MATLab Code user manual for dynamic analysis of Rotating Beams

Summer Internship

June - August 2018

By
Prasad ADHAV

Guided by

Dr. Chandra Shekhar PRASAD (IT CAS)

Contents

1 Input Variables **2**

1.1 For Beam_main_v2_0.m 2

1.2 Beam_main_v2_0.m 2

1.3 For data.m 4

1.4 cinput.m 5

2 data.m **5**

3 mass_mat.m **12**

4 stiff_mat.m **13**

5 Standalone codes **15**

5.1 shape_func_plots.m 15

5.2 natfreq_v_mode_plot.m 15

5.3 convergence.m 15

5.4 animation_mode_shape.m 15

5.5 rotating_animation.m 15

The following short report consists of the user manual for the Matlab code written during the internship on the topic of "Mathematical and numerical modeling of rotating beam". The code calculates the natural frequencies of rotating as well as non-rotating beams. The mode shapes are displayed in

1 Input Variables

In this section the input variables will be defined. These variables are to be defined by the user. Default values for each variable is provided as well, if the user wishes to use the default value, he/she just has to press `return` for each input prompt in command window.

1.1 For Beam_main_v2_0.m

The variables to be inserted in the main script are as follows. Default values present in the code are shown in parenthesis.

`E = Young's Modulus in` N/m^2. $(2e + 08)$

`rho = Density in` kg/m^3. (8000)

`L = Length of beam in meters.` (1)

`a = type of cross-section.` (1)

Four type of cross-sections are available as follows:
`1 = rectangle`
`2 = square`
`3 = circle`
`4 = 4 digit NACA profile`

`HMMS = the number of modes shapes to be displayed.` (4)

`lambda = the rotating speed of the beam.` (0)

`nen = number of elements to be used.` (50)

1.2 Beam_main_v2_0.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    MATLab code for Rotating and Non-Rotating Natural Frequencies of      %
%                  cantilevered, uniform cross-sectional beam              %
%               Institute of Thermomechanics,CAS (June - Aug 2018)        %
%          By:- Prasad ADHAV(UPC/ECN)     Guide:- Chandra PRASAD (CAS)     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
clc;
close all;
%% Inputs/Initialisations
disp('To input different valuses please type in values when prompted');
```

```
disp(' ');
disp('Default data for Steel is given,')
disp('to use that default data please press "Enter"');

E = cinput('Input the Young`s Modulus of material(N-m^2) =', 2e+08);
rho = cinput('Input density of material =', 8000);
L = cinput('Length of beam (in mt)', 1);

disp('Select type of cross section');
disp(' ');
disp('1 = Rectangular c/s, I/P breadth (b), height (h)');
disp('2 = Square c/s, I/P side (s)');
disp('3 = Circle c/s, I/P radius (r)');
disp('4 = NACA Profile');
disp(' ');

a = cinput('Select Profile',1);
%Warning for incorrect input for 'a' variable
if a > 4
    F = warndlg('***Error: Profile not available***');
    waitfor(F)
    display('Type in:>> Beam_main_v2_0')
    pause(3)
    return
end
disp('How many mode shapes would you like to Display?')
HMMS = cinput('Enter the number of modes and mode shapes to computed: ', 4);

%By default zero for stationary beam, can be changed for rotating beam
lambda = cinput('Enter value of lambda (Rotating speed) =\n',0);
nen = cinput('Enter number of elements to be used',50); %Number of elements

%[Ix,Iy,m,Px,Py,b,h,Xnx] = data(a,rho,HMMS,L,nen,E);

tic
Dof=2; %Degrees of Freedom per Node
l = L/nen;%Element length
%% Calling functions

[Ix,Iy,m,Px,Py,b,h,Xnx] = data(a,rho,HMMS,L,nen,E); %Input user dats
omega = lambda*sqrt((E*Ix)/(m*(L^4)));
[GM] = mass_mat(nen,l,m,Dof);             %Mass matrix
[GK] = stiff_mat(E,Ix,nen,l,m,omega);   %Stiffness matrix

%% Results
% Global Stiffness Matrix: Applying Boundary Conditions for a Cantilever Beam
% Example for 4 elements is given

