

Codice Python

Mattia Gianelli

Marzo - Aprile 2021

```
1 import time
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy import interpolate
5 from numpy import sqrt
6 from numpy import zeros, array, dot
7 from scipy.linalg import eig
8 import math
9 import sectionproperties.pre.sections as sections
10 from sectionproperties.analysis.cross_section import
    CrossSection
11 import os.path
12
13 t = time.time() # Time starting
14
15 # Acquisition and data handling
16
17 print('Which geometry?')
18 name_geometry = str(input())
19 file_handle = open(name_geometry+'.txt', 'r') #
    Input geometry & opens a handle to your file, in read
    mode
20 lines_list = file_handle.readlines() # Read in all
    the lines of your file into a list of lines
21 x, y, z = (float(val) for val in lines_list[0].split
    ()) # Extract dimensions from lines, cast values to
    float
22 P = [[float(val) for val in line.split()] for line in
    lines_list[0:]] # Triple-nested list comprehension
    to get data
23 file_handle.close() # Close the file
24 if os.path.isfile(name_geometry+'_Prop.txt'):
25     file_handle2 = open(name_geometry+'_Prop.txt', 'r
    ') # Input geometric properties
26     lines_list2 = file_handle2.readlines()
27     Beam = lines_list2[0].split()
28 else:
29     file_handle2 = open(name_geometry+'_Prop.txt', 'a
    ') # Write geometric properties into a txt file
30     print('Type of element? Prismatic or Not
    Prismatic')
31     file_handle2.write("{}\t{}".format('Type:', str(
    input(''))))
32     file_handle2.write('\n')
```



```

67 Mn = np.zeros((dofs * 2, dofs * 2)) # Mass Matrix
    of generic beam element
68 Gn = np.zeros((dofs * 2, dofs * 2)) # Geometric
    Stiffness Matrix of generic beam element
69 Gn_e = np.zeros((dofs * 2, dofs * 2))
70 R = np.zeros((dofs * 2, dofs * 2))
71 rho = 7850 # Density, kg/m^3
72 print('Boundary conditions? 0 = Free, 1 = Restrained
    at the hub')
73 BC = int(input())
74 if BC == 0:
75     w = 0 # Rotational speed, in rad/s
76 elif BC == 1:
77     print('Rotational speed? [RPM]')
78     w = float(input()) * 2 * math.pi / 60 #
    Rotational speed, in rad/s
79 E = 205e9 # Young Modulus, Pa
80 nu = 0.3 # Poisson Ratio
81 G = E / (2 + 2 * nu) # Shear Modulus, Pa
82 conv = 1000 # Conversion factor (from mm to m)
83 A = np.zeros(int(ns)) # Cross-section Area
84 xct = np.zeros(int(ns)) # X-Distance between Shear
    Center and Centroid
85 yct = np.zeros(int(ns))
86 xcc = np.zeros(int(ns)) # X-Distance between Global
    Axis and Centroid
87 ycc = np.zeros(int(ns))
88 IxR = np.zeros(int(ns)) # Principal Second Moment
    of Inertia
89 IyR = np.zeros(int(ns))
90 Ix = np.zeros(int(ns)) # Second Moment of Inertia
    in global coordinate
91 Iy = np.zeros(int(ns))
92 k_x = np.zeros(int(ns)) # Shear Factor in X
    direction
93 k_y = np.zeros(int(ns))
94 Ip = np.zeros(int(ns)) # Polar Moment of Inertia
95 It = np.zeros(int(ns)) # De Saint Venant Torsional
    Constant
96 Cm = np.zeros(int(ns)) # Warping Constant
97 delta = np.zeros(int(ns))
98 dx = np.zeros(int(ns))
99 dy = np.zeros(int(ns))
100

```

```

101
102 # Function Definition
103
104
105 def shearparameter(E, IM, G, K, A, L):
106     return 12 * E * IM / (G * K * A * L**2)
107
108
109 def rotationmatrix(delta, R):
110     c = math.cos(delta)
111     s = math.sin(delta)
112
113     R[0][0], R[3][3], R[6][6], R[7][7], R[10][10], R
114     [13][13] = 1, 1, 1, 1, 1, 1
115     R[1][1], R[2][2], R[4][4], R[5][5] = c, c, c, c
116     R[8][8], R[9][9], R[11][11], R[12][12] = c, c, c
117     , c
118     R[1][2], R[2][1], R[4][5], R[5][4] = s, -s, s, -
119     s
120     R[8][9], R[9][8], R[11][12], R[12][11] = s, -s,
121     s, -s
122
123     return R
124
125
126 def traslationmatrix(dx, dy, dofs):
127     T = np.diag(np.ones(dofs * 2))
128
129     T[0][0], T[3][3], T[6][6], T[7][7], T[10][10], T
130     [13][13] = 1, 1, 1, 1, 1, 1
131     T[1][1], T[2][2], T[4][4], T[5][5] = (1 + dx), (
132     1 + dy), (1 + dy), (1 + dx)
133     T[8][8], T[9][9], T[11][11], T[12][12] = (1 + dx
134     ), (1 + dy), (1 + dy), (1 + dx)
135
136     return T
137
138
139 for r in range(n):
140     for c in range(3):
141         P[r][c] = P[r][c] / 1000 # From mm to m
142
143 if flag == 1:
144     file_handle3 = open(name_geometry+'_Prop.txt', '

```

```
137 a') # Write geometric properties into a txt file
138     for i in range(ns): # For sections
139         print('Section:', i + 1)
140         points = []
141         facets = []
142         hole = []
143         # Acquire points
144         for h in range(nps):
145             xp[h] = P[h + i * nps][0]
146             yp[h] = P[h + i * nps][1]
147             zp[h] = P[h + i * nps][2]
148             points = points + [[xp[h], yp[h]]]
149             facets = facets + [[h, h + 1]]
150             control_point = [[(max(xp) + min(xp)) * 0.5
, (max(yp) + np.mean(yp)) * 0.5]]
151             # Create geometry and mesh
152             facets.pop(-1) # to keep if the first and
last point coincide
153             facets[-1][1] = 0
154             points.pop(-1)
155             perimeter = list(range(0, len(facets)))
156
157             geometry = sections.CustomSection(points,
facets, hole, control_point, perimeter=perimeter)
158             # geometry.plot_geometry()
159             mesh = geometry.create_mesh(mesh_sizes=[2.5
])
160
161             # create a CrossSection analysis object
162
163             section = CrossSection(geometry, mesh)
164             # section.display_mesh_info()
165             # section.plot_mesh() # plot the mesh
166
167             # perform a geometric and warping analysis
168             section.calculate_geometric_properties()
169             section.calculate_warping_properties()
170
171             # section.plot_centroids() # plot the
centroids
172             # section.display_results() # print the
results to the terminal
173
174             A[i] = section.get_area()
```



```

204 Axis and X-Global References
205     gamma = np.zeros(int(ns))
206     r_s = np.zeros(int(ns)) # Vector od radial
    distances
207     Kg = np.zeros((dofs * ns, dofs * ns)) # Global
    Stiffness Matrix
208     Mg = np.zeros((dofs * ns, dofs * ns)) # Global
    Mass Matrix
209     Mgr = np.zeros((ns, ns)) # Reduced Global Mass
    Matrix
210     F_c = np.zeros(int(ns)) # Centrifugal Force
211     sigma_c = np.zeros(int(ns)) # Centrifugal
    Stress
212     F_e = np.zeros(int(ns)) # Axial Element Force
213     r_s[0] = P[0][2]
214     if flag == 0:
215         for i in range(ns): # Create Section
216             zL = zp[0]
217             for h in range(nps):
218                 xp[h] = P[h + i * nps][0]
219                 yp[h] = P[h + i * nps][1]
220                 zp[h] = P[h + i * nps][2]
221                 # ax.scatter(xp, yp, zp, c='r', marker
    ='.') # 3D plot
222                 # plt.plot(xp, yp, label=i) # 2D plot
223                 if i >= 1:
224                     L[i - 1] = zp[0] - zL # Length of
    elements
225                     r_s[i] = L[i - 1] + r_s[i - 1]
226
227                     A[i] = Prop[0][0]
228                     xct[i] = abs(Prop[0][5] - Prop[0][7])
    # Distance centroid-shear center
229                     yct[i] = abs(Prop[0][6] - Prop[0][8])
230                     IyR[i] = Prop[0][1]
231                     IxR[i] = Prop[0][2]
232                     Ip[i] = IxR[i] + IyR[i] + A[i] * (xct[i
    ] ** 2 + yct[i] ** 2)
233                     It[i] = Prop[0][3]
234                     Cm[i] = Prop[0][4]
235                     k_x[i] = Prop[0][9] / A[i]
236                     k_y[i] = Prop[0][10] / A[i]
237             else:
238                 for i in range(ns): # Create Section

```

```

239         zL = zp[0]
240         for h in range(nps):
241             xp[h] = P[h + i * nps][0]
242             yp[h] = P[h + i * nps][1]
243             zp[h] = P[h + i * nps][2]
244             # ax.scatter(xp, yp, zp, c='r', marker
= '.') # 3D plot
245             # plt.plot(xp, yp, label=i) # 2D plot
246             if i >= 1:
247                 L[i - 1] = zp[0] - zL # Length of
elements
248                 r_s[i] = L[i - 1] + r_s[i - 1]
249
250         for e in range(ns - 1):
251             phi_x_1 = shearparameter(E, IyR[e], G, k_x[e
], A[e], L[e]) # First node
252             phi_y_1 = shearparameter(E, IxR[e], G, k_y[e
], A[e], L[e])
253             phi_x_2 = shearparameter(E, IyR[e + 1], G,
k_x[e + 1], A[e + 1], L[e]) # Second node
254             phi_y_2 = shearparameter(E, IxR[e + 1], G,
k_y[e + 1], A[e + 1], L[e])
255
256             # Mass Matrix Coefficients for Global Mass
Matrix
257
258             m11 = 1 / 3
259             m18 = 1 / 6
260             m22 = (13 / 35 + 7 / 10 * phi_x_1 + 1 / 3 *
phi_x_1 ** 2 + 6 * IyR[e] / (5 * A[e] * L[e] ** 2
)) * (
261                 1 + phi_x_1) ** -2
262             m99 = (13 / 35 + 7 / 10 * phi_x_2 + 1 / 3 *
phi_x_2 ** 2 + 6 * IyR[e + 1] / (5 * A[e + 1] * L[e
] ** 2)) * (
263                 1 + phi_x_2) ** -2
264             m29 = (9 / 70 + 3 / 10 * phi_x_1 + 1 / 6 *
phi_x_1 ** 2 - 6 * IyR[e] / (5 * A[e] * L[e] ** 2
)) * (
265                 1 + phi_x_1) ** -2
266             m92 = (9 / 70 + 3 / 10 * phi_x_2 + 1 / 6 *
phi_x_2 ** 2 - 6 * IyR[e + 1] / (5 * A[e + 1] * L[e
] ** 2)) * (
267                 1 + phi_x_2) ** -2

