to enter values through Matlab command window. The program to run depends on the type of forces scenario you are observing.
  All pin joints: Run defbeam-allpins program
  Fixed joints at the ends: Run defbeam-bothfixed program
  Fixed-pin joints: Run defbeam-fixed-pin program
  In case there are roller joints, run the programs considering there are pins (As we have considered loading only about axis perpendicular to the beams).
- If the code returns value of NaN, try to replace the values of modulus of elasticity without multiplying by 10^9 in the Matlab command method of input.
  If the program returns huge numbers in fractions, enable application of 'vpa()' function in the programs of programs by de-commenting the comments.
  The code might give little difference from the values obtained due to restrictions of the computers representing numbers in decimal forms.
- Data about loadings is to be provided as per the dimensions asked for in the table. The maximum number of any type of loads is seven, since cases other than this are rarely experienced. However, this is not a limitation of the program. The program can take any number of values. Figure 2 and 3 depicts the input and output screen of our GUI. Apart from the bending, SFD and BMD can also be obtained.
- The final output shows shear force diagram, bending moment diagram,initial configuration of beam and final deflected beam. However deflections for the beams have been normalized with respect to maximum value.

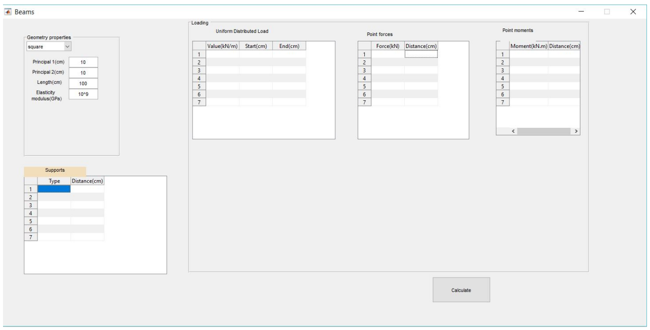

Figure 2: GUI designed for taking inputs

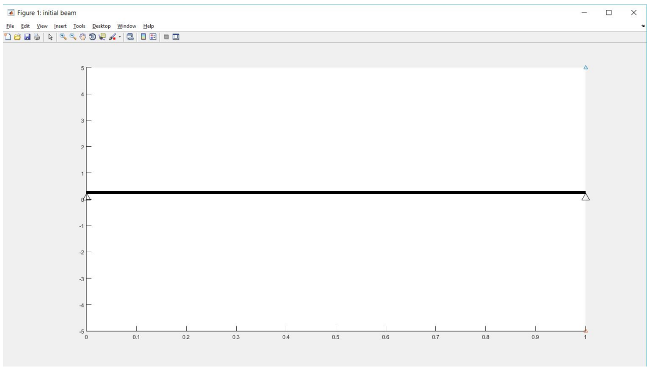

Figure 3: Output Screen 1

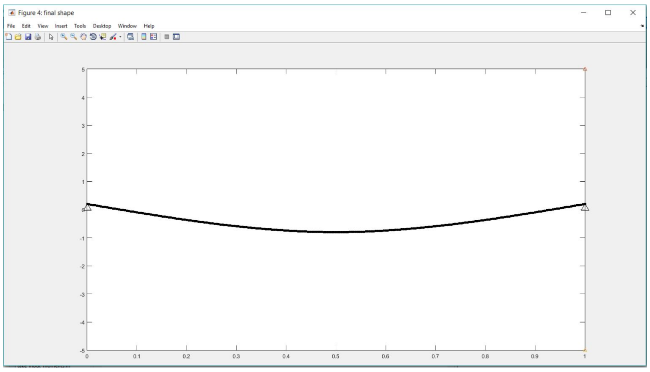

Figure 4: Output Screen 2

5 Validation

We tried focusing on the following objectives while designing our project.

- To check whether final module meets to the expectations.
- To test for actual application.
- Involves executing the code.
- To detect, correct and report the defects.

5.1 Test Cases

The validation was done for the combination of following criteria. In total, a total of 49 test cases were obtained.
Cases Possible for types of Supports:

- Pin – Pin

- Fixed – Fixed
- Roller – Pin
- Roller – Fixed
- Pin – Fixed
- Fixed – Free (Cantilever)

Cases possible for Loadings:

- Point loads (PL)
- Uniformly Distributed loads (UDL)
- Moments (M)
- PL + UDL
- PL + M
- UDL + M
- PL + UDL + M

Cases for cross-section:

- Rectangular
- I section
- Circular

5.2 Testing Methodology

The test cases were solved for the cases considered Using a FEA solver. Simultaneously, we also solved for the cases considered Using our developed module. We compared the analytical solutions for all the cases considered. While comparing focused on support reactions, SFD and BMD, and critical values, deflections at critical points and maximum deflection was analyzed.

5.3 Results and Deviations

Our validation has been done for the following cases and the code is found to be working properly for all kinds of loading scenarios with supports only at the ends of the beam:
(1) Pin - Pin support
(2) Pin - Roller support
(3) Fixed - Pin support
(4) Fixed - Roller support
(5) Fixed - Fixed support
Our validation has also been done for following cases but code was found NOT working properly for any kind of loading scenarios:
(1) Fixed - Free support
(2) For supports at any position other than ends of the beam
(3) For more than two supports
All the validation done and result obtained has been added in **Appendix B** as list of figures.

6 Appendix A - Matlab Codes

Matlab Codes

In this appendix, the different Matlab fragments used by us are provided. A brief description of the function of each code has also been given at the beginning of the code which explains the basic function of each code.

A| Code used for GUI (Front-end)

The following Matlab code was used to create a GUI.The user needs to input certain inputs in the GUI input to get the desired output.

```
function varargout = Beams(varargin)
% BEAMS MATLAB code for Beams.fig
%       BEAMS, by itself, creates a new BEAMS or raises the existing
%       singleton*.
%
%       H = BEAMS returns the handle to a new BEAMS or the handle to
%       the existing singleton*.
%
%       BEAMS('CALLBACK',hObject,eventData,handles,...) calls the local
%       function named CALLBACK in BEAMS.M with the given input arguments.
%
%       BEAMS('Property','Value',...) creates a new BEAMS or raises the
%       existing singleton*.  Starting from the left, property value pairs are
%       applied to the GUI before Beams_OpeningFcn gets called.  An
%       unrecognized property name or invalid value makes property application
%       stop.  All inputs are passed to Beams_OpeningFcn via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Beams

% Last Modified by GUIDE v2.5 09-Nov-2017 15:02:36

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Beams_OpeningFcn, ...
                   'gui_OutputFcn',  @Beams_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```


% --- Executes just before Beams is made visible.
function Beams_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Beams (see VARARGIN)

% Choose default command line output for Beams
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

set(handles.uitable1,'Data',cell(7,2));
set(handles.uitable3,'Data',cell(7,3));
set(handles.uitable4,'Data',cell(7,2));
set(handles.uitable5,'Data',cell(7,2));

% UIWAIT makes Beams wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Beams_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in xyz.
function xyz_Callback(hObject, eventdata, handles)
cont=cellstr(get(hObject,'String'));
shape=cont(get(hObject,'Value'));
if(strcmp(shape,'square'))
    shapeval=1;
elseif(strcmp(shape,'circle'))
    shapeval=2;
elseif (strcmp(shape,'rectangle'))
    shapeval=3;
end
assignin('base','shape',shapeval) ;

% hObject    handle to xyz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns xyz contents as cell array
%        contents{get(hObject,'Value')} returns selected item from xyz



% --- Executes during object creation, after setting all properties.
```

```
function xyz_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xyz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p1_Callback(hObject, eventdata, handles)
% hObject    handle to p1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p1 as text
%        str2double(get(hObject,'String')) returns contents of p1 as a double


% --- Executes during object creation, after setting all properties.
function p1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function p2_Callback(hObject, eventdata, handles)
% hObject    handle to p2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of p2 as text
%        str2double(get(hObject,'String')) returns contents of p2 as a double


% --- Executes during object creation, after setting all properties.
function p2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function l_Callback(hObject, eventdata, handles)
% hObject    handle to l (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of l as text
%        str2double(get(hObject,'String')) returns contents of l as a double


% --- Executes during object creation, after setting all properties.
function l_CreateFcn(hObject, eventdata, handles)
% hObject    handle to l (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function e_Callback(hObject, eventdata, handles)
% hObject    handle to e (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of e as text
%        str2double(get(hObject,'String')) returns contents of e as a double


% --- Executes during object creation, after setting all properties.
function e_CreateFcn(hObject, eventdata, handles)
% hObject    handle to e (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    P1=str2double(get(handles.p1,'string'));
    P2=str2double(get(handles.p2,'string'));
    L=str2double(get(handles.l,'string'));
    E=str2double(get(handles.e,'string'));

    Supports_data=get(handles.uitable1,'Data')
    S_x=Supports_data(:,2)
    S_type=Supports_data(:,1)

```

```
    S_x(cellfun('isempty',S_x)) = [];
    S_x=str2double(S_x)

    S_type(cellfun('isempty',S_type)) = []

    UDL_data=get(handles.uitable3,'Data');
    UDL_value=UDL_data(:,1);
    UDL_start=UDL_data(:,2);
    UDL_end=UDL_data(:,3);

    UDL_value(cellfun('isempty',UDL_value)) = [];
    UDL_value=str2double(UDL_value)

    UDL_start(cellfun('isempty',UDL_start)) = [];
    UDL_start=str2double(UDL_start)


    UDL_end(cellfun('isempty',UDL_end)) = [];
    UDL_end=str2double(UDL_end)

    PF_data=get(handles.uitable4,'Data');

    PF_value=PF_data(:,1);
    PF_value(cellfun('isempty',PF_value)) = [];
    PF_value=str2double(PF_value)

    PF_x=PF_data(:,2);
    PF_x(cellfun('isempty',PF_x)) = [];
    PF_x=str2double(PF_x)

    PM_data=get(handles.uitable5,'Data');

    PM_value=PM_data(:,1);
    PM_value(cellfun('isempty',PM_value)) = [];
    PM_value=str2double(PM_value)

    PM_x=PM_data(:,2);
    PM_x(cellfun('isempty',PM_x)) = [];
    PM_x=str2double(PM_x)
% --- Executes during object creation, after setting all properties.
function uitable1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% --- Executes when entered data in editable cell(s) in uitable1.
function uitable1_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
%       Indices: row and column indices of the cell(s) edited
%       PreviousData: previous data for the cell(s) edited
%       EditData: string(s) entered by the user
%       NewData: EditData or its converted form set on the Data property. Empty if Data was not
    ↪ changed
%       Error: error string when failed to convert EditData to appropriate value for Data
% handles    structure with handles and user data (see GUIDATA)

% --- Executes when selected cell(s) is changed in uitable1.
function uitable1_CellSelectionCallback(hObject, eventdata, handles)
% hObject    handle to uitable1 (see GCBO)
```

```
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
%       Indices: row and column indices of the cell(s) currently selecteds
% handles    structure with handles and user data (see GUIDATA)
     %data=get(hObject,'data');
     %v=data;
     %retreivedata(2,1)
% --- Executes when entered data in editable cell(s) in uitable3.
function uitable3_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to uitable3 (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
%       Indices: row and column indices of the cell(s) edited
%       PreviousData: previous data for the cell(s) edited
%       EditData: string(s) entered by the user
%       NewData: EditData or its converted form set on the Data property. Empty if Data was not
    ↪ changed
%       Error: error string when failed to convert EditData to appropriate value for Data
% handles    structure with handles and user data (see GUIDATA)
% --- Executes when entered data in editable cell(s) in uitable4.
function uitable4_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to uitable4 (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
%       Indices: row and column indices of the cell(s) edited
%       PreviousData: previous data for the cell(s) edited
%       EditData: string(s) entered by the user
%       NewData: EditData or its converted form set on the Data property. Empty if Data was not
    ↪ changed
%       Error: error string when failed to convert EditData to appropriate value for Data
% handles    structure with handles and user data (see GUIDATA)
% --- Executes when entered data in editable cell(s) in uitable5.
function uitable5_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to uitable5 (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
%       Indices: row and column indices of the cell(s) edited
%       PreviousData: previous data for the cell(s) edited
%       EditData: string(s) entered by the user
%       NewData: EditData or its converted form set on the Data property. Empty if Data was not
    ↪ changed
%       Error: error string when failed to convert EditData to appropriate value for Data
% handles    structure with handles and user data (see GUIDATA)
```

B| Codes for Computing (Back-end)

B.1 Main Functions

These are the main functions used to compile the whole back-end code fragments.

B.1.1 Clamped Joint Case

This function handles the Clamped joined cases for the parameters provided.

```
syms t; %Need symbolic toolbox for function to work
syms a;
syms b; %for unknown moments at ends
addPrompt = ' :  '; %For changing characters added in front of prompts for taking input
e = 0.01; %For chaning the least value of x while plotting graphs
E = 0; %stores the modulus of elasticity of the beam
```

```
I = 0; %stores the calculated value of inertia

l = 0; %length of beam

ns = 0; %number of supports
np = 0; %number of point forces supplied externally
nw = 0; %number of different magnitudes of distributed loads
nm = 0; %number of moments supplied externally

m_net_applied = 0; %stores net sum of the external moments applied

xs = []; %stores x coordinates of supports with respect to origin
xs_new = []; %stores new coordinates after including the boundary condition
xpf = []; %stores x coordinate of point forces externally applied with respect to origin
xm = []; %stores x coordinates of moments applied

xsw = []; %stores start coordinates of each different distributed load range
xew = []; %stores end coordinates of each different distributed load range

xwnet = []; %stores the net x coordinate at which the force can be considered to act

X_pf = []; %difference of every point from every other consecutive point, for point forces
X_w = []; %difference of every point from every other consecutive point, for distributes forces
X_s = []; %difference of every point from every other consecutive point, for distributes forces


pf = []; %stores magnitudes of point forces acting
m = []; %stores magnitudes of moments applied externally
w = []; %stores magnitudes of distributed loads
wnet = []; %stores values of effective loads acting because of the distributed loads

sf = []; %stores the forces acting at the supports after being calculated

V_func = []; %stores the expression form of shear force (variable used is t)
M_func = []; %stores the expression form of bending moment (variable used is t)
slope_func = []; %stores the expression form of slope of deflection (variable used is t)
def_func = []; %stores the expression form of slope of deflection (variable used is t)

xchanges_V = []; %coordinates where shear function changes its definition x(i)<= x < x(i+1)
xchanges_M = []; %coordinates where bending moment function changes its definition
xchanges_slope = []; %coordinates where slope of deflection function changes its definition
xchanges_def = []; %coordinates where deflection function changes its definition

ch = 0; %for taking choice of the input

disp('Hello, welcome to DefBeam input');
disp('Please enter the parameters as asked in SI units for a SYMMETRICAL beam');

%Taking input modulus of elasticity of the beam
prompt = 'Enter modulus of elasticity of beam';
E = input(strcat(prompt,addPrompt));
E = ensure_input_number(E,prompt, addPrompt);

%Taking input inertia of the beam
prompt = 'Enter I of beam';
I = input(strcat(prompt,addPrompt));
I = ensure_input_number(I,prompt, addPrompt);

% Taking input length of beam
```

```
prompt = 'Length of beam';
l = input(strcat(prompt,addPrompt));
l = ensure_input_number(l,prompt, addPrompt);


%Taking input: number of supports
%These supports will be fixed joints (different from end points)
disp('All supports are fixed joints');
prompt = 'Number of supports';
ns = input(strcat(prompt,addPrompt));
ns = ensure_input_number(ns,prompt,addPrompt);


%Taking input: x coordinate of joints
disp('Specifying coordinates of points, that is, fixed joints');
disp('Origin is the beginning of the beam');

for i=1:ns
    prompt = 'Enter coordinate of support ';
    prompt = strcat(prompt,num2str(i)); %To add support number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    xs = [xs, x];  %Appending number to the array

end

%%%%%%%%%%%%% POINT FORCES %%%%%%%%%%%%%%%%%%%%%%
disp('Time to enter point Forces!');
%Taking input: number of point forces
%These forces are applied externally
disp('Point forces are applied externally, upward direction being positive');
prompt = 'Number of point forces';
np = input(strcat(prompt,addPrompt));
np = ensure_input_number(np,prompt,addPrompt);

[xpf, pf] = take_input_pL(np,addPrompt);

%%%%%%%%%%%%%%%  DISTRIBUTED LOAD  %%%%%%%%%%%%%%%%%%%%%%%
disp('Describe the distributed loads!');
%Taking input: number of different distributed loads
disp('Distributed loads, upward direction being positive');
prompt = 'Number of distributed loads';
nw = input(strcat(prompt,addPrompt));
nw = ensure_input_number(nw,prompt,addPrompt);

[xsw, xew, w] = take_input_dL(nw,addPrompt);

%%%%%%%%%%%%%   MOMENTS   %%%%%%%%%%%%%%%%%%%%%%
disp('Time to enter Moments supplied!');
%Taking input: number of moments supplied
%These moments are applied externally
disp('Moments are applied externally');
prompt = 'Number of moments supplied';
nm = input(strcat(prompt,addPrompt));
nm = ensure_input_number(nm,prompt,addPrompt);

[ xm, m ] = take_input_moments(nm,addPrompt);

