Developing Python Code for Static and Modal Analysis of Plane Truss Structure

R Vara Prasad, Assistant Professor, Anil Neerukonda Institute of Technology & Sciences, Visakhapatnam & India, vara.remo@gmail.com

Divya Dunga , Student, Anil Neerukonda Institute of Technology & Sciences, Visakhapatnam & India, divyadunga250@gmail.com

Deepak Kumar.B, Student, Anil Neerukonda Institute of Technology & Sciences, Visakhapatnam & India, drocks673@gmail.com

Bhargavi Sri, Student, Anil Neerukonda Institute of Technology & Sciences, Visakhapatnam & India, bhargavishree415@gmail.com

D Santosh Sunil Varma, Student, Anil Neerukonda Institute of Technology & Sciences, Visakhapatnam & India, sunilvarma024@gmail.com

G S Nethikonda, Student, Anil Neerukonda Institute of Technology & Sciences, Visakhapatnam & India, srinethi97531@gmail.com

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

Keywords — ANSYS APDL, Automation, Jupyter Notebook, MATLAB, Python, Truss.

I. Introduction

There is no confidential that engineering problems that were decade ago are considered troublesome except on special supercomputers, now routinely run on modest commodity personal computers. However, it is not quite as well recognized that these progresses have also been be associated by important changes in programming languages, all of which aims at greatly reducing the effort of writing codes. Scientific applications are no exceptions; many legacy programs written in Fortran, C or even C++ could probably be rewritten in a more concise way using a variety of scripting languages, such as MATLAB, Scilab, IDL, Mathematica and others. Scripting languages have various benefits over 'conventional' languages, these include:

(1) Scripting languages develop portable codes,

(2)Requires less memory management responsibility by the programmer and

(3) Allow data types to be dynamically set at run time.

All of these factors contribute to making codes less bug prone and so ultimately lead to increased productivity and shorter code development cycles. These advantages, unfortunately, have a price; scripting languages tend to be inferior in raw numerical performance when benchmarked against compiled codes. Our experience, however, has been that scripting languages, and in particular Python, often perform above expectation, and are thus well positioned to find the optimum between satisfactory execution time and acceptable program development cost.

II. Methodology

The present work consists of two parts. In the first part a fully working python code was developed using finite element procedure in the jupyter notebook(Jupyter Notebook (open source code), which began as the iPython Notebook project, is a development environment for writing and executing Python code. Jupyter Notebook is often used for exploratory data analysis and visualization.
Project Jupyter is the top-level project name for all of the subprojects under development, which includes Jupyter Notebook. Jupyter Notebooks can also run code for other programming languages such as Julia and R.) which is accompanied by python 3.9.10.in the second part, the results obtained using python are compared with the results obtained from ANSYS APDL 2021 R21 software. The python developed in this current work is applicable for any type of truss structure of isotropic material and different loading and support conditions. This python code is used to investigate both static and vibrational analysis (A technique used to determine a structure's vibration characteristics: – Natural frequencies – Mode shapes – Mode participation factors (how much a given mode participates in a given direction). It is most fundamental of all the dynamic analysis types. Modal analysis allows the design to avoid resonant vibrations or to vibrate at a specified frequency (speakers, for example). Gives engineers an idea of how the design will respond to different types of dynamic loads. Helps in calculating solution controls (time steps, etc.) for other dynamic analyses. Recommendation: Because a structure's vibration characteristics determine how it responds to any type of dynamic load, always perform a modal analysis first before trying any other dynamic analysis) of any truss structure. To check whether the code developed was giving exact results, the present work considered railway bridge truss problem and the data was taken from the textbook of introduction to finite elements by chandrapatla and ashok belugundu.