GK(1,:) = [];%1st row is deleted we get 9x10 (2*Nel+1) x 2*(Nel+1))
```

```
GK(:,1) = [];%1st column is deleted we get 9x9 (2*Nel+1 x 2*Nel+1)
GK(1,:) = [];%1st row is deleted we get 8x9 (2*Nel x 2*Nel+1))
GK(:,1) = [];%1st row is deleted we get 8x8 (2*Nel x 2*Nel)

% Global Mass Matrix: Applying Boundary Conditions for a Cantilever Beam
% Row 1,2 and columns 1,2 deleted to get a reduced global mass matrix
% The procedure is same as explained above
GM(1,:) =[];
GM(:,1) = [];
GM(1,:) =[];
GM(:,1) = [];

% Calculating Eigen values(D) and Eigen vectors(V) of Reduced Global matrices
[V,D] = eig(GK,GM);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Natural Frequencies from the Reduced Global Matrices in rad/s
Frequency = zeros(1,2*nen);
for i = 1:(2*nen)
    Frequency(i) = sqrt(D(i,i));
end
Frequency = vpa(Frequency,6);
disp('Frequency in rad/s');
disp(Frequency);
kk = 1;
while kk <= 6;
    fprintf('Mode shape # %2f corresponds to nat. freq: %3.3f\n', kk, Frequency(kk) );
    kk = kk + 1;
end

% Natural Frequencies of rotating blade in non-dimensional form
Non_dim_nat_freq = zeros(1,2*nen);
for i=1:(2*nen)
    Non_dim_nat_freq(i) = Frequency(i)/sqrt((E*Ix)/(m*(L^4)));
end
Non_dim_nat_freq = vpa(Non_dim_nat_freq,6);
disp('Non-dimensional Frequency');
disp(Non_dim_nat_freq);


%% Post Process
```


1.3 For data.m

`[Ix,Iy,M,Px,Py,b,h,Xnx] = data(a,rho,HMMS,L,nen,E)`

Once the inputs are given for the required variables above, a function is called, which calculates various properties of the beam such as moment of inertia, mass etc. After the initial basic inputs, some more inputs are required from the user specifying the dimensions of the cross-sections. According to the cross-section selected, only inputs for that particular cross-section are asked.

```
b = breadth of rectangle/square

h = width of rectangle/square

r = radius of circle
```

The fourth cross-section available is 4 digit NACA profile. Three NACA profile can be used since the moment of inertia of only three NACA profiles is pre-computed. The three profiles available are NACA 0012, NACA 0015 and NACA 0018.

```
typeNACA = numeric input foe NACA profile (0012)

trail = open(1)/closed(0) trailing edge of NACA profile.  (0)
```

By default, the trailing edge is set to 0 or closed trailing edge, since that's what the simulation required.

1.4 cinput.m

The function `cinput` can be seen a lot of times in the `Beam_main_v_0.m` script and `data.m`, which asks for user input, if user does not want to specify an input but wants to use the default values provided, the user can just press the return key. This function is used because there are a lot of user inputs, and it is monotonous to input same inputs every time, hence to avoid such such unwanted repetitions. Also, if the Matlab inbuilt function `input` is used, and if user fails to enter a value for required input, the script will stop. Another use of this function is that it avoids the usage of an if loop every time one wants the same functionality, but wants to use the default function `input`.

```
function res = cinput(s,value)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% res = cinput(statement,value)
% used to assign default values if user does not give any inputs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

text = [s ' (default ' num2str(value) ')= '];
res = input(text);
if isempty(res)
    res = value;
end
```

This concludes the input variables required from the user. Now, individual function scripts and the main scripts are seen to understand the working of the code.

2 data.m

In data.m function script, the inputs required are defined in the Beam main script, which can be seen in line 1. This function mainly asks for some extra inputs as seen in section 1 and calculations the coordinates of cross-section, area, mass and moment of inertia of the beam. Along with this it also calculates a matrix (`Xnx`) which contains the information which is used to plot the mode shapes.
The first `if` loop is very straight forward to understand (line $16-212$). According to user input for the profile selected, the area,mass,moment of inertia and profile coordinates are calculated.To verify if the coordinates are correctly calculated, the cross-sections of the beam profile are plotted. Since all the data for NACA profile $3D$ plot cannot be taken, it is plotted in data.m itself, the $3D$ plots for other three profiles is done in

main script.
Mode shapes are calculated from line 215 – 254. And according to the user input,up to 6 mode shapes can be plotted, as seen from line 256 – 323.
The animation of the mode shapes ($2D$) is done in the this function as well, which can be seen from line 325 – 425. Just un-comment the code, and when the Beam_main_v_2_0.m is ran,the animations will be shown one after another. A separate stand alone code is written for $3D$ animations of single and multiple rotating beams (in rotating_animation.m).