```

```
%%%%%%%%%%%%%% CALCULATING NET DISTRIBUTED FORCES AND NET POINT OF APPLICATION %%%%%%%%%%%%%
for i=1:nw
    wnet = [wnet nForce_dL(w(i),xsw(i),xew(i))]; %appends the new w to the array
    xwnet = [xwnet nPoint_dL(w(i),xsw(i),xew(i))]; %appends the new  x-ccordinate to the array
end


%%%%%%%%%%%%%% CALCULATING SUPPORT FORCES %%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% FOR fixed JOINT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m_net_applied = sum(m)-a+b; %calculating net sum of the external moments applied
m = [-a m b];
xm = [0 xm l];
nm = nm+2;
[sf] = sf_fixed(ns,np,nw,xs,xpf,xwnet,pf,w,wnet,m_net_applied); %calculates support forces

%%%%%%%%%%%%%% FINDING SHEAR FORCE EXPRESSIONS %%%%%%%%%%%%%%%%%%%%%%%
[V_func, xchanges_V] = find_shear_func( l,ns,np,nw,xs,xpf,xsw,xew,sf,pf,w ); %finds the expressions and the values of x where shear force changes

%%%%%%%%%%%%%% FINDING BENDING MOMENT EXPRESSIONS %%%%%%%%%%%%%%%%%%%%
[ M_func, xchanges_M ] = find_moment_func( l,nm,xm,m,V_func,xchanges_V ); %finds the expressions and the values of x where bending moment changes

%%%%%%%%%%%%%% FINDING SLOPE AND DEFLECTION EXPRESSIONS %%%%%%%%%%%%%%%%%%
[ xchanges_def, def_func,xchanges_slope, slope_func ] = find_deflection_func( ns,xs,M_func, xchanges_M, E, I ); %finds the expressions and the values of x where function of deflection changes

%%%%%%%%%%%%%%%%%%%%%%%%% DONE IN TERMS OF VARIABLES a and b %%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%% INVOLVING VARIABLES %%%%%%%%%%%%%%%%%%%%%%

%Finding variables
[a, b] = find_fixedends_moments( slope_func, xchanges_slope, def_func, xchanges_def );

%substitute variables in arrays of expressions
sf = find_expressions_from_symbols(a, b, sf);
m = find_expressions_from_symbols(a, b, m);
V_func = find_expressions_from_symbols(a, b, V_func);
M_func = find_expressions_from_symbols(a, b, M_func);
slope_func = find_expressions_from_symbols(a, b, slope_func);
def_func = find_expressions_from_symbols(a, b, def_func);


%%%%%%%%%%%%%% PLOTTING GRAPHS %%%%%%%%%%%%%%%%
% sfd_from_func( l,e,V_func,xchanges_V );
% figure;
% bmd_from_func( l,e,M_func,xchanges_M );
% figure;
% slope_d_from_func( l,e,slope_func,xchanges_slope );
% figure;
% def_d_from_func( l,e,def_func,xchanges_def );
```

B.1.2 Pin Joint Case

This function handles the Pin Joint cases for the parameters provided.

```
syms t; %Need symbolic toolbox for function to work
syms a;
syms b; %for unknown moments at ends
addPrompt = ' :  '; %For changing characters added in front of prompts for taking input
e = 0.01; %For chaning the least value of x while plotting graphs
E = 0; %stores the modulus of elasticity of the beam
I = 0; %stores the calculated value of inertia

l = 0; %length of beam

ns = 0; %number of supports
np = 0; %number of point forces supplied externally
nw = 0; %number of different magnitudes of distributed loads
nm = 0; %number of moments supplied externally

m_net_applied = 0; %stores net sum of the external moments applied

xs = []; %stores x coordinates of supports with respect to origin
xs_new = []; %stores new coordinates after including the boundary condition
xpf = []; %stores x coordinate of point forces externally applied with respect to origin
xm = []; %stores x coordinates of moments applied

xsw = []; %stores start coordinates of each different distributed load range
xew = []; %stores end coordinates of each different distributed load range

xwnet = []; %stores the net x coordinate at which the force can be considered to act

X_pf = []; %difference of every point from every other consecutive point, for point forces
X_w = []; %difference of every point from every other consecutive point, for distributes forces
X_s = []; %difference of every point from every other consecutive point, for distributes forces


pf = []; %stores magnitudes of point forces acting
m = []; %stores magnitudes of moments applied externally
w = []; %stores magnitudes of distributed loads
wnet = []; %stores values of effective loads acting because of the distributed loads

sf = []; %stores the forces acting at the supports after being calculated

V_func = []; %stores the expression form of shear force (variable used is t)
M_func = []; %stores the expression form of bending moment (variable used is t)
slope_func = []; %stores the expression form of slope of deflection (variable used is t)
def_func = []; %stores the expression form of slope of deflection (variable used is t)

xchanges_V = []; %coordinates where shear function changes its definition x(i)<= x < x(i+1)
xchanges_M = []; %coordinates where bending moment function changes its definition
xchanges_slope = []; %coordinates where slope of deflection function changes its definition
xchanges_def = []; %coordinates where deflection function changes its definition

ch = 0; %for taking choice of the input

disp('Hello, welcome to DefBeam input');
disp('Please enter the parameters as asked in SI units for a SYMMETRICAL beam');

%Taking input modulus of elasticity of the beam
prompt = 'Enter modulus of elasticity of beam';
E = input(strcat(prompt,addPrompt));
E = ensure_input_number(E,prompt, addPrompt);
```

```

%Taking input inertia of the beam
prompt = 'Enter I of beam';
I = input(strcat(prompt,addPrompt));
I = ensure_input_number(I,prompt, addPrompt);

% Taking input length of beam
prompt = 'Length of beam';
l = input(strcat(prompt,addPrompt));
l = ensure_input_number(l,prompt, addPrompt);


%Taking input: number of supports
%These supports will be fixed joints (different from end points)
disp('All supports are fixed joints');
prompt = 'Number of supports';
ns = input(strcat(prompt,addPrompt));
ns = ensure_input_number(ns,prompt,addPrompt);


%Taking input: x coordinate of joints
disp('Specifying coordinates of points, that is, fixed joints');
disp('Origin is the beginning of the beam');

for i=1:ns
    prompt = 'Enter coordinate of support ';
    prompt = strcat(prompt,num2str(i)); %To add support number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    xs = [xs, x];  %Appending number to the array

end

%%%%%%%%%%%%% POINT FORCES %%%%%%%%%%%%%%%%%%%%%%%
disp('Time to enter point Forces!');
%Taking input: number of point forces
%These forces are applied externally
disp('Point forces are applied externally, upward direction being positive');
prompt = 'Number of point forces';
np = input(strcat(prompt,addPrompt));
np = ensure_input_number(np,prompt,addPrompt);

[xpf, pf] = take_input_pL(np,addPrompt);

%%%%%%%%%%%%%%%  DISTRIBUTED LOAD  %%%%%%%%%%%%%%%%%%%%%%%
disp('Describe the distributed loads!');
%Taking input: number of different distributed loads
disp('Distributed loads, upward direction being positive');
prompt = 'Number of distributed loads';
nw = input(strcat(prompt,addPrompt));
nw = ensure_input_number(nw,prompt,addPrompt);

[xsw, xew, w] = take_input_dL(nw,addPrompt);

%%%%%%%%%%%%%   MOMENTS   %%%%%%%%%%%%%%%%%%%%%%
disp('Time to enter Moments supplied!');
%Taking input: number of moments supplied
%These moments are applied externally
```

```
disp('Moments are applied externally');
prompt = 'Number of moments supplied';
nm = input(strcat(prompt,addPrompt));
nm = ensure_input_number(nm,prompt,addPrompt);

[ xm, m ] = take_input_moments(nm,addPrompt);


%%%%%%%%%%%%%% CALCULATING NET DISTRIBUTED FORCES AND NET POINT OF APPLICATION %%%%%%%%%%%%%
for i=1:nw
    wnet = [wnet nForce_dL(w(i),xsw(i),xew(i))]; %appends the new w to the array
    xwnet = [xwnet nPoint_dL(w(i),xsw(i),xew(i))]; %appends the new  x-ccordinate to the array
end


%%%%%%%%%%%%%% CALCULATING SUPPORT FORCES %%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% FOR fixed JOINT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m_net_applied = sum(m)-a+b; %calculating net sum of the external moments applied
m = [-a m b];
xm = [0 xm l];
nm = nm+2;
[sf] = sf_fixed(ns,np,nw,xs,xpf,xwnet,pf,w,wnet,m_net_applied); %calculates support forces

%%%%%%%%%%%%%% FINDING SHEAR FORCE EXPRESSIONS %%%%%%%%%%%%%%%%%%%%%%%
[V_func, xchanges_V] = find_shear_func( l,ns,np,nw,xs,xpf,xsw,xew,sf,pf,w ); %finds the expressions
    ↪ and the values of x where shear force changes

%%%%%%%%%%%%%% FINDING BENDING MOMENT EXPRESSIONS %%%%%%%%%%%%%%%%%%%
[ M_func, xchanges_M ] = find_moment_func( l,nm,xm,m,V_func,xchanges_V ); %finds the expressions and
    ↪  the values of x where bending moment changes

%%%%%%%%%%%%%% FINDING SLOPE AND DEFLECTION EXPRESSIONS %%%%%%%%%%%%%%%%%%
[ xchanges_def, def_func,xchanges_slope, slope_func ] = find_deflection_func( ns,xs,M_func,
    ↪ xchanges_M, E, I ); %finds the expressions and the values of x where function of deflection
    ↪ changes

%%%%%%%%%%%%%%%%%%%%%%%%% DONE IN TERMS OF VARIABLES a and b %%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%% INVOLVING VARIABLES %%%%%%%%%%%%%%%%%%%%%%

%Finding variables
[a, b] = find_fixedends_moments( slope_func, xchanges_slope, def_func, xchanges_def );

%substitute variables in arrays of expressions
sf = find_expressions_from_symbols(a, b, sf);
m = find_expressions_from_symbols(a, b, m);
V_func = find_expressions_from_symbols(a, b, V_func);
M_func = find_expressions_from_symbols(a, b, M_func);
slope_func = find_expressions_from_symbols(a, b, slope_func);
def_func = find_expressions_from_symbols(a, b, def_func);


%%%%%%%%%%%%%% PLOTTING GRAPHS %%%%%%%%%%%%%%%%
% sfd_from_func( l,e,V_func,xchanges_V );
% figure;
% bmd_from_func( l,e,M_func,xchanges_M );
% figure;
% slope_d_from_func( l,e,slope_func,xchanges_slope );
```

```
% figure;
% def_d_from_func( l,e,def_func,xchanges_def );
```

B.2 Slope Graph from Slope Function

This function plots the slope diagram by using the bending slope functions.

```
%%%Matlab Code Slope_d_from_func.m
function [ slope, x ] = slope_d_from_func( l,e,slope_func,xchanges_slope )
%BMD_FROM_FUNC Plots the slope diagram by using the bending slope functions
%   e--> discrete values of x to be taken

    syms t;
    x = [];
    slope = [];
    n = length(xchanges_slope);

    for i=1:n
        if i < n
            x1 = [xchanges_slope(i):e:xchanges_slope(i+1)];
        else
            x1 = [xchanges_slope(i)];
        end
        x = [x x1];
        n2 = length(x1);
        for j=1:n2
            syms t;
            t = x1(j);
            slope = [slope subs(slope_func(i))];
        end
    end

    plot(x,slope);
    title('Slope graph');
end
```

B.3 Bending Moment Diagrams

This function plots bending moment diagram from the various input parameters provided to it.

```
%Matlab code bmd.m
function [ M, x ] = bmd( l,e,ns,np,nw,nm,xs,xpf,xsw,xew,xm,sf,pf,w,m  )
%BMD Makes bending moment diagram from various parameters given
%

    x = [0:e:l];
    M = [];
    for i=1:length(x)
        M = [M find_moment(x(i),e,ns,np,nw,nm,xs,xpf,xsw,xew,xm,sf,pf,w,m)];
    end
    plot(x,M);


end
```

B.4 Bending Moment Diagram from Function

This function plots the bending diagram by using the bending moment functions.

```
%Matlab Code bmd_from_func.m
function [ M, x ] = bmd_from_func( l,e,M_func,xchanges_M )
%BMD_FROM_FUNC Plots the bending diagram by using the bending moment functions
%   e--> discrete values of x to be taken
    syms t;
    x = [0:e:l];
    M = [];
    ichanges = 1;
    for i=1:length(x)
        i

        syms t;
        if ichanges < length(xchanges_M)
            if xchanges_M(ichanges) <= x(i) < xchanges_M(ichanges + 1)
                func = M_func(ichanges);
                t = x(i);
                val = subs(func);
                M = [M val];
            else
                ichanges = ichanges + 1;
                if ichanges < length(xchanges_M)
                    syms t;
                    func = M_func(ichanges);
                    t = x(i);
                    val = subs(func);
                    M = [M val];

                end
            end
        end
        clc
    end

    plot(x,M);

end
```

