FEM_viga

August 1, 2021

[1]:
```
import numpy as np
import pandas as pd
from scipy.linalg import eig
import matplotlib.pyplot as plt
pi = np.pi
```

1 Propriedades da viga

[2]:
```
L = 0.5
b = 0.02
h = 0.005
A = b*h
I = b*h**3/12
pho = 2700
E = 7.1e10
```

2 Análise

[3]:
```
n = np.array([4, 6, 100]) # Número de elementos da viga
resultados = dict() # Dicionário para armazenar os resultados

for N in n:
    # Montando as matrizes do elemento
    a = L/2/N
    Me = (pho*A*a/105)*np.array([[78, 22*a, 27, -13*a], [22*a, 8*a**2, 13*a,␣
↪-6*a**2], [27, 13*a, 78, -22*a], [-13*a, -6*a**2, -22*a, 8*a**2]])
    Ke = (E*I/(2*a**3))*np.array([[3, 3*a, -3, 3*a], [3*a, 4*a**2, -3*a,␣
↪2*a**2], [-3, -3*a, 3, -3*a], [3*a, 2*a**2, -3*a, 4*a**2]])

    # Montando as matrizes do sistema
    Nn = N + 1
    Ngl = Nn*2
    M = np.zeros((Ngl,Ngl))
    K = np.zeros((Ngl,Ngl))
    for j in range(1,N+1):
        Mee = np.zeros((Ngl,Ngl))
```

```
        Mee[(2*j-2):(2*j+2),(2*j-2):(2*j+2)] = Me
        M = M + Mee
        Kee = np.zeros((Ngl,Ngl))
        Kee[(2*j-2):(2*j+2),(2*j-2):(2*j+2)] = Ke
        K = K + Kee

    # Aplicando as condicoes de contorno geometricas
    cc = [1, 2, Ngl-1] # Graus de Liberdade que devem ser restritos em ordem crescente.
    for j in range(1,len(cc)+1):
        M = np.delete(M, cc[j-1]-j, axis=1)
        M = np.delete(M, cc[j-1]-j, axis=0)
        K = np.delete(K, cc[j-1]-j, axis=1)
        K = np.delete(K, cc[j-1]-j, axis=0)

    # Problema de autovalor generalizado. W é um vetor e A uma matrix com as colunas normalizadas
    W, Vc = eig(K, M)

    # Ordenando os autovalores e a matriz de autovetores
    idx = W.argsort()
    W = W[idx]
    Vc = Vc[:,idx]

    # Normalizando os autovetores pela matriz de massa, de forma que A'@M@A = I
    m_r = np.diagonal(Vc.T @ M @ Vc)
    m_r = np.reciprocal(np.sqrt(m_r))
    for a in range(Vc.shape[1]):
        Vc[:,a] *= m_r[a] # multiplica cada coluna pelo fator de escala

    resultados[N] = dict()
    resultados[N]['fn'] = (W**0.5/(2*pi)).real

    ## Montando as formas modais
    # Incluindo os GL das condicoes de contorno
    for c in cc:
        Vc = np.insert(Vc, c-1, 0, axis=0)

    resultados[N] = dict()
    resultados[N]['fn'] = (W**0.5/(2*pi)).real
    resultados[N]['V'] = Vc[0::2, :] # Modos de deslocamento (Gl 1,3,5,...) https://stackoverflow.com/questions/509211/understanding-slice-notation
    resultados[N]['theta'] = Vc[1::2, :] # Modos angulares (Gl 2,4,6,...)

    # Faz as formas modais terem a mesma orientação
    for j in range(1, W.size):
        if np.sum(resultados[N]['V'][:,j-1]) >= 0:
```

```
            pass
        else:
            resultados[N]['V'][:,j-1] *= -1
```