```
1  function [Ix,Iy,M,Px,Py,b,h,Xnx] = data(a,rho,HMMS,L,nen,E)
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  %data prompts user to input various values from user
4  %and calculate moment of inertias, mass, area, profile coordinates
5  %and mode shapes
6  %a = type of cross section
7  %Ix, Iy = moment of inertias in respective directions
8  %m = Mass of beam per unit length
9  %Px, Py are variables containing profile cordinates
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11
12 disp(' ');
13 disp('Give required inputs according to the c/s');
14 disp(' ');
15
16 if a == 1
17
18     disp('You have selected rectangle c/s');
19     b = cinput('Breadth of rectangle (in mt)',0.05);
20     h = cinput('Height of rectangle (in mt)',0.01);
21     Ix = (b*h^3)/12;
22     Iy = (b^3*h)/12;
23     area = b*h;
24     M = area*rho;
25
26     figure(1)
27     title('Beam Cross section')
28     Px = [0 b b 0];
29     Py = [0 0 h h];
30     %plot(Px, Py,'b')
31     patch(Px, Py, 'b');
32     axis equal;
33     xlabel('b'); ylabel('h');
34
35 elseif a == 2
36
37     disp('You have selected square c/s');
38     b = cinput('Side of square (in mts)',0.05);
39     h = b;
40     Ix = (b*h^3)/12;
41     Iy = (b^3*h)/12;
42     area = b*h;
43     M = area*rho;
```

```

    figure(1)
    title('Beam Cross section')
    Px = [0 b b 0];
    Py = [0 0 h h];
    %plot(Px, Py, 'b');
    patch(Px, Py, 'b');
    axis equal;
    xlabel('b'); ylabel('h');

elseif a == 3

    disp('You have selected Circular c/s');
    b = cinput('Radius of Circle',0.05);
    h = b;
    Ix = b^4*pi/4;
    Iy = b^4*pi/4;
    area = pi*b*h;
    M = area*rho;
    %Center
    x = 0;
    yy = 0;
    th = 0:pi/50:2*pi;
    Px = b*cos(th) + x;
    Py = b*sin(th) + yy;

    figure(1)
    title('Beam Cross section')
    %plot(Px, Py,'b');
    patch(Px, Py, 'b');
    axis equal;
    xlabel('x'); ylabel('y');

elseif a == 4

    disp('Select NACA Profile');
    disp('Available NACA Profile Moment of inertia');
    disp('NACA 0012');
    disp('NACA 0015');
    disp('NACA 0018');
    typeNACA = cinput('input NACA profile = ','0012');
    c = cinput('input chord length (in mt)',0.1);

    % NACA Profile Calculations
    for a = 4
        % Extract values from type of airfoil string
        Minit = str2double(typeNACA(1));
        Pinit = str2double(typeNACA(2));
        Tinit = str2double(typeNACA(3:4));

        if Minit>0
```

```
            disp('***Warning***')
            disp('Moment of inertia not available')
            disp('Only profile visible')
            disp(' ')
        end

        % Number of grid points
        gridPts = 50;

        % Constants
        a0 = 0.2969;
        a1 = -0.1260;
        a2 = -0.3516;
        a3 = 0.2843;
        disp(' ')
        disp('Selet Trrailing');
        disp('trail = 0, closed trailing edge');
        disp('trail = 1, open trailing edge')
        disp('Default selection is closed trailing edge')
        disp(' ')

        trail = cinput('Select Trailing edge type =',0);
        if trail == 1
            a4 = -0.1015;% Open trailing edge
        else
            a4 = -0.1036;% Closed trailing edge
        end