B.5 Array Conversion

This function converts the cell form to arrays. This function was created so that the GUI interface takes cell array as input.

```
%Matlab Code conver_array.m
function [ a ] = convert_array( f )
%CONVERT_ARRAY Converts the cell form to arrays
%   This function was made as the GUI interface takes cell array as input

    a = cell2mat(f)


end
```

B.6 Decimal Expression Conversion

This code converts the long form decimal expression obtained after the computation to approximate forms.

```
%Matlab code conver_decimal_expression.m
function [ f ] = convert_decimal_expression( exp )
%CONVERT_DECIMAL_EXPRESSION Converts expressions to approximate forms
%   Made to convert weird fractions to decimals in the expressions

    f = []; %stores decimal form of expressions
    n = length(exp);

    for i=1:n
        f = [f vpa(exp(i))]; %making the array of expressions
    end

end
```

B.7 Slope Diagram from Slope Function

This function plots the slope diagram by using the bending slope functions. It works only for the discrete values of x.

```
%Matlab Code def_d_from_func.m
function [ def, x ] = def_d_from_func( l,e,def_func,xchanges_def )
%BMD_FROM_FUNC Plots the slope diagram by using the bending slope functions
%   e--> discrete values of x to be taken
    syms t;
    x = [];
    def = [];
    n = length(xchanges_def);

    for i=1:n
        if i < n
            x1 = [xchanges_def(i):e:xchanges_def(i+1)];
        else
            x1 = [xchanges_def(i)];
        end
        x = [x x1];
        n2 = length(x1);
        for j=1:n2
            syms t;
            t = x1(j);
            def = [def subs(def_func(i))];
        end
    end
    plot(x,def);
    title('Deflection of beam graph');
end
```

B.8 Plots for different Cases of inputs

This function plots the the graphs for different cases of inputs. It takes the inputs from the function defined earlier.

```
function [ output_args ] = deflection_compare_from_func_d( e,def_funcs,xchanges_defs )
%DEFLECTION_COMPARE_FROM_FUNC_D Compares the deflections obtained in
```

```
%different scenarios in the form of graphs
%   def_funcs is the array of def funcs and xchanges_Defs is the arrays of
%   xchanges of the respective def funcs

syms t;
n = length(def_funcs);
%length of def_funcs and xchanges_Defs should be same

for i = 1:n
    n1 = length(xchanges_defs{i});

    for j = 1:n1
        if j > 1
            x = [xchanges_defs{i}[j-1]:e:xchanges_defs{i}[j]];
            %t = x;
            %y = subs(def_funcs{i}(j));
            %y = def_funcs{i}(j);
        end
    end


end
```

B.9 Difference Points

This function calculates difference of elements in reference array with every o

```
function [ X ] = diff_points( m, n, x_ref, x_com )
%DIFF_POINTS Calculates difference of elements in reference array (x_ref) with every other element
%of the other array (x_com) and returns a matrix
% Parameters passed are the dimensions of the matrix to be formed. The array of
% elements supplied is also a parameter

X = zeros(m,n);

for i=1:m
    for j=1:n
        X(i,j) = x_com(j) - x_ref(i);
    end
end

end
```

B.10 Number Input

This functions checks if the parameter provided by the user is a number. If the input is not a number, the user will be warned to input a number. We used this function throuout our document.

```
function [ n ] = ensure_input_number( p, prompt, addPrompt )
%CHECK_INPUT_NUMBER This function ensures that the value of the parameter
%passed becomes a number. If input parameter is not a number, the function keeps prompting till a
    ↪ number is entered. Returns a number.
while (isa(p,'string') || isa(p,'char'))
    disp('Wrong input. Please enter a number.');
    p = input(strcat(prompt,addPrompt));
end

```

```
n = p;


end
```

B.11 Inertia of Circle

This function finds the inertia of a circle on giving the value of radius.

```
function [ I ] = find_circle_inertia( r )
%FIND_CIRCLE_INERTIA Finds second moment of inertia of circular cross
%section
%
    I = pi * r^4/4;

end
```

B.12 Deflection

This function finds piecewise function of deflection and its slope along with coordinates.

```
function [ xchanges_def, def_func,xchanges_slope, slope_func] = find_deflection_func( ns,xs,M_func,
    ↪ xchanges_M, E, I )
%FIND_DEFLECTION_FUNC Finds piecewise function of deflection and its slopes along with xcoordinates
    ↪ where function changes its definition (constants of
%integration not found out yet

    syms t;

    xchanges_def = []; % stores x coordinates where functions change their definition
    xchanges_slope = []; %stores xcoordinates where slope function changes its definition
    def_func = []; %stores the expressions in piecewise definitions of the function definitions
    slope_func = []; %stores the function form of slope

    xchanges_def = xchanges_M; %As function changes value when M changes its value
    xchanges_slope = xchanges_def; %As function changes value when M changes its value

    %[xchanges_slope, slope_func] = find_slope_func(ns,xs,M_func,xchanges_M, E, I);
    n = length(xchanges_def); %length of varying arrays

    values = zeros(n,1); %C1x + C2
    A = zeros(n,n); %matrix for helping to find first constant of integration
    C1 = zeros(n,1); %Second constant of integration after integrating Slope

    %%%%%%%%%%%%%%%%%%%% FINDING DOUBLE INTEGRATION OF MOMENT PART ONLY OF DEFLECTION
        ↪ %%%%%%%%%%%%%%%%%%%%
    syms t;
    for i = 1:n
        def_func = [def_func int(int(M_func(i),t),t)];
    end

    %%%%%%%%%%%%%%%%%%%% FINDING SINGLE INTEGRATION OF MOMENT PART ONLY OF DEFLECTION SLOPE
        ↪ %%%%%%%%%%%%%%%%%%%%
    syms t;
    for i = 1:n
        slope_func = [slope_func int(M_func(i),t)];
    end
```

```

%%%%%%%%%%%%%%%%%%%%% FINDING FIRST CONSTANT OF INTEGRATION  %%%%%%%%%%%%%%%%%%%%%
xstart = xs(1);
xend = xs(ns);
istart = 0; %stores index of first support coordinate
iend = 0; %stores index of last support coordinate

for i=1:n
    if (xchanges_def(i) == xstart)
        istart = i;
    end
    if (xchanges_def(i) == xend)
        iend = i;
    end
end

n2 = iend - istart; %sub part of beam from which support starts and till it ends

A = zeros(n2,n2);
C1 = zeros(n2,1);
B = zeros(n2,1);
msum = 0;

for i=istart:iend-1
    syms t;
    A(1,i - istart + 1) = xchanges_def(i+1) - xchanges_def(i);

    func = def_func(i);
    t = xchanges_def(i+1);
    msum = msum + subs(func);
    t = xchanges_def(i);
    msum = msum - subs(func);

    A(i - istart + 2,i - istart + 1) = 1;
    A(i - istart + 2,i - istart + 2) = -1;

    syms t;
    func1 = slope_func(i+1);
    func2 = slope_func(i);
    t = xchanges_def(i+1);
    val1 = subs(func1);
    val2 = subs(func2);
    B(i - istart + 2,1) = val1 - val2;
end
B(1,1) = - msum;
C1 = inv(A) * B;


%including deflection and slope function using first constant of integration
for i=istart:iend-1
    syms t;
    slope_func(i) = slope_func(i) + C1(i-istart+1);
    def_func(i) = def_func(i) + C1(i-istart+1) * t;
end
%A*C1 = B


%finding constants if any after remaining part of the beam for the deflection
istart2 = iend;
```

```
    iend2 = n;
    if 1 < istart2 < iend2
        for i=istart2:iend2
            func1 = slope_func(i-1);
            func2 = slope_func(i);
            t = xchanges_def(i);
            val = subs(func1) - subs(func2);
            slope_func(i) = slope_func(i) + val;
            syms t;
            def_func(i) = def_func(i) + val*t;

            %Ensuring continuity of deflection equation
            syms t;
            func1 = def_func(i-1);
            func2 = def_func(i);
            t = xchanges_def(i);
            val = subs(func1) - subs(func2);
            def_func(i) = def_func(i) + val;
        end
    end


    %finding constants if any for previous part of start of the beam
    istart2 = istart - 1;
    iend2 = 1;
    i = istart2;
    while i >=iend2
        func1 = slope_func(i+1);
        func2 = slope_func(i);
        t = xchanges_def(i+1);
        val = subs(func1) - subs(func2);
        slope_func(i) = slope_func(i) - val;
        syms t;
        def_func(i) = def_func(i) - val*t;

        %ensuring continuity of the deflection function
        syms t;
        func1 = def_func(i+1);
        func2 = def_func(i);
        t = xchanges_def(i+1);
        val = subs(func1) - subs(func2);
        def_func(i) = def_func(i) + val;

        i = i - 1;
    end

    %Ensuring continuity of deflection between supports (from start of
    %first support till end of first support
    for i=istart+1:iend
        syms t;
            func1 = def_func(i-1);
            func2 = def_func(i);
            t = xchanges_def(i);
            val = subs(func1) - subs(func2);
            def_func(i) = def_func(i) + val;
    end


    slope_func = slope_func/(E*I);
```

```
    def_func = def_func/(E*I);
    slope_func = convert_decimal_expression(slope_func);
    def_func = convert_decimal_expression(def_func);

end
```

B.13 Deflection Constants

This function finds constant of integrating Moment.

```
function [ def_func ] = find_deflections_constants( ns,xs,xchanges_def,def_func )
%FIND_DEFLECTIONS_CONSTANTS Finds constants after integrating M
%
    syms t;
    syms A;
    syms B;
    id = 0;

    E = zeros(length(xchanges_def),1);
    C = zeros(length(xchanges_def),2);
    D = zeros(length(xchanges_def),1);

    % E + C*[A B] = D
    %forming equations at supports
    for i=1:ns
        syms t;
        while xchanges_def(id)~=xs(i)
            id = id + 1;
        end
        t = xs(i);
        syms A;
        syms B;
        def_func(id) = subs(t);
        syms t;
        C(id,1) = diff(def_func(id),A);
        C(id,2) = diff(def_func(id),B);
        D(id,1) = 0; %deflections at supports is 0
        A = 0;
        B = 0;
        E(id,1) = subs(def_func(id));
    end


    %continuity equations at intermediate points




end
```

B.14 Inertia of I-Beam

This function finds Moment of inertia of an I-Beam.

```
function [ I ] = find_i_inertia( b1,h1,b2,h2,b3,h3 )
%FIND_I_INERTIA Finds second moment of inertia of IBeam
%   A2 is the top part, where 0 is initially
```

```
    I = 0;
    A1 = b1*h1;
    A2 = b2*h2;
    A3 = b3*h3;

    N = (h3/2*A3 + (h3 + (h1/2)) * A1 + (h3 + h1 + (h2/2)))/(A1+A2+A3);

    I1 = find_rect_inertia(b1,h1) + A1*((h1/2)+h3 - N)^2; %middle
    I2 = find_rect_inertia(b2,h2) + A2*(h3 + h1 + (h2/2) - N)^2; %top
    I3 = find_rect_inertia(b3,h3) + A3*((h3/2) - N)^2; %bottom

    I = I1 + I2 + I3;

    %b1, h1 --> middle part of Ibeam
    %b2, h2 --> top part of ibeam
    %b3, h3 --> bottom part of the beam




end
```

B.15 Inertia

This function differentiates different cases of Inertia.

```
function [ I ] = find_inertia( type, args )
%FIND_INERTIA Summary of this function goes here
%   Detailed explanation goes here

    I = 0;
    if strcmp(type,'RECT')
        b = args(1);
        d = args(2);
        I = find_rect_inertia(b,d);
    else
        if strcmp(type,'CIRCLE')
            r = args(1);
            I = find_circle_inertia(r);
        else
            if strcmp(type,'I')
                b1 = args(1);
                h1 = args(2);
                b2 = args(3);
                h2 = args(4);
                b3 = args(5);
                h3 = args(6);
                I = find_i_inertia(b1,h1,b2,h2,b3,h3);
            end
        end
    end



end
```

B.16 Moment

This function finds Moment.

```
function [ M ] = find_moment( xi,e,ns,np,nw,nm,xs,xpf,xsw,xew,xm,sf,pf,w,m )
%FIND_MOMENT Finds moment at a particular point from shear force
%
    M = 0;
    x = [0:e:xi];
    for i=1:length(x)
        V = find_shear( x(i),ns,np,nw,xs,xpf,xsw,xew,sf,pf,w );
        M = M - V*e;
    end

    for i=1:nm
        if xm(i) <= xi
            M = M - m;
        end
    end


end
```

B.17 Find Moment

```
function [ M_func, xchanges_M ] = find_moment_func( l,nm,xm,m,V_func,xchanges_V )
%FIND_MOMENT_FUNC_FUNCTION Finds the functional form of bending moment
%
  syms t;

  M_func = []; %stores the functions of bending moments
  xchanges_M = []; %stores the coordinates where bending moment function changes its definition
  im = 1; %stores iterates over the moments supplied externally
  iv = 1; %stores iterates ovr the shear force functions
  val1 = 0; %stores helps in converting the integrated shear functions to the relevant coordinates
  n = 0; %stores length of the varying arrays

  xchanges_M = sort([xchanges_V xm]);
  xchanges_M = unique(xchanges_M,'sorted');

  n = length(xchanges_M); %number of points where m changes


  %integrating shear functions
  iv = 1;
  if length(xchanges_V) > 0
      for i=1:n
          if iv < length(xchanges_V)
              if xchanges_M(i) >= xchanges_V(iv) && xchanges_M(i) < xchanges_V(iv+1)
                  M_func = [M_func int(-V_func(iv),t)];
              else
                  iv = iv + 1;
                  M_func = [M_func int(-V_func(iv),t)];
              end
          else
              if xchanges_M(i) >= xchanges_V(iv)
                  M_func = [M_func int(-V_func(iv),t)];
              end
```

```

            end
        end
    else
        for i=1:n
            M_func = [M_func 0];
        end
    end

    %converting to relevant coordinates t --> x
    for i=1:n
        syms t;
        func = M_func(i);
        t = xchanges_M(i);
        val = subs(func);
        M_func(i) = func - val;
    end

    %Adding effects due to externally applied moments

    im = 1;
    if length(xm) > 0
      for i = 1:n
          if im <= length(xm)
              if xchanges_M(i) == xm(im)
                  M_func(i) = M_func(i) - m(im);
                  im = im + 1;
              end
          end
      end
    end

    %adding effects due to previous moments
    for i = 1:n-1
        syms t;
        func = M_func(i);
        t = xchanges_M(i+1);
        val = subs(func);
        M_func(i+1) = M_func(i+1) + val;
    end

    M_func = convert_decimal_expression(M_func);

end
```

This function differentiates different cases of Moment.

