Escola Politécnica da Universidade de São Paulo
Engenharia Elétrica

MAP3121 - Métodos Numéricos e Aplicações

André Lucas Pierote Rodrigues Vasconcelos - NUSP: 11356540 - Turma: 01
Leonardo Isao Komura - NUSP: 11261656 - Turma: 03

São Paulo, SP
2021

SUMÁRIO

1. Introdução

Matrizes tridiagonais simétricas são objetos operacionais de grande relevância para diversas aplicações, pois, seus autovalores e autovetores carregam informações importantes e de grande utilidade para alguns sistemas.

Assim sendo, esse relatório tem como objetivo **transformar uma matriz simétrica em outra tridiagonal simétrica, com auxílio do algoritmo de Householder,** para o cálculo dos autovalores e autovetores dessa matriz. Além disso, o código também possibilita o uso dos valores obtidos para um aplicação: **calcular as frequências e os modos de vibração em treliças planas.**

2. Informações gerais

O código foi desenvolvido por meio da interface de programação *Visual Studio Code*, e feito na linguagem de programação *Python 3.7.9+*.

Como bibliotecas, foram utilizadas:

- *math*: para cálculos matemáticos;
- *numpy*: para manipulação de vetores e matrizes;
- *matplotlib*: para a construção de gráficos;
- *sys*: para a assistência na representação de matrizes.

Para a testagem do programa, foi utilizado um computador com um processador *Intel(R) Core(TM) i5-8400 @ 2.80GHz*, logo, o número de iterações e tempo de execução são referentes a ele e podem variar dependendo da máquina utilizada.

O código está contido inteiramente no arquivo **EP2.py** e há um arquivo chamado **LEIA-ME.txt** que apresenta instruções para sua execução.

3. Como executar o código

Ao iniciar o programa, o usuário irá deparar-se com a interface inicial, onde haverá um cabeçalho e, a seguinte, um menu para selecionar qual operação será realizada.

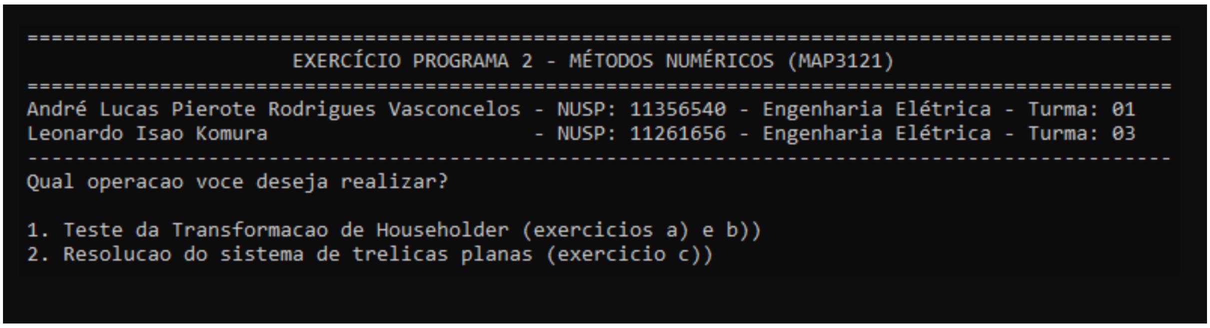

Figura 1 - Interface do usuário inicial

3.a) Operação 1 - Transformação de Householder

A operação 1 realiza a transformação de Householder e o algoritmo QR com deslocamento espectral dada uma matriz especificada pelo usuário. Selecionada essa opção, serão pedidos:

I. O que o usuário pretende fazer, seja digitar as entradas da matriz ou carregar um determinado arquivo de input;
II. Os valores da matriz ou qual input específico o usuário quer utilizar (depende do escolhido anteriormente);

Dadas as entradas, o programa irá realizar cálculos e retornará as matrizes simétricas como tridiagonais simétricas, as quais são posteriormente utilizadas para o cálculo dos autovetores e os autovalores através do algoritmo QR feito no EP1 com deslocamento espectral, e seus valores teóricos.

Além disso, serão impressas outras informações pedidas pelos itens a) e b).

3.b) Operação 2 - Aplicação para treliças planas

A operação 2 realiza o cálculo das posições de um número determinado de massas presas em molas em função de suas posições iniciais. Selecionada essa operação, serão pedidos:

I. Qual o número de nós;
II. Número de nós não fixos;
III. Número de barras;
IV. Densidade massa (rho);
V. Área da seção transversal (A);
VI. Módulo de elasticidade (E).
VII. Nós que formam cada barra, junto de seu ângulo formado com a horizontal e seu comprimento.

Esses valores devem ser escritos num arquivo com o nome “input-c” que segue o padrão descrito pelo enunciado.

Dadas as entradas, o programa apresentará as cinco menores frequências e os modos de vibração de cada nó respectivos para cada uma delas.