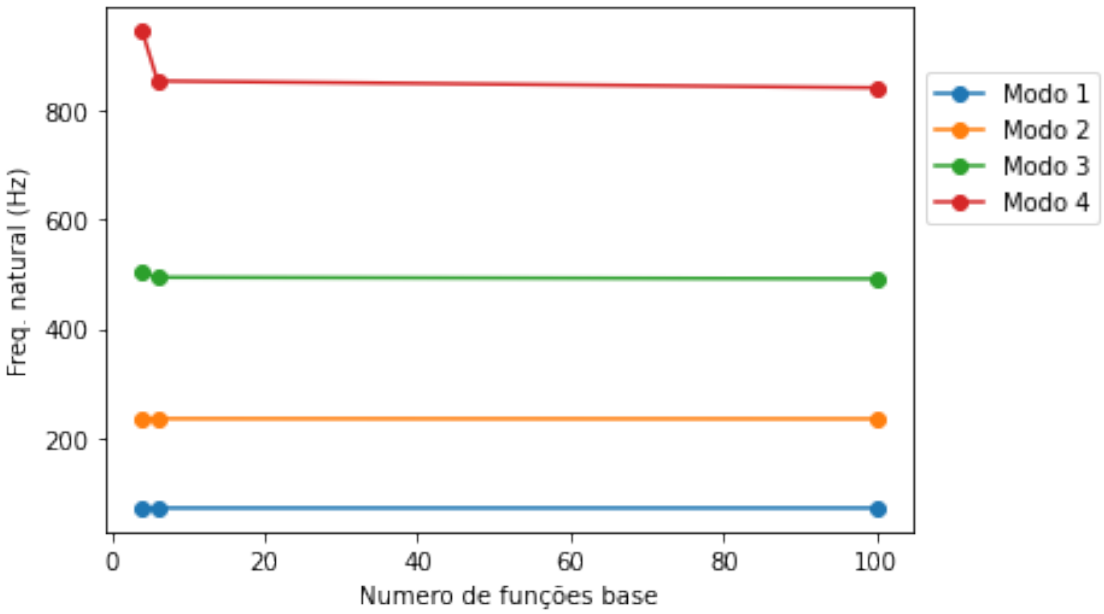
3 Frequências Naturais

[4]:
```
pd.options.display.float_format = "{:.2f}".format
pd.DataFrame(data=[resultados[n_b]['fn'] for n_b in n], index=[f"n={i}" for i␣
↪in n], columns=[f"f_{j}" for j in range(1, len(resultados[n.
↪max()]['fn'])+1)])
```

[4]:
```
        f_1    f_2    f_3    f_4     f_5     f_6     f_7     f_8     f_9  \
n=4   72.70 236.90 502.29 943.23 1537.81 2455.32 3554.73     NaN     NaN
n=6   72.66 235.74 493.87 852.31 1317.34 2018.76 2790.04 3842.05 5216.77
n=100 72.65 235.44 491.22 840.01 1281.82 1816.63 2444.46 3165.31 3979.17

         f_10  …      f_190      f_191      f_192      f_193      f_194  \
n=4       NaN  …        NaN        NaN        NaN        NaN        NaN
n=6   6878.27  …        NaN        NaN        NaN        NaN        NaN
n=100 4886.05  … 2290716.68 2305151.89 2318171.35 2329712.14 2339717.10

           f_195      f_196      f_197      f_198      f_199
n=4          NaN        NaN        NaN        NaN        NaN
n=6          NaN        NaN        NaN        NaN        NaN
n=100 2348135.58 2354924.20 2360047.47 2363478.25 2365198.26

[3 rows x 199 columns]
```

3.1 Comparação das frequências naturais

[5]:
```
fn_N=np.zeros((4, len(n)))
for j in range(0,4): # numero de modos
    for k in range(len(n)): # Número de simulações
        fn_N[j, k] = resultados[n[k]]['fn'][j]

for j in range(fn_N[:,0].size):
    plt.plot(n, fn_N[j,:], marker='o')

plt.xlabel('Numero de funções base')
plt.ylabel('Freq. natural (Hz)')
plt.legend(['Modo 1', 'Modo 2','Modo 3', 'Modo 4'], loc='upper left',␣
↪bbox_to_anchor=(1, 0.9))
plt.show()
```



3.2 Comparação das formas modais

[6]:
```
fig, axs = plt.subplots(2,2)

for n_b in n:
    x = np.linspace(0, L, len(resultados[n_b]['V'][:,0]))

    axs[0,0].set_title('Primeiro modo')
    axs[0,0].plot(x, resultados[n_b]['V'][:,0], label=f"{n_b} elementos")
    axs[0,0].set_xlabel('x [m]')
    axs[0,0].set_ylabel('Forma modal')

    axs[0,1].set_title('Segundo modo')
    axs[0,1].plot(x, resultados[n_b]['V'][:,1])
    axs[0,1].set_xlabel('x [m]')
    axs[0,1].set_ylabel('Forma modal')

    axs[1,0].set_title('Terceiro modo')
    axs[1,0].plot(x, resultados[n_b]['V'][:,2])
    axs[1,0].set_xlabel('x [m]')
    axs[1,0].set_ylabel('Forma modal')

    axs[1,1].set_title('Quarto modo modo')
    axs[1,1].plot(x, resultados[n_b]['V'][:,3])
    axs[1,1].set_xlabel('x [m]')
    axs[1,1].set_ylabel('Forma modal')
```

```
fig.legend(loc='upper left', bbox_to_anchor=(1, 0.9))
fig.tight_layout()
plt.show()
```