B.18 Particular deflection

This function finds particular deflection.

```
function [ defunc_part ] = find_particular_deflection_func( M_func )
%FIND_PARTICULAR_DEFLECTION_FUNC Finds integration of particular expression
% of moments passed

syms t;
syms C1;
syms C2;

```

```
defunc_part = int(int(M_func,t),t) + C1*t + C2;

end
```

B.19 Inertia of Rectangle

This function finds Inertia of Rectangle on giving the input of dimensions.

```
function [ I ] = find_rect_inertia( b,d )
%FIND_RECT_INERTIA Finds second moment of inertia of rectangular cross
%section
%
    I = b*d^3/12;


end
```

B.20 Shear Force

```
function [ V ] = find_shear( x,ns,np,nw,xs,xpf,xsw,xew,sf,pf,w )
%FIND_SHEAR Finds shear at a point
%   Takes various parameters as input

    V = 0;

    %Adding effects due to point forces
    for i=1:np
        if xpf(i) <= x
            V = V - pf(i);
        end
    end

    %Adding effects due to support forces
    for i=1:ns
        if xs(i) <= x
            V = V - sf(i);
        end
    end

    %Adding effects due to distributed forces
    for i=1:nw
        if xsw(i) <= x
            %calculating net force which has to be taken
            if xew(i) <= x
                xtaken = xew(i);
            else
                xtaken = x;
            end
            V = V - nForce_dL( w(i), xsw(i), xtaken );
        end
    end
end
```

B.21

This function finds the shear force.

```
function [ V ] = find_shear_from_func( x, V_func, xchanges_V )
%FIND_SHEAR_FROM_FUNC Finds shear from the expression form of the shear
%force at a particular x coordinate
%
    V = 0;
    syms t;

    n = length(xchanges_V); %number of points where the function fluctuates

    for i=1:n
        if i < n && xchanges_V(i) <= x && x < xchanges_V(i+1)
            t = x;
            V = subs(V_func(i));
        else
            if i == n && x >= xchanges_V(i)
                t = x;
                V = subs(V_func(i));
            end
        end
    end

end
```

B.22 Shear Force

This function finds the shear force from function.

```
function [ V_func, xchanges_V ] = find_shear_func( l,ns,np,nw,xs,xpf,xsw,xew,sf,pf,w )
%FIND_SHEAR_FUNCTION Finds the functional form of shear force
%
    syms t;
    V_func = [];
    xchanges_V = []; % coordinates of x where shear value fluctuates
    xchanges_V = sort([xs xpf xsw xew]);

    if (xchanges_V(length(xchanges_V))~=l)
        xchanges_V = [xchanges_V l];
    end
    if (xchanges_V(1)~=0)
        xchanges_V = [0 xchanges_V];
    end

    xchanges_V = unique(xchanges_V,'sorted');
    n = length(xchanges_V); % total number of coordinates where x changes

    for i=1:n
        V_func = [V_func (find_shear(xchanges_V(i),ns,np,nw,xs,xpf,xsw,xew,sf,pf,w)+t*0)]; %finds
            ↪ value of shears only due to support and point forces
    end
    %superimposing effects of distributed loads
    iw = 1;
    for i=1:n
        if iw <= nw
            if xchanges_V(i)==xsw(iw)
                V_func(i) = V_func(i) - w(iw)*(t - xchanges_V(i));
            else
                if xchanges_V(i) > xsw(iw) && xchanges_V(i) < xew(iw)
```

```
                    V_func(i) = V_func(i) - w(iw)*(t - xchanges_V(i));
                else
                    if xchanges_V(i) >= xew(iw)
                        iw = iw+1;
                    end
                end
            end

        end

    V_func = convert_decimal_expression(V_func);
end
```

B.23 Slope of Curvature

This function finds the slope of the curvature in the beam.

```
function [ xchanges_slope,slope_func ] = find_slope_func( ns,xs,M_func,xchanges_M, E, I )
%FIND_SLOPE_FUNC Finds the slope in the curvatures of the beam by
%integrating the funcs
%
    syms t;

    xchanges_slope = []; %stores coordinates where the slope function changes its definition
    slope_func = []; %stores expressions of the slopes of the functions
    n = length(xchanges_M); %stores length of arrays

    xchanges_slope = xchanges_M; %as slope definition will change where moment definition changes

    %Adding integrated forms of moment functions
    for i=1:n
        slope_func = [slope_func int(M_func(i))];
    end

    %converting to relevant coordinates
    for i=1:n-1
        syms t;
        func = slope_func(i+1);
        t = xchanges_slope(i);
        val = subs(func);
        slope_func(i+1) = func - val;
    end

%     %ensuring continuity in slopes
%     for i = 1:n-1
%         syms t;
%         func1 = slope_func(i);
%         func2 = slope_func(2);
%         t = xchanges_slope(i);
%         val1 = subs(func1);
%         val2 = subs(func2);
%
%     end


    slope_func = slope_func/(E*I); %EIdv/dx = -Mx + c

end
```

B.24 Net Force

This function finds the net force acting in the given range.

```
function [ wnet ] = nForce_dL( w, xStart, xEnd )
%NETFORCE_DISTRIBUTEDLOADS Calculates the net force acting in the given range of distributed load
%   Takes input as the distributed load value and the range in which it
%   acts

wnet = w*abs((xEnd - xStart));


end
```

B.25 Net Point of Action

This function finds net point of action for distributed loads.

```
function [ x ] = nPoint_dL( w, xStart, xEnd )
%NETPOINT_DISTRIBUTEDLOADS Finds net point of action of distributed loads
%   Takes input as the distributed load value and the range in which it
%   acts

    x = (xStart+xEnd)/2;


end
```

B.26 Support Forces at Pins

This function calculates support forces in case of all pin supports.

```
function [ F ] = sf_all_pins(ns,np,nw,xs,xpf,xwnet,pf,w,wnet,mnet)
%SF_BOTH_PINS Calculates support forces in case of all pin supports
%   Takes parameters as ns,np,nw,xs,xpf,xwnet,p,w,m
F = [];
X_pf = diff_points(ns,np,xs,xpf) %Forming the difference between points of the matrix for point forces
X_s = diff_points(ns,ns,xs,xs) %Forming the difference between points of the matrix for support forces (unknown for now)
X_w = diff_points(ns,nw,xs,xwnet) %Forming the difference between points of the matrix for distributed forces with the help of net forces found

A = [];

% X_s*F' + X_pf*P' + X_df*W' + M = 0
% F' = - X_s^-1 * (X_pf*P' + X_df*W' + M) --> Equations used
if np~=0
    A = X_pf * pf';
end
if nw~=0
    if length(A)~=0
        A = A + X_w * wnet';
        %disp('case1');
    else
        A = X_w * wnet';
        %disp('case2');
    end
```

```
end

if length(A)~=0
    A = A + mnet*ones(ns,1);
else
    A = mnet*ones(ns,1);
end

F = - pinv(X_s) * (A);

end
```

B.27 Shear Force Diagram

This function plots the shear force Diagram.

```
function [ V, x ] = sfd( l,e,ns,np,nw,xs,xpf,xsw,xew,sf,pf,w )
%SFD Plots the shear force diagram
%   Detailed explanation goes here
    x = [0:e:l];
    V = [];
    for i=1:length(x)
        V = [V find_shear(x(i),ns,np,nw,xs,xpf,xsw,xew,sf,pf,w)];
    end
    plot(x,V);

end
```

B.28 Slope Diagram from Function

This function plots the slope diagram by using the bending slope functions.

```
%%%Matlab Code Slope_d_from_func.m
function [ slope, x ] = slope_d_from_func( l,e,slope_func,xchanges_slope )
%BMD_FROM_FUNC Plots the slope diagram by using the bending slope functions
%   e--> discrete values of x to be taken

    syms t;
    x = [];
    slope = [];
    n = length(xchanges_slope);

    for i=1:n
        if i < n
            x1 = [xchanges_slope(i):e:xchanges_slope(i+1)];
        else
            x1 = [xchanges_slope(i)];
        end
        x = [x x1];
        n2 = length(x1);
        for j=1:n2
            syms t;
            t = x1(j);
            slope = [slope subs(slope_func(i))];
        end
    end

```

```
    plot(x,slope);
    title('Slope graph');
end
```

B.29 Inertia of Circle

```
function [ r ] = take_input_circle_inertia( addPrompt )
%TAKE_INPUT_CIRCLE_INERTIA Takes inputs required to find inertia of a
%circular cross-section
%
prompt = 'Enter radius';
r = input(strcat(prompt,addPrompt));
r = ensure_input_number(r,prompt,addPrompt);


end
```

This fragments takes and stores the input of the radius of the circular beam for use in calculation of Inertia.

B.30 Distributed Load Coordinates

This function takes the input of positions of distributed loads with start and end coordinates and also the magnitude of force applied on it. It also stores them for further calculations.

```
function [ xsw, xew, w ] = take_input_dL( nw, addPrompt )
%TAKE_INPUT_DISTRIBUTEDLOADS Takes input of distributed loads with start and end coordinates and
%magnitudes of force/length depending on the number of loads given as input

xsw = []; %stores start coordinates of each different distributed load range
xew = []; %stores end coordinates of each different distributed load range
w = []; %stores magnitudes of distributed loads

%Taking input: x coordinate of point forces applied
disp('Specifying ranges of distributed loads');
disp('Origin is the beginning of the beam, as mentioned before.');

for i=1:nw

    prompt = 'Enter START coordinate of range of distributed load ';
    prompt = strcat(prompt,num2str(i)); %To add load number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    xsw = [xsw, x];  %Appending number to the array

    prompt = 'Enter END coordinate of range of distributed load ';
    prompt = strcat(prompt,num2str(i)); %To add load number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    xew = [xew, x];  %Appending number to the array

    prompt = 'Enter force per length of load ';
    prompt = strcat(prompt,num2str(i)); %To add load number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

```

```
    w = [w, x];  %Appending number to the array

end


end
```

B.31 Inertia of I Beam

This function finds the moment of Inertia of an I beam by taking the required parameters.

```
function [ b1,h1,b2,h2,b3,h3 ] = take_input_ibeam_inertia( addPrompt )
%TAKE_INPUT_IBEAM_INERTIA Takes input of the parameters required to find
%the inertia of ibeam
%

prompt = 'Enter b1';
b1 = input(strcat(prompt,addPrompt));
b1 = ensure_input_number(b1,prompt,addPrompt);

prompt = 'Enter h1';
h1 = input(strcat(prompt,addPrompt));
h1 = ensure_input_number(h1,prompt,addPrompt);

prompt = 'Enter b2';
b2 = input(strcat(prompt,addPrompt));
b2 = ensure_input_number(b2,prompt,addPrompt);

prompt = 'Enter h2';
h2 = input(strcat(prompt,addPrompt));
h2 = ensure_input_number(h2,prompt,addPrompt);

prompt = 'Enter b3';
b3 = input(strcat(prompt,addPrompt));
b3 = ensure_input_number(b3,prompt,addPrompt);

prompt = 'Enter h3';
h3 = input(strcat(prompt,addPrompt));
h3 = ensure_input_number(h3,prompt,addPrompt);



end
```

B.32 Moment Input

This function takes input of point loads with coordinates and magnitudes, also store them for further calculation.

```
function [ xm, m ] = take_input_moments( nm, addPrompt )
%TAKE_INPUT_MOMENTS Takes input of moments with coordinates and
%magnitudes depending on the number of moments given as input

%Taking input: x coordinate of point forces applied
disp('Specifying coordinates of moments applied externally');
disp('Origin is the beginning of the beam, as mentioned before.');

```

```
xm = []; %stores x coordinates of moments applied
m = []; %stores magnitudes of moments applied externally

for i=1:nm

    prompt = 'Enter coordinate of moment ';
    prompt = strcat(prompt,num2str(i)); %To add moment number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    xm = [xm, x];  %Appending number to the array

    prompt = 'Enter moment ';
    prompt = strcat(prompt,num2str(i)); %To add moment number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    m = [m, x];  %Appending number to the array

end


end
```

B.33 Point loads

This function takes input of point loads with coordinates and magnitudes, also store them for further calculation.

```
function [ xpf, pf ] = take_input_pL( np, addPrompt )
%TAKE_INPUT_POINTLOADS Takes input of point loads with coordinates and
%magnitudes depending on the number of loads given as input

%Taking input: x coordinate of point forces applied
disp('Specifying coordinates of point forces applied externally');
disp('Origin is the beginning of the beam, as mentioned before.');

xpf = []; %stores x coordinate of point forces externally applied with respect to origin
pf = []; %stores magnitudes of point forces acting

for i=1:np

    prompt = 'Enter coordinate of point force ';
    prompt = strcat(prompt,num2str(i)); %To add force number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    xpf = [xpf, x];  %Appending number to the array

    prompt = 'Enter force ';
    prompt = strcat(prompt,num2str(i)); %To add force number to the prompt
    x = input( strcat(prompt,addPrompt) );
    x = ensure_input_number(x,prompt,addPrompt);

    pf = [pf, x];  %Appending number to the array

end

```

```

end
```

B.34 Inertia of Rectangle

This function takes the dimensions of a rectangular beam and gives the inertia of rectangle as output.

```
function [ b, d ] = take_input_rect_inertia( addPrompt )
%TAKE_INPUT_RECT_INERTIA Takes input for finding inertia of rectangle
%
prompt = 'Enter b';
b = input( strcat(prompt,addPrompt) );
b = ensure_input_number(b,prompt,addPrompt);

prompt = 'Enter d';
d = input( strcat(prompt,addPrompt) );
d = ensure_input_number(d,prompt,addPrompt);


end
```

7 Appendix B - List of Figures

List of Figures

Case 1: Pin - Roller Support (Only PL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 5m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=2m	5N	D						
Section - Rect	X=4m	3.5N	D						
b = 50.4 mm									
h = 101.6 mm									
I = 4.44x10^6 mm^4									

Figure 5: Test Case 1

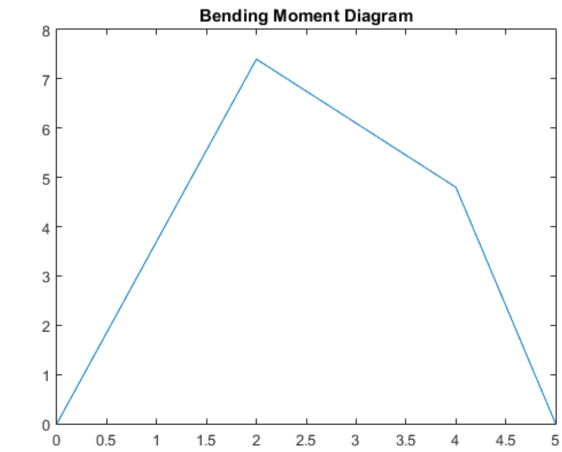

Figure 6: Bending Moment diagram

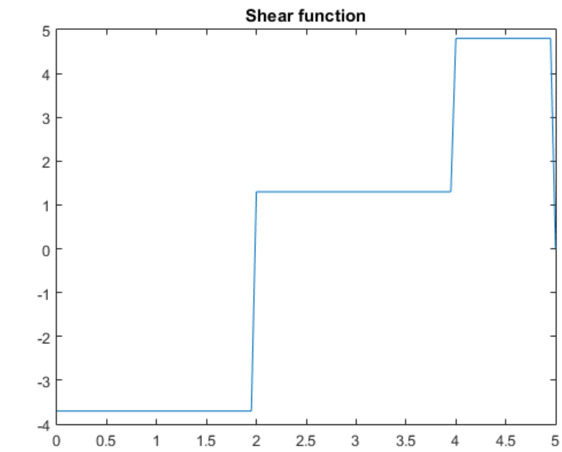

Figure 7: Shear Force Diagram

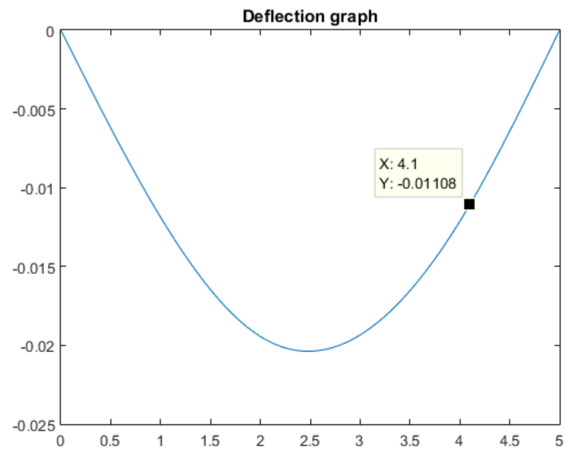

Figure 8: Deflection of Beam

Case 2: Pin - Roller Support (Only PM)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 5m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa				X=3m	4.5N-m	CCW			
Section - Rect				X=5m	3N-m	CCW			
b = 50.4 mm									
h = 101.6 mm									
I = 4.44x10^6 mm^4									