```

```

268         m33 = (13 / 35 + 7 / 10 * phi_y_1 + 1 / 3 *
phi_y_1 ** 2 + 6 * IxR[e] / (5 * A[e] * L[e] ** 2
)) * (
269             1 + phi_y_1) ** -2
270         m1010 = (13 / 35 + 7 / 10 * phi_y_2 + 1 / 3
* phi_y_2 ** 2 + 6 * IxR[e + 1] / (5 * A[e + 1] * L
[e] ** 2)) * (
271             1 + phi_y_2) ** -2
272         m310 = (9 / 70 + 3 / 10 * phi_y_1 + 1 / 6 *
phi_y_1 ** 2 - 6 * IxR[e] / (5 * A[e] * L[e] ** 2
)) * (
273             1 + phi_y_1) ** -2
274         m103 = (9 / 70 + 3 / 10 * phi_y_2 + 1 / 6 *
phi_y_2 ** 2 - 6 * IxR[e + 1] / (5 * A[e + 1] * L[e
] ** 2)) * (
275             1 + phi_y_2) ** -2
276         m26 = L[e] * (11 / 210 + 11 / 210 * phi_x_1
+ 1 / 24 * phi_x_1 ** 2 + (1 / 10 - 1 / 2 * phi_x_1
) * IyR[e] / (
277             A[e] * L[e] ** 2)) * (1 +
phi_x_1) ** -2
278         m913 = L[e] * (
279             11 / 210 + 11 / 210 * phi_x_2 +
1 / 24 * phi_x_2 ** 2 + (1 / 10 - 1 / 2 * phi_x_2
) * IyR[e + 1] / (
280             A[e + 1] * L[e] ** 2)) * (1
+ phi_x_2) ** -2
281         m35 = -L[e] * (11 / 210 + 11 / 210 * phi_y_1
+ 1 / 24 * phi_y_1 ** 2 + (1 / 10 - 1 / 2 * phi_y_1
) * IxR[e] / (
282             A[e] * L[e] ** 2)) * (1 +
phi_y_1) ** -2
283         m1012 = -L[e] * (11 / 210 + 11 / 210 *
phi_y_2 + 1 / 24 * phi_y_2 ** 2 + (1 / 10 - 1 / 2 *
phi_y_2) * IxR[e] / (
284             A[e + 1] * L[e] ** 2)) * (1 +
phi_y_2) ** -2
285         m213 = -L[e] * (13 / 420 + 3 / 40 * phi_x_1
+ 1 / 24 * phi_x_1 ** 2 - (1 / 10 - 1 / 2 * phi_x_1
) * IyR[e] / (
286             A[e] * L[e] ** 2)) * (1 +
phi_x_1) ** -2
287         m132 = -L[e] * (
288             13 / 420 + 3 / 40 * phi_x_2 + 1

```

```

288 / 24 * phi_x_2 ** 2 - (1 / 10 - 1 / 2 * phi_x_2) *
    IyR[e + 1] / (
289         A[e + 1] * L[e] ** 2)) * (1
    + phi_x_1) ** -2
290     m312 = L[e] * (13 / 420 + 3 / 40 * phi_y_1
    + 1 / 24 * phi_y_1 ** 2 - (1 / 10 - 1 / 2 * phi_y_1
    ) * IxR[e] / (
291         A[e] * L[e] ** 2)) * (1 +
    phi_y_1) ** -2
292     m123 = L[e] * (13 / 420 + 3 / 40 * phi_y_2
    + 1 / 24 * phi_y_2 ** 2 - (1 / 10 - 1 / 2 * phi_y_2
    ) * IxR[e + 1] / (
293         A[e + 1] * L[e] ** 2)) * (1 +
    phi_y_2) ** -2
294     m44 = 13 / 35 * Ip[e] / A[e] + 6 / (5 * L[e]
    ] ** 2) * Cm[e] / A[e]
295     m1111 = 13 / 35 * Ip[e + 1] / A[e + 1] + 6
    / (5 * L[e] ** 2) * Cm[e + 1] / A[e + 1]
296     m411 = 9 / 70 * Ip[e] / A[e] - 6 / (5 * L[e]
    ] ** 2) * Cm[e] / A[e]
297     m114 = 9 / 70 * Ip[e + 1] / A[e + 1] - 6 / (
    5 * L[e] ** 2) * Cm[e + 1] / A[e + 1]
298     m55 = L[e] ** 2 * (1 / 105 + 1 / 60 *
    phi_y_1 + 1 / 120 * phi_y_1 ** 2 + (
299         2 / 15 + 1 / 6 * phi_y_1 + 1 / 3
    * phi_y_1 ** 2) * IxR[e] / (A[e] * L[e] ** 2)) * (1
    + phi_y_1) ** -2
300     m1212 = L[e] ** 2 * (1 / 105 + 1 / 60 *
    phi_y_2 + 1 / 120 * phi_y_2 ** 2 + (
301         2 / 15 + 1 / 6 * phi_y_2 + 1 / 3
    * phi_y_2 ** 2) * IxR[e + 1] / (A[e + 1] * L[e] **
    2)) * (1 + phi_y_2) ** -2
302     m66 = L[e] ** 2 * (1 / 105 + 1 / 60 *
    phi_x_1 + 1 / 120 * phi_x_1 ** 2 + (
303         2 / 15 + 1 / 6 * phi_x_1 + 1 / 3
    * phi_x_1 ** 2) * IyR[e] / (A[e] * L[e] ** 2)) * (1
    + phi_x_1) ** -2
304     m1313 = L[e] ** 2 * (1 / 105 + 1 / 60 *
    phi_x_2 + 1 / 120 * phi_x_2 ** 2 + (
305         2 / 15 + 1 / 6 * phi_x_2 + 1 / 3
    * phi_x_2 ** 2) * IyR[e + 1] / (A[e + 1] * L[e] **
    2)) * (1 + phi_x_2) ** -2
306     m512 = -L[e] ** 2 * (1 / 140 + 1 / 60 *
    phi_y_1 + 1 / 120 * phi_y_1 ** 2 + (

```

```

307         1 / 30 + 1 / 6 * phi_y_1 - 1 / 6
      * phi_y_1 ** 2) * IxR[e] / (A[e] * L[e] ** 2)) * (1
      + phi_y_1) ** -2
308         m125 = -L[e] ** 2 * (1 / 140 + 1 / 60 *
phi_y_2 + 1 / 120 * phi_y_2 ** 2 + (
309         1 / 30 + 1 / 6 * phi_y_2 - 1 / 6
      * phi_y_2 ** 2) * IxR[e + 1] / (A[e + 1] * L[e] **
2)) * (1 + phi_y_2) ** -2
310         m613 = -L[e] ** 2 * (1 / 140 + 1 / 60 *
phi_x_1 + 1 / 120 * phi_x_1 ** 2 + (
311         1 / 30 + 1 / 6 * phi_x_1 - 1 / 6
      * phi_x_1 ** 2) * IyR[e] / (A[e] * L[e] ** 2)) * (1
      + phi_x_1) ** -2
312         m136 = -L[e] ** 2 * (1 / 140 + 1 / 60 *
phi_x_2 + 1 / 120 * phi_x_2 ** 2 + (
313         1 / 30 + 1 / 6 * phi_x_2 - 1 / 6
      * phi_x_2 ** 2) * IyR[e + 1] / (A[e + 1] * L[e] **
2)) * (1 + phi_x_2) ** -2
314         m24 = yct[e] * (13 / 35 + 7 / 10 * phi_x_1
      + 1 / 3 * phi_x_1 ** 2) * (1 + phi_x_1) ** -2
315         m34 = xct[e] * (13 / 35 + 7 / 10 * phi_y_1
      + 1 / 3 * phi_y_1 ** 2) * (1 + phi_y_1) ** -2
316         m45 = -L[e] * xct[e] * (11 / 210 + 11 / 210
      * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
      ) ** -2
317         m46 = L[e] * yct[e] * (11 / 210 + 11 / 210
      * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
      ) ** -2
318         m211 = yct[e] * (9 / 70 + 3 / 10 * phi_x_1
      + 1 / 6 * phi_x_1 ** 2) * (1 + phi_x_1) ** -2
319         m112 = yct[e + 1] * (9 / 70 + 3 / 10 *
phi_x_2 + 1 / 6 * phi_x_2 ** 2) * (1 + phi_x_2) ** -
2
320         m311 = -xct[e] * (9 / 70 + 3 / 10 * phi_y_1
      + 1 / 6 * phi_y_1 ** 2) * (1 + phi_y_1) ** -2
321         m113 = -xct[e + 1] * (9 / 70 + 3 / 10 *
phi_y_2 + 1 / 6 * phi_y_2 ** 2) * (1 + phi_y_2) ** -
2
322         m49 = yct[e] * (9 / 70 + 3 / 10 * phi_x_1 +
1 / 6 * phi_x_1 ** 2) * (1 + phi_x_1) ** -2
323         m94 = yct[e + 1] * (9 / 70 + 3 / 10 *
phi_x_2 + 1 / 6 * phi_x_2 ** 2) * (1 + phi_x_2) ** -
2
324         m410 = -xct[e] * (9 / 70 + 3 / 10 * phi_y_1

```

```

324 + 1 / 6 * phi_y_1 ** 2) * (1 + phi_y_1) ** -2
325     m104 = -xct[e + 1] * (9 / 70 + 3 / 10 *
    phi_y_2 + 1 / 6 * phi_y_2 ** 2) * (1 + phi_y_2) ** -
    2
326     m412 = L[e] * xct[e] * (13 / 420 + 3 / 40 *
    phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
327     m124 = L[e] * xct[e + 1] * (13 / 420 + 3 /
    40 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
    ) ** -2
328     m413 = -L[e] * yct[e] * (13 / 420 + 3 / 40
    * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
329     m134 = -L[e] * yct[e + 1] * (13 / 420 + 3 /
    40 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
330     m511 = -L[e] * xct[e] * (13 / 420 + 3 / 40
    * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
331     m115 = -L[e] * xct[e + 1] * (13 / 420 + 3 /
    40 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
    ) ** -2
332     m611 = L[e] * yct[e] * (13 / 420 + 3 / 40 *
    phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
333     m116 = L[e] * yct[e + 1] * (13 / 420 + 3 /
    40 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
334     m911 = yct[e + 1] * (13 / 35 + 7 / 10 *
    phi_x_2 + 1 / 3 * phi_x_2 ** 2) * (1 + phi_x_2) ** -
    2
335     m1011 = -xct[e + 1] * (13 / 35 + 7 / 10 *
    phi_y_2 + 1 / 3 * phi_y_2 ** 2) * (1 + phi_y_2) ** -
    2
336     m1112 = L[e] * xct[e + 1] * (11 / 210 + 11
    / 210 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 +
    phi_y_2) ** -2
337     m1113 = -L[e] * yct[e + 1] * (11 / 210 + 11
    / 210 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 +
    phi_x_2) ** -2
338     m27 = L[e] * yct[e] * (11 / 210 + 11 / 210
    * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
339     m37 = L[e] * xct[e] * (11 / 210 + 11 / 210