III. STATIC AND VIBRATIONAL ANALYSIS USING PYTHON CODE

This is the following python code developed in jupyter notebook by importing various scientific computing libraries:

```
# importing libraries
import math
import matplotlib.pyplot as plot
import numpy as np
# creating path for a file
file = open(r"D:\Remo desktop\PhD work\Python Project 2021\out.txt","w+")
# creating input data and print
noofnodes=int(input("enter the total number of nodes"))
file.write("the total number of nodes= "+str(noofnodes)+ '\n')
noofelements=int(input("enter the total number of elements"))
file.write("the total number of elements= "+str(noofelements)+ '\n')
coordinates={}
for i in range(1,noofnodes+1):
coordinates[i]=list(map(int,input("enter coordinates for node "+ str(i)+ ' in mm :').split(",")))
    file.write('coordinates for node' + str(i)+ 'in mm : '+ str(coordinates[i])+ '\n')
area=float(input("enter area in mm-square"))
file.write("area in mm-square= "+str(area)+ '\n')
elasticity=float(input("enter the modulus of elasticity in N/mm-square"))
file.write("modulus of elasticity in N/mm-square= "+str(elasticity)+ '\n')
startend={}
for i in range(1,noofelements+1):
startend[i]=list(map(int,input("enter the start node and end node for element "+ str(i)+ ' :').split(",")))
file.write('the start node and end node for element '+ str(i)+ ' :'+str(startend[i])+ '\n')
noofsuppnodes=int(input("enter the total number of nodes having supports"))
file.write('the total number of nodes having supports '+str(noofsuppnodes)+ '\n')
nodesandtypesupport=[]
for i in range(noofsuppnodes):
    x=int(input("enter the node number having support"))
    print("enter the type of support")
    print("h for hinged/fixed")
    print("hrs for horizontal roller support")
    print("vrs for vertical roller support")
    t=input()
    file.write('the node number having support is '+str(x)+' type of support is '+t+ '\n')
    nodesandtypesupport.append([x,t.lower()])
print(nodesandtypesupport)
noofloadnodes=int(input("enter total number of loaded nodes "))
file.write('total number of loaded nodes '+str(noofloadnodes)+ '\n')
loadnodes=[]
typeofload={}
for i in range(noofloadnodes):
    x=int(input("enter the node number having load "))
    loadnodes.append(x)
     typeofload[x]=list(map(float,input('enter the horizontal and vertical loads in N ').split(',')))
     file.write('the node number having load is '+str(x)+' horizontal and vertical loads in N is '+str(typeofload[x])+ '\n')
density=float(input("enter density in gm/mm-cube "))
poission=float(input("enter poission's ratio "))
```

```
    file.write('density in gm/mm-cube is '+str(density)+ '\n')
    file.write('poissions ratio  is '+str(poission)+ '\n')
    # calculating length of element direction cosines
    lengthofelements={}
    cos={}
    sin={}
    for i in range(1,noofelements+1):
        x,y=startend[i][0],startend[i][1]
        a,b=coordinates[x][0],coordinates[x][1]
        c,d=coordinates[y][0],coordinates[y][1]
        l=math.sqrt((d-b)**2+(c-a)**2)
        lengthofelements[i]=l
        cos[i]=(c-a)/l
        sin[i]=(d-b)/l
    for i in range(1,noofelements+1):
        print("length of elements "+str(i)+": ",lengthofelements[i])
        print("cos value of elements "+str(i)+": ",cos[i])
        print("sin value of elements "+str(i)+": ",sin[i])
    # calculating element stiffness matrix and element mass matrix
    stiffness={}
    for i in range(1,noofelements+1):
        l=cos[i]
        m=sin[i]
        mat=[[l**2,l*m,-l**2,-l*m],[l*m,m**2,-l*m,-m**2],[-l**2,-l*m,l**2,l*m],[-l*m,-m**2,l*m,m**2]]
        a=(area*elasticity)/lengthofelements[i]
        for j in range(4):
            for k in range(4):
                mat[j][k]=a*mat[j][k]
        print("stiffness matrix of element "+str(i))
        for j in range(4):
            for k in range(4):
                print(mat[j][k],end='    ')
            print()
        print()
        print()
        stiffness[i]=mat
    mass={}
    for i in range(1,noofelements+1):
         mat=[[2,0,1,0],[0,2,0,1],[1,0,2,0],[0,1,0,2]]
        a=(area*density*lengthofelements[i])/6
        for j in range(4):
            for k in range(4):
                mat[j][k]=a*mat[j][k]
        print("mass matrix of element "+str(i))
        for j in range(4):
            for k in range(4):
                print(mat[j][k],end='    ')
        print()
        print()
        print()
        mass[i]=mat
    # calculating Global Stiffness matrix and global mass matrix
    ndof = noofnodes * 2
    print(ndof)
    globstifmat=[]
for i in range(ndof):
    globstifmat.append([0]*ndof)
x=0
asinode={}
for i in range(1,noofnodes+1):
    asinode[i]=[x,x+1]
    x+=2
e=startend
n=asinode
d={}
for i in range(1,noofelements+1):
    x=[]
    p=e[i]
    q=p[0]
    r=p[1]
    x.extend([n[q][0],n[q][1],n[r][0],n[r][1]])
    d[i]=x
print(d)
    for i in range(ndof):
    for j in range(ndof):
        s=0
        for k in d:
            if(i in d[k] and j in d[k]):
                zzz=stiffness[k]
                s+=zzz[d[k].index(i)][d[k].index(j)]
        globstifmat[i][j]=s
print("global stiffness matrix ")
for j in range(ndof):
    for k in range(ndof):
        print(globstifmat[j][k],end='    ')
 print()
print()
print(len(globstifmat))
print(len(globstifmat[0]))
ndof = noofnodes * 2
print(ndof)
globmassmat=[]
for i in range(ndof):
    globmassmat.append([0]*ndof)
for i in range(ndof):
    for j in range(ndof):
        s=0
        for k in d:
            if(i in d[k] and j in d[k]):
                zzz=mass[k]
                s+=zzz[d[k].index(i)][d[k].index(j)]
        globmassmat[i][j]=s
print("global mass matrix ")
for j in range(ndof):
 for k in range(ndof):
  print(globmassmat[j][k],end='    ')
   print()
print()
# calculating global load vector
globloadvector=[0]*noofnodes*2
for i in typeofload:
    globloadvector[asinode[i][0]]=typeofload[i][0]
    globloadvector[asinode[i][1]]=typeofload[i][1]
print(globloadvector)
print(nodesandtypesupport)
coltorem=[]
rowtorem=[]
for i in nodesandtypesupport:
    if(i[1]=='h'):
        coltorem.extend(asinode[i[0]])
        rowtorem.extend(asinode[i[0]])
    elif(i[1]=='hrs'):
        coltorem.append(asinode[i[0]][1])
        rowtorem.append(asinode[i[0]][1])
    elif(i[1]=='vrs'):
        coltorem.append(asinode[i[0]][0])
        rowtorem.append(asinode[i[0]][0])
print(coltorem)
print(rowtorem)
npglobstifmat=np.array(globstifmat)
onpglobstifmat=npglobstifmat
print(npglobstifmat)
print(npglobstifmat.shape)
# Reducing Global Stiffness Matrix, global mass matrix
and global load vector
#removing row
npglobstifmat=np.delete(npglobstifmat, rowtorem, 0)
print(npglobstifmat)
print(npglobstifmat.shape)
#removing column
npglobstifmat=np.delete(npglobstifmat, rowtorem, 1)
print(npglobstifmat)
rednpglobstifmat=npglobstifmat
print(npglobstifmat.shape)
#mass matrix removing rows nd columns
npglobmassmat=np.array(globmassmat)
onpglobmassmat=npglobmassmat
print(npglobmassmat)
npglobmassmat=np.delete(npglobmassmat, rowtorem, 0)
print(npglobmassmat)
npglobmassmat=np.delete(npglobmassmat, rowtorem, 1)
print(npglobmassmat)
rednpglobmassmat=npglobmassmat
print(npglobmassmat.shape)
print(globloadvector)
#coverting to column vector
npglobloadvector=np.array(globloadvector)
orignpglobloadvector=npglobloadvector
print(npglobloadvector.shape)
#removing rows
npglobloadvector=np.delete(npglobloadvector,
rowtorem, 0)
print(npglobloadvector)
print(npglobloadvector.shape)
# Solving KQ = F  to determine Nodal Displacments
#linear eqn solving
a = npglobstifmat
b = npglobloadvector
nodaldisplacement = np.linalg.solve(a, b)
print("nodal displacement matrix in mm")
print(nodaldisplacement)
#changing dimension
xx=[]
zz=list(nodaldisplacement)
i=0
s=set(rowtorem)
for j in range(noofnodes*2):
    if(j not in s):
        xx.append(nodaldisplacement[i])
        i+=1
    else:
        xx.append(0)
print(xx)
file.write('nodal displacements in mm are ' + str(xx)+'\n')
#changing to col matrix
npnodaldisplacement=np.array(xx)
print(npnodaldisplacement.shape)
print('nodal displacement vector in mm')
print(npnodaldisplacement)
# calculating Element Stresses
elementstresses={}
for i in range(1,noofelements+1):
    l=cos[i]
    m=sin[i]
    m1=np.array([-l,-m,l,m]).reshape(1,-1)
    pq=[]
    for z in d[i]:
        pq.append(npnodaldisplacement[z])
    zz=np.array(pq).reshape(-1,1)
    a=(elasticity/lengthofelements[i])
    b=np.matmul(m1,zz)
    elementstresses[i]=a*b
print("element stresses in N/mm-square")
for i in elementstresses:
print("element"+str(i)+"="+str(elementstresses[i][0][0]))
 file.write('element '+str(i)+' stress in N/mm-square=
'+str(elementstresses[i][0][0])+'\n')
#reaction of support
npglobstifmat=np.array(globstifmat)
x=np.matmul(npglobstifmat,npnodaldisplacement)
y=x-orignpglobloadvector
reactionmat=y
print("reaction at support nodes in N")
print(reactionmat)
j=0
for i in nodesandtypesupport:
    if(i[1]=='h'):
        print('reaction node at R'+str(i[0])+"X in N
"+str(reactionmat[(i[0]-1)*2]))
        print('reaction node at R'+str(i[0])+"Y in N
"+str(reactionmat[(i[0]-1)*2+1]))
        file.write('reaction node at R'+str(i[0])+"X in N
"+str(reactionmat[(i[0]-1)*2])+ '\n')
        file.write('reaction node at R'+str(i[0])+"Y in N
"+str(reactionmat[(i[0]-1)*2+1])+ '\n')
    elif(i[1]=='hrs'):
        print('reaction node at R'+str(i[0])+"Y in N
"+str(reactionmat[(i[0]-1)*2+1]))
        file.write('reaction node at R'+str(i[0])+"Y in N
"+str(reactionmat[(i[0]-1)*2+1])+ '\n')
    elif(i[1]=='vrs'):
        print('reaction node at R'+str(i[0])+"X in N
"+str(reactionmat[(i[0]-1)*2]))
      file.write('reaction  node  at  R'+str(i[0])+"X  in  N
"+str(reactionmat[(i[0]-1)*2])+ '\n')
  #modal analysis
  #(globalstiffnessmatrix-(lambda)globalmassmatrix)X
  class Eigen(object):
    def _init_(self, *args, **kwargs):
      return super()._init_(*args, **kwargs)
       def Rescale(self, x):
      max = x.max()
      min = x.min()
      s = x.shape
      n = s[0]
      amax = max
      if abs(min) > max: amax = abs(min)
      for i in range(n):
        x[i] = x[i] / max
      return x
   def RescaleEigenVectors(self, evec):
      dims = evec.shape
      ndofs = dims[0]
      for i in range(ndofs):
        evec[:,i] = self.Rescale(evec[:,i])
      return evec
   def GetOrthogonalVector(self, ndofs, trial, mg, ev,evec):
      const = 0
      s = [ndofs]
      sumcu= np.zeros(s)
      for e in range(ev  ):
        U = evec[:,e]
        const += trial @ mg @ U
        cu = [x * const for x in U]
        sumcu += cu
      trial = trial - sumcu
      return trial
    def Solve(self,kg, mg, tolerance = 0.00001 ):
      dims = kg.shape
      ndofs = dims[0]
      s = (ndofs,ndofs)
      evec = np.zeros(s)
      s = (ndofs)
      eval = np.zeros(ndofs)
      trial = np.ones(ndofs)
      eigenvalue0 = 0
      for ev in range(ndofs):
        print("Computing eigenvalue and eigen vector " +
str(ev) + "... " , end="")
     converged = False
        uk_1 = trial
        k = 0
        while converged == False:
          k += 1
         if ev > 0:
            uk_1                                        =
self.GetOrthogonalVector(ndofs,uk_1,mg,ev,evec)
          vk_1 = mg @ uk_1
          uhatk = np.linalg.solve(kg,vk_1)
          vhatk = mg @ uhatk
          uhatkt = np.transpose(uhatk)
          eigenvalue = (uhatkt @ vk_1)/(uhatkt @ vhatk)
          denominator = math.sqrt(uhatkt @ vhatk)
          uk = uhatk/denominator
          tol  =  abs((eigenvalue  -  eigenvalue0)  /
eigenvalue)
          if tol <= tolerance:
            converged = True
            evec[:,ev] = uk
            eval[ev] = eigenvalue
            print("Eigenvalue = " + str(eigenvalue))
            print('no of iterations= ',k)
          else:
            eigenvalue0 = eigenvalue
            uk_1 = uk
            if k > 1000:
              evec[:,ev] = uk
              eval[ev] = eigenvalue
              print ("could not converge. Tolerance = "
+ str(tol))
              break
     self.eigenvalues = eval
        return evec
    rednpglobstifmat.shape
    rednpglobmassmat.shape
    # compute eigenvalues and eigen vectors
    e = Eigen()
    evec = e.Solve(rednpglobstifmat,rednpglobmassmat)
    evect = e.RescaleEigenVectors(evec)
    eval = e.eigenvalues
    neval = len(eval)
    print("eigen values")
    eigvalues=e.eigenvalues
    print(e.eigenvalues)
    file.write("eigen values are "+str(eigvalues)+'\n')
    print(len(e.eigenvalues))
    print("eigen vectors")
    print(len(evect))
    print(evect)
    for i in evect:
    file.write("eigen vectors are "+str(list(i))+'\n')
    from scipy.linalg import eigvalsh
    ab=eigvalsh(rednpglobstifmat,rednpglobmassmat)
    print(ab[0])
    f1=np.sqrt(ab[0])
    f1
    f2=np.sqrt(ab[0])
    f2=f2/(2*3.14)
    f2
    #frequency
    freq=[]
    for i in eigvalues:
    freq.append(math.sqrt(i)/(2*3.14))
    print("natural frequency in hertz",freq)
    file.write("natural frequency in hertz "+str(freq)+ '\n')
    import matplotlib.pyplot as plt
    z=1
    for i in evect:
      time      = np.array(i)
      amplitude  = np.sin(time)
    plot.plot(time, amplitude)
    plot.title('mode'+str(z))
    plot.xlabel('vector')
    plot.ylabel('sin(vector)')
    plot.grid(True, which='both')
    plot.axhline(y=0, color='k')
    plot.savefig(r"D:\Remo desktop\PhD work\Python
Project 2021\mode"+str(z)+'.png', bbox_inches='tight')
    plot.show()
    plot.show()
      z+=1
  file.close()
```