4. Código

4.a) Transformação de Householder

O objetivo dessa primeira parte era promover transformações numa matriz A, de forma que ela virasse uma T tridiagonal simétrica. Em outras palavras, temos que zerar linhas e colunas a fim de fazer isso acontecer.

A primeira etapa para fazer a transformação de Householder foi a passagem

$$\bar{w}_i = \bar{a}_i + \delta \frac{||\bar{a}_i||}{||e||} e = \bar{a}_i + \delta ||\bar{a}_i|| e$$

Dessa maneira, definiu-se os vetores “alpha” e “e”, para que assim pudessem ser calculados ω e ω_{barra}

```
def Householder(n, matrix):
    A = matrix.copy()
    Ht = np.eye(n)
    for i in range(n-2):
        HtFinal = Ht.copy()
        Nova = A.copy()
        a = np.zeros(n)
        for j in range(n-1-i):
            a[j+1+i] = Nova[j+1+i][i]

        #print("a = ", a)

        alpha = np.zeros(n-1-i)
        for j in range(n-1-i):
            alpha[j] = a[j+1+i]

        #print("alpha = ",alpha)

        norm2 = np.dot(a,a)
        norm = math.sqrt(norm2)

        e = np.zeros(n)
        e[i+1] = norm

        #print("e = ", e)

        w = np.zeros(n-i)
        if A[i+1][i] > 0:
            w = np.add(np.array(a),np.array(e))
        else:
            w = np.subtract(np.array(a),np.array(e))

        #print("w = ", w)

        w_barra = np.zeros(n-1-i)
        for j in range(n-1-i):
            w_barra[j] = w[j+1+i]
        #print("wbarra = ",w_barra)
```

Figura 2- Cálculo de α, e, ω, ω_{barra}

O cálculo final foi

$$H_w x = x - 2\frac{w \cdot x}{w \cdot w} w = x - w = -y$$

Percebeu-se logo que para cada linha/coluna aplicado Householder, ou seja, para cada $H\omega_{1barra} * linha/coluna\ do\ vetor$ podíamos fazer o conta com a multiplicação $vetor - \omega_{1barra}$ quando era a primeira linha/coluna da matriz, nas demais deve-se utilizar os produtos escalares $2w(w \cdot x \div w \cdot w)$.

```
    Hwa = np.zeros(n-1-i)
    Hwa = np.subtract(alpha, w_barra)

    for k in range(n-1-i):
        Nova[k+1+i][i] = Hwa[k]
    for j in range(n-1-i):
        Nova[i][j+1+i] = Nova[j+1+i][i]

    #HwA - Submatriz
    col = np.zeros(n-1-i)
    for j in range(n-1-i):
        for k in range(n-1-i):
            col[k] = A[k+1+i][j+1+i]
        for k in range(n-1-i):
            Nova[k+1+i][j+1+i] = A[k+1+i][j+1+i] - (2*np.dot(w_barra,col)/np.dot(w_barra,w_barra))*w_barra[k]

    A = Nova.copy()

    #HwAHw - Submatriz
    lin = np.zeros(n-1-i)
    for j in range(n-1-i):
        for k in range(n-1-i):
            lin[k] = A[j+1+i][k+1+i]
        for k in range(n-1-i):
            Nova[j+1+i][k+1+i] = A[j+1+i][k+1+i] - (2*np.dot(w_barra,lin)/np.dot(w_barra,w_barra))*w_barra[k]

    #Ht = IHw1Hw2...
    lin = np.zeros(n-1-i)
    for j in range(n-1):
        for k in range(n-1-i):
            lin[k] = Ht[j+1][k+1+i]
        for k in range(n-1-i):
            HtFinal[j+1][k+1+i] = Ht[j+1][k+1+i] - (2*np.dot(w_barra,lin)/np.dot(w_barra,w_barra))*w_barra[k]

    Ht = HtFinal.copy()
    A = Nova.copy()
    #print("Ht %d: "%(i+1))
    #print(Ht)

return Nova, HtFinal
```

Figura 3- Multiplicações envolvendo a matriz Hω

Assim, com a assistência do algoritmo QR com deslocamento espectral, fez-se o seguinte passo a passo:

- Aplica-se Householder na matriz A, obtendo a matriz T tridiagonal simétrica e a matriz H transposta (Ht);
- Aplica-se o algoritmo QR em T e obtém-se seus autovetores (V) e autovalores;
- Realiza-se o produto Ht*V para obter os autovetores ortogonais de A;