```

```

339 * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
340     m47 = 11 / 210 * L[e] * Ip[e] / A[e] + 1 / (
10 * L[e]) * Cm[e] / A[e]
341     m57 = L[e] ** 2 * xct[e] * (1 / 105 + 1 / 60
    * phi_y_1 + 1 / 120 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
342     m67 = L[e] ** 2 * yct[e] * (1 / 105 + 1 / 60
    * phi_x_1 + 1 / 120 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
343     m77 = 1 / 105 * L[e] ** 2 * Ip[e] / A[e] + 2
    / 15 * Cm[e] / A[e]
344     m1414 = 1 / 105 * L[e] ** 2 * Ip[e + 1] / A[
e + 1] + 2 / 15 * Cm[e + 1] / A[e + 1]
345     m214 = -L[e] * yct[e] * (13 / 420 + 3 / 40
    * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
346     m142 = -L[e] * yct[e + 1] * (13 / 420 + 3 /
40 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
347     m314 = -L[e] * xct[e] * (13 / 420 + 3 / 40
    * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
348     m143 = -L[e] * xct[e + 1] * (13 / 420 + 3 /
40 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
    ) ** -2
349     m414 = -13 / 420 * L[e] * Ip[e] / A[e] + 1
    / (10 * L[e]) * Cm[e] / A[e]
350     m144 = -13 / 420 * L[e] * Ip[e + 1] / A[e +
1] + 1 / (10 * L[e]) * Cm[e + 1] / A[e + 1]
351     m514 = -L[e] ** 2 * xct[e] * (1 / 140 + 1 /
60 * phi_y_1 + 1 / 120 * phi_y_1 ** 2) * (1 +
phi_y_1) ** -2
352     m145 = -L[e] ** 2 * xct[e + 1] * (1 / 140 +
1 / 60 * phi_y_2 + 1 / 120 * phi_y_2 ** 2) * (1 +
phi_y_2) ** -2
353     m614 = -L[e] ** 2 * yct[e] * (1 / 140 + 1 /
60 * phi_x_1 + 1 / 120 * phi_x_1 ** 2) * (1 +
phi_x_1) ** -2
354     m146 = -L[e] ** 2 * yct[e + 1] * (1 / 140 +
1 / 60 * phi_x_2 + 1 / 120 * phi_x_2 ** 2) * (1 +
phi_x_2) ** -2
355     m79 = L[e] * yct[e] * (13 / 420 + 3 / 40 *
phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1

```

```

355 ) ** -2
356     m97 = L[e] * yct[e + 1] * (13 / 420 + 3 / 40
      * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
      ) ** -2
357     m710 = L[e] * xct[e] * (13 / 420 + 3 / 40 *
      phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
      ) ** -2
358     m107 = L[e] * xct[e + 1] * (13 / 420 + 3 /
      40 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
      ) ** -2
359     m711 = 13 / 420 * L[e] * Ip[e] / A[e] - 1
      / (10 * L[e]) * Cm[e] / A[e]
360     m117 = 13 / 420 * L[e] * Ip[e + 1] / A[e + 1
      ] - 1 / (10 * L[e]) * Cm[e + 1] / A[e + 1]
361     m712 = -L[e] ** 2 * xct[e] * (1 / 140 + 1 /
      60 * phi_y_1 + 1 / 120 * phi_y_1 ** 2) * (1 +
      phi_y_1) ** -2
362     m127 = -L[e] ** 2 * xct[e + 1] * (1 / 140 +
      1 / 60 * phi_y_2 + 1 / 120 * phi_y_2 ** 2) * (1 +
      phi_y_2) ** -2
363     m713 = -L[e] ** 2 * yct[e] * (1 / 140 + 1 /
      60 * phi_x_1 + 1 / 120 * phi_x_1 ** 2) * (1 +
      phi_x_1) ** -2
364     m137 = -L[e] ** 2 * yct[e + 1] * (1 / 140 +
      1 / 60 * phi_x_2 + 1 / 120 * phi_x_2 ** 2) * (1 +
      phi_x_2) ** -2
365     m714 = -1 / 140 * L[e] ** 2 * Ip[e] / A[e
      ] - 1 / 30 * Cm[e] / A[e]
366     m147 = -1 / 140 * L[e] ** 2 * Ip[e + 1] / A[
      e + 1] - 1 / 30 * Cm[e + 1] / A[e + 1]
367     m914 = -L[e] * yct[e + 1] * (11 / 210 + 11
      / 210 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 +
      phi_x_2) ** -2
368     m1014 = -L[e] * xct[e + 1] * (11 / 210 + 11
      / 210 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 +
      phi_y_2) ** -2
369     m1114 = -11 / 210 * L[e] * Ip[e + 1] / A[e
      + 1] - 1 / (10 * L[e]) * Cm[e + 1] / A[e + 1]
370     m1214 = L[e] ** 2 * xct[e + 1] * (1 / 105 +
      1 / 60 * phi_y_2 + 1 / 120 * phi_y_2 ** 2) * (1 +
      phi_y_2) ** -2
371     m1314 = L[e] ** 2 * yct[e + 1] * (1 / 105 +
      1 / 60 * phi_x_2 + 1 / 120 * phi_x_2 ** 2) * (1 +
      phi_x_2) ** -2

```

```

372
373     M1 = rho * A[e] * L[e]
374     M2 = rho * A[e + 1] * L[e]
375
376     Mn[0][0], Mn[7][7], Mn[0][7], Mn[7][0] = M1
    * m11, M2 * m11, M1 * m18, M2 * m18
377     Mn[1][1], Mn[8][8], Mn[1][8], Mn[8][1] = M1
    * m22, M2 * m99, M1 * m29, M2 * m92
378     Mn[2][2], Mn[9][9], Mn[2][9], Mn[9][2] = M1
    * m33, M2 * m1010, M1 * m310, M2 * m103
379     Mn[1][5], Mn[1][12], Mn[5][8], Mn[8][12] =
M1 * m26, M1 * m213, -M1 * m213, -M2 * m913
380     Mn[5][1], Mn[12][1], Mn[8][5], Mn[12][8] =
M1 * m26, M2 * m132, -M2 * m132, -M2 * m913
381     Mn[2][4], Mn[2][11], Mn[4][9], Mn[9][11] =
M1 * m35, M1 * m312, -M1 * m312, -M2 * m1012
382     Mn[4][2], Mn[11][2], Mn[9][4], Mn[11][9] =
M1 * m35, M2 * m123, -M2 * m123, -M2 * m1012
383     Mn[3][3], Mn[10][10], Mn[3][10], Mn[10][3
] = M1 * m44, M2 * m1111, M1 * m411, M2 * m114
384     Mn[4][4], Mn[11][11] = M1 * m55, M2 * m1212
385     Mn[5][5], Mn[12][12] = M1 * m66, M2 * m1313
386     Mn[4][11], Mn[11][4], Mn[5][12], Mn[12][5
] = M1 * m512, M2 * m125, M1 * m613, M2 * m136
387
388     Mn[1][3], Mn[2][3], Mn[1][6], Mn[2][6] = M1
    * m24, M1 * m34, M1 * m27, M1 * m37
389     Mn[3][1], Mn[3][2], Mn[6][1], Mn[6][3] = M1
    * m24, M1 * m34, M1 * m27, M1 * m37
390     Mn[3][6], Mn[6][3], Mn[6][6], Mn[13][13] =
M1 * m47, M1 * m47, M1 * m77, M2 * m1414
391     Mn[3][4], Mn[3][5], Mn[4][6], Mn[5][6] = M1
    * m45, M1 * m46, M1 * m57, M1 * m67
392     Mn[4][3], Mn[5][3], Mn[6][4], Mn[6][5] = M1
    * m45, M1 * m46, M1 * m57, M1 * m67
393     Mn[1][10], Mn[1][13], Mn[2][10], Mn[2][13
] = M1 * m211, M1 * m214, M1 * m311, M1 * m314
394     Mn[10][1], Mn[13][1], Mn[10][2], Mn[13][2
] = M2 * m112, M2 * m142, M2 * m113, M2 * m143
395     Mn[3][8], Mn[3][9], Mn[3][11], Mn[3][12] =
M1 * m49, M1 * m410, M1 * m412, M1 * m413
396     Mn[8][3], Mn[9][3], Mn[11][3], Mn[12][3] =
M2 * m94, M2 * m104, M2 * m124, M2 * m134
397     Mn[3][13], Mn[13][3] = M1 * m414, M2 * m144

```

```

398     Mn[4][10], Mn[4][13], Mn[5][10], Mn[5][13
] = M1 * m511, M1 * m514, M1 * m611, M1 * m614
399     Mn[10][4], Mn[13][4], Mn[10][5], Mn[13][5
] = M2 * m115, M2 * m145, M2 * m116, M2 * m146
400     Mn[6][8], Mn[6][9], Mn[6][11], Mn[6][12] =
M1 * m79, M1 * m710, M1 * m712, M1 * m713
401     Mn[8][6], Mn[9][6], Mn[11][6], Mn[12][6] =
M2 * m97, M2 * m107, M2 * m127, M2 * m137
402     Mn[6][10], Mn[10][6], Mn[6][13], Mn[13][6
] = M1 * m711, M2 * m117, M1 * m714, M2 * m147
403     Mn[8][10], Mn[8][13], Mn[10][11], Mn[10][13
] = M2 * m911, M2 * m914, M2 * m1011, M2 * m1014
404     Mn[10][8], Mn[13][8], Mn[11][10], Mn[13][10
] = M2 * m911, M2 * m914, M2 * m1011, M2 * m1014
405     Mn[10][11], Mn[10][12], Mn[11][13], Mn[12][
13] = M2 * m1112, M2 * m1113, M2 * m1214, M2 * m1314
406     Mn[11][10], Mn[12][10], Mn[13][11], Mn[13][
12] = M2 * m1112, M2 * m1113, M2 * m1214, M2 * m1314
407     Mn[10][13], Mn[13][10], Mn[13][13], Mn[13][
13] = M2 * m1114, M2 * m1114, M2 * m77, M2 * m77
408     Mn = Mn.T # For element convention
409
410     for r in range(len(Kn)):
411         for c in range(len(Kn)):
412             Mg[r + e * dofs][c + e * dofs] = Mg[
r + e * dofs][c + e * dofs] + Mn[r][c] # Global
Mass Matrix
413
414     for i in range(ns):
415         j = i
416         Mgr[i][j] = Mg[i * dofs][j * dofs]
417
418     for i in range(ns - 1):
419         j = i
420         Mgr[i][j + 1] = Mg[i * dofs][j * dofs + dofs
]
421         Mgr[i + 1][j] = Mg[j * dofs + dofs][i * dofs
]
422
423     F_c[0] = w ** 2 * np.dot(r_s, Mgr[0][:])
424     for i in range(1, len(Mgr)):
425         F_c[i] = w ** 2 * np.dot(r_s, Mgr[i][:]) +
F_c[i - 1]
426

```

```

427     sigma_c = (F_c / A)[::-1]
428     F_e = sigma_c * A
429
430     for e in range(ns - 1):
431         phi_x_1 = shearparameter(E, IyR[e], G, k_x[e
], A[e], L[e])
432         phi_y_1 = shearparameter(E, IxR[e], G, k_y[e
], A[e], L[e])
433         phi_x_2 = shearparameter(E, IyR[e + 1], G,
k_x[e + 1], A[e + 1], L[e])
434         phi_y_2 = shearparameter(E, IxR[e + 1], G,
k_y[e + 1], A[e + 1], L[e])
435
436         # Torsional curvature
437
438         Lambda_1 = sqrt((G * It[e]) / (E * Cm[e]))
# Dimension of m^-1
439         Lambda_2 = sqrt((G * It[e + 1]) / (E * Cm[e
+ 1]))
440         e_1 = L[e] * Lambda_1
441         e_2 = L[e] * Lambda_2
442         C_1 = math.cosh(e_1)
443         C_2 = math.cosh(e_2)
444         S_1 = math.sinh(e_1)
445         S_2 = math.sinh(e_2)
446         k0_1 = (G * It[e]) / (2 * (1 - C_1) + e_1 *
S_1)
447         k0_2 = (G * It[e + 1]) / (2 * (1 - C_2) +
e_2 * S_2)
448
449         # Stiffness Matrix Coefficients
450
451         k11 = E * A[e] / L[e]
452         k88 = E * A[e + 1] / L[e]
453         k22 = 12 * E * IyR[e] / (L[e] ** 3 * (1 +
phi_x_1))
454         k99 = 12 * E * IyR[e + 1] / (L[e] ** 3 * (1
+ phi_x_2))
455         k33 = 12 * E * IxR[e] / (L[e] ** 3 * (1 +
phi_y_1))
456         k1010 = 12 * E * IxR[e + 1] / (L[e] ** 3 * (
1 + phi_y_2))
457         k26 = 6 * E * IyR[e] / (L[e] ** 2 * (1 +
phi_x_1))