        %% Calculations

        % Actual percentage values of airfoil properties
        M = Minit/100;
        P = Pinit/10;
        T = Tinit/100;
        b = M;
        h = T;

        % Airfoil grid
        x = linspace(0,1,gridPts)';
        z = linspace(0,1,gridPts);

        % Camber and Gradient
        yc     = ones(gridPts,1);
        dyc_dx = ones(gridPts,1);
        theta  = ones(gridPts,1);
        for i = 1:1:gridPts
            if (x(i) >= 0 && x(i) < P)
                yc(i)     = (M/P^2)*((2*P*x(i))-x(i)^2);
                dyc_dx(i) = ((2*M)/(P^2))*(P-x(i));
        elseif (x(i) >=P && x(i) <=1)
                yc(i)     = (M/(1-P)^2)*(1-(2*P)+(2*P*x(i))-(x(i)^2));
```

```
                dyc_dx(i) = ((2*M)/((1-P)^2))*(P-x(i));
            end
        theta(i) = atan(dyc_dx(i));
        end

        % Thickness distribution
        yt = 5*T.*((a0.*sqrt(x)) + (a1.*x) + (a2.*x.^2) + (a3.*x.^3) + (a4.*x.^4));

        % Upper surface points
        xu1 = x(:) - yt(:).*sin(theta);
        yu1 = yc(:) + yt(:).*cos(theta);

        % Lower surface points
        xl1 = x(:) + yt(:).*sin(theta);
        yl1 = yc(:) - yt(:).*cos(theta);

        xu = c*xu1;
        yu = c*yu1;

        xl = c*xl1;
        yl = c*yl1;

        Px = [xu xl];
        Py = [yu yl];

        [XU,~] = meshgrid(xu,z);
        YU = meshgrid(yu);
        [XL,ZZ] = meshgrid(xu,z);
        YL = meshgrid(yl);

        %% Plot the airfoil c/s (with lines)
        figure(1);
        title('Beam Cross section')
        hold on; grid on;
        axis equal;
        patch(xu,yu,'r-');
        plot(xl,yl,'b-');
        xlim([-0.05 0.15]);  ylim([-0.05 0.05]);
        xlabel('Chord length'); ylabel('Thickness distribution');

        col = ZZ;
        figure(112)
        s1 = surf(XU,ZZ,YU,col);
        s1.EdgeColor = 'none';
        hold on
        fill(YU,XU,ZZ)
        s2 = surf(XL,ZZ,YL,col);
        s2.EdgeColor = 'none';
        %fill(XL,ZZ,YL)
        hold off
```

```
        grid on;
        axis image
        xlabel('Chord Length');zlabel('Thickness distribution');ylabel('
           Beam Length');
    end
    %Moment of inertia
    if Tinit == 12
        Ix = 1.0294e-04;% NACA 0012
    elseif Tinit ==15
        Ix = 1.6084e-04;% NACA 0015
    else
        Ix = 2.3161e-04;% NACA 0018
    end
    Iy = 1; %Dummy value

    area = 2*trapz(xu,yu); %c/s Area calculation
    M = area*rho;
end


%% Mode Shapes Plot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if HMMS >= 7
    disp('    ')
    warning('NOTE: Up to 6 mode shapes (plots) are displayed via the script
       .');
    disp('    ')
end
    Nm = 3*HMMS;
    jj = 1;
   while jj <= Nm;
         betaNL(jj) = fzero(@(betaNL)cosh(betaNL)*cos(betaNL)+1,[jj jj+3]);
         jj = jj+3;
   end

    index = (betaNL~=0);
    betaNLall = (betaNL(index))';
    %fprintf('betaNL value is %2.3f\n', betaNLall);
    betaN = (betaNLall/L)';
k = 1;
wn = ones(1,length(betaN));
while k <= length(betaN);
    wn(k) = betaN(k)^2*sqrt((E*Ix)/(rho*area));
    k = k+1;
end

x = linspace(0, L, nen);
xl = x./L;
sigmaN = zeros(1, HMMS);
for ii = 1:HMMS;
    sigmaN(ii) = (sinh(betaN(ii)*L)-sin(betaN(ii)*L))/(cosh(betaN(ii)*L)+
```