- Pega-se os autovalores e forma-se uma matriz Lambda com seus valores na diagonal e, então, realiza-se o produto V*Lambda*Vt, sendo Vt a transposta da matriz V de autovetores. Como resultado final obtém-se, novamente, a matriz tridiagonal semelhante, que deve ser a própria T.

```python
# Aplicacao de Householder
print("   Aplicacao de Householder: ")
T, Ht = Householder(n, A)
print("   Matriz T = H*A*Ht: ")
print(T)
print("\n")
print("   Matriz Ht: ")
print(Ht)

# Aplicacao do algoritmo QR com deslocamento espectral
a = np.zeros(n)
b = np.zeros(n-1)
for i in range(n):
    a[i] = T[i][i]
for i in range(n-1):
    b[i] = T[i+1][i]
g = b.copy()

V, Autovalores, k = QR_deslocamento(a, b, g, n, 10**(-6), Ht)

print("\n")
print("   Aplicacao do algoritmo QR na matriz T: ")
print("   Autovetores de T: ")
print(V)
print("\n")

print("   Autovalores de T: ", Autovalores)

print("\n")
print("   Matriz Ht*V (autovetores ortogonais de A): ")
HtV = np.matmul(Ht,V)
print(HtV)

Lambda = np.eye(n)
for i in range(n):
    Lambda[i][i] = Autovalores[i]

Vlambda = np.matmul(V, Lambda)
T_QR = np.matmul(Vlambda, np.transpose(V))
print("\n")
print("   Forma diagonal semelhante: ")
print("   Matriz V*Lambda*Vt: ")
print(T_QR)
```

Figura 4 - Algoritmo QR sendo aplicado para encontrar autovalores e autovetores

Para o restante dos itens a) e b) foram realizadas as seguintes operações:

- Primeiramente são impressos os autovalores teóricos e os obtidos, para comparação;
- Realiza-se os produtos A*v e lambda*v, sendo v um autovetor e lambda o autovalor correspondente a ele. Em sequência, são impressos os resultados para comparação.

- Por fim, checa-se se a matriz de autovetores é ortogonal. Para isso, basta verificar se o produto entre a matriz V e sua transposta resulta numa matriz identidade.

```
if arquivo==1:
    print("\n")
    print("   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    print("   Restante da resolucao do exercicio a): ")
    print("\n")
    print("   Autovalores esperados: (7, 2, -1 e -2)")
    print("   Autovalores obtidos pela matriz T (H*A*Ht): ", Autovalores)
    print("\n")
    print("   Verificacao se A*v = Lambda*v: ")
    for i in range(n):
        vtrans = np.zeros(n)
        for j in range(n):
            vtrans[j] = V[j][i]
        vetor = np.transpose(vtrans)
        valor = Autovalores[i]
        Av = np.matmul(A, vetor)
        LambdaV = valor*np.array(vetor)
        print("   v = ", vtrans)
        print("   Lambda = ", valor)
        print("   A*v    =   ", np.transpose(Av))
        print("   Lambda*v = ", np.transpose(LambdaV))
        print("\n")
    print("   Verificacao se V é ortogonal (V * Vt = I) (erro absoluto de 1e-5): ")
    I = np.matmul(V, np.transpose(V))
    print("   Matriz V*Vt: ")
    print(I)
    check = True
    for i in range(n):
        for j in range(n):
            check = check * math.isclose(I[i][j], ident[i][j], abs_tol=1e-5)
    if(check == True):
        print("   Ela é ortogonal")
    else:
        print("   Ela não é ortogonal")

else:
    print("\n")
    print("   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    print("   Restante da resolucao do exercicio b): ")
    print("\n")
    valores = np.zeros(n)
    for i in range(n):
        valores[i] = 0.5/(1-math.cos(((2*(i+1)-1)*math.pi)/((2*n)+1)))
    print("   Autovalores esperados: ", valores)
    print("   Autovalores obtidos: ", Autovalores)
    print("\n")
    print("   Verificacao se A*v = Lambda*v: ")
    for i in range(n):
        vtrans = np.zeros(n)
        for j in range(n):
            vtrans[j] = V[j][i]
        vetor = np.transpose(vtrans)
        valor = Autovalores[i]
        Av = np.matmul(A, vetor)
        LambdaV = valor*np.array(vetor)
        print("   v = ", vtrans)
        print("   Lambda = ", valor)
        print("   A*v    =   ", np.transpose(Av))
        print("   Lambda*v = ", np.transpose(LambdaV))
        print("\n")
    print("   Verificacao se V é ortogonal (V * Vt = I) (erro absoluto de 1e-5): ")
    I = np.matmul(V, np.transpose(V))
    print("   Matriz V*Vt: ")
    print(I)
    check = True
    for i in range(n):
        for j in range(n):
            check = check * math.isclose(I[i][j], ident[i][j], abs_tol=1e-5)
    if(check == True):
        print("   Ela é ortogonal")
    else:
        print("   Ela não é ortogonal")
```

Figura 5 - Testes específicos pedidos nos itens a) e b)

4.b) Aplicação para treliças planas

Após a leitura do arquivo (onde obtemos as informações descritas no item 3.b)) , calculamos a matriz K da seguinte forma:

$$K^{\{i,j\}} = \frac{AE}{L_{\{i,j\}}} \cdot \begin{pmatrix} C^2 & CS & -C^2 & -CS \\ CS & S^2 & -CS & -S^2 \\ -C^2 & -CS & C^2 & CS \\ -CS & -S^2 & CS & S^2 \end{pmatrix}$$

Nesse caso, tanto a área da seção transversal quanto o módulo de elasticidade são dados, a matriz de senos e cossenos foi feita multiplicando cada elemento na posição por seu respectivo valor. Também pegou-se dois casos, aquele que os valores de de i e j estão no intervalo (1 a 12) e aqueles que não estão (13,14) feitos separadamente. Vale lembrar que o impacto no K é dependente apenas do nó i (que deve estar entre 1 e 12), no entanto j pode variar até 14 caso essa última condição seja satisfeita.