```

```

458     k913 = 6 * E * IyR[e + 1] / (L[e] ** 2 * (1
      + phi_x_2))
459     k510 = 6 * E * IxR[e] / (L[e] ** 2 * (1 +
      phi_y_1))
460     k1012 = 6 * E * IxR[e + 1] / (L[e] ** 2 * (1
      + phi_y_2))
461     k44 = k0_1 * Lambda_1 * S_1
462     k1111 = k0_2 * Lambda_2 * S_2
463     k47 = k0_1 * (C_1 - 1)
464     k144 = k0_2 * (C_2 - 1)
465     k77 = k0_1 * (C_1 * L[e] - S_1 / Lambda_1)
466     k1414 = k0_2 * (C_2 * L[e] - S_2 / Lambda_2)
467     k714 = k0_1 * (S_1 / Lambda_1 - L[e])
468     k147 = k0_2 * (S_2 / Lambda_1 - L[e])
469     k55 = (4 + phi_y_1) * E * IxR[e] / (L[e] * (
      1 + phi_y_1))
470     k1212 = (4 + phi_y_2) * E * IxR[e + 1] / (L[
      e] * (1 + phi_y_2))
471     k66 = (4 + phi_x_1) * E * IyR[e] / (L[e] * (
      1 + phi_x_1))
472     k1313 = (4 + phi_x_2) * E * IyR[e + 1] / (L[
      e] * (1 + phi_x_2))
473     k512 = (2 - phi_y_1) * E * IxR[e] / (L[e
      ] * (1 + phi_y_1))
474     k125 = (2 - phi_y_2) * E * IxR[e + 1] / (L[e
      ] * (1 + phi_y_2))
475     k613 = (2 - phi_x_1) * E * IyR[e] / (L[e
      ] * (1 + phi_x_1))
476     k136 = (2 - phi_x_2) * E * IyR[e + 1] / (L[e
      ] * (1 + phi_x_2))
477
478     Kn[0][0], Kn[7][7], Kn[0][7], Kn[7][0] = k11
      , k88, -k11, -k88
479     Kn[1][1], Kn[8][8], Kn[1][8], Kn[8][1] = k22
      , k99, -k22, -k99
480     Kn[2][2], Kn[9][9], Kn[2][9], Kn[9][2] = k33
      , k1010, -k33, -k1010
481     Kn[1][5], Kn[1][12], Kn[5][8], Kn[8][12], Kn
      [5][1], \
482     Kn[12][1], Kn[8][5], Kn[12][8] = k26, k26, -
      k26, -k913, k26, k913, -k913, -k913
483     Kn[2][4], Kn[2][11], Kn[4][9], Kn[9][11], Kn
      [4][2], \
484     Kn[11][2], Kn[9][4], Kn[11][9] = -k510, -

```

```

484 k510, k510, k1012, -k510, -k1012, k1012, k1012
485     Kn[3][3], Kn[10][10], Kn[3][10], Kn[10][3
    ] = k44, k1111, -k44, -k1111
486     Kn[4][4], Kn[11][11] = k55, k1212
487     Kn[5][5], Kn[12][12] = k66, k1313
488     Kn[4][11], Kn[11][4], Kn[5][12], Kn[12][5
    ] = k512, k125, k613, k136
489     Kn[3][6], Kn[6][3], Kn[6][6], Kn[13][13] =
    k47, k47, k77, k1414
490     Kn[3][13], Kn[13][3], Kn[6][10], Kn[10][6
    ] = k47, k144, -k47, -k144
491     Kn[6][13], Kn[13][6], Kn[10][13], Kn[13][10
    ] = k714, k147, -k144, -k144
492     Kn = Kn.T
493
494     # Geometric Stiffness Matrix Coefficients
495
496     g22 = (6 / 5 + 2 * phi_x_1 + phi_x_1 ** 2
    ) * (1 + phi_x_1) ** -2
497     g99 = (6 / 5 + 2 * phi_x_2 + phi_x_2 ** 2
    ) * (1 + phi_x_2) ** -2
498     g33 = (6 / 5 + 2 * phi_y_1 + phi_y_1 ** 2
    ) * (1 + phi_y_1) ** -2
499     g1010 = (6 / 5 + 2 * phi_y_2 + phi_y_2 ** 2
    ) * (1 + phi_y_2) ** -2
500     g44 = Ip[e] / A[e]
501     g1111 = Ip[e + 1] / A[e + 1]
502     g55 = L[e] ** 2 * (2 / 15 + 1 / 6 * phi_y_1
    + 1 / 12 * phi_y_1 ** 2) * (1 + phi_y_1) ** -2
503     g1212 = L[e] ** 2 * (2 / 15 + 1 / 6 *
    phi_y_2 + 1 / 12 * phi_y_2 ** 2) * (1 + phi_y_2
    ) ** -2
504     g66 = L[e] ** 2 * (2 / 15 + 1 / 6 * phi_x_1
    + 1 / 12 * phi_x_1 ** 2) * (1 + phi_x_1) ** -2
505     g1313 = L[e] ** 2 * (2 / 15 + 1 / 6 *
    phi_x_2 + 1 / 12 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
506     g510 = L[e] / 10 * (1 + phi_y_1) ** -2
507     g1012 = L[e] / 10 * (1 + phi_y_2) ** -2
508     g26 = L[e] / 10 * (1 + phi_x_1) ** -2
509     g913 = L[e] / 10 * (1 + phi_x_2) ** -2
510     g512 = -L[e] ** 2 * (1 / 30 + 1 / 6 *
    phi_y_1 + 1 / 12 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2

```

```

511         g125 = -L[e] ** 2 * (1 / 30 + 1 / 6 *
    phi_y_2 + 1 / 12 * phi_y_2 ** 2) * (1 + phi_y_2
    ) ** -2
512         g613 = -L[e] ** 2 * (1 / 30 + 1 / 6 *
    phi_x_1 + 1 / 12 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
513         g136 = -L[e] ** 2 * (1 / 30 + 1 / 6 *
    phi_x_2 + 1 / 12 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
514
515         Gn[1][1], Gn[8][8], Gn[1][8], Gn[8][1] = g22
    , g99, -g22, -g99
516         Gn[2][2], Gn[9][9], Gn[2][9], Gn[9][2] = g33
    , g1010, -g33, -g1010
517         Gn[1][5], Gn[1][12], Gn[5][8], Gn[8][12] =
    g26, g26, -g26, -g913
518         Gn[5][1], Gn[12][1], Gn[8][5], Gn[12][8] =
    g26, g913, -g913, -g913
519         Gn[2][4], Gn[2][11], Gn[4][9], Gn[9][11] = -
    g510, -g510, g510, g1012
520         Gn[4][2], Gn[11][2], Gn[9][4], Gn[11][9] = -
    g510, -g1012, g1012, g1012
521         Gn[3][3], Gn[10][10], Gn[3][10], Gn[10][3
    ] = g44, g1111, -g44, -g1111
522         Gn[4][4], Gn[11][11] = g55, g1212
523         Gn[5][5], Gn[12][12] = g66, g1313
524         Gn[4][11], Gn[11][4], Gn[5][12], Gn[12][5
    ] = g512, g125, g613, g136
525         Gn = Gn.T
526
527         Gn_e = Gn * F_e[e] / L[e]
528
529         for r in range(len(Kn)):
530             for c in range(len(Kn)):
531                 Kg[r + e * dofs][c + e * dofs] = Kg[
    r + e * dofs][c + e * dofs] + Kn[r][c] + Gn_e[r][c]
532
533         Kg = Kg - Mg * w ** 2 # Mass spin softening
534
535         # Eigenvalues
536
537         if BC == 1:
538             (Q, V) = eig(Kg[dofs:, dofs:], Mg[dofs:,
    dofs:]) # Constrained Beam at lowest section

```

```

539     elif BC == 0:
540         (Q, V) = eig(Kg, Mg) # Free Beam
541
542         omega = sqrt(Q)
543         oma = array(omega)
544         fn = oma / (2 * math.pi)
545
546         fn_real = fn.real
547
548         order = fn_real.ravel().argsort() # preserve
order for both eigenvalues & vectors
549
550         mf = len(Q)
551         NN = zeros(mf, 'f')
552         FN = zeros(mf, 'f')
553
554         for i in range(0, mf):
555             NN[i] = float(i + 1)
556             FN[i] = fn_real[order[i]]
557
558         mfs = mf
559         if mfs > 100:
560             mfs = 100
561
562         print(" ")
563         print("Natural Frequencies ")
564         print(" ")
565         print("%8.7s" % 'fn [Hz]')
566
567         for i in range(0, 20):
568             print("%8.5g" % (FN[i]))
569
570         # Mass Normalize Eigenvectors
571
572         if BC == 1:
573             QQQ = dot(V.T, dot(Mg[dofs:, dofs:], V)) #
Constrained Beam at lowest section
574         elif BC == 0:
575             QQQ = dot(V.T, dot(Mg, V)) # Free Beam
576
577         for i in range(0, mf):
578             nf = sqrt(QQQ[i, i])
579             for j in range(0, mf):
580                 V[j, i] /= nf

```

```

581
582     # Sort Eigenvectors
583
584     MS = zeros((mf, mf), 'f')
585
586     for i in range(0, mf):
587         MS[0:mf, i] = V[0:mf, order[i]]
588     print('Mode shapes? 0 = No, 1 = Yes')
589     answer = int(input())
590     while answer == 1:
591         MSV = np.zeros(ns)
592         MSVx = np.zeros(ns)
593         MSVy = np.zeros(ns)
594         Lb = np.zeros(ns)
595         m = 0
596         print('Modal shape of what frequencies? ', '
\n')
597         ind_wn = int(input()) - 1
598         mod = 'flessionale'
599         print('FN:   ', FN[ind_wn])
600         uz = 0
601         ux = 1
602         uy = 2
603         for i in range(ns - 1):
604             Lb[i + 1] = r_s[i + 1] - r_s[0]
605             if m <= max(abs(MS[i * dofs + uz][ind_wn
]), abs(MS[i * dofs + ux][ind_wn]), abs(MS[i * dofs
+ uy][ind_wn]))):
606                 m = max(abs(MS[i * dofs + uz][ind_wn
]), abs(MS[i * dofs + ux][ind_wn]), abs(MS[i * dofs
+ uy][ind_wn]))
607                 MSV[i + 1] = MS[i * dofs + uz][ind_wn]
608                 MSVx[i + 1] = MS[i * dofs + ux][ind_wn]
609                 MSVy[i + 1] = MS[i * dofs + uy][ind_wn]
610                 Lb = Lb / max(abs(Lb))
611                 Lbnew = np.linspace(Lb[0], Lb[-1], 100)
612                 spl = interpolate.splrep(Lb, MSV, s=0)
613                 spline = interpolate.splev(Lbnew, spl, der=0
)
614                 splx = interpolate.splrep(Lb, MSVx, s=0)
615                 splinex = interpolate.splev(Lbnew, splx, der
=0)
616                 sply = interpolate.splrep(Lb, MSVy, s=0)
617                 spliney = interpolate.splev(Lbnew, sply, der