```
        cos(betaN(ii)*L));
end
Tc = '(cosh(betaN(ii).*x(jj))-cos(betaN(ii).*x(jj)))- sigmaN(ii).*(sinh(betaN(ii).*x(jj))-sin(betaN(ii)*x(jj)))';
Xnx = zeros(length(betaN),length(x));

for ii = 1:length(betaN)
    for jj = 1:length(x)
        Xnx(ii,jj) = eval(Tc);
    end
end
```

3 mass_mat.m

As seen in the main script, another function named `mass_mat` is called. `mas_mat.m` computes the global mass matrix. It uses the shape functions provided, and computes elemental mass matrix. And then the assembly of the mass matrix is done. The output of this function is the global mass matrix `GM`. The variables and there significance is provided in the comments in the code.

```
function [GM] = mass_mat(nen,l,m,Dof)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The given function calculates the assembles global mass matrix for
% the calculation of natural freequencies of rotating beam.
% nen = number of elements
% l = element length
% m = mass per unit length
% The function mass_mat gives the global mass matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

GM = zeros(2*(nen+1),2*(nen+1));

% M = m*[(13/35)*l, (11/210)*l^2, (9/70)*l, (-13/420)*l^2;...
%       (11/210)*l^2, (1/105)*l^3, (13/420)*l^2, (-1/140)*l^3;...
%       (9/70)*l, (13/420)*l^2, (13/35)*l, (-11/210)*l;...
%       (-13/420)*l^2, (-1/140)*l^3, (-11/210)*l^2, (1/105)*l^3];

format loose
syms x; %Symbolic calculations initiated

H_1 = ((2*(x/l)^3)-(3*(x/l)^2)+1);
H_2 = ((((x/l)^3)-(2*(x/l)^2)+(x/l))*l);
H_3 = (-(2*(x/l)^3)+(3*(x/l)^2));
H_4 = ((((x/l)^3)-((x/l)^2))*l);
H = [H_1 H_2 H_3 H_4].'; %Shape function matrix

%Mass matrix
M = zeros(Dof*2);
for i = 1:(Dof*2)
    for j = 1:(Dof*2)
        M(i,j) = (m*int((H(i,1)*H(j,1)),0,l));% Elemental Mass Matrix
    end
end

for n = 1:nen
   for i = (2*(n-1)+1):(2*(n+1))
       for j = (2*(n-1)+1):(2*(n+1))

           GM(i,j) = GM(i,j) + M((i-(2*(n-1))),((j-(2*(n-1)))));

       end
   end
end
```

4 stiff_mat.m

The function `stiff_mat` computes the global stiffness matrix. The required input variables and there significance is provided in the comments, the output is global stiffness matrix `GK`.
Since the stiffness matrix is a bit complex to compute in one go,it is divided in to 4 parts as K_1, K_2, K_3 and K_4. These matrices are the assembled individually as GK_1, GK_2, GK_3 and GK_4 respectively. Then they are simply added or subtracted depending on the sign. The stiffness matrix from the internship report can be used to verify the stiffness matrix.

```
function [GK] = stiff_mat(E,Ix,nen,l,m,omega)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The given function calculates the assembles global stiffness matrix for
% the calculation of natural freequencies of rotating beam
% The following inputs are required
% E = Young' modulus
% Ix = moment of inertia about x-axis
% nen = number of elements
% l = element length
% m = mass per unit length
% omega =  rotating speed of beam
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Elemental stiffness matrices
K_1 = (E*Ix)*[(12/l^3), (6/l^2), (-12/l^3), (6/l^2);...
                        (6/l^2), (4/l), (-6/l^2), (2/l);...
                        (-12/l^3), (-6/l^2), (12/l^3), (-6/l^2);...
                        (6/l^2), (2/l), (-6/l^2), (4/l)];

K_2 = [(6/5*l), (6/l^2), (-12/l^3), (6/l^2);...
                        (6/l), (4/l), (-6/l^2), (2/l);...
                        (-12/l^3), (-6/l^2), (12/l^3), (-6/l^2);...
                        (6/l^2), (2/l), (-6/l^2), (4/l)];

