In [1]:

```
import numpy as np
from scipy import integrate
from scipy.misc import derivative
from scipy.linalg import eig
import matplotlib.pyplot as plt

pi = np.pi
```

Método de Rayleigth-Ritz

In [2]:

```
L = 1.5                             # Comprimento da viga [m]
rho = 7700                          # Densidade do material [kg/m**3]
E = 2.1e11                          # Modulo de elasticidade [Pa]
A1 = pi * 0.03**2 / 4                # Area da secao 1 [m**2]
A2 = pi * 0.06**2 / 4                # Area da secao 2 [m**2]
A3 = pi * 0.03**2 / 4                # Area da secao 3 [m**2]
I1 = (pi/4)*(0.03/2)**4              # Momento de area da secao 1 [m**4]
I2 = (pi/4)*(0.06/2)**4              # Momento de area da secao 2 [m**4]
I3 = (pi/4)*(0.03/2)**4              # Momento de area da secao 3 [m**4]
m_eng = 20                          # Massa da engrenagem [kg]
J_eng = 0.25 * m_eng * 0.1**2       # Momento de inercia de massa da engreagem [kg*m**2]
k_mola = 2*10e3                      # Rigidez esquivalente das molas em paralelo [N/m]


def eta(j):
    if j==1:
        return 1.875/L
    if j>1:
        return (j-0.5)*pi/L

def D(etaj):
    etajL = etaj*L
    return ((np.cos(etajL) + np.cosh(etajL))/(np.sin(etajL) - np.sinh(etajL))).tolist()

def d(x, etaj):
    etajx = etaj*x
    D_ = D(etaj)
```

```
    # Precisa reformular para evitar erros de arredondamento
    return np.sin(etajx) + D_*np.cos(etajx) - 0.5*((D_ - 1)*np.exp(-etajx) + (D_ + 1)*np.exp(etajx))
```

In [3]:

```
n = np.array([4, 8, 12, 16])                # Array com os numeros de funcoes base
N = np.max(n)                               # No. de funcoes que serao utilizadas para montar as matrizes K e M
K = np.zeros((N, N))                        # Pre-alocando matriz de rigidez
M = np.zeros((N, N))                        # Pre-alocando matriz de massa

L1 = np.linspace(0, L/3, 50000)
L2 = np.linspace(L/3, 2*L/3, 50000)
L3 = np.linspace(2*L/3, L, 50000)

didj = lambda x, i, j: d(x, eta(i))*d(x, eta(j))
diff2 = lambda x, i, j: derivative(d, x, n=2, dx=1e-6, args = (eta(i),))*derivative(d, x, n=2, dx=1e-6, args = (eta(j),))

for i in range(1, N+1):
    for j in range(i, N+1):

        M[i-1, j-1] = rho*A1*integrate.simpson(didj(L1, i, j), L1) \
            + rho*A2*integrate.simpson(didj(L2, i, j), L2) \
            + rho*A3*integrate.simpson(didj(L3, i, j), L3) \
            + m_eng*d(L/3, eta(i))*d(L/3, eta(j)) \
            + J_eng*derivative(d, L/3, dx=1e-6, args = (eta(i),))*derivative(d, L/3, dx=1e-6, args = (eta(j),))

        K[i-1, j-1] = E*I1*integrate.simpson(diff2(L1, i, j), L1) \
            + E*I2*integrate.simpson(diff2(L2, i, j), L2) \
            + E*I3*integrate.simpson(diff2(L3, i, j), L3) \
            + k_mola*d(2*L/3, eta(i))*d(2*L/3, eta(j))

# Espelhando a matriz
M = (M + M.T - np.diag(np.diagonal(M))).real
K = (K + K.T - np.diag(np.diagonal(K))).real

# Recortando as matrizes para menos funções base
dict_M = {}
dict_K = {}
for j in n:
    dict_M[j] = M[:j, :j]
    dict_K[j] = K[:j, :j]
```

In [4]:

```
N = 16
# Problema de autovalor generalizado. W é um vetor e A uma matriz com as colunas normalizadas (norm=1)
W, A = eig(dict_K[N], dict_M[N])
```

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

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

print(np.sqrt(W)/(2*pi))
```

```
[9.70273703e+00 4.80689665e+01 1.33788809e+02 4.45151673e+02
 6.20919942e+02 7.85391654e+02 1.26420604e+03 1.75497249e+03
 1.91275366e+03 2.81396056e+03 3.56186954e+03 3.83744499e+03
 5.31189572e+03 6.39125728e+03 9.08108806e+03 1.00337819e+04]
```