```

```

617 =0)
618
619     fig = plt.figure()
620     plt.plot(Lb, MSV, 'o', Lbnew, spline, '--')
621     plt.ylim(-m * 1.2, m * 1.2)
622     plt.xlabel('Asse z')
623     plt.ylabel('Ampiezza')
624     plt.title('Frequenza naturale: %d°' % (
ind_wn - 1) + ' %s' % mod)
625     plt.show()
626
627     fig2 = plt.figure()
628     ax = fig2.add_subplot(1, 1, 1, projection='
3d')
629     ax.scatter(Lbnew, splinex, spliney, c='r',
marker='.') # 3D plot
630     ax.set_ylim3d(-m * 1.2, m * 1.2)
631     ax.set_zlim3d(-m * 1.2, m * 1.2)
632     plt.title('Frequenza naturale: %d°' % 2 +
' %s' % mod)
633     ax.set_xlabel('Lunghezza trave
adimensionalizzata')
634     ax.set_ylabel('Asse Y')
635     ax.set_zlabel('Asse Z')
636     plt.show()
637     print('Continue? 0 = No 1 = Yes')
638     answer = int(input())
639
640 else: # Not Prismatic, more sections are included
641     L = np.zeros(int(ns) - 1)
642     r = np.zeros(int(ns))
643     r[0] = P[0][2] # Position of hub section
644     Kg = np.zeros((dofs * (ns * 2 - 1), dofs * (ns
* 2 - 1)))
645     Mg = np.zeros((dofs * (ns * 2 - 1), dofs * (ns
* 2 - 1)))
646     Mgr = np.zeros((ns * 2 - 1, ns * 2 - 1))
647     F_c = np.zeros(int(ns * 2)-1)
648     sigma_c = np.zeros(int(ns * 2)-1)
649     F_e = np.zeros(int(ns * 2)-2)
650     L_v = np.zeros(int(ns * 2 - 2)) # Length of sub
-elements
651     L_v[0] = L[0]
652     r_v = np.zeros(int(ns * 2) - 1) # Distance of

```

```

652 sub-elements
653     r_v[0] = r[0]
654     A_c = np.zeros(int(ns * 2) - 1) # Centrifugal
Area
655     if flag == 0:
656         for i in range(ns):
657             zL = zp[0]
658             for h in range(nps):
659                 xp[h] = P[h + i * nps][0]
660                 yp[h] = P[h + i * nps][1]
661                 zp[h] = P[h + i * nps][2]
662                 # ax.scatter(xp, yp, zp, c='r', marker
= '.') # 3D plot
663                 # plt.plot(xp, yp, label=i) # 2D plot
664                 xcc[i] = Prop[i][5]
665                 ycc[i] = Prop[i][6]
666                 if i >= 1:
667                     L[i - 1] = sqrt((xcc[i] - xcc[i - 1
] ) ** 2 + (ycc[i] - ycc[i - 1] ) ** 2 + (zp[0] - zL
) ** 2)
668                     r[i] = L[i - 1] + r[i - 1]
669                     L_v[(i - 1) * 2] = L[i - 1] / 2
670                     L_v[(i - 1) * 2 + 1] = L[i - 1] / 2
671                     r_v[i * 2 - 1] = L_v[i * 2 - 2] +
r_v[i * 2 - 2]
672                     r_v[i * 2] = L_v[i * 2 - 2] + r_v[i
* 2 - 1]
673                     A[i] = Prop[i][0]
674                     xct[i] = abs(Prop[i][5] - Prop[i][7])
675                     yct[i] = abs(Prop[i][6] - Prop[i][8])
676                     It[i] = Prop[i][3]
677                     Cm[i] = Prop[i][4]
678                     dx[i] = abs(Prop[i][5] - Prop[0][5])
679                     dy[i] = abs(Prop[i][6] - Prop[0][6])
680                     delta[i] = (Prop[i][11] + 90) * math.pi
/ 180 # Oxyz (engine system) as reference
681                     IxR[i] = Prop[i][2]
682                     IyR[i] = Prop[i][1]
683                     k_x[i] = Prop[i][10] / A[i]
684                     k_y[i] = Prop[i][9] / A[i]
685                     Ip[i] = IxR[i] + IyR[i] + A[i] * (xct[i
] ** 2 + yct[i] ** 2)
686                 else:
687                     for i in range(ns):

```

```

688         zL = zp[0]
689         for h in range(nps):
690             xp[h] = P[h + i * nps][0]
691             yp[h] = P[h + i * nps][1]
692             zp[h] = P[h + i * nps][2]
693             # ax.scatter(xp, yp, zp, c='r', marker
        ='.') # 3D plot
694             # plt.plot(xp, yp, label=i) # 2D plot
695             if i >= 1:
696                 L[i - 1] = sqrt((xcc[i] - xcc[i - 1]
        ]) ** 2 + (ycc[i] - ycc[i - 1]) ** 2 + (zp[0] - zL
        ) ** 2)
697                 r[i] = L[i - 1] + r[i - 1]
698                 L_v[(i - 1) * 2] = L[i - 1] / 2
699                 L_v[(i - 1) * 2 + 1] = L[i - 1] / 2
700                 r_v[i * 2 - 1] = L_v[i * 2 - 2] +
r_v[i * 2 - 2]
701                 r_v[i * 2] = L_v[i * 2 - 2] + r_v[i
        * 2 - 1]
702         e = 0
703         for el in range(int(ns-1)*2):
704             phi_x_1 = shearparameter(E, IyR[e], G, k_x[e
        ], A[e], L_v[el]) # Dimensionless, first node
705             phi_y_1 = shearparameter(E, IxR[e], G, k_y[e
        ], A[e], L_v[el])
706             phi_x_2 = shearparameter(E, IyR[e], G, k_x[e
        ], A[e], L_v[el]) # Dimensionless, second node
707             phi_y_2 = shearparameter(E, IxR[e], G, k_y[e
        ], A[e], L_v[el])
708
709             R = rotationmatrix(delta[e], R) # Rotation
        Matrix for pre-twisted beams
710             T = traslationmatrix(dx[e], dy[e], dofs) #
        Traslation Matrix for centroids
711
712             # Mass Matrix Coefficients for Global Mass
        Matrix
713
714             m11 = 1 / 3
715             m18 = 1 / 6
716             m22 = (13 / 35 + 7 / 10 * phi_x_1 + 1 / 3 *
        phi_x_1 ** 2 + 6 * IyR[e] / (5 * A[e] * L_v[el] ** 2
        )) * (
717                 1 + phi_x_1) ** -2

```

```

718         m99 = (13 / 35 + 7 / 10 * phi_x_2 + 1 / 3 *
phi_x_2 ** 2 + 6 * IyR[e] / (5 * A[e] * L_v[eL] ** 2
)) * (
719             1 + phi_x_2) ** -2
720         m29 = (9 / 70 + 3 / 10 * phi_x_1 + 1 / 6 *
phi_x_1 ** 2 - 6 * IyR[e] / (5 * A[e] * L_v[eL] ** 2
)) * (
721             1 + phi_x_1) ** -2
722         m92 = (9 / 70 + 3 / 10 * phi_x_2 + 1 / 6 *
phi_x_2 ** 2 - 6 * IyR[e] / (5 * A[e] * L_v[eL] ** 2
)) * (
723             1 + phi_x_2) ** -2
724         m33 = (13 / 35 + 7 / 10 * phi_y_1 + 1 / 3 *
phi_y_1 ** 2 + 6 * IxR[e] / (5 * A[e] * L_v[eL] ** 2
)) * (
725             1 + phi_y_1) ** -2
726         m1010 = (13 / 35 + 7 / 10 * phi_y_2 + 1 / 3
* phi_y_2 ** 2 + 6 * IxR[e] / (5 * A[e] * L_v[eL
] ** 2)) * (
727             1 + phi_y_2) ** -2
728         m310 = (9 / 70 + 3 / 10 * phi_y_1 + 1 / 6 *
phi_y_1 ** 2 - 6 * IxR[e] / (5 * A[e] * L_v[eL] ** 2
)) * (
729             1 + phi_y_1) ** -2
730         m103 = (9 / 70 + 3 / 10 * phi_y_2 + 1 / 6 *
phi_y_2 ** 2 - 6 * IxR[e] / (5 * A[e] * L_v[eL] ** 2
)) * (
731             1 + phi_y_2) ** -2
732         m26 = L_v[eL] * (11 / 210 + 11 / 210 *
phi_x_1 + 1 / 24 * phi_x_1 ** 2 + (1 / 10 - 1 / 2 *
phi_x_1) * IyR[e] / (
733             A[e] * L_v[eL] ** 2)) * (1 +
phi_x_1) ** -2
734         m913 = L_v[eL] * (
735             11 / 210 + 11 / 210 * phi_x_2 +
1 / 24 * phi_x_2 ** 2 + (1 / 10 - 1 / 2 * phi_x_2
) * IyR[e] / (
736             A[e] * L_v[eL] ** 2)) * (1
+ phi_x_2) ** -2
737         m35 = -L_v[eL] * (11 / 210 + 11 / 210 *
phi_y_1 + 1 / 24 * phi_y_1 ** 2 + (1 / 10 - 1 / 2 *
phi_y_1) * IxR[e] / (
738             A[e] * L_v[eL] ** 2)) * (1 +
phi_y_1) ** -2

```

```

739         m1012 = -L_v[eL] * (11 / 210 + 11 / 210 *
phi_y_2 + 1 / 24 * phi_y_2 ** 2 + (1 / 10 - 1 / 2 *
phi_y_2) * IxR[e] / (
740             A[e] * L_v[eL] ** 2)) * (1 +
phi_y_2) ** -2
741         m213 = -L_v[eL] * (13 / 420 + 3 / 40 *
phi_x_1 + 1 / 24 * phi_x_1 ** 2 - (1 / 10 - 1 / 2 *
phi_x_1) * IyR[e] / (
742             A[e] * L_v[eL] ** 2)) * (1 +
phi_x_1) ** -2
743         m132 = -L_v[eL] * (
744             13 / 420 + 3 / 40 * phi_x_2 + 1
/ 24 * phi_x_2 ** 2 - (1 / 10 - 1 / 2 * phi_x_2) *
IyR[e] / (
745             A[e] * L_v[eL] ** 2)) * (1
+ phi_x_1) ** -2
746         m312 = L_v[eL] * (13 / 420 + 3 / 40 *
phi_y_1 + 1 / 24 * phi_y_1 ** 2 - (1 / 10 - 1 / 2 *
phi_y_1) * IxR[e] / (
747             A[e] * L_v[eL] ** 2)) * (1 +
phi_y_1) ** -2
748         m123 = L_v[eL] * (13 / 420 + 3 / 40 *
phi_y_2 + 1 / 24 * phi_y_2 ** 2 - (1 / 10 - 1 / 2 *
phi_y_2) * IxR[e] / (
749             A[e] * L_v[eL] ** 2)) * (1 +
phi_y_2) ** -2
750         m44 = 13 / 35 * Ip[e] / A[e] + 6 / (5 * L_v[
eL] ** 2) * Cm[e] / A[e]
751         m1111 = 13 / 35 * Ip[e] / A[e] + 6 / (5 *
L_v[eL] ** 2) * Cm[e] / A[e]
752         m411 = 9 / 70 * Ip[e] / A[e] - 6 / (5 * L_v[
eL] ** 2) * Cm[e] / A[e]
753         m114 = 9 / 70 * Ip[e] / A[e] - 6 / (5 * L_v[
eL] ** 2) * Cm[e] / A[e]
754         m55 = L_v[eL] ** 2 * (1 / 105 + 1 / 60 *
phi_y_1 + 1 / 120 * phi_y_1 ** 2 + (
755             2 / 15 + 1 / 6 * phi_y_1 + 1 / 3
* phi_y_1 ** 2) * IxR[e] / (A[e] * L_v[eL] ** 2
)) * \
756             (1 + phi_y_1) ** -2
757         m1212 = L_v[eL] ** 2 * (1 / 105 + 1 / 60 *
phi_y_2 + 1 / 120 * phi_y_2 ** 2 + (
758             2 / 15 + 1 / 6 * phi_y_2 + 1 / 3
* phi_y_2 ** 2) * IxR[e] / (A[e] * L_v[eL] ** 2