OBS*: Para o armazenamento dos ângulos e comprimentos, foram montadas duas matrizes 14x14 onde cada linha representa o primeiro nó e cada coluna o segundo nó. Vale ressaltar que ela é simétrica, pois {i,j} = {j,i}

```python
# Definindo a matriz de rigidez K (excluindo as barras conectadas aos nos 13 e 14)
n = 2*n_nfixos
Kij = np.zeros((4, 4))
K = np.zeros((n, n))
for i in range(n_nfixos):
    for j in range(i):
        if (comprimento[i][j] != 0):
            Kij[0][0] = (secao*E/comprimento[i][j]) * (math.cos(np.radians(theta[i][j])))**2
            Kij[1][1] = (secao*E/comprimento[i][j]) * (math.sin(np.radians(theta[i][j])))**2
            Kij[2][2] = (secao*E/comprimento[i][j]) * (math.cos(np.radians(theta[i][j])))**2
            Kij[3][3] = (secao*E/comprimento[i][j]) * (math.sin(np.radians(theta[i][j])))**2
            Kij[1][0] = (secao*E/comprimento[i][j]) * math.cos(np.radians(theta[i][j]))*math.sin(np.radians(theta[i][j]))
            Kij[0][1] = Kij[1][0]
            Kij[2][0] = (secao*E/comprimento[i][j]) * -(math.cos(np.radians(theta[i][j])))**2
            Kij[0][2] = Kij[2][0]
            Kij[2][1] = (secao*E/comprimento[i][j]) * -(math.cos(np.radians(theta[i][j])))*math.sin(np.radians(theta[i][j]))
            Kij[1][2] = Kij[2][1]
            Kij[3][0] = (secao*E/comprimento[i][j]) * -(math.cos(np.radians(theta[i][j])))*math.sin(np.radians(theta[i][j]))
            Kij[0][3] = Kij[3][0]
            Kij[3][1] = (secao*E/comprimento[i][j]) * -(math.sin(np.radians(theta[i][j])))**2
            Kij[1][3] = Kij[3][1]
            Kij[3][2] = (secao*E/comprimento[i][j]) * math.cos(np.radians(theta[i][j]))*math.sin(np.radians(theta[i][j]))
            Kij[2][3] = Kij[3][2]

            K[(2*(i+1))-2][(2*(i+1))-2] += Kij[0][0]
            K[(2*(i+1))-2][2*(i+1)-1]   += Kij[0][1]
            K[(2*(i+1))-2][(2*(j+1))-2] += Kij[0][2]
            K[(2*(i+1))-2][2*(j+1)-1]   += Kij[0][3]
            K[2*(i+1)-1][(2*(i+1))-2]   += Kij[1][0]
            K[2*(i+1)-1][2*(i+1)-1]     += Kij[1][1]
            K[2*(i+1)-1][(2*(j+1))-2]   += Kij[1][2]
            K[2*(i+1)-1][2*(j+1)-1]     += Kij[1][3]
            K[(2*(j+1))-2][(2*(i+1))-2] += Kij[2][0]
            K[(2*(j+1))-2][2*(i+1)-1]   += Kij[2][1]
            K[(2*(j+1))-2][(2*(j+1))-2] += Kij[2][2]
            K[(2*(j+1))-2][2*(j+1)-1]   += Kij[2][3]
            K[2*(j+1)-1][(2*(i+1))-2]   += Kij[3][0]
            K[2*(j+1)-1][2*(i+1)-1]     += Kij[3][1]
            K[2*(j+1)-1][(2*(j+1))-2]   += Kij[3][2]
            K[2*(j+1)-1][2*(j+1)-1]     += Kij[3][3]
```