Figure 9: Test Case 2

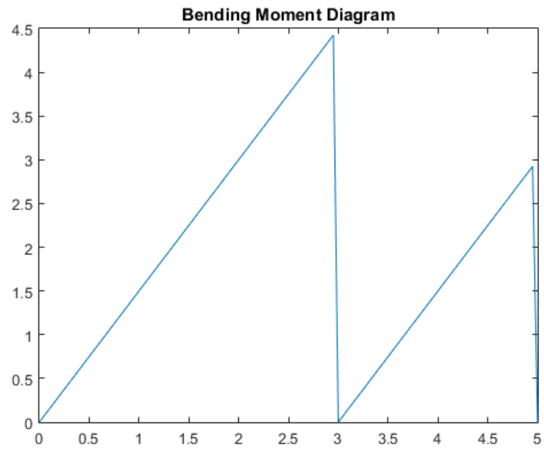

Figure 10: Bending Moment diagram

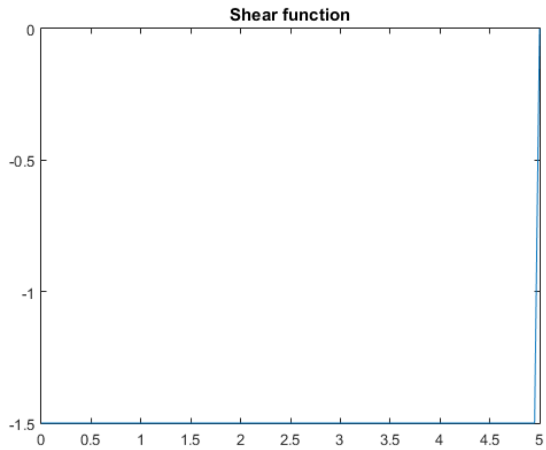

Figure 11: Shear Force Diagram

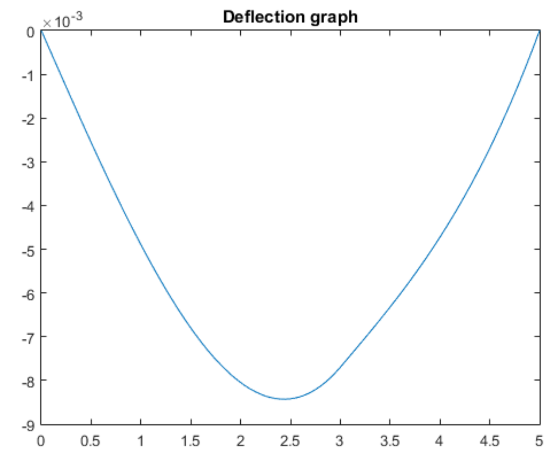

Figure 12: Deflection of Beam

Case 3: Pin - Roller Support (Only UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 5m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa							X=1to3m	10N/m	D
Section - Rect							X=4to5m	4N/m	D
b = 50.4 mm									
h = 101.6 mm									
I = 4.44x10^6 mm^4									

Figure 13: Test Case 3

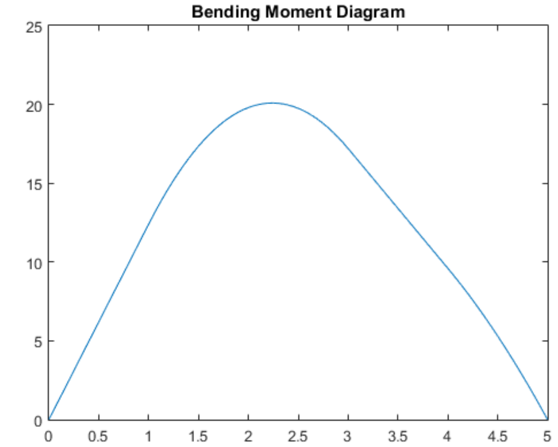

Figure 14: Bending Moment diagram

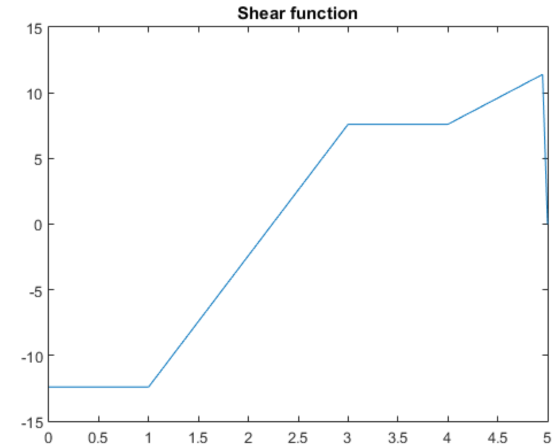

Figure 15: Shear Force Diagram

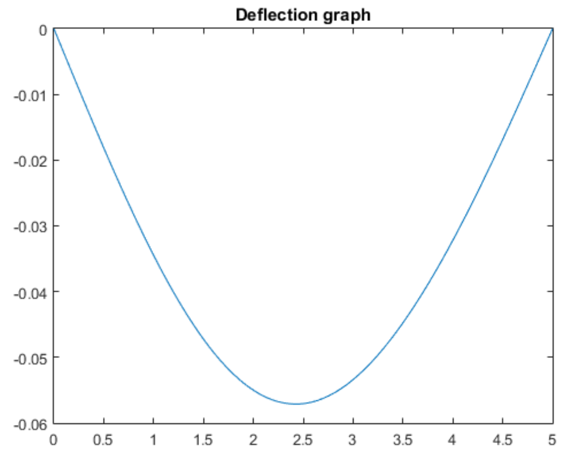

Figure 16: Deflection of Beam

Case 4: Pin - Roller Support (PL+PM)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 5m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=1m	100N	D	X=4m	2200N-m	CW			
Section - Rect	X=3m	350N-m	D						
b = 50.4 mm									
h = 101.6 mm									
I = 4.44x10^6 mm^4									

Figure 17: Test Case 4

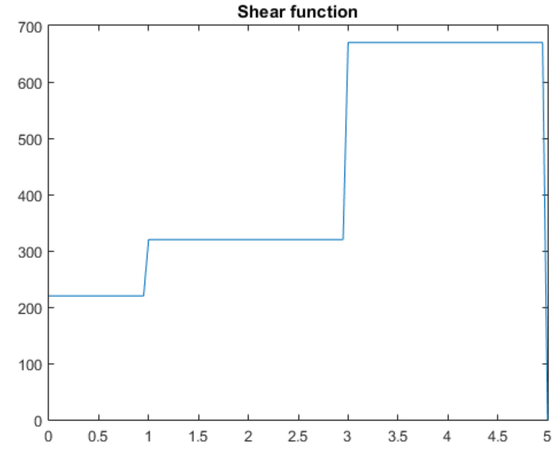

Figure 18: Shear Force Diagram

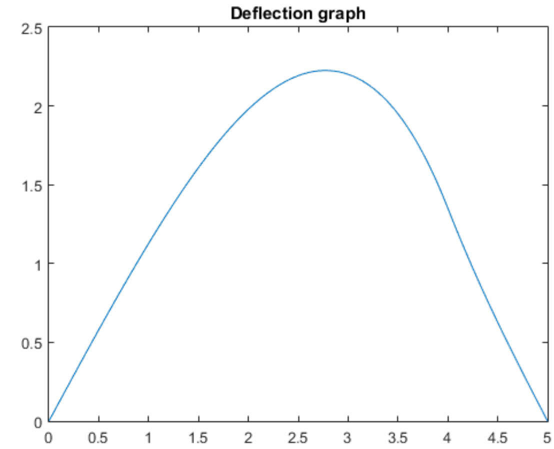

Figure 19: Deflection of Beam

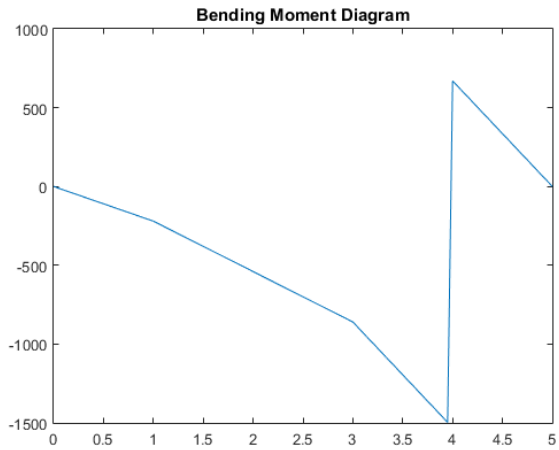

Figure 20: Bending Moment diagram

Case 5: Pin - Roller Support (PL+UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 5m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=2m	2000N	D				X=2.5to4m	45N/m	D
Section - Rect	X=4.5m	1200N	D						
b = 50.4 mm									
h = 101.6 mm									
I = 4.44x10^6 mm^4									

Figure 21: Test Case 5

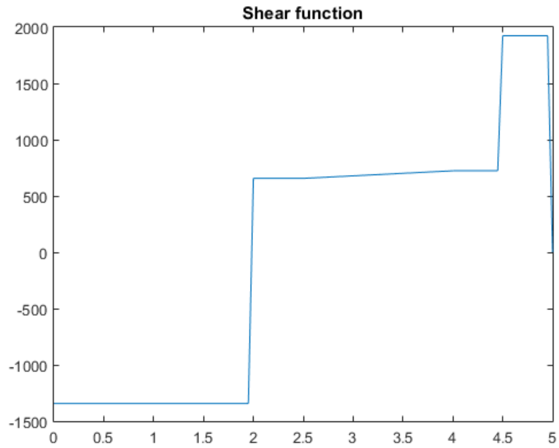

Figure 22: Shear Force Diagram

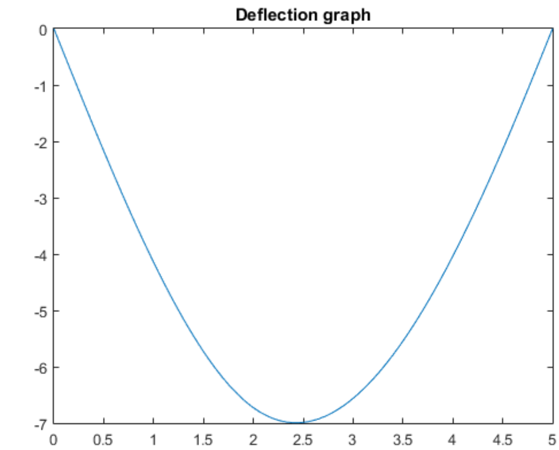

Figure 23: Deflection of Beam

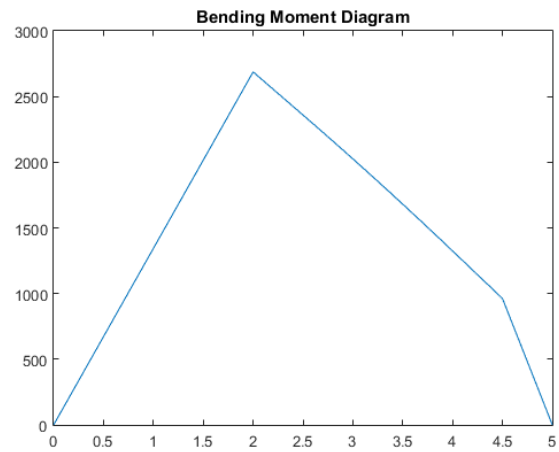

Figure 24: Bending Moment diagram

Case 6: Pin - Roller Support (PM+UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 5m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa				X=0.6m	350N-m	CW	X=3to5m	400N/m	D
Section - Rect				X=2.8m	850N-m	CCW			
b = 50.4 mm									
h = 101.6 mm									
I = 4.44x10^6 mm^4									

Figure 25: Test Case 6

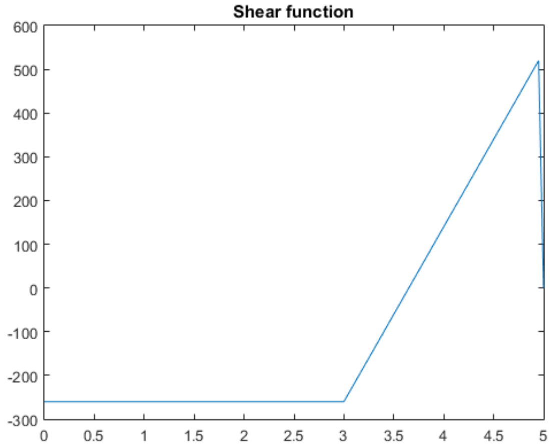

Figure 26: Shear Force Diagram

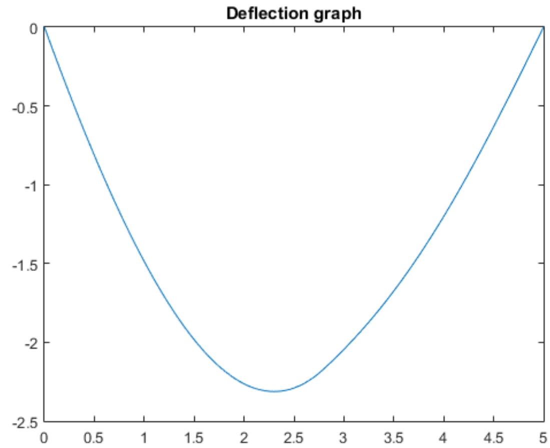

Figure 27: Deflection of Beam

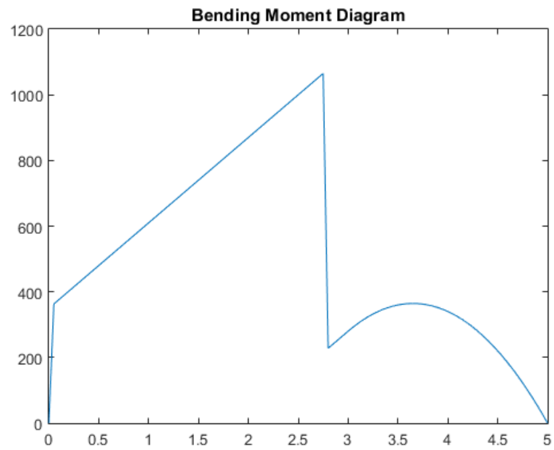

Figure 28: Bending Moment diagram

Case 7: Pin - Roller Support (PL+PM+UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 5m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=2.5m	2300N	D	X=0m	500N-m	CCW	X=3.2to5m	380N/m	D
Section - Rect				X=5m	650N-m	CCW			
b = 50.4 mm									
h = 101.6 mm									
I = 4.44x10^6 mm^4									