This code is saved as projectcode.ipynb.To verify whether the code giving correct results or not, the present work considers a railway bridge truss structure as shown

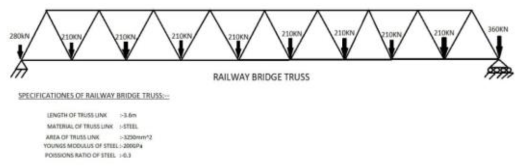

280kN
210KN
210KN
210KN
210KN
210KN
210KN
210KN
210KN
360KN
RAILWAY BRIDGE TRUSS
SPECIFICATIONES OF RAILWAY BRIDGE TRUSS:--
LENGTH OF TRUSS LINK :-3.6m
MATERIAL OF TRUSS LINK :-STEEL
AREA OF TRUSS LINK :-3250mm^2
YOUNGS MODULUS OF STEEL :-200GPa
POISSIONS RATIO OF STEEL :-0.3

Fig.3.1: Railway Bridge Truss Structure

Now run this python code by clicking run in the jupyter notebook interface. The code prompts the following input data:

The total number of nodes = 19
The total number of elements = 35
Coordinates for node1in mm: [0, 0]
Coordinates for node2in mm: [1800, 3118]
Coordinates for node3in mm: [3600, 0]
Coordinates for node4in mm: [5400, 3118]
Coordinates for node5in mm: [7200, 0]
Coordinates for node6in mm: [9000, 3118]
Coordinates for node7in mm: [10800, 0]
Coordinates for node8in mm: [12600, 3118]
Coordinates for node9in mm: [14400, 0]
Coordinates for node10in mm: [16200, 3118]
Coordinates for node11in mm: [18000, 0]
Coordinates for node12in mm: [19800, 3118]
Coordinates for node13in mm: [21600, 0]
Coordinates for node14in mm: [23400, 3118]
Coordinates for node15in mm: [25200, 0]
Coordinates for node16in mm: [27000, 3118]
Coordinates for node17in mm: [28800, 0]
Coordinates for node18in mm: [30600, 3118]
Coordinates for node19in mm: [32400, 0]
Area in mm^2 = 3250.0
Modulus of elasticity in N/mm^2 = 200000.0
The start node and end node for element 1: [1, 3]
The start node and end node for element 2: [3, 5]
The start node and end node for element 3: [5, 7]
The start node and end node for element 4: [7, 9]
The start node and end node for element 5: [9, 11]
The start node and end node for element 6: [11, 13]
The start node and end node for element 7: [13, 15]
The start node and end node for element 8: [15, 17]
The start node and end node for element 9: [17, 19]
The start node and end node for element 10: [2, 4]
The start node and end node for element 11: [4, 6]
The start node and end node for element 12: [6, 8]
The start node and end node for element 13: [8, 10]
The start node and end node for element 14: [10, 12]
The start node and end node for element 15: [12, 14]
The start node and end node for element 16: [14, 16]
The start node and end node for element 17: [16, 18]
The start node and end node for element 18: [1, 2]
The start node and end node for element 19: [2, 3]
The start node and end node for element 20: [3, 4]
The start node and end node for element 21: [4, 5]
The start node and end node for element 22: [5, 6]
The start node and end node for element 23: [6, 7]
The start node and end node for element 24: [7, 8]
The start node and end node for element 25: [8, 9]
The start node and end node for element 26: [9, 10]
The start node and end node for element 27: [10, 11]
The start node and end node for element 28: [11, 12]
The start node and end node for element 29: [12, 13]
The start node and end node for element 30: [13, 14]
The start node and end node for element 31: [14, 15]
The start node and end node for element 32: [15, 16]
The start node and end node for element 33: [16, 17]
The start node and end node for element 34: [17, 18]
The start node and end node for element 35: [18, 19]
The total number of nodes having supports 2
The node number having support is 1 type of support is h
The node number having support is 19 type of support is hrs
Total number of loaded nodes 10
The node number having load is 1 horizontal and vertical loads in N is [0.0, -280000.0]
The node number having load is 3 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 5 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 7 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 9 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 11 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 13 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 15 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 17 horizontal and vertical loads in N is [0.0, -210000.0]
The node number having load is 19 horizontal and

vertical loads in N is [0.0, -360000.0]

Density in gm/mm^3 is 0.00785

Poissions ratio is 0.3

IV. RESULTS AND DISCUSSIONS

The results obtained from python code is verified by doing analysis of railway-bridge in ANSYS APDL software. The static results are compared for validation for checking the code is correct or not.