K_3 = [(3/5), (1/10), (-3/5), (-l^2/70);...
                        (1/10), (l^2/30), (-1/10), (-l^2/60);...
                        (-3/5), (-1/10), (3/5), (l^2/70);...
                        (-l^2/70), (-l^2/60), (l^2/70), (l^2/30)];

K_4= [(6*l/35), (l^2/28), (-6*l/35), 0;...
                        (l^2/28), (l^3/105), (-l^2/28), (-l^3/140);...
                        (-6*l/35), (-l^2/28), (6*l/35), 0;...
                        0, (-l^2/140), 0, (l^3/105)];

% Calculating where X is the position of element from Fixed End
%Initiating empty stiffness matrix of size of global stiffness matrix
GK1 = zeros(2*(nen+1),2*(nen+1));

for n = 1:nen
    for i = (2*(n-1)+1):(2*(n+1))
        for j = (2*(n-1)+1):(2*(n+1))
            GK1(i,j) = GK1(i,j)+K_1((i-(2*(n-1))),((j-(2*(n-1)))));
```

```matlab
        end
    end
end

X = zeros(1,nen);
for n = 1:nen
    for j = n:nen+1
        X(j) = (j-1)*1;
    end
end

% Calculating A value for each element
A = zeros(1,2*nen+2);
for i = 1:nen
    for j = i:nen
        A(i) = A(i)+((X(j+1))^2)-((X(j))^2);
    end
end


% Global Stiffness Matrix:- Assembly of Element Stiffness Matrices for K_2
GK2 = zeros(2*(nen+1),2*(nen+1));
for n = 1:nen
    for i = (2*(n-1)+1):(2*(n+1))
        for j = (2*(n-1)+1):(2*(n+1))
            GK2(i,j) = GK2(i,j)+(m*(omega^2))*(A(n)/2)*(K_2((i-(2*(n-1))),((j-(2*(n-1))))));
        end
    end
end


% Global Stiffness Matrix:- Assembly of Element Stiffness Matrices for K_3
GK3 = zeros(2*(nen+1),2*(nen+1));
for n = 1:nen
    for i = (2*(n-1)+1):(2*(n+1))
        for j = (2*(n-1)+1):(2*(n+1))
            GK3(i,j) = GK3(i,j)+(m*(omega^2))*(X(n)/2)*(K_3((i-(2*(n-1))),((j-(2*(n-1))))));
        end
    end
end


% Global Stiffness Matrix:- Assembly of Element Stiffness Matrices for K_4
GK4 = zeros(2*(nen+1),2*(nen+1));
for n = 1:nen
    for i = (2*(n-1)+1):(2*(n+1))
        for j = (2*(n-1)+1):(2*(n+1))
            GK4(i,j) = GK4(i,j)+(m*(omega^2)/2)*(K_4((i-(2*(n-1))),((j-(2*(n-1))))));
```

```
        end
    end
end

% Global Stiffness Matrix according to the equation
GK = GK1 + GK2 - GK3 - GK4;
end
```

5 Standalone codes

In folder "Standalone codes", the standalone codes can be found for various extra computations and results visualization. These codes are kept as standalone so as keep the main code less complex and easy to manipulate according as needed. Also, animations take a lot of time to be displayed which may not be desired is one is interested just in the natural frequency values for different conditions.

5.1 shape_func_plots.m

This code plots the cubic shape functions used in the numerical modeling of the rotating beam. This code is straight forward and just has plots.

5.2 natfreq_v_mode_plot.m

This code has all the data collected for various cross-sections, at various rotating speeds, and for various modes for 50 elements. This code also just plots the natural frequencies for different modes, and illustrates the centrifugal stiffening effect discussed in the report.

5.3 convergence.m

As the name suggest this code is used to illustrate the convergence study of implementation. The number of elements used is varied in this case and only first mode, for non-rotating and one rotating speed

5.4 animation_mode_shape.m

In this standalone script, variables are predefined and one can see the $2D$ animation of first 4 modes of the beam. The variables used in this script are same as mentioned in section 1, although here the user is not prompted to input any values, the values are predefined.

5.5 rotating_animation.m

In rotating animation code, one can see the illustration of multiple beams rotating and vibrating in first 4 modes. In this script as well the needed input variables are predefined and user is not prompted to input any values. The variables used are same as described in section 1.