Figure 29: Test Case 7

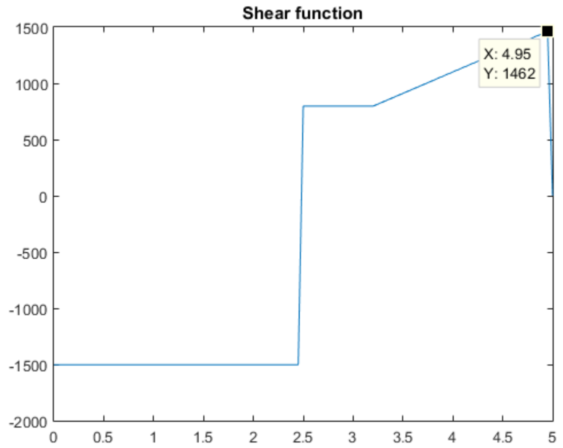

Figure 30: Shear Force Diagram

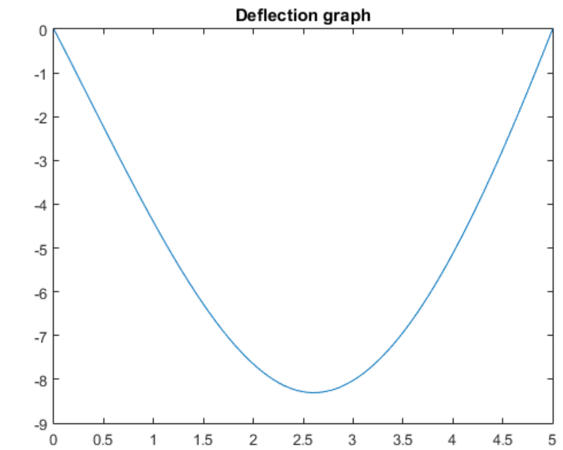

Figure 31: Deflection of Beam

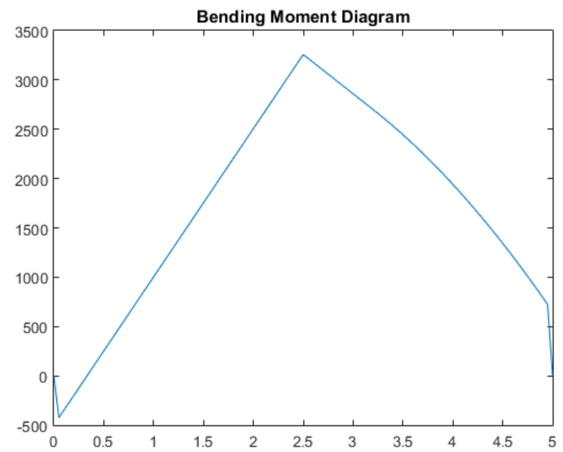

Figure 32: Bending Moment diagram

Case 8: Fixed - Fixed Support (Only PL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 4m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 179.3 GPa	X=2.5m	800N	D						
Section - I									
h2 =8.001=h3									
h1 = 189.988, b1 = 6.2223									
b2 =101.98= b3									
I = 19.55583x10^6 mm^4									

Figure 33: Test Case 8

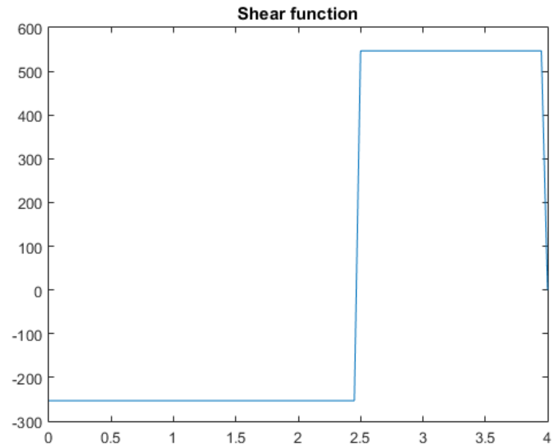

Figure 34: Shear Force Diagram

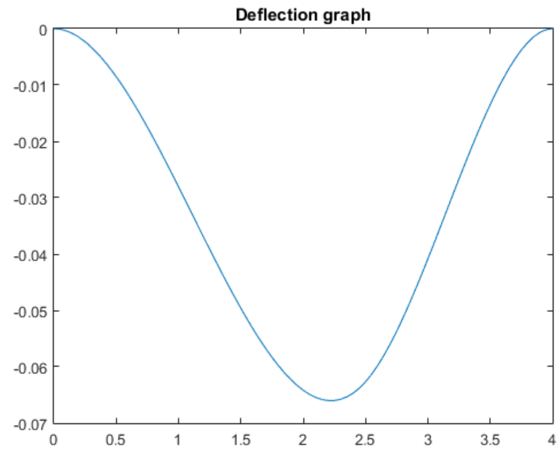

Figure 35: Deflection of Beam

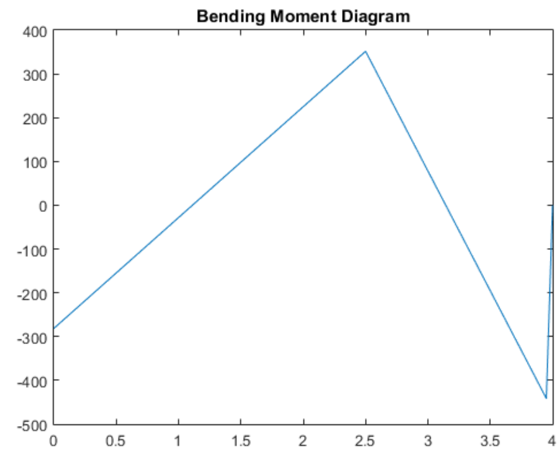

Figure 36: Bending Moment diagram

Case 9: Fixed - Fixed Support (Only PM)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 4m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 179.3 GPa				X=1m	5700N-m	CW			
Section - I				X=3.5m	920N-m	CCW			
h2 =8.001=h3									
h1 = 189.988, b1 = 6.2223									
b2 =101.98= b3									
I = 19.55583x10^6 mm^4									

Figure 37: Test Case 9

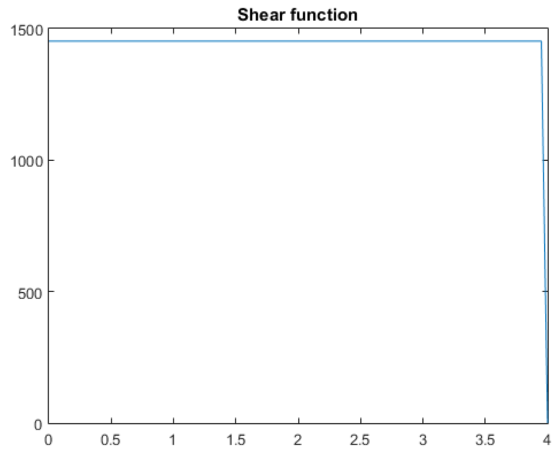

Figure 38: Shear Force Diagram

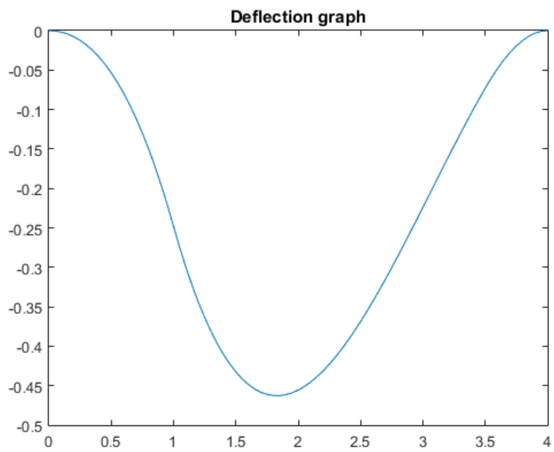

Figure 39: Deflection of Beam

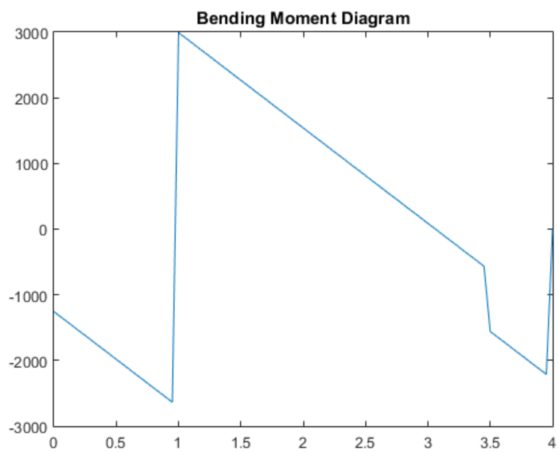

Figure 40: Bending Moment diagram

Case 10: Fixed - Fixed Support (Only UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 4m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 179.3 GPa							X=1.2to3m	2400N/m	D
Section - I							X=3.5to4m	640N/m	D
h2 =8.001=h3									
h1 = 189.988, b1 = 6.2223									
b2 =101.98= b3									
I = 19.55583x10^6 mm^4									

Figure 41: Test Case 10

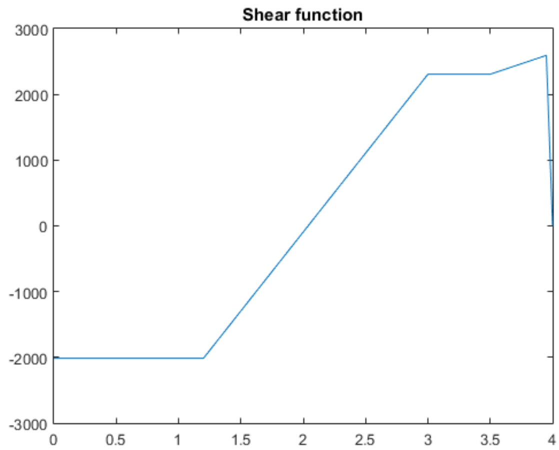

Figure 42: Shear Force Diagram

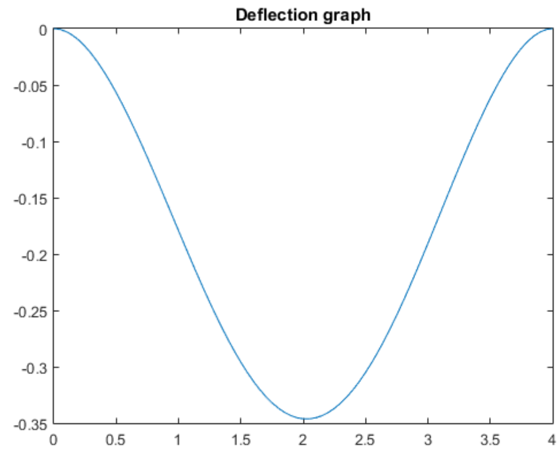

Figure 43: Deflection of Beam

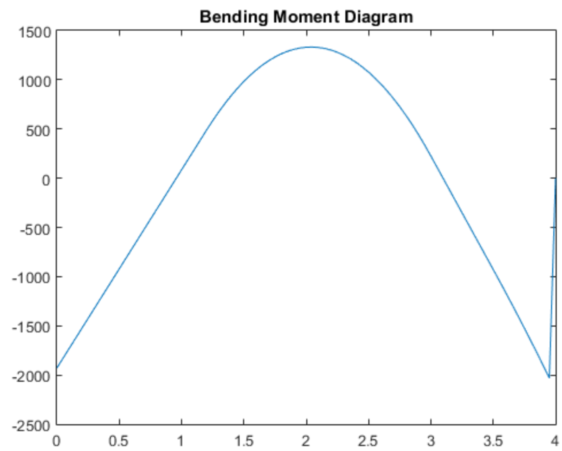

Figure 44: Bending Moment diagram

Case 11: Fixed - Fixed Support (PL+PM)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 4m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 179.3 GPa	X=1.2m	1550N	D	X=3.5m	3500N-m	CW			
Section - I	X=2.5m	3580N	D						
h2 =8.001=h3									
h1 = 189.988, b1 = 6.2223									
b2 =101.98= b3									
I = 19.55583x10^6 mm^4									

Figure 45: Test Case 11

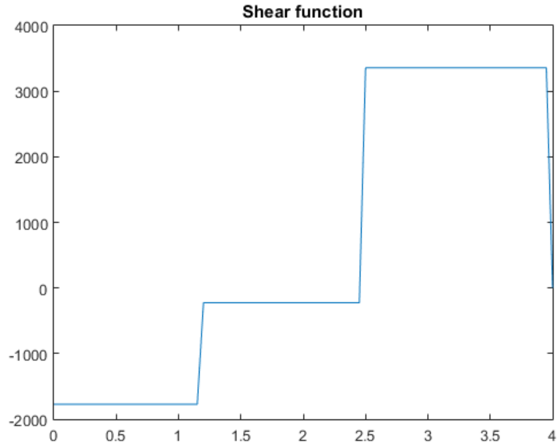

Figure 46: Shear Force Diagram

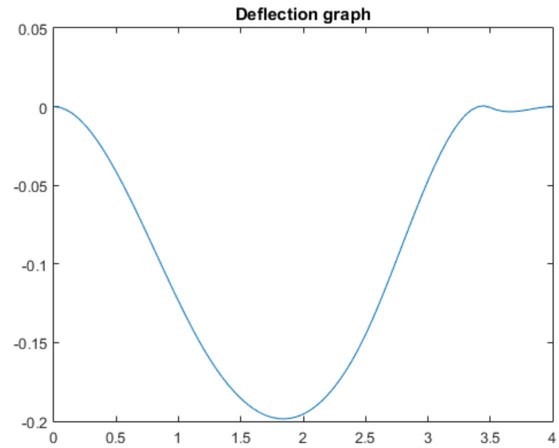

Figure 47: Deflection of Beam

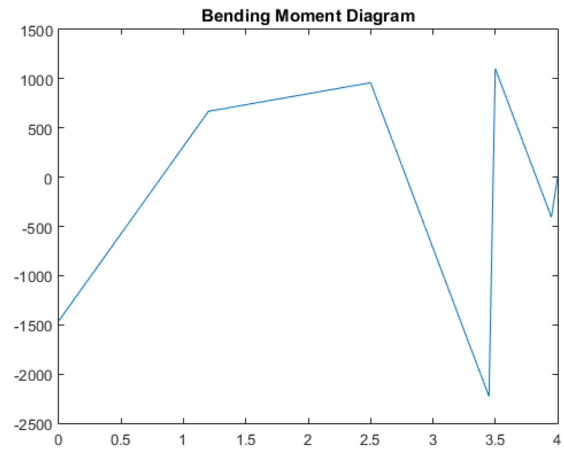

Figure 48: Bending Moment diagram

Case 12: Fixed - Fixed Support (PL+UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 4m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 179.3 GPa	X=0.5m	4100N	D				X=1.5to3m	5800N/m	D
Section - I	X=3.5m	1750N	D						
h2 =8.001=h3									
h1 = 189.988, b1 = 6.2223									
b2 =101.98= b3									
I = 19.55583x10^6 mm^4									