By observing Table 4.1, Table 4.2 and Table 4.3 results obtained from python code and ANSYS APDL are compared for nodal displacements, element stresses and support reactions for validation. The python code result values are exactly matched with ANSYS APDL result values which shows that the code developed in the current work is validated and is suitable for any truss structure for static and vibrational analysis.

Node Number	Nodal displacements in mm obtained from Ansys APDL in mm		Nodal displacements in mm obtained from Python Code in mm	
	Qx	Qy	Qx	Qy
1	0.000	0.000	0.000	0.000
2	80.572	-52.717	80.572	-52.717
3	2.686	-103.880	2.686	-103.880
4	75.201	-150.400	75.201	-150.400
5	10.072	-192.650	10.072	-192.650
6	65.801	-227.920	65.801	-227.920
7	20.815	-257.000	20.815	-257.000
8	53.715	-277.540	53.715	-277.540
9	33.572	-290.720	33.572	-290.720
10	40.286	-294.590	40.286	-294.590
11	47.001	-290.720	47.001	-290.720
12	26.857	-277.540	26.857	-277.540
13	59.758	-257.000	59.758	-257.000
14	14.772	-227.920	14.772	-227.920
15	70.501	-192.650	70.501	-192.650
16	5.372	-150.400	5.372	-150.400
17	77.887	-103.880	77.887	-103.880
18	7.9378E-13	-52.717	7.9378E-13	-52.717
19	80.572	0.000	80.572	0.000

Table 4.1: Validation of Nodal Displacements.

Reaction forces	Reaction forces from Ansys APDL in N	Reaction forces from Python Code in N
R1x	1.1816E-08	1.1816E-08
R1y	1119999.9	1119999.9
R19y	1199999.9	1199999.9

Table 4.2: Validation of Reaction Forces.

As the code is validated, the modal analysis of given railway-bridge truss structure is done and the natural frequency and mode shapes are shown. The mode shapes for the corresponding 35 natural frequencies are plotted in the Fig: 4.1(mode1-mode35) The 35 natural frequency

Element Number	Element stresses from Ansys APDL in N/mm2	Element stresses from Python code in N/mm2
1	149.21	149.21
2	410.32	410.32
3	596.83	596.83
4	708.74	708.74
5	746.04	746.04
6	708.74	708.74
7	596.83	596.83
8	410.32	410.32
9	149.21	149.21
10	-298.42	-298.42
11	-522.23	-522.23
12	-671.44	-671.44
13	-746.04	-746.04
14	-746.04	-746.04
15	-671.44	-671.44
16	-522.23	-522.23
17	-298.42	-298.42
18	-298.44	-298.44
19	298.44	298.44
20	-223.83	-223.83
21	223.83	223.83
22	-149.22	-149.22
23	149.22	149.22
24	-74.61	-74.61
25	74.61	74.61
26	0	0
27	2.78E-12	2.78E-12
28	74.61	74.61
29	-74.61	-74.61
30	149.22	149.22
31	-149.22	-149.22
32	223.83	223.83
33	-223.83	-223.83
34	298.44	298.44
35	-298.44	-298.44

Table 4.3: Validation of Element Stresses

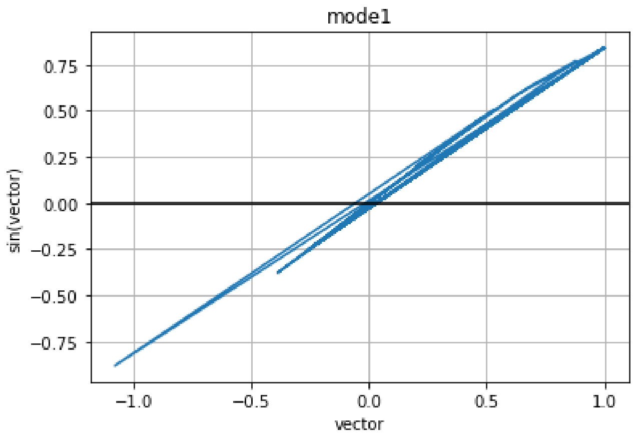


Fig.4.1: Mode Shape 1 for corresponding frequency 1.

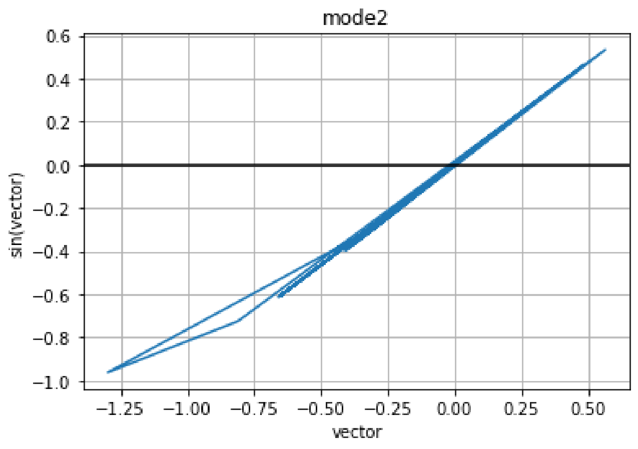


Fig.4.2: Mode Shape 2 for corresponding frequency 2.

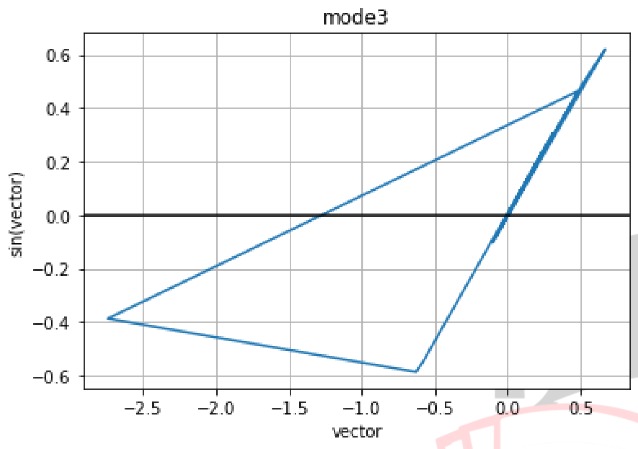


Fig.4.3: Mode Shape 3 for corresponding frequency 3.

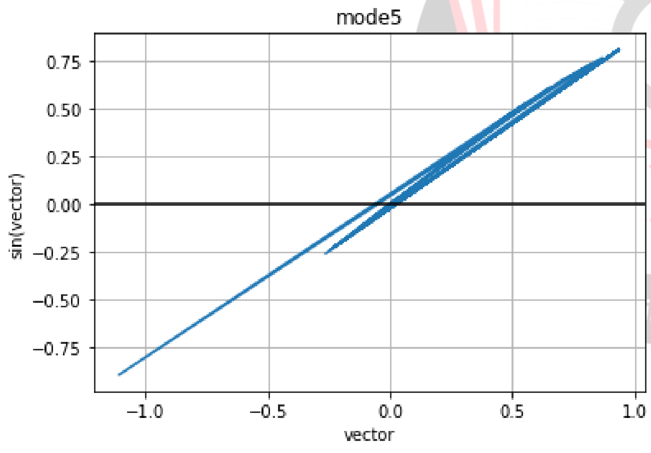


Fig.4.4: Mode Shape 5 for corresponding frequency 5.