Figura 6 - Matriz de rigidez K

```
# Adicionando a contribuição das barras {11,14} e {12, 13}
for i in range(n_nfixos):
    for j in range(12,14,1):
        if (comprimento[i][j] != 0):
            Kij[0][0] = (secao*E/comprimento[i][j]) * (math.cos(np.radians(theta[i][j])))**2
            Kij[1][1] = (secao*E/comprimento[i][j]) * (math.sin(np.radians(theta[i][j])))**2
            Kij[2][2] = (secao*E/comprimento[i][j]) * (math.cos(np.radians(theta[i][j])))**2
            Kij[3][3] = (secao*E/comprimento[i][j]) * (math.sin(np.radians(theta[i][j])))**2
            Kij[1][0] = (secao*E/comprimento[i][j]) * math.cos(np.radians(theta[i][j]))*math.sin(np.radians(theta[i][j]))
            Kij[0][1] = Kij[1][0]
            Kij[2][0] = (secao*E/comprimento[i][j]) * -(math.cos(np.radians(theta[i][j])))**2
            Kij[0][2] = Kij[2][0]
            Kij[2][1] = (secao*E/comprimento[i][j]) * -(math.cos(np.radians(theta[i][j])))*math.sin(np.radians(theta[i][j]))
            Kij[1][2] = Kij[2][1]
            Kij[3][0] = (secao*E/comprimento[i][j]) * -(math.cos(np.radians(theta[i][j])))*math.sin(np.radians(theta[i][j]))
            Kij[0][3] = Kij[3][0]
            Kij[3][1] = (secao*E/comprimento[i][j]) * -(math.sin(np.radians(theta[i][j])))**2
            Kij[1][3] = Kij[3][1]
            Kij[3][2] = (secao*E/comprimento[i][j]) * math.cos(np.radians(theta[i][j]))*math.sin(np.radians(theta[i][j]))
            Kij[2][3] = Kij[3][2]

            K[(2*(i+1))-2][(2*(i+1))-2] += Kij[0][0]
            K[(2*(i+1))-2][2*(i+1)-1]   += Kij[0][1]
            K[2*(i+1)-1][(2*(i+1))-2]   += Kij[1][0]
            K[2*(i+1)-1][2*(i+1)-1]     += Kij[1][1]
```

Figura 7 - Matriz de rigidez K para valores fora do intervalo

Calculamos a matriz M conforme explicitado no enunciado, ou seja, uma matriz 24x24 diagonal com entradas

$$M_{2i-1,2i-1} = m_i,$$
$$M_{2i,2i} = m_i,$$

para $i = 1, 2, \ldots, 12.$

O Ktil é também calculado de forma explícita ao descrito no enunciado, sendo

$$\tilde{K} = M^{-\frac{1}{2}} K M^{-\frac{1}{2}}$$

Para calcularmos $M^{\frac{-1}{2}}$ utilizamos o conceito de potenciação de matrizes, do tipo:

$$A^n = BC^nB^t$$

No caso específico, B é a matriz de autovetores, enquanto C é a matriz de autovalores e M=A. Para descobrirmos os autovalores e os autovetores, foi utilizado o algoritmo QR na matriz M.

Após isso, já foram obtidos dados suficientes para a resolução do problema proposto. Portanto, dada a seguinte EDO:

$$Mx'' + Kx = 0$$

E sua decomposição:

$$Kz = \omega^2 Mz$$

É possível deduzir as seguintes expressões:

$$x(t) = ze^{iwt} = z(Acos\omega t + Bisen\omega t)$$

$$x'(t) = - z\omega(Acos\omega t - Bisen\omega t)$$

$$x''(t) = - z\omega^2(Acos\omega t + Bisen\omega t) = - \omega^2 x(t)$$

$x(0) = zA = ze^{iw0} = z$ -> Logo, A=1

$x'(0) = z\omega Bi = 0$ -> Logo, B=0

e

$$z = M^{\frac{-1}{2}} y$$

Ademais, temos que os autovalores de Ktil são ω^2 e os autovetores são iguais a y, segundo a seguinte expressão:

$$\tilde{K}y = \omega^2 y$$

Logo, com o auxílio dessas informações, pode-se obter a frequência ω e os modos de vibração z.

```
# Definindo a matriz M
M = np.zeros((n,n))
for i in range(n_nfixos):
    mi = 0.00
    for j in range(n_nos):
        mij = secao*comprimento[i][j]*rho
        mi += (mij/2)
    M[(2*(i+1))-2][(2*(i+1))-2] = mi
    M[(2*(i+1))-1][(2*(i+1))-1] = mi

# Definindo a matriz K~
ident = np.eye(n)
a = np.zeros(n)
b = np.zeros(n-1)
for i in range(n):
    a[i] = M[i][i]
g = b.copy()

# Calculando M^(-1/2)
B, Lambda, k = QR_deslocamento(a, b, g, n, 10**(-6), ident)
C = np.zeros((n,n))
for i in range(n):
    C[i][i] = Lambda[i]**(-0.5)

BC = np.matmul(B,C)
Minv = np.matmul(BC, np.transpose(B)) #Minv = M^(-1/2)

MK = np.matmul(Minv,K)
Ktil = np.matmul(MK, Minv)
```