Figure 49: Test Case 12

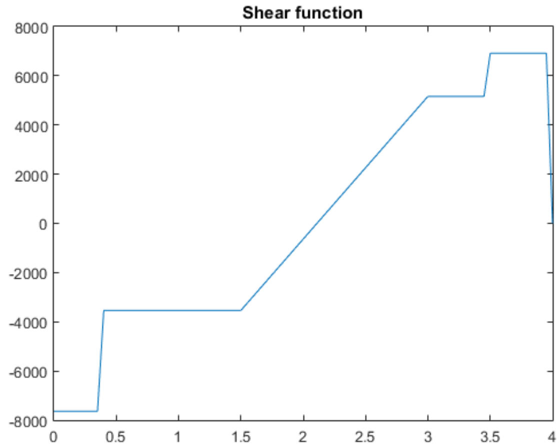

Figure 50: Shear Force Diagram

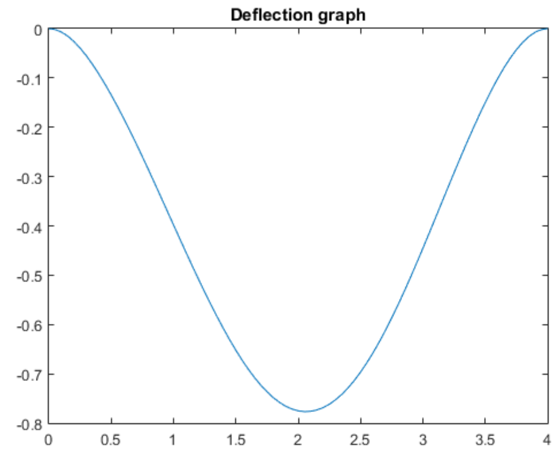

Figure 51: Deflection of Beam

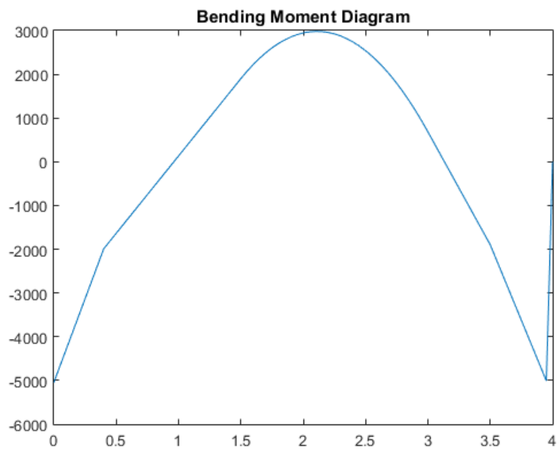

Figure 52: Bending Moment diagram

Case 13: Fixed - Fixed Support (PM+UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 4m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 179.3 GPa				X=0.3m	7800N-m	CW	X=3to4m	1300N/m	D
Section - I				X=4m	770N-m	CCW			
h2 =8.001=h3									
h1 = 189.988, b1 = 6.2223									
b2 =101.98= b3									
I = 19.55583x10^6 mm^4									

Figure 53: Test Case 13

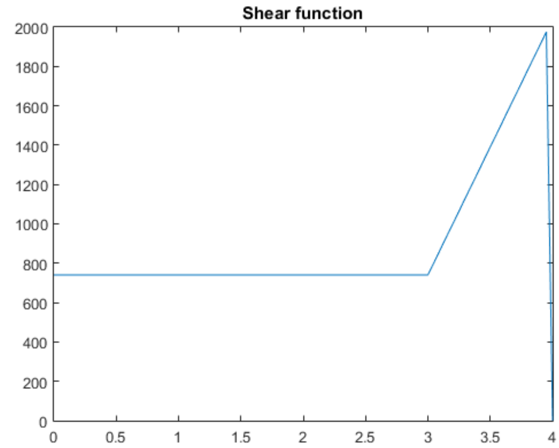

Figure 54: Shear Force Diagram

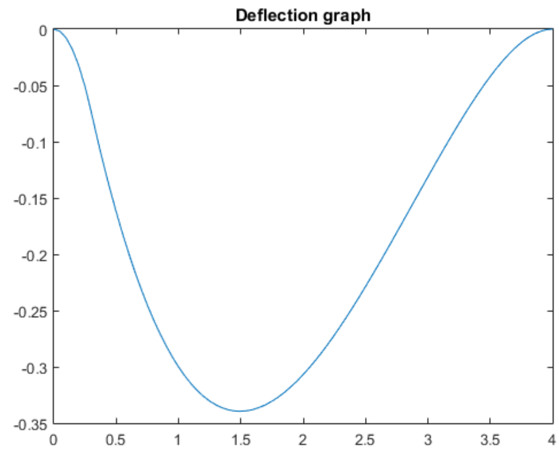

Figure 55: Deflection of Beam

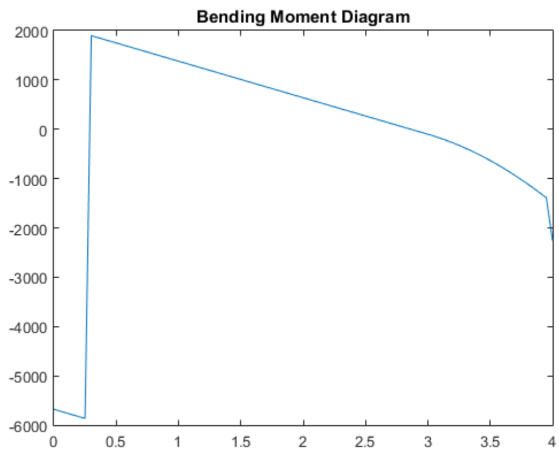

Figure 56: Bending Moment diagram

Case 14: Fixed - Fixed Support (PM+PL+UDL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 4m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 179.3 GPa	X=1.5m	4300N	D	X=0.7m	6980N-m	CCW	X=2to4m	990N/m	D
Section - I				X=1.8m	24000N-m	CCW			
h2 =8.001=h3									
h1 = 189.988, b1 = 6.2223									
b2 =101.98= b3									
I = 19.55583x10^6 mm^4									

Figure 57: Test Case 14

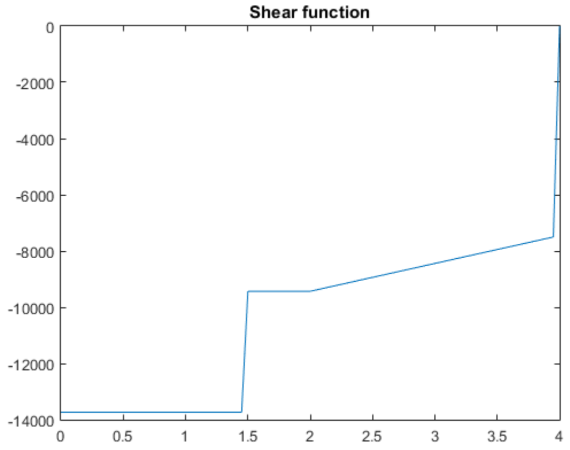

Figure 58: Shear Force Diagram

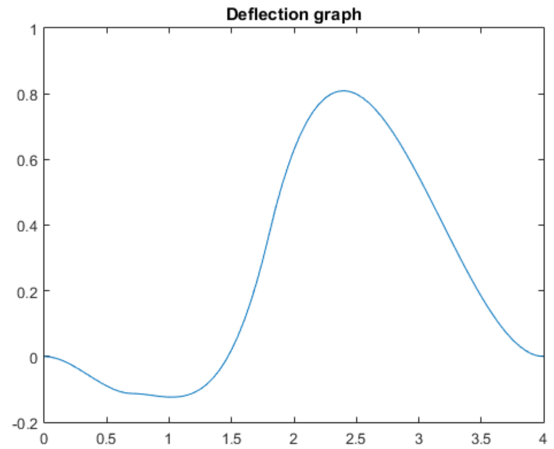

Figure 59: Deflection of Beam

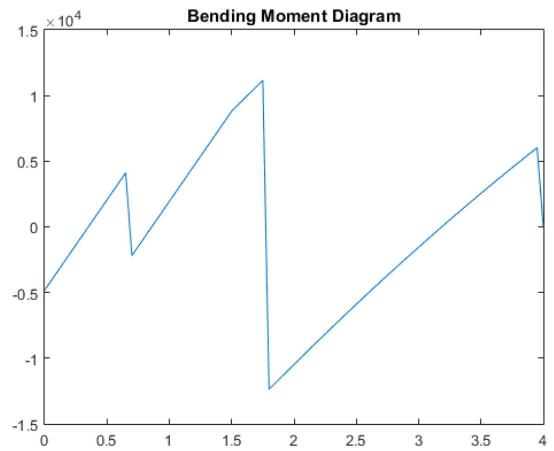

Figure 60: Bending Moment diagram

Case 15: Pin - Pin Support (Only PM)									
Beam Geometry	Point Loads			Moments			UDL		
L = 1.2 m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=0.5m	2800N	D						
Section - Circle	X=1.2m	1000N	D						
D = 25.4 mm									
I = 2.043x10^4 mm^4									

Figure 61: Test Case 15

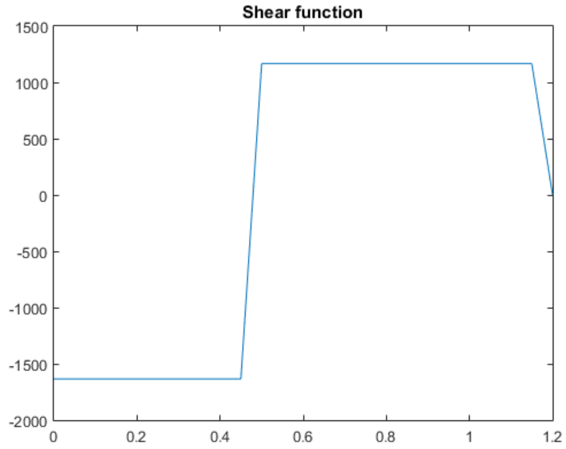

Figure 62: Shear Force Diagram

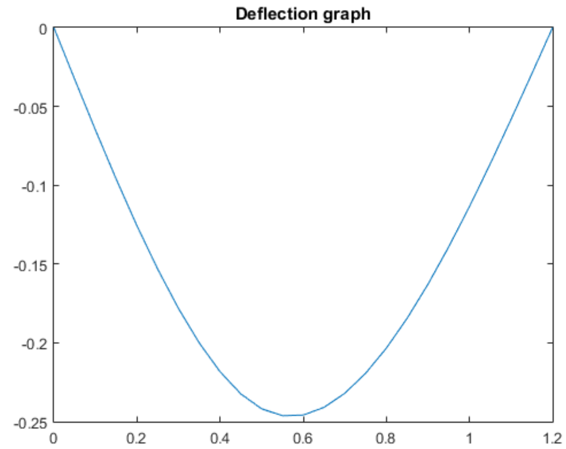

Figure 63: Deflection of Beam

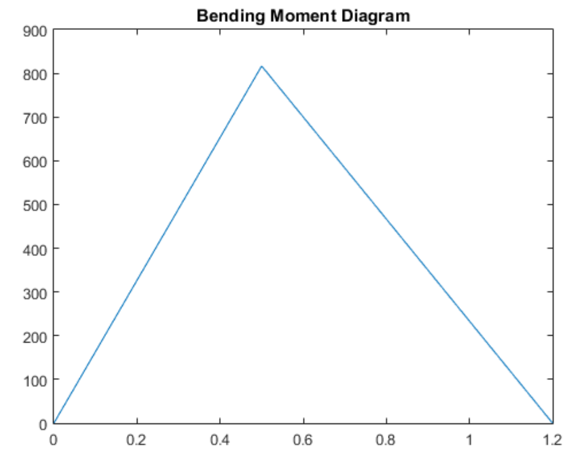

Figure 64: Bending Moment diagram

Case 15: Pin - Pin Support (Only PM)									
Beam Geometry	Point Loads			Moments			UDL		
L = 1.2 m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=0.5m	2800N	D						
Section - Circle	X=1.2m	1000N	D						
D = 25.4 mm									
I = 2.043x10^4 mm^4									

Figure 65: Test Case 14

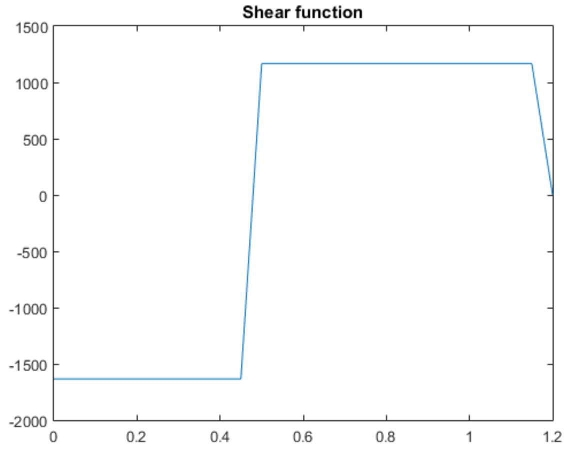

Figure 66: Shear Force Diagram

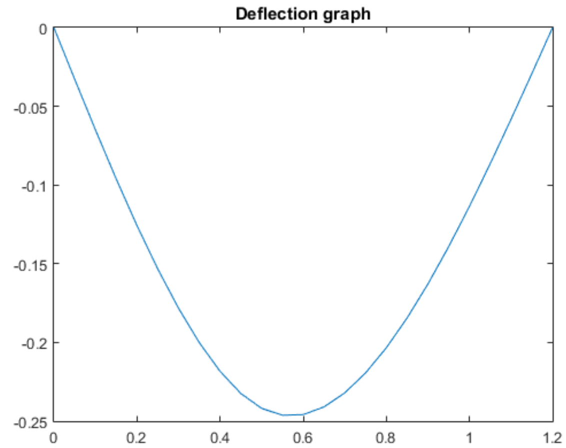

Figure 67: Deflection of Beam

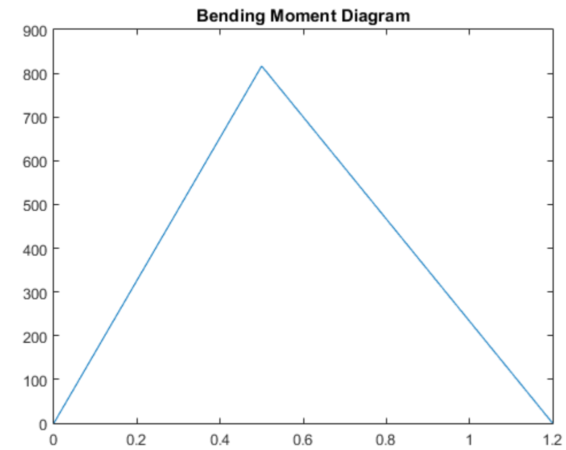

Figure 68: Bending Moment diagram

Case 16: Pin - Pin Support (Only PL)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 1.2 m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa				X=0.3m	1655N-m	CCW			
Section - Circle				X=1.1m	2710N-m	CCW			
D = 25.4 mm									
I = 2.043x10^4 mm^4									