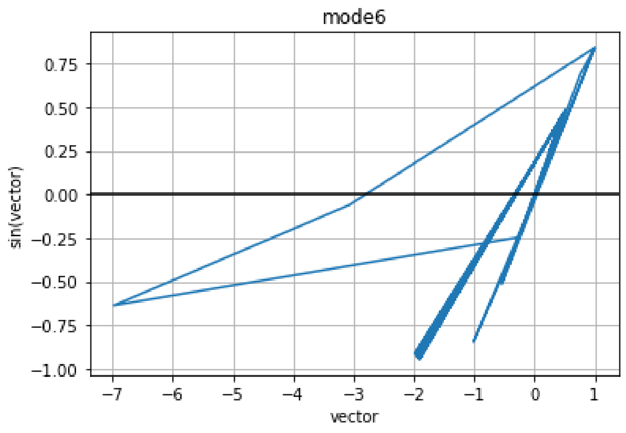


Fig.4.5: Mode Shape 6 for corresponding frequency 6

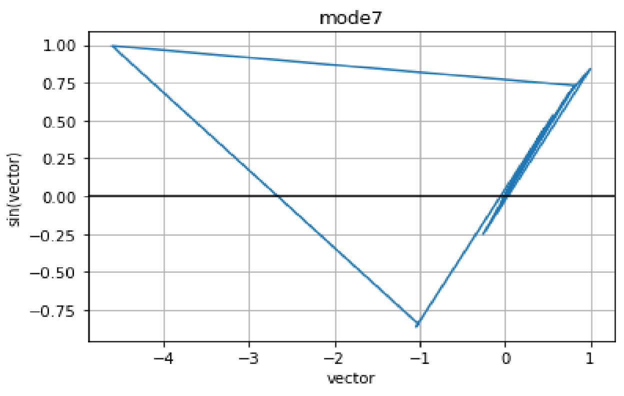


Fig.4.6: Mode Shape 7 for corresponding frequency 7.

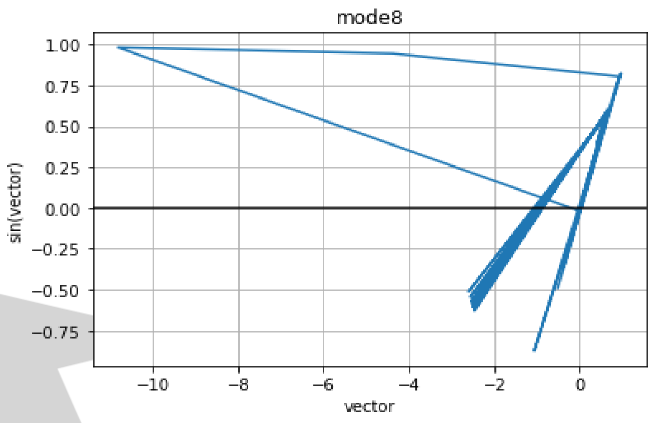


Fig.4.7: Mode Shape 8 for corresponding frequency 8.

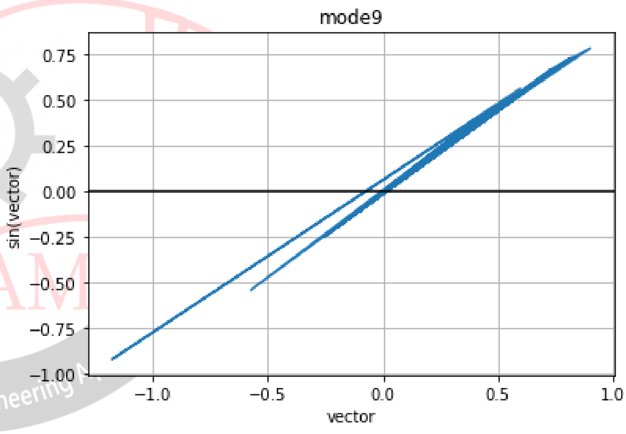


Fig.4.9: Mode Shape 9 for corresponding frequency 9.

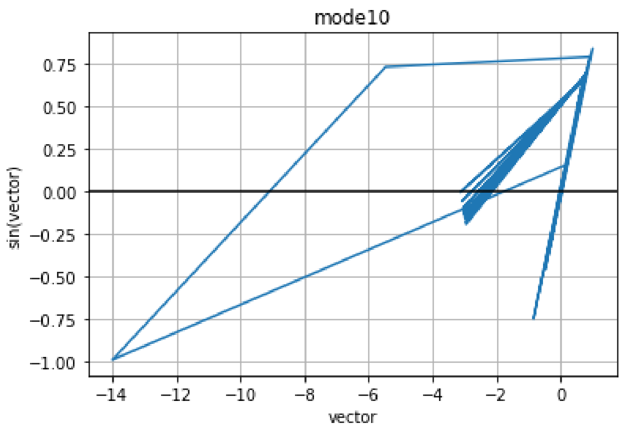


Fig.4.10: Mode Shape 10 for corresponding frequency 10.

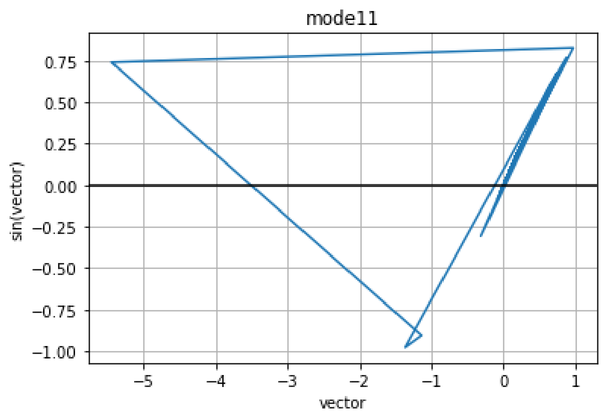
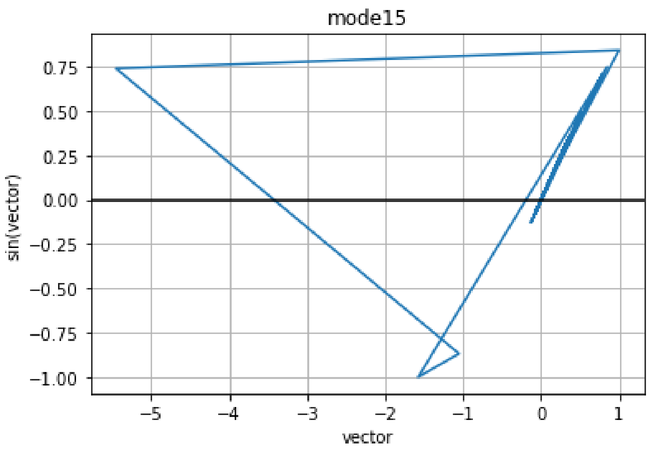


Fig.4.11: Mode Shape 11 for corresponding frequency 11.