Figura 8 - Matrizes M, Ktil e $M^{\frac{-1}{2}}$

```
freq = np.zeros(n)
for i in range(n):
    freq[i] = math.sqrt(abs(Autovalores[i]))

minfreqs = np.zeros(5)
for i in range(5):
    minimo = min(freq)
    minfreqs[i] = minimo
    index = np.argwhere(freq==minimo)
    freq = np.delete(freq, index)
print("   5 menores frequencias (rad/s): ", minfreqs)
```

Figura 9 - 5 menores frequências

```
Y = np.zeros((n,5))
for j in range(5):
    value = minfreqs[j]
    pos = np.where(minfreqs == value)
    for i in range(n):
        Y[i][j] = V[i][pos]

Z = np.zeros((n,5))
autovector = np.zeros(n)
for j in range(5):
    for i in range(n):
        autovector[i] = Y[i][j]
    autovectorT = np.transpose(autovector)
    vector = np.matmul(Minv, autovectorT)
    for i in range(n):
        Z[i][j] = vector[i]

print("\n")
print("   Matriz dos modos de vibração: ")
print("   (Cada coluna refere-se a uma das 5 menores frequência em ordem crescente da esquerda para direita)")
print("   (Cada linha par (0, 2, 4, ...) refere-se ao deslocamento horizontal, enquanto cada linha ímpar (1, 3, 5, ...) refere-se ao vertical)")
print("\n")
with np.printoptions(threshold=sys.maxsize, linewidth = 200):
    print("   Z = ")
    print(Z)
```

Figura 10 - Modos de vibração

OBS* Há uma grande parte comentada abaixo do código. Ela serve para a impressão dos gráficos dos deslocamentos horizontais dos nós em função do tempo.

5. Questões

a)

```
   Matriz A:
[[ 2.00000   4.00000   1.00000   1.00000 ]
 [ 4.00000   2.00000   1.00000   1.00000 ]
 [ 1.00000   1.00000   1.00000   2.00000 ]
 [ 1.00000   1.00000   2.00000   1.00000 ]]


   Aplicacao de Householder:
   Matriz T = H*A*Ht:
[[ 2.00000   -4.24264   0.00000   0.00000 ]
 [ -4.24264   3.00000   1.41421   0.00000 ]
 [ 0.00000   1.41421   2.00000   -0.00000 ]
 [ 0.00000   0.00000   -0.00000   -1.00000 ]]


   Matriz Ht:
[[ 1.00000   0.00000   0.00000   0.00000 ]
 [ 0.00000   -0.94281   0.33333   0.00000 ]
 [ 0.00000   -0.23570   -0.66667   -0.70711 ]
 [ 0.00000   -0.23570   -0.66667   0.70711 ]]


   Aplicacao do algoritmo QR na matriz T:
   Autovetores de T:
[[ 0.63246   0.70711   0.31623   0.00000 ]
 [ 0.63246   -0.70711   0.31623   0.00000 ]
 [ 0.31623   0.00000   -0.63246   -0.70711 ]
 [ 0.31623   0.00000   -0.63246   0.70711 ]]


   Autovalores de T:  [ 7.00000   -2.00000   2.00000   -1.00000 ]


   Matriz Ht*V (autovetores ortogonais de A):
[[ 0.63246   0.70711   0.31623   0.00000 ]
 [ -0.49088   0.66667   -0.50896   -0.23570 ]
 [ -0.58350   0.16667   0.79432   -0.02860 ]
 [ -0.13628   0.16667   -0.10011   0.97140 ]]


   Forma diagonal semelhante:
   Matriz V*Lambda*Vt:
[[ 2.00000   4.00000   1.00000   1.00000 ]
 [ 4.00000   2.00000   1.00000   1.00000 ]
 [ 1.00000   1.00000   1.00000   2.00000 ]
 [ 1.00000   1.00000   2.00000   1.00000 ]]
```

```
   Restante da resolucao do exercicio a):


   Autovalores esperados: (7, 2, -1 e -2)
   Autovalores obtidos pela matriz T (H*A*Ht):  [ 7.00000   -2.00000   2.00000   -1.00000 ]


   Verificacao se A*v = Lambda*v:
   v =  [ 0.63246   0.63246   0.31623   0.31623 ]
   Lambda =  7.000000000000001
   A*v    =    [ 4.42719   4.42719   2.21359   2.21359 ]
   Lambda*v =  [ 4.42719   4.42719   2.21359   2.21359 ]


   v =  [ 0.70711   -0.70711   0.00000   0.00000 ]
   Lambda =  -1.9999999999999978
   A*v    =    [ -1.41421   1.41421   0.00000   0.00000 ]
   Lambda*v =  [ -1.41421   1.41421   -0.00000   -0.00000 ]


   v =  [ 0.31623   0.31623   -0.63246   -0.63246 ]
   Lambda =  1.9999999999999993
   A*v    =    [ 0.63246   0.63246   -1.26491   -1.26491 ]
   Lambda*v =  [ 0.63246   0.63246   -1.26491   -1.26491 ]


   v =  [ 0.00000   0.00000   -0.70711   0.70711 ]
   Lambda =  -1.0000000000000002
   A*v    =    [ -0.00000   -0.00000   0.70711   -0.70711 ]
   Lambda*v =  [ -0.00000   -0.00000   0.70711   -0.70711 ]


   Verificacao se V é ortogonal (V * Vt = I) (erro absoluto de 1e-5):
   Matriz V*Vt:
[[ 1.00000   -0.00000   0.00000   0.00000 ]
 [ -0.00000   1.00000   0.00000   0.00000 ]
 [ 0.00000   0.00000   1.00000   0.00000 ]
 [ 0.00000   0.00000   0.00000   1.00000 ]]
   Ela é ortogonal
```