Figure 69: Test Case 16

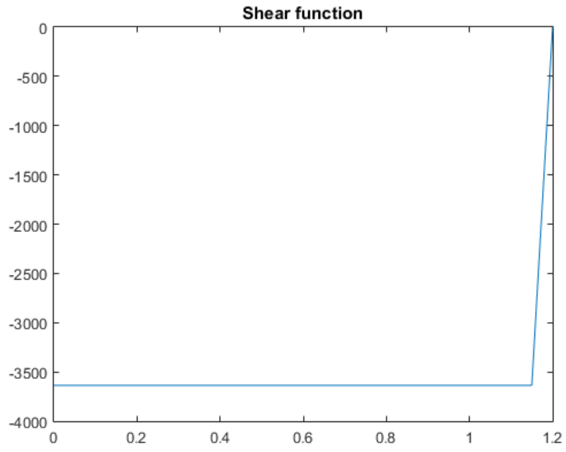

Figure 70: Shear Force Diagram

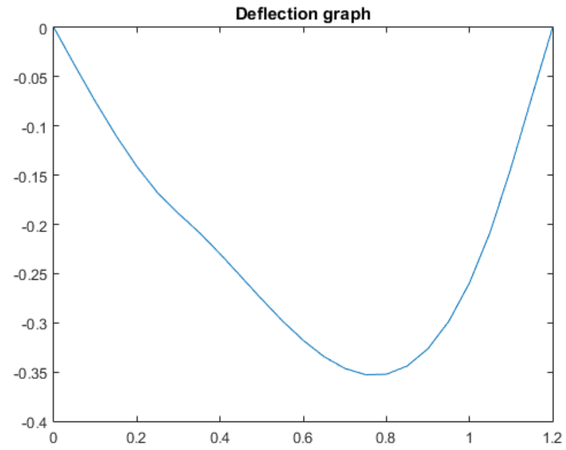

Figure 71: Deflection of Beam

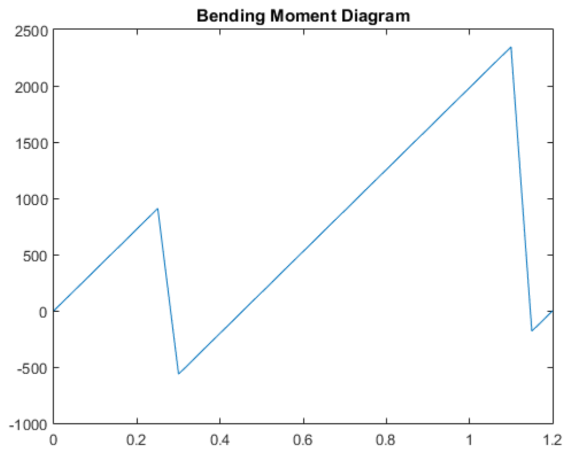

Figure 72: Bending Moment diagram

Case 17: Pin - Pin Support (Only UDL)									
Beam Geometry	Point Loads			Moments			UDL		
L = 1.2 m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa							X=0to0.5m	1800N/m	D
Section - Circle							X=0.8to1m	4300N/m	D
D = 25.4 mm									
I = 2.043x10^4 mm^4									

Figure 73: Test Case 17

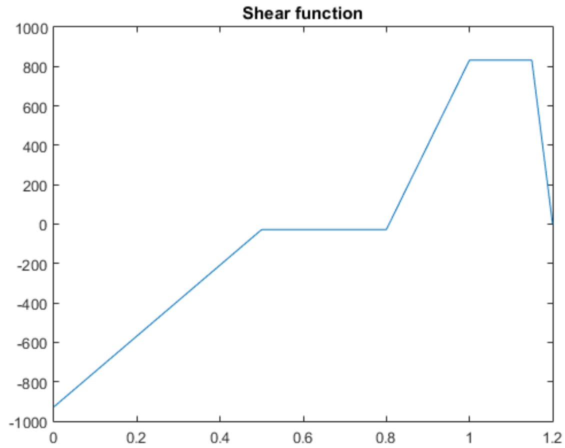

Figure 74: Shear Force Diagram

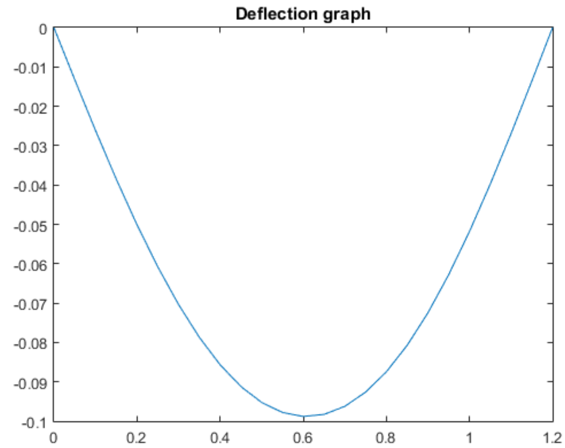

Figure 75: Deflection of Beam

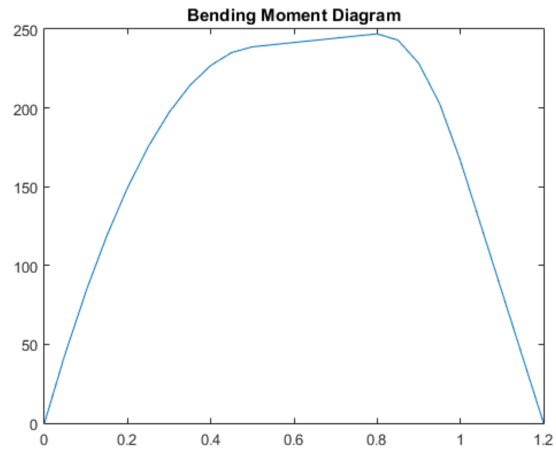

Figure 76: Bending Moment diagram

Case 18: Pin - Pin Support (PL+PM)									
Beam Geometry	**Point Loads**			**Moments**			**UDL**		
L = 1.2 m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=0.7m	1110N	D	X=1.15m	2800N-m	CW			
Section - Circle	X=1.15m	6500N	D						
D = 25.4 mm									
I = 2.043x10^4 mm^4									

Figure 77: Test Case 18

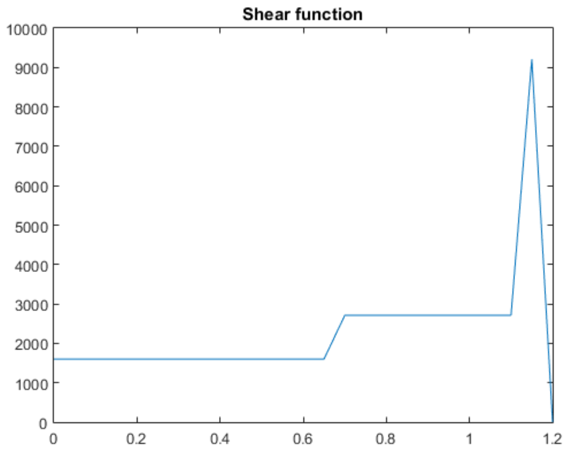

Figure 78: Shear Force Diagram

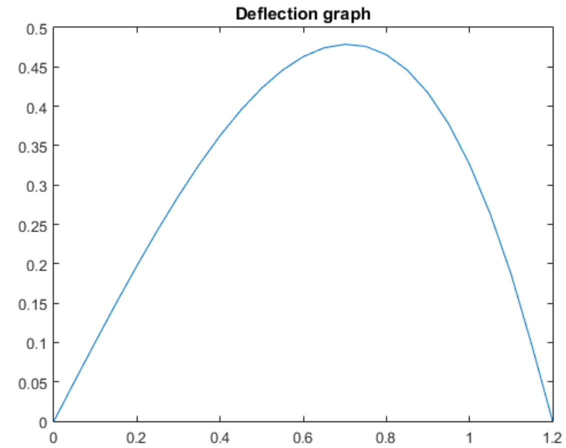

Figure 79: Deflection of Beam

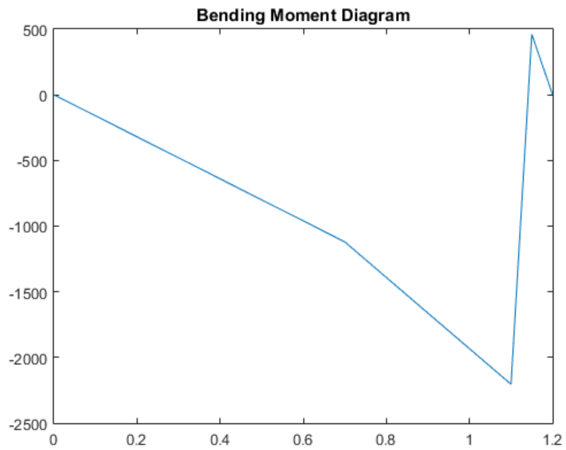

Figure 80: Bending Moment diagram

Case 19: Pin - Pin Support (PL+UDL)									
Beam Geometry	Point Loads			Moments			UDL		
L = 1.2 m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa	X=0.4m	2750N	D				X=0.6to1m	4225N/m	D
Section - Circle	X=1.1m	2800N	D						
D = 25.4 mm									
I = 2.043x10^4 mm^4									

Figure 81: Test Case 19

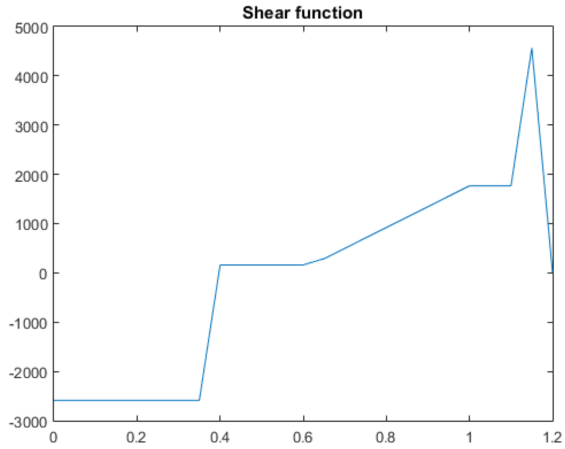

Figure 82: Shear Force Diagram

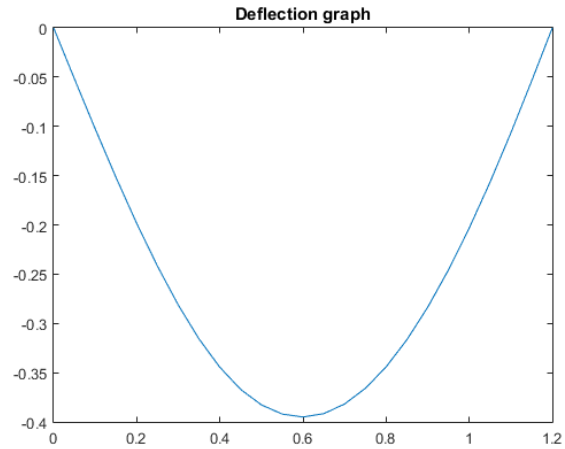

Figure 83: Deflection of Beam

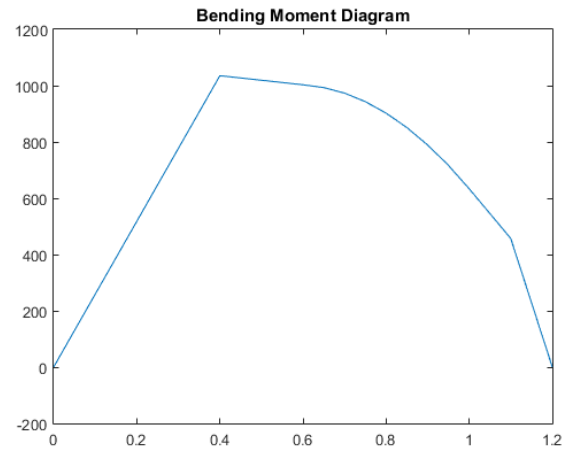

Figure 84: Bending Moment diagram

Case 20: Pin - Pin Support (PM+UDL)									
Beam Geometry	Point Loads			Moments			UDL		
L = 1.2 m	POS	MAG	DIR	POS	MAG	DIR	POS	MAG	DIR
E = 193.1 GPa				X=1m	7800N-m	CW	X=0.3to1m	4600N/m	D
Section - Circle				X=1.1m	8900N-m	CCW			
D = 25.4 mm									
I = 2.043x10^4 mm^4									

Figure 85: Test Case 14

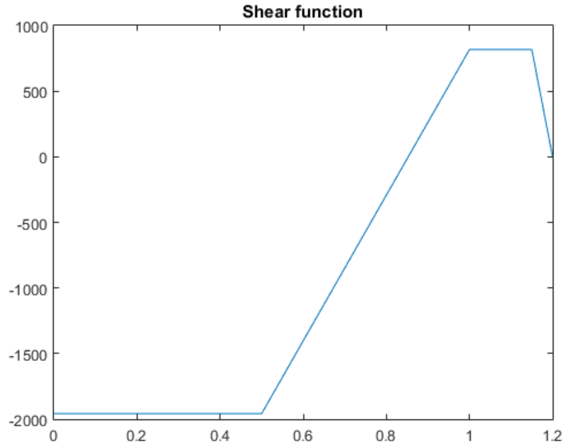

Figure 86: Shear Force Diagram

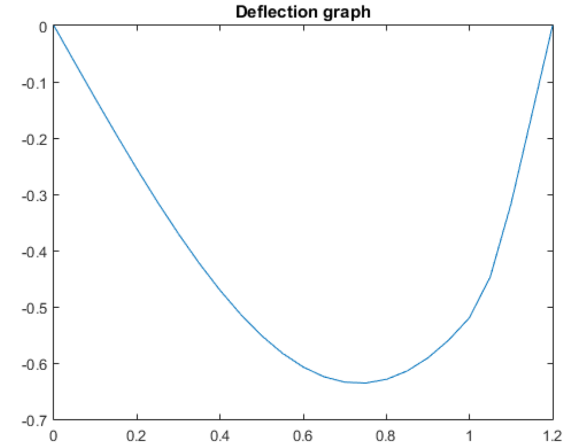

Figure 87: Deflection of Beam

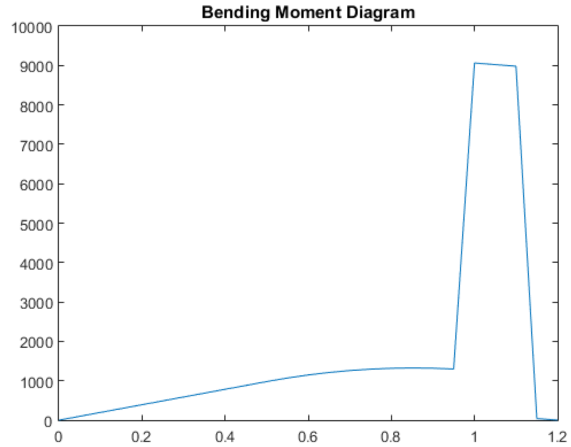

Figure 88: Bending Moment diagram