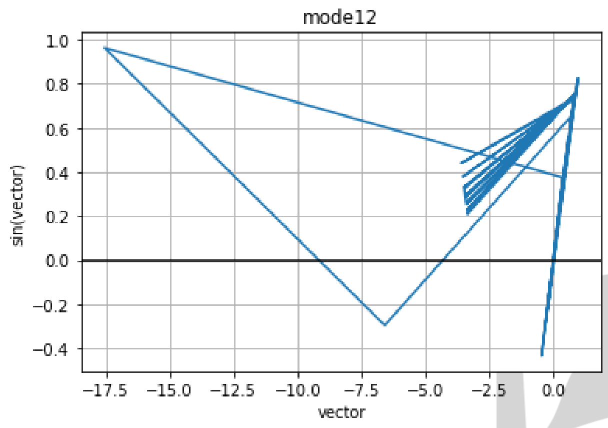
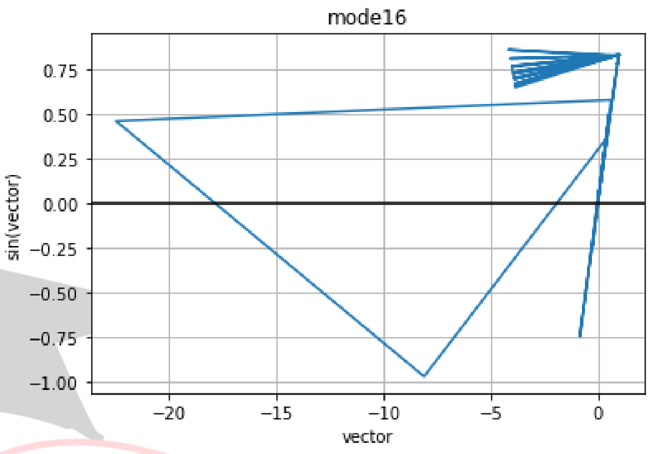
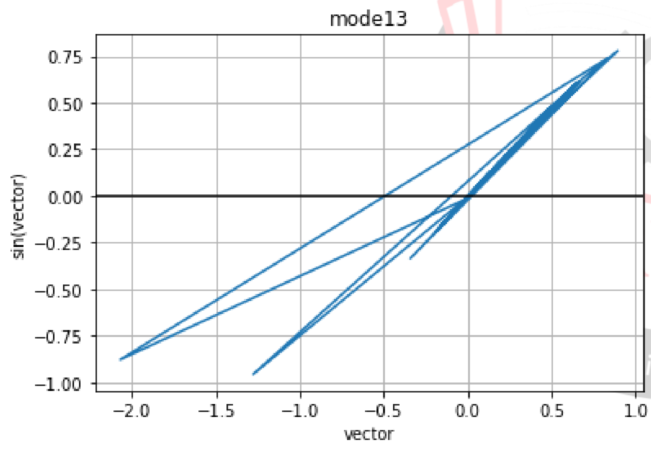
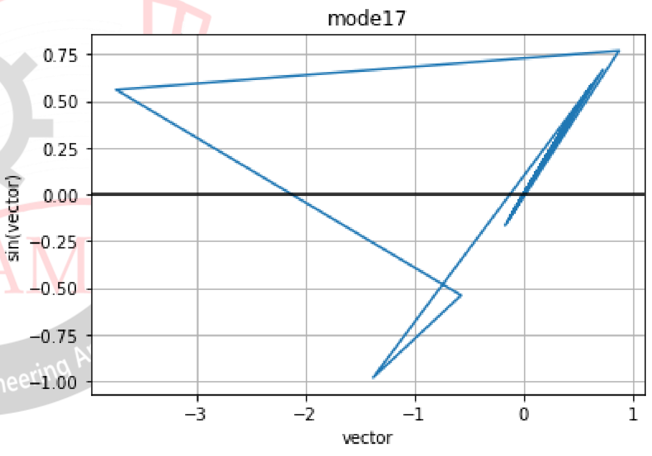
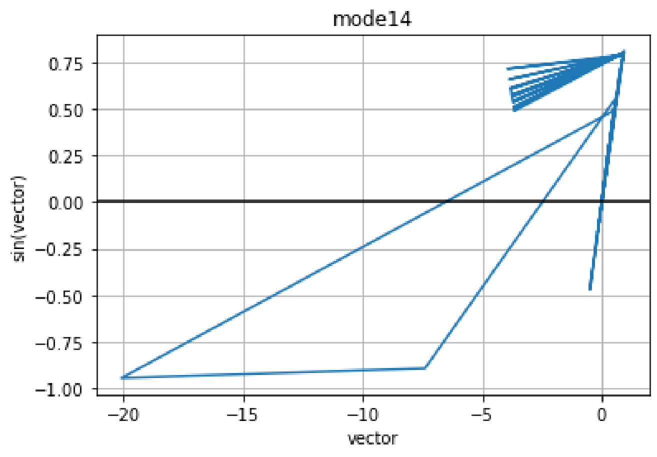


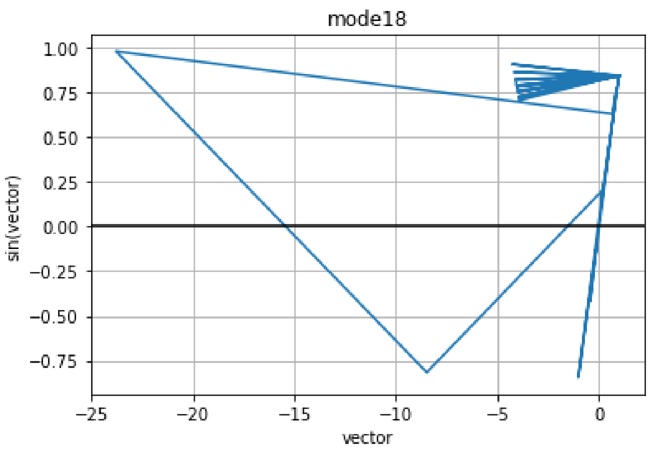
Fig.4.12: Mode Shape 12 for corresponding frequency 12.