b)

```
   Matriz A:
[[ 20.00000   19.00000   18.00000  ...  3.00000   2.00000   1.00000 ]
 [ 19.00000   19.00000   18.00000  ...  3.00000   2.00000   1.00000 ]
 [ 18.00000   18.00000   18.00000  ...  3.00000   2.00000   1.00000 ]
 ...
 [ 3.00000   3.00000   3.00000  ...  3.00000   2.00000   1.00000 ]
 [ 2.00000   2.00000   2.00000  ...  2.00000   2.00000   1.00000 ]
 [ 1.00000   1.00000   1.00000  ...  1.00000   1.00000   1.00000 ]]


   Aplicacao de Householder:
   Matriz T = H*A*Ht:
[[ 20.00000   -49.69909   0.00000  ...  0.00000   0.00000   0.00000 ]
 [ -49.69909   152.20000   -16.52453  ...  0.00000   0.00000   0.00000 ]
 [ 0.00000   -16.52453   17.02222  ...  0.00000   0.00000   0.00000 ]
 ...
 [ 0.00000   0.00000   0.00000  ...  0.29454   0.01727   0.00000 ]
 [ 0.00000   0.00000   0.00000  ...  0.01727   0.27596   0.00868 ]
 [ 0.00000   0.00000   0.00000  ...  0.00000   0.00868   0.26027 ]]


   Matriz Ht:
[[ 1.00000   0.00000   0.00000  ...  0.00000   0.00000   0.00000 ]
 [ 0.00000   -0.38230   -0.51360  ...  0.00000   -0.00000   0.00000 ]
 [ 0.00000   -0.36218   -0.35141  ...  -0.00000   0.00000   -0.00000 ]
 ...
 [ 0.00000   -0.06036   0.13321  ...  -0.01826   0.23817   0.51803 ]
 [ 0.00000   -0.04024   0.09084  ...  -0.30301   -0.39878   -0.44693 ]
 [ 0.00000   -0.02012   0.04603  ...  0.31466   0.30028   0.26071 ]]


   Aplicacao do algoritmo QR na matriz T:
   Autovetores de T:
[[ 0.31212   0.31029   -0.30663  ...  0.07117   -0.04768   0.02391 ]
 [ 0.31029   0.29396   -0.26217  ...  -0.19873   0.13860   -0.07116 ]
 [ 0.30663   0.26217   -0.17970  ...  0.28502   -0.21659   0.11675 ]
 ...
 [ 0.07117   -0.19873   -0.28502  ...  -0.30664   -0.24840   -0.13860 ]
 [ 0.04768   -0.13859   -0.21659  ...  0.24841   0.17970   0.09425 ]
 [ 0.02391   -0.07117   -0.11676  ...  -0.13860   -0.09424   -0.04768 ]]


   Autovalores de T:  [ 170.40427   19.00810   6.89678  ...  0.26369   0.25596   0.25147 ]


   Matriz Ht*V (autovetores ortogonais de A):
[[ 0.31212   0.31029   -0.30663  ...  0.07117   -0.04768   0.02391 ]
 [ -0.01218   -0.09448   0.19681  ...  -0.01623   -0.04851   0.03525 ]
 [ -0.32272   -0.17531   0.02052  ...  -0.39780   0.24080   -0.00647 ]
 ...
 [ -0.03201   -0.07603   -0.16389  ...  -0.12824   0.10156   -0.08010 ]
 [ 0.14299   -0.11530   0.42372  ...  -0.05672   0.05612   -0.12902 ]
 [ 0.26221   -0.39796   0.00593  ...  0.05246   0.06996   -0.07361 ]]
```

```
   Forma diagonal semelhante:
   Matriz V*Lambda*Vt:
[[ 20.00000   19.00000   18.00000  ...  3.00000   2.00000   1.00000 ]
 [ 19.00000   19.00000   18.00000  ...  3.00000   2.00000   1.00000 ]
 [ 18.00000   18.00000   18.00000  ...  3.00000   2.00000   1.00000 ]
 ...
 [ 3.00000   3.00000   3.00000  ...  3.00000   2.00000   1.00000 ]
 [ 2.00000   2.00000   2.00000  ...  2.00000   2.00000   1.00000 ]
 [ 1.00000   1.00000   1.00000  ...  1.00000   1.00000   1.00000 ]]
```

```
Restante da resolucao do exercicio b):


Autovalores esperados:  [ 170.40427   19.00810   6.89678  ...  0.26369   0.25596   0.25147 ]
Autovalores obtidos:  [ 170.40427   19.00810   6.89678  ...  0.26369   0.25596   0.25147 ]