```

```

758 )) * \
759         (1 + phi_y_2) ** -2
760         m66 = L_v[e] ** 2 * (1 / 105 + 1 / 60 *
phi_x_1 + 1 / 120 * phi_x_1 ** 2 + (
761         2 / 15 + 1 / 6 * phi_x_1 + 1 / 3
* phi_x_1 ** 2) * IyR[e] / (A[e] * L_v[e] ** 2
)) * \
762         (1 + phi_x_1) ** -2
763         m1313 = L_v[e] ** 2 * (1 / 105 + 1 / 60 *
phi_x_2 + 1 / 120 * phi_x_2 ** 2 + (
764         2 / 15 + 1 / 6 * phi_x_2 + 1 / 3
* phi_x_2 ** 2) * IyR[e] / (A[e] * L_v[e] ** 2
)) * \
765         (1 + phi_x_2) ** -2
766         m512 = L_v[e] ** 2 * (1 / 140 + 1 / 60 *
phi_y_1 + 1 / 120 * phi_y_1 ** 2 + (
767         1 / 30 + 1 / 6 * phi_y_1 - 1 / 6
* phi_y_1 ** 2) * IxR[e] / (A[e] * L_v[e] ** 2
)) * \
768         (1 + phi_y_1) ** -2
769         m125 = -L_v[e] ** 2 * (1 / 140 + 1 / 60 *
phi_y_2 + 1 / 120 * phi_y_2 ** 2 + (
770         1 / 30 + 1 / 6 * phi_y_2 - 1 / 6
* phi_y_2 ** 2) * IxR[e] / (A[e] * L_v[e] ** 2
)) * \
771         (1 + phi_y_2) ** -2
772         m613 = -L_v[e] ** 2 * (1 / 140 + 1 / 60 *
phi_x_1 + 1 / 120 * phi_x_1 ** 2 + (
773         1 / 30 + 1 / 6 * phi_x_1 - 1 / 6
* phi_x_1 ** 2) * IyR[e] / (A[e] * L_v[e] ** 2
)) * \
774         (1 + phi_x_1) ** -2
775         m136 = -L_v[e] ** 2 * (1 / 140 + 1 / 60 *
phi_x_2 + 1 / 120 * phi_x_2 ** 2 + (
776         1 / 30 + 1 / 6 * phi_x_2 - 1 / 6
* phi_x_2 ** 2) * IyR[e] / (A[e] * L_v[e] ** 2
)) * \
777         (1 + phi_x_2) ** -2
778         m24 = yct[e] * (13 / 35 + 7 / 10 * phi_x_1
+ 1 / 3 * phi_x_1 ** 2) * (1 + phi_x_1) ** -2
779
780         m34 = xct[e] * (13 / 35 + 7 / 10 * phi_y_1
+ 1 / 3 * phi_y_1 ** 2) * (1 + phi_y_1) ** -2
781

```

```

782         m45 = -L_v[el] * xct[e] * (11 / 210 + 11 /
210 * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 +
phi_y_1) ** -2
783         m46 = L_v[el] * yct[e] * (11 / 210 + 11 /
210 * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 +
phi_x_1) ** -2
784         m211 = yct[e] * (9 / 70 + 3 / 10 * phi_x_1
+ 1 / 6 * phi_x_1 ** 2) * (1 + phi_x_1) ** -2
785         m112 = yct[e] * (9 / 70 + 3 / 10 * phi_x_2
+ 1 / 6 * phi_x_2 ** 2) * (1 + phi_x_2) ** -2
786         m311 = -xct[e] * (9 / 70 + 3 / 10 * phi_y_1
+ 1 / 6 * phi_y_1 ** 2) * (1 + phi_y_1) ** -2
787         m113 = -xct[e] * (9 / 70 + 3 / 10 * phi_y_2
+ 1 / 6 * phi_y_2 ** 2) * (1 + phi_y_2) ** -2
788         m49 = yct[e] * (9 / 70 + 3 / 10 * phi_x_1 +
1 / 6 * phi_x_1 ** 2) * (1 + phi_x_1) ** -2
789         m94 = yct[e] * (9 / 70 + 3 / 10 * phi_x_2 +
1 / 6 * phi_x_2 ** 2) * (1 + phi_x_2) ** -2
790         m410 = -xct[e] * (9 / 70 + 3 / 10 * phi_y_1
+ 1 / 6 * phi_y_1 ** 2) * (1 + phi_y_1) ** -2
791         m104 = -xct[e] * (9 / 70 + 3 / 10 * phi_y_2
+ 1 / 6 * phi_y_2 ** 2) * (1 + phi_y_2) ** -2
792         m412 = L_v[el] * xct[e] * (13 / 420 + 3 / 40
* phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
) ** -2
793         m124 = L_v[el] * xct[e] * (13 / 420 + 3 / 40
* phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
) ** -2
794         m413 = -L_v[el] * yct[e] * (13 / 420 + 3 /
40 * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
) ** -2
795         m134 = -L_v[el] * yct[e] * (13 / 420 + 3 /
40 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
) ** -2
796         m511 = -L_v[el] * xct[e] * (13 / 420 + 3 /
40 * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
) ** -2
797         m115 = -L_v[el] * xct[e] * (13 / 420 + 3 /
40 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
) ** -2
798         m611 = L_v[el] * yct[e] * (13 / 420 + 3 / 40
* phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
) ** -2
799         m116 = L_v[el] * yct[e] * (13 / 420 + 3 / 40

```

```

799 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
800     m911 = yct[e] * (13 / 35 + 7 / 10 * phi_x_2
    + 1 / 3 * phi_x_2 ** 2) * (1 + phi_x_2) ** -2
801     m1011 = -xct[e] * (13 / 35 + 7 / 10 *
    phi_y_2 + 1 / 3 * phi_y_2 ** 2) * (1 + phi_y_2) ** -
    2
802     m1112 = L_v[e] * xct[e] * (11 / 210 + 11 /
    210 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 +
    phi_y_2) ** -2
803     m1113 = L_v[e] * yct[e] * (11 / 210 + 11 /
    210 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 +
    phi_x_2) ** -2
804     m27 = L_v[e] * yct[e] * (11 / 210 + 11 /
    210 * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 +
    phi_x_1) ** -2
805     m37 = L_v[e] * xct[e] * (11 / 210 + 11 /
    210 * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 +
    phi_y_1) ** -2
806     m47 = 11 / 210 * L_v[e] * Ip[e] / A[e] + 1
    / (10 * L_v[e]) * Cm[e] / A[e]
807     m57 = L_v[e] ** 2 * xct[e] * (1 / 105 + 1
    / 60 * phi_y_1 + 1 / 120 * phi_y_1 ** 2) * (1 +
    phi_y_1) ** -2
808     m67 = L_v[e] ** 2 * yct[e] * (1 / 105 + 1
    / 60 * phi_x_1 + 1 / 120 * phi_x_1 ** 2) * (1 +
    phi_x_1) ** -2
809     m77 = 1 / 105 * L_v[e] ** 2 * Ip[e] / A[e
    ] + 2 / 15 * Cm[e] / A[e]
810     m1414 = 1 / 105 * L_v[e] ** 2 * Ip[e] / A[e
    ] + 2 / 15 * Cm[e] / A[e]
811     m214 = -L_v[e] * yct[e] * (13 / 420 + 3 /
    40 * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
812     m142 = -L_v[e] * yct[e] * (13 / 420 + 3 /
    40 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
813     m314 = -L_v[e] * xct[e] * (13 / 420 + 3 /
    40 * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
814     m143 = -L_v[e] * xct[e] * (13 / 420 + 3 /
    40 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
    ) ** -2
815     m414 = -13 / 420 * L_v[e] * Ip[e] / A[e] +

```

```

815 1 / (10 * L_v[e]) * Cm[e] / A[e]
816     m144 = -13 / 420 * L_v[e] * Ip[e] / A[e] +
      1 / (10 * L_v[e]) * Cm[e] / A[e]
817     m514 = -L_v[e] ** 2 * xct[e] * (1 / 140 + 1
      / 60 * phi_y_1 + 1 / 120 * phi_y_1 ** 2) * (1 +
      phi_y_1) ** -2
818     m145 = -L_v[e] ** 2 * xct[e] * (1 / 140 + 1
      / 60 * phi_y_2 + 1 / 120 * phi_y_2 ** 2) * (1 +
      phi_y_2) ** -2
819     m614 = -L_v[e] ** 2 * yct[e] * (1 / 140 + 1
      / 60 * phi_x_1 + 1 / 120 * phi_x_1 ** 2) * (1 +
      phi_x_1) ** -2
820     m146 = -L_v[e] ** 2 * yct[e] * (1 / 140 + 1
      / 60 * phi_x_2 + 1 / 120 * phi_x_2 ** 2) * (1 +
      phi_x_2) ** -2
821     m79 = L_v[e] * yct[e] * (13 / 420 + 3 / 40
      * phi_x_1 + 1 / 24 * phi_x_1 ** 2) * (1 + phi_x_1
      ) ** -2
822     m97 = L_v[e] * yct[e] * (13 / 420 + 3 / 40
      * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 + phi_x_2
      ) ** -2
823     m710 = L_v[e] * xct[e] * (13 / 420 + 3 / 40
      * phi_y_1 + 1 / 24 * phi_y_1 ** 2) * (1 + phi_y_1
      ) ** -2
824     m107 = L_v[e] * xct[e] * (13 / 420 + 3 / 40
      * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 + phi_y_2
      ) ** -2
825     m711 = 13 / 420 * L_v[e] * Ip[e] / A[e] - 1
      / (10 * L_v[e]) * Cm[e] / A[e]
826     m117 = 13 / 420 * L_v[e] * Ip[e] / A[e] - 1
      / (10 * L_v[e]) * Cm[e] / A[e]
827     m712 = -L_v[e] ** 2 * xct[e] * (1 / 140 + 1
      / 60 * phi_y_1 + 1 / 120 * phi_y_1 ** 2) * (1 +
      phi_y_1) ** -2
828     m127 = -L_v[e] ** 2 * xct[e] * (1 / 140 + 1
      / 60 * phi_y_2 + 1 / 120 * phi_y_2 ** 2) * (1 +
      phi_y_2) ** -2
829     m713 = -L_v[e] ** 2 * yct[e] * (1 / 140 + 1
      / 60 * phi_x_1 + 1 / 120 * phi_x_1 ** 2) * (1 +
      phi_x_1) ** -2
830     m137 = -L_v[e] ** 2 * yct[e] * (1 / 140 + 1
      / 60 * phi_x_2 + 1 / 120 * phi_x_2 ** 2) * (1 +
      phi_x_2) ** -2
831     m714 = -1 / 140 * L_v[e] ** 2 * Ip[e] / A[e]