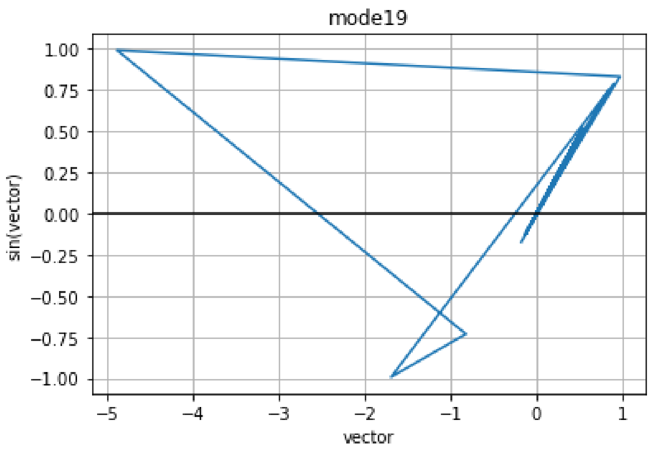


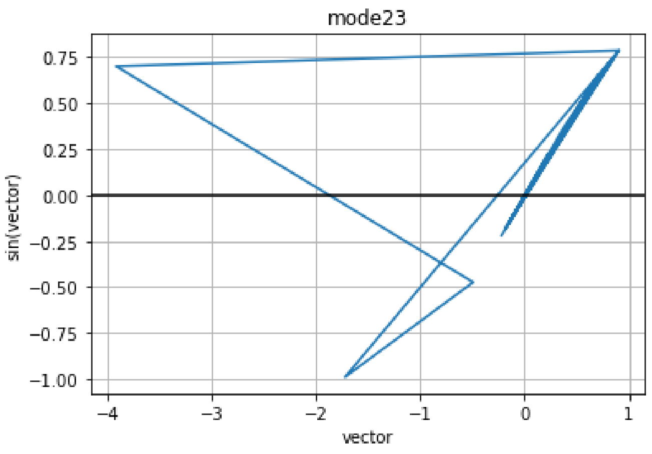


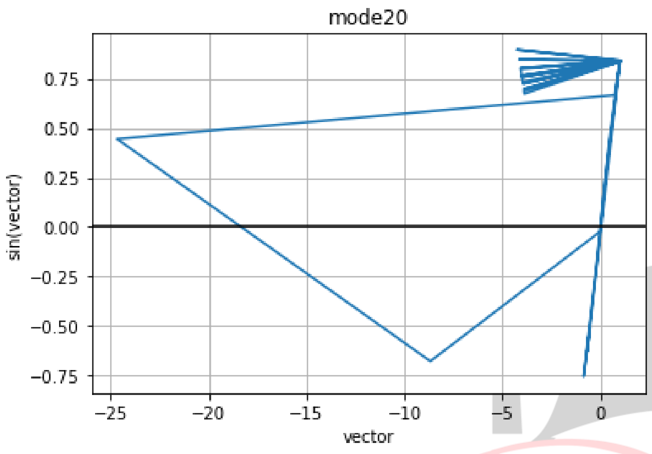


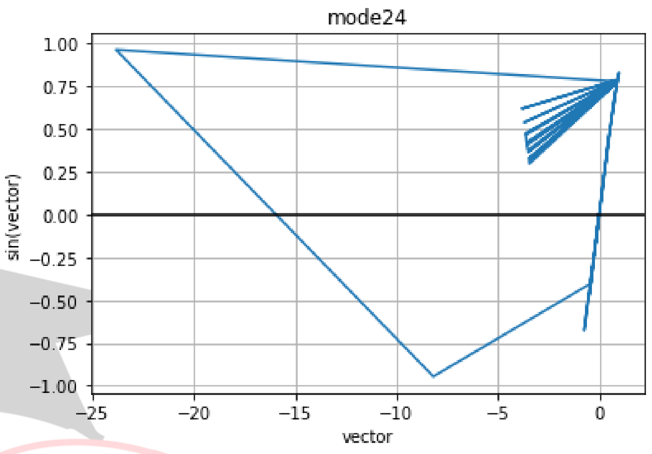


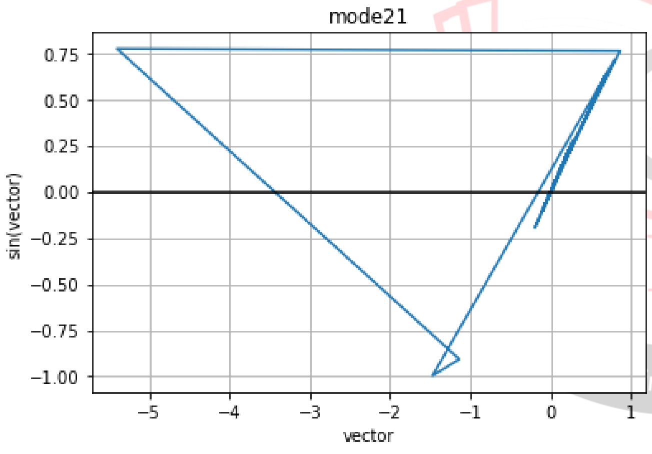


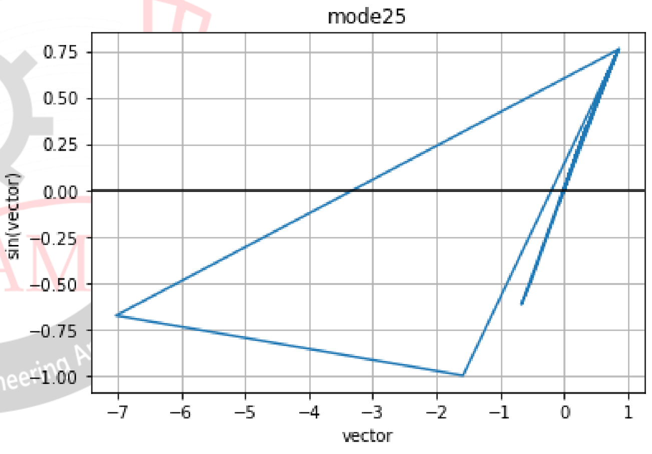

mode19
sin(vector)
vector
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−5
−4
−3
−2
−1
0
1

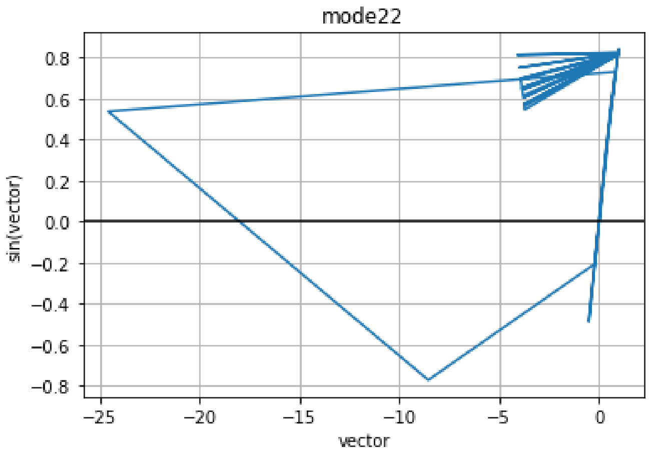

mode23
sin(vector)
vector
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−4
−3
−2
−1
0
1

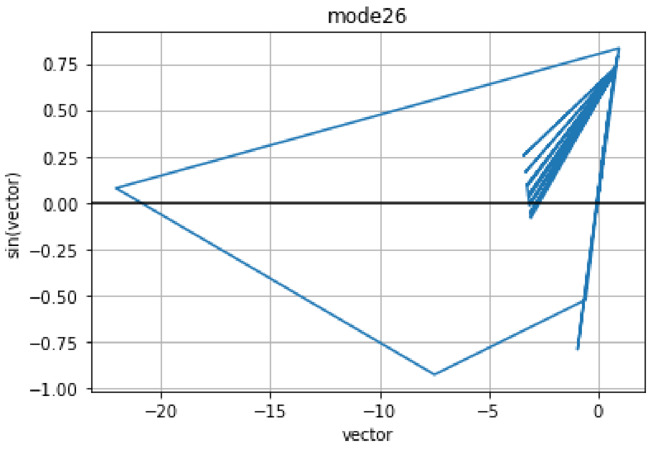

mode20
sin(vector)
vector
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−25
−20
−15
−10
−5
0

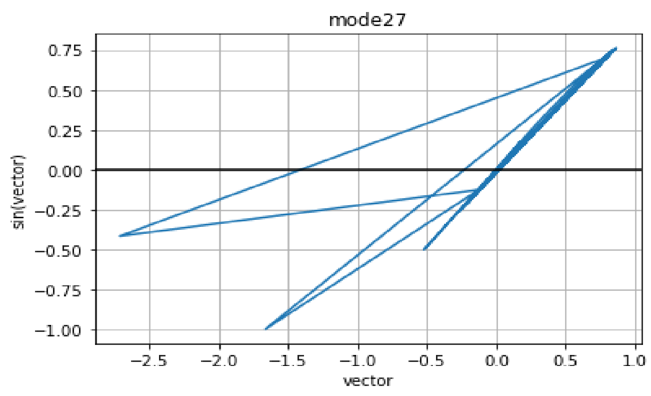

mode24
sin(vector)
vector
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−25
−20
−15
−10
−5
0

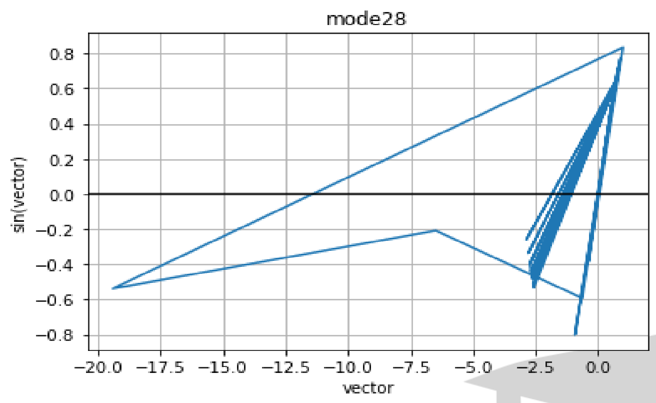

mode21
sin(vector)
vector
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−5
−4
−3
−2
−1
0
1

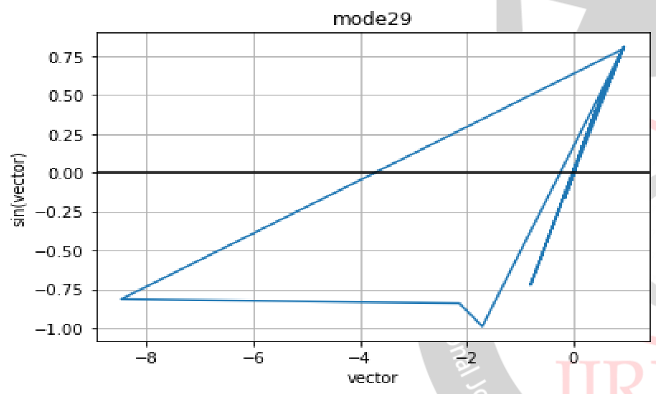

mode25
sin(vector)
vector
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−7
−6
−5
−4
−3
−2
−1
0
1

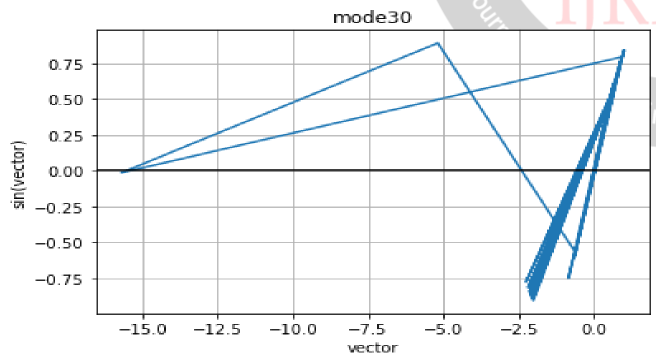

mode22
sin(vector)
vector
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−25
−20
−15
−10
−5
0

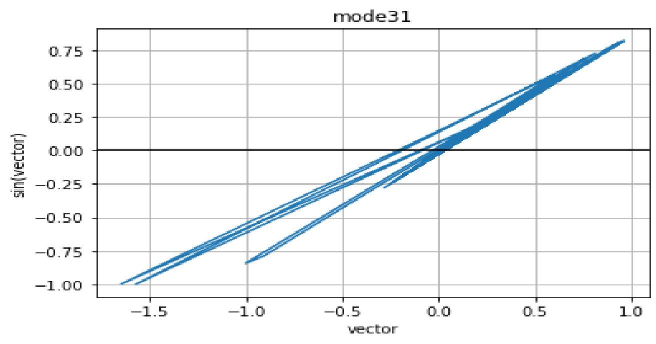

mode26
sin(vector)
vector
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−20
−15
−10
−5
0

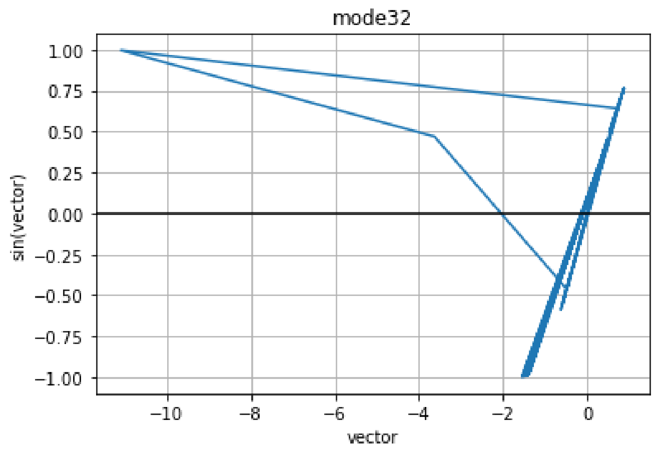


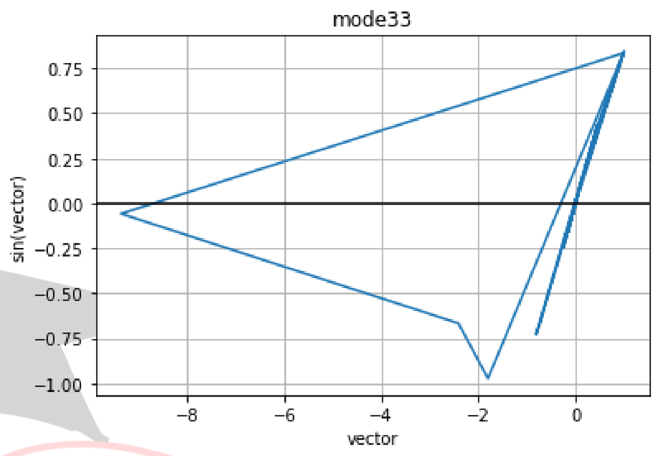


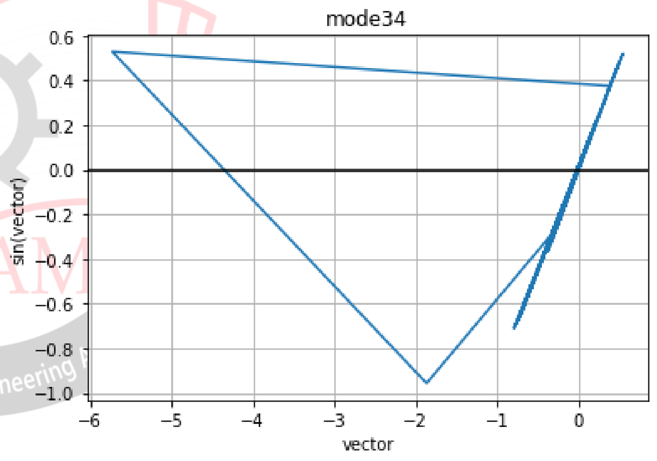












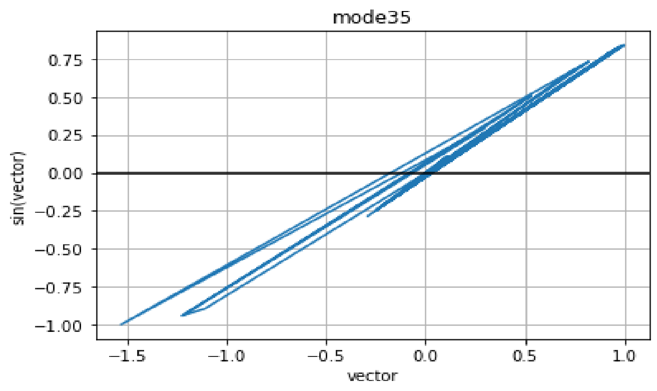


Fig.4.13: Mode Shapes From 13 to 35 for corresponding Natural Frequencies From 13 to 35.

S.No	Natural frequency from Python Code In Hertz
1	0.00770
2	0.02299
3	0.02991
4	0.05340
5	0.07532
6	0.07105
7	0.05334
8	0.02723
9	0.01277
10	0.01068
11	0.02301
12	0.02379
13	0.00970
14	0.01411
15	0.00855
16	0.00795
17	0.00823
18	0.00790
19	0.00805
20	0.00786
21	0.00795
22	0.00783
23	0.00789
24	0.00780
25	0.00785
26	0.00788
27	0.00782
28	0.00784
29	0.00780
30	0.00781
31	0.00778
32	0.00779
33	0.00777
34	0.00778
35	0.00777

Table 4.4: Natural frequencies of Railway Bridge.

V. CONCLUSION AND FUTURE SCOPE

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

The python code which we developed is suitable for all truss structures with any material. Our code is restricted to linear properties only. The code which we developed is used for automation for conducting stress analysis and vibration analysis.

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

This Python code developed for any type of truss structure. In future it can be upgraded to work for all structural members like beams plates, columns etc. Also, same procedure can be followed to develop python code that can work for plates with notches. Also we can implement in composite materials also.

References

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

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

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

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

[5] scikit-fem: A Python package for finite element assembly **Tom Gustafsson**1 **and G. D. McBain DOI:** 10.21105/joss.02369

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

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

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

[9] Python Scientific Lecture Notes: https://scipy-lectures.github.io/

[10] Scientific computing tools for python: http://www.scipy.org/

[11] NumPy: http://www.numpy.org/

[12] SciPy library: http://www.scipy.org/scipylib/

[13] Matplotlib: http://matplotlib.org/

[14] IPython: http://ipython.org/

[15] Text book of introduction to finite element analysis by Chandrapatla and Ashok Belugundu