```

```

831 ] - 1 / 30 * Cm[e] / A[e]
832     m147 = -1 / 140 * L_v[eL] ** 2 * Ip[e] / A[e]
      ] - 1 / 30 * Cm[e] / A[e]
833     m914 = -L_v[eL] * yct[e] * (11 / 210 + 11 /
      210 * phi_x_2 + 1 / 24 * phi_x_2 ** 2) * (1 +
      phi_x_2) ** -2
834     m1014 = -L_v[eL] * xct[e] * (11 / 210 + 11
      / 210 * phi_y_2 + 1 / 24 * phi_y_2 ** 2) * (1 +
      phi_y_2) ** -2
835     m1114 = -11 / 210 * L_v[eL] * Ip[e] / A[e]
      ] - 1 / (10 * L_v[eL]) * Cm[e] / A[e]
836     m1214 = L_v[eL] ** 2 * xct[e] * (1 / 105 + 1
      / 60 * phi_y_2 + 1 / 120 * phi_y_2 ** 2) * (1 +
      phi_y_2) ** -2
837     m1314 = L_v[eL] ** 2 * yct[e] * (1 / 105 + 1
      / 60 * phi_x_2 + 1 / 120 * phi_x_2 ** 2) * (1 +
      phi_x_2) ** -2
838
839     M1 = rho * A[e] * L_v[eL]
840     M2 = rho * A[e] * L_v[eL]
841
842     Mn[0][0], Mn[7][7], Mn[0][7], Mn[7][0] = M1
      * m11, M2 * m11, M1 * m18, M2 * m18
843     Mn[1][1], Mn[8][8], Mn[1][8], Mn[8][1] = M1
      * m22, M2 * m99, M1 * m29, M2 * m92
844     Mn[2][2], Mn[9][9], Mn[2][9], Mn[9][2] = M1
      * m33, M2 * m1010, M1 * m310, M2 * m103
845     Mn[1][5], Mn[1][12], Mn[5][8], Mn[8][12] =
      M1 * m26, M1 * m213, -M1 * m213, -M2 * m913
846     Mn[5][1], Mn[12][1], Mn[8][5], Mn[12][8] =
      M1 * m26, M2 * m132, -M2 * m132, -M2 * m913
847     Mn[2][4], Mn[2][11], Mn[4][9], Mn[9][11] =
      M1 * m35, M1 * m312, -M1 * m312, -M2 * m1012
848     Mn[4][2], Mn[11][2], Mn[9][4], Mn[11][9] =
      M1 * m35, M2 * m123, -M2 * m123, -M2 * m1012
849     Mn[3][3], Mn[10][10], Mn[3][10], Mn[10][3]
      ] = M1 * m44, M2 * m1111, M1 * m411, M2 * m114
850     Mn[4][4], Mn[11][11] = M1 * m55, M2 * m1212
851     Mn[5][5], Mn[12][12] = M1 * m66, M2 * m1313
852     Mn[4][11], Mn[11][4], Mn[5][12], Mn[12][5]
      ] = M1 * m512, M2 * m125, M1 * m613, M2 * m136
853
854     Mn[1][3], Mn[2][3], Mn[1][6], Mn[2][6] = M1
      * m24, M1 * m34, M1 * m27, M1 * m37

```

```

855     Mn[3][1], Mn[3][2], Mn[6][1], Mn[6][3] = M1
      * m24, M1 * m34, M1 * m27, M1 * m37
856     Mn[3][6], Mn[6][3], Mn[6][6], Mn[13][13] =
M1 * m47, M1 * m47, M1 * m77, M2 * m1414
857     Mn[3][4], Mn[3][5], Mn[4][6], Mn[5][6] = M1
      * m45, M1 * m46, M1 * m57, M1 * m67
858     Mn[4][3], Mn[5][3], Mn[6][4], Mn[6][5] = M1
      * m45, M1 * m46, M1 * m57, M1 * m67
859     Mn[1][10], Mn[1][13], Mn[2][10], Mn[2][13
] = M1 * m211, M1 * m214, M1 * m311, M1 * m314
860     Mn[10][1], Mn[13][1], Mn[10][2], Mn[13][2
] = M2 * m112, M2 * m142, M2 * m113, M2 * m143
861     Mn[3][8], Mn[3][9], Mn[3][11], Mn[3][12] =
M1 * m49, M1 * m410, M1 * m412, M1 * m413
862     Mn[8][3], Mn[9][3], Mn[11][3], Mn[12][3] =
M2 * m94, M2 * m104, M2 * m124, M2 * m134
863     Mn[3][13], Mn[13][3] = M1 * m414, M2 * m144
864     Mn[4][10], Mn[4][13], Mn[5][10], Mn[5][13
] = M1 * m511, M1 * m514, M1 * m611, M1 * m614
865     Mn[10][4], Mn[13][4], Mn[10][5], Mn[13][5
] = M2 * m115, M2 * m145, M2 * m116, M2 * m146
866     Mn[6][8], Mn[6][9], Mn[6][11], Mn[6][12] =
M1 * m79, M1 * m710, M1 * m712, M1 * m713
867     Mn[8][6], Mn[9][6], Mn[11][6], Mn[12][6] =
M2 * m97, M2 * m107, M2 * m127, M2 * m137
868     Mn[6][10], Mn[10][6], Mn[6][13], Mn[13][6
] = M1 * m711, M2 * m117, M1 * m714, M2 * m147
869     Mn[8][10], Mn[8][13], Mn[10][11], Mn[10][13
] = M2 * m911, M2 * m914, M2 * m1011, M2 * m1014
870     Mn[10][8], Mn[13][8], Mn[11][10], Mn[13][10
] = M2 * m911, M2 * m914, M2 * m1011, M2 * m1014
871     Mn[10][11], Mn[10][12], Mn[11][13], Mn[12][
13] = M2 * m1112, M2 * m1113, M2 * m1214, M2 * m1314
872     Mn[11][10], Mn[12][10], Mn[13][11], Mn[13][
12] = M2 * m1112, M2 * m1113, M2 * m1214, M2 * m1314
873     Mn[10][13], Mn[13][10], Mn[13][13], Mn[13][
13] = M2 * m1114, M2 * m1114, M2 * m77, M2 * m77
874
875     MnT = np.dot(np.dot(T.T, Mn.T), T)    # Mn.T
      for Element Numbering Convention
876     MnR = np.dot(np.dot(R.T, MnT), R)
877
878     for r in range(len(Kn)):
879         for c in range(len(Kn)):

```

```

880             Mg[r + eL * dofs][c + eL * dofs] =
Mg[r + eL * dofs][c + eL * dofs] + MnR[r][c] #
Global Mass Matrix
881         if divmod(eL, 2)[1] == 0:
882             e = e + 1
883         A_c[0] = A[0]
884         A_c[-1] = A[-1]
885         i = 1
886         for e in range(1, len(A_c)-1):
887             if divmod(e, 2)[1] == 0:
888                 A_c[e] = A[i]
889                 i = i + 1
890             else:
891                 A_c[e] = 0.5 * (A[i - 1] + A[i])
892
893         for i in range(ns*2-1):
894             j = i
895             Mgr[i][j] = Mg[i * dofs][j * dofs] #
Reduced Global Mass Matrix
896
897         for i in range(ns * 2 - 2):
898             j = i
899             Mgr[i][j + 1] = Mg[i * dofs][j * dofs + dofs
]
900             Mgr[i + 1][j] = Mg[j * dofs + dofs][i * dofs
]
901
902         F_c[0] = w ** 2 * np.dot(r_v, Mgr[0][:]) #
Centrifugal Force
903         for i in range(1, len(Mgr)):
904             F_c[i] = w ** 2 * np.dot(r_v, Mgr[i][:]) +
F_c[i - 1]
905         sigma_c = (F_c / A_c)[::-1] # Centrifugal
Stress
906         Fe = sigma_c * A_c # Axial Force for element
907         for i in range(len(L_v)):
908             F_e[i] = (Fe[i]+Fe[i+1])/2
909         e = 0
910
911         for eL in range(int(ns-1)*2):
912             phi_x_1 = shearparameter(E, IyR[e], G, k_x[e
], A[e], L_v[eL]) # Dimensionless
913             phi_y_1 = shearparameter(E, IxR[e], G, k_y[e
], A[e], L_v[eL])

```

```

914     phi_x_2 = shearparameter(E, IyR[e], G, k_x[e
], A[e], L_v[e]) # Dimensionless
915     phi_y_2 = shearparameter(E, IxR[e], G, k_y[e
], A[e], L_v[e])
916
917     R = rotationmatrix(delta[e], R)
918     T = traslationmatrix(dx[e], dy[e], dofs)
919
920     # Torsional curvature
921
922     Lambda_1 = sqrt((G * It[e]) / (E * Cm[e]))
923     Lambda_2 = sqrt((G * It[e]) / (E * Cm[e]))
924     e_1 = L_v[e] * Lambda_1
925     e_2 = L_v[e] * Lambda_2
926     C_1 = math.cosh(e_1)
927     C_2 = math.cosh(e_2)
928     S_1 = math.sinh(e_1)
929     S_2 = math.sinh(e_2)
930     k0_1 = (G * It[e]) / (2 * (1 - C_1) + e_1 *
S_1)
931     k0_2 = (G * It[e]) / (2 * (1 - C_2) + e_2 *
S_2)
932
933     # Stiffness Matrix Coefficients
934
935     k11 = E * A[e] / L_v[e]
936     k88 = E * A[e] / L_v[e]
937     k22 = 12 * E * IyR[e] / (L_v[e] ** 3 * (1
+ phi_x_1))
938     k99 = 12 * E * IyR[e] / (L_v[e] ** 3 * (1
+ phi_x_2))
939     k33 = 12 * E * IxR[e] / (L_v[e] ** 3 * (1
+ phi_y_1))
940     k1010 = 12 * E * IxR[e] / (L_v[e] ** 3 * (1
+ phi_y_2))
941     k26 = 6 * E * IyR[e] / (L_v[e] ** 2 * (1 +
phi_x_1))
942     k913 = 6 * E * IyR[e] / (L_v[e] ** 2 * (1
+ phi_x_2))
943     k510 = 6 * E * IxR[e] / (L_v[e] ** 2 * (1
+ phi_y_1))
944     k1012 = 6 * E * IxR[e] / (L_v[e] ** 2 * (1
+ phi_y_2))
945     k44 = k0_1 * Lambda_1 * S_1

```

```

946         k1111 = k0_2 * Lambda_2 * S_2
947         k47 = k0_1 * (C_1 - 1)
948         k144 = k0_2 * (C_2 - 1)
949         k77 = k0_1 * (C_1 * L_v[eL] - S_1 / Lambda_1
)
950         k1414 = k0_2 * (C_2 * L_v[eL] - S_2 /
Lambda_2)
951         k714 = k0_1 * (S_1 / Lambda_1 - L_v[eL])
952         k147 = k0_2 * (S_2 / Lambda_1 - L_v[eL])
953         k55 = (4 + phi_y_1) * E * IxR[e] / (L_v[eL
] * (1 + phi_y_1))
954         k1212 = (4 + phi_y_2) * E * IxR[e] / (L_v[eL
]* (1 + phi_y_2))
955         k66 = (4 + phi_x_1) * E * IyR[e] / (L_v[eL
] * (1 + phi_x_1))
956         k1313 = (4 + phi_x_2) * E * IyR[e] / (L_v[eL
] * (1 + phi_x_2))
957         k512 = (2 - phi_y_1) * E * IxR[e] / (L_v[eL
] * (1 + phi_y_1))
958         k125 = (2 - phi_y_2) * E * IxR[e] / (L_v[eL
] * (1 + phi_y_2))
959         k613 = (2 - phi_x_1) * E * IyR[e] / (L_v[eL
] * (1 + phi_x_1))
960         k136 = (2 - phi_x_2) * E * IyR[e] / (L_v[eL
] * (1 + phi_x_2))
961
962         Kn[0][0], Kn[7][7], Kn[0][7], Kn[7][0] = k11
, k88, -k11, -k88
963         Kn[1][1], Kn[8][8], Kn[1][8], Kn[8][1] = k22
, k99, -k22, -k99
964         Kn[2][2], Kn[9][9], Kn[2][9], Kn[9][2] = k33
, k1010, -k33, -k1010
965         Kn[1][5], Kn[1][12], Kn[5][8], Kn[8][12], Kn
[5][1], \
966         Kn[12][1], Kn[8][5], Kn[12][8] = k26, k26, -
k26, -k913, k26, k913, -k913, -k913
967         Kn[2][4], Kn[2][11], Kn[4][9], Kn[9][11], Kn
[4][2], \
968         Kn[11][2], Kn[9][4], Kn[11][9] = -k510, -
k510, k510, k1012, -k510, -k1012, k1012, k1012
969         Kn[3][3], Kn[10][10], Kn[3][10], Kn[10][3
] = k44, k1111, -k44, -k1111
970         Kn[4][4], Kn[11][11] = k55, k1212
971         Kn[5][5], Kn[12][12] = k66, k1313

```

```

972     Kn[4][11], Kn[11][4], Kn[5][12], Kn[12][5
    ] = k512, k125, k613, k136
973     Kn[3][6], Kn[6][3], Kn[6][6], Kn[13][13] =
    k47, k47, k77, k1414
974     Kn[3][13], Kn[13][3], Kn[6][10], Kn[10][6
    ] = k47, k144, -k47, -k144
975     Kn[6][13], Kn[13][6], Kn[10][13], Kn[13][10
    ] = k714, k147, -k144, -k144
976
977     KnT = np.dot(np.dot(T.T, Kn.T), T)
978     KnR = np.dot(np.dot(R.T, KnT), R)
979
980     # Geometric Stiffness Matrix Coefficients
981
982     g22 = (6 / 5 + 2 * phi_x_1 + phi_x_1 ** 2
    ) * (1 + phi_x_1) ** -2
983     g99 = (6 / 5 + 2 * phi_x_2 + phi_x_2 ** 2
    ) * (1 + phi_x_2) ** -2
984     g33 = (6 / 5 + 2 * phi_y_1 + phi_y_1 ** 2
    ) * (1 + phi_y_1) ** -2
985     g1010 = (6 / 5 + 2 * phi_y_2 + phi_y_2 ** 2
    ) * (1 + phi_y_2) ** -2
986     g44 = Ip[e] / A[e]
987     g1111 = Ip[e] / A[e]
988     g55 = L_v[e] ** 2 * (2 / 15 + 1 / 6 *
    phi_y_1 + 1 / 12 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
989     g1212 = L_v[e] ** 2 * (2 / 15 + 1 / 6 *
    phi_y_2 + 1 / 12 * phi_y_2 ** 2) * (1 + phi_y_2
    ) ** -2
990     g66 = L_v[e] ** 2 * (2 / 15 + 1 / 6 *
    phi_x_1 + 1 / 12 * phi_x_1 ** 2) * (1 + phi_x_1
    ) ** -2
991     g1313 = L_v[e] ** 2 * (2 / 15 + 1 / 6 *
    phi_x_2 + 1 / 12 * phi_x_2 ** 2) * (1 + phi_x_2
    ) ** -2
992     g510 = L_v[e] / 10 * (1 + phi_y_1) ** -2
993     g1012 = L_v[e] / 10 * (1 + phi_y_2) ** -2
994     g26 = L_v[e] / 10 * (1 + phi_x_1) ** -2
995     g913 = L_v[e] / 10 * (1 + phi_x_2) ** -2
996     g512 = -L_v[e] ** 2 * (1 / 30 + 1 / 6 *
    phi_y_1 + 1 / 12 * phi_y_1 ** 2) * (1 + phi_y_1
    ) ** -2
997     g125 = -L_v[e] ** 2 * (1 / 30 + 1 / 6 *

```

```

997 phi_y_2 + 1 / 12 * phi_y_2 ** 2) * (1 + phi_y_2
) ** -2
998      g613 = -L_v[eL] ** 2 * (1 / 30 + 1 / 6 *
phi_x_1 + 1 / 12 * phi_x_1 ** 2) * (1 + phi_x_1
) ** -2
999      g136 = -L_v[eL] ** 2 * (1 / 30 + 1 / 6 *
phi_x_2 + 1 / 12 * phi_x_2 ** 2) * (1 + phi_x_2
) ** -2
1000
1001      Gn[1][1], Gn[8][8], Gn[1][8], Gn[8][1] =
g22, g99, -g22, -g99
1002      Gn[2][2], Gn[9][9], Gn[2][9], Gn[9][2] =
g33, g1010, -g33, -g1010
1003      Gn[1][5], Gn[1][12], Gn[5][8], Gn[8][12] =
g26, g26, -g26, -g913
1004      Gn[5][1], Gn[12][1], Gn[8][5], Gn[12][8] =
g26, g913, -g913, -g913
1005      Gn[2][4], Gn[2][11], Gn[4][9], Gn[9][11
] = -g510, -g510, g510, g1012
1006      Gn[4][2], Gn[11][2], Gn[9][4], Gn[11][9
] = -g510, -g1012, g1012, g1012
1007      Gn[3][3], Gn[10][10], Gn[3][10], Gn[10][3
] = g44, g1111, -g44, -g1111
1008      Gn[4][4], Gn[11][11] = g55, g1212
1009      Gn[5][5], Gn[12][12] = g66, g1313
1010      Gn[4][11], Gn[11][4], Gn[5][12], Gn[12][5
] = g512, g125, g613, g136
1011
1012      GnT = np.dot(np.dot(T.T, Gn.T), T)
1013      GnR = np.dot(np.dot(R.T, GnT), R)
1014
1015      Gn_e = GnR * F_e[eL] / L_v[eL]
1016
1017      for r in range(len(Kn)):
1018          for c in range(len(Kn)):
1019              Kg[r + eL * dofs][c + eL * dofs] =
Kg[r + eL * dofs][c + eL * dofs] + KnR[r][c] + Gn_e
[r][c]
1020          if divmod(eL, 2)[1] == 0:
1021              e = e + 1
1022
1023      Kg = Kg - Mg * w ** 2 # Mass spin softening
1024
1025      # Eigenvalues

```

```

1026
1027     if BC == 1:
1028         (Q, V) = eig(Kg[dofs:, dofs:], Mg[dofs:,
1029         dofs:]) # Constrained Beam at lowest section
1029     elif BC == 0:
1030         (Q, V) = eig(Kg, Mg) # Free Beam
1031
1032     omega = sqrt(Q)
1033     oma = array(omega)
1034     fn = oma / (2 * math.pi)
1035
1036     fn_real = fn.real
1037
1038     order = fn_real.ravel().argsort() # preserve
1039     order for both eigenvalues & vectors
1040
1041     mf = len(Q)
1042     NN = zeros(mf, 'f')
1043     FN = zeros(mf, 'f')
1044
1045     for i in range(0, mf):
1046         NN[i] = float(i + 1)
1047         FN[i] = fn_real[order[i]]
1048
1049     mfs = mf
1050     if mfs > 100:
1051         mfs = 100
1052
1053     print(" ")
1054     print("Natural Frequencies ")
1055     print(" ")
1056     print("%8.7s" % 'fn [Hz]')
1057
1058     for i in range(0, 20):
1059         print("%8.5g" % (FN[i]))
1060
1061     # Mass Normalize Eigenvectors
1062
1063     if BC == 1:
1064         QQQ = dot(V.T, dot(Mg[dofs:, dofs:], V))
1065         # Constrained Beam at lowest section
1066     elif BC == 0:
1067         QQQ = dot(V.T, dot(Mg, V)) # Free Beam

```

```

1067     for i in range(0, mf):
1068         nf = sqrt(QQQ[i, i])
1069         for j in range(0, mf):
1070             V[j, i] /= nf
1071
1072     # Sort Eigenvectors
1073
1074     MS = zeros((mf, mf), 'f')
1075
1076     for i in range(0, mf):
1077         MS[0:mf, i] = V[0:mf, order[i]]
1078     print('Cycle for Mode shapes? 0 = No, 1 = Yes')
1079     answer = int(input())
1080     while answer == 1:
1081         MSV = np.zeros(ns * 2 - 1)
1082         MSVx = np.zeros(ns * 2 - 1)
1083         MSVy = np.zeros(ns * 2 - 1)
1084         Lb = np.zeros(ns * 2 - 1)
1085         m = 0
1086         print('Modal shape of what frequency? ', '\
n')
1087         ind_wn = int(input()) - 1
1088         mod = 'flessionale'
1089         print('FN:   ', FN[ind_wn])
1090         uz = 0
1091         ux = 1
1092         uy = 2
1093         for i in range(ns * 2 - 2):
1094             Lb[i + 1] = r_v[i + 1] - r_v[0]
1095             if m <= max(abs(MS[i * dofs + uz][
ind_wn]), abs(MS[i * dofs + ux][ind_wn]), abs(MS[i
* dofs + uy][ind_wn])):
1096                 m = max(abs(MS[i * dofs + uz][
ind_wn]), abs(MS[i * dofs + ux][ind_wn]), abs(MS[i
* dofs + uy][ind_wn]))
1097                 MSV[i + 1] = MS[i * dofs + uz][ind_wn]
1098                 MSVx[i + 1] = MS[i * dofs + ux][ind_wn]
1099                 MSVy[i + 1] = MS[i * dofs + uy][ind_wn]
1100             Lb = Lb / max(abs(Lb))
1101             Lbnew = np.linspace(Lb[0], Lb[-1], 100)
1102             spl = interpolate.splrep(Lb, MSV, s=0)
1103             spline = interpolate.splev(Lbnew, spl, der=
0)
1104             splx = interpolate.splrep(Lb, MSVx, s=0)

```

```
1105     splinex = interpolate.splev(Lbnew, splx,  
    der=0)  
1106     sply = interpolate.splrep(Lb, MSVy, s=0)  
1107     spliney = interpolate.splev(Lbnew, sply,  
    der=0)  
1108  
1109     fig = plt.figure()  
1110     plt.plot(Lb, MSV, 'o', Lbnew, spline, '--')  
1111     plt.ylim(-m * 1.2, m * 1.2)  
1112     plt.xlabel('Asse z')  
1113     plt.ylabel('Ampiezza')  
1114     plt.title('Frequenza naturale: %d°' % (  
    ind_wn - 1) + ' %s' % mod)  
1115     plt.show()  
1116  
1117     fig2 = plt.figure()  
1118     ax = fig2.add_subplot(1, 1, 1, projection='  
    3d')  
1119     ax.scatter(Lbnew, splinex, spliney, c='r',  
    marker='.') # 3D plot  
1120     ax.set_ylim3d(-m * 1.2, m * 1.2)  
1121     ax.set_zlim3d(-m * 1.2, m * 1.2)  
1122     plt.title('Frequenza naturale: %d°' % 1 +  
    ' %s' % mod)  
1123     ax.set_xlabel('Lunghezza trave  
    adimensionalizzata')  
1124     ax.set_ylabel('Asse Y')  
1125     ax.set_zlabel('Asse Z')  
1126     plt.show()  
1127     print('Continue? 0 = No 1 = Yes')  
1128     answer = int(input())  
1129     elapsed = time.time() - t  
1130     print('Time=', elapsed, 's', 'or %1.2g min' % float  
    (elapsed/60))
```