Verificacao se A*v = Lambda*v:
v =  [ 0.31212   0.31029   0.30663  ...  0.07117   0.04768   0.02391 ]
Lambda =  170.40426750542773
A*v    =    [ 53.18629   52.87417   52.25177  ...  12.12758   8.12481   4.07436 ]
Lambda*v =  [ 53.18629   52.87417   52.25177  ...  12.12758   8.12481   4.07436 ]


v =  [ 0.31029   0.29396   0.26217  ...  -0.19873   -0.13859   -0.07117 ]
Lambda =  19.008099491009183
A*v    =    [ 5.89796   5.58767   4.98342  ...  -3.77746   -2.63442   -1.35280 ]
Lambda*v =  [ 5.89796   5.58767   4.98342  ...  -3.77746   -2.63442   -1.35280 ]


v =  [ -0.30663   -0.26217   -0.17970  ...  -0.28502   -0.21659   -0.11676 ]
Lambda =  6.896784892743416
A*v    =    [ -2.11479   -1.80816   -1.23935  ...  -1.96571   -1.49379   -0.80527 ]
Lambda*v =  [ -2.11479   -1.80816   -1.23935  ...  -1.96571   -1.49379   -0.80527 ]


v =  [ 0.30118   0.21659   0.07117  ...  -0.31212   -0.27440   -0.15962 ]
Lambda =  3.560482807695399
A*v    =    [ 1.07235   0.77117   0.25340  ...  -1.11129   -0.97700   -0.56831 ]
Lambda*v =  [ 1.07235   0.77117   0.25340  ...  -1.11129   -0.97700   -0.56831 ]


v =  [ -0.29396   -0.15962   0.04768  ...  -0.27440   -0.30663   -0.19873 ]
Lambda =  2.1880801951103925
A*v    =    [ -0.64321   -0.34925   0.10433  ...  -0.60041   -0.67094   -0.43483 ]
Lambda*v =  [ -0.64321   -0.34925   0.10433  ...  -0.60041   -0.67094   -0.43483 ]


v =  [ 0.28502   0.09424   -0.15962  ...  -0.17970   -0.31029   -0.23318 ]
Lambda =  1.4939898290587417
A*v    =    [ 0.42581   0.14080   -0.23846  ...  -0.26847   -0.46357   -0.34837 ]
Lambda*v =  [ 0.42581   0.14080   -0.23846  ...  -0.26847   -0.46357   -0.34837 ]


v =  [ -0.27440   -0.02391   0.24841  ...  -0.04768   -0.28502   -0.26217 ]
Lambda =  1.0954523500713775
A*v    =    [ -0.30059   -0.02619   0.27212  ...  -0.05223   -0.31222   -0.28720 ]
Lambda*v =  [ -0.30059   -0.02619   0.27212  ...  -0.05223   -0.31222   -0.28720 ]


v =  [ -0.26217   0.04768   0.30118  ...  -0.09424   0.23318   0.28502 ]
Lambda =  0.8461219550212756
A*v    =    [ -0.22183   0.04034   0.25484  ...  -0.07974   0.19730   0.24116 ]
Lambda*v =  [ -0.22183   0.04034   0.25484  ...  -0.07974   0.19730   0.24116 ]


v =  [ -0.24841   0.11676   0.31029  ...  0.21659   -0.15962   -0.30118 ]
Lambda =  0.6802549888118138
A*v    =    [ -0.16898   0.07943   0.21107  ...  0.14734   -0.10858   -0.20488 ]
Lambda*v =  [ -0.16898   0.07943   0.21107  ...  0.14734   -0.10858   -0.20488 ]
```

```
v =  [ 0.23318   -0.17970   -0.27440  ...  0.29396   -0.07117   -0.31029 ]
Lambda =  0.5647697268334025
A*v    =    [ 0.13170   -0.10149   -0.15497  ...  0.16602   -0.04019   -0.17524 ]
Lambda*v =  [ 0.13170   -0.10149   -0.15497  ...  0.16602   -0.04019   -0.17524 ]


v =  [ -0.21659   0.23318   0.19873  ...  0.31029   0.02391   -0.31212 ]
Lambda =  0.48155512239007514
A*v    =    [ -0.10430   0.11229   0.09570  ...  0.14942   0.01151   -0.15030 ]
Lambda*v =  [ -0.10430   0.11229   0.09570  ...  0.14942   0.01151   -0.15030 ]


v =  [ 0.19873   -0.27440   -0.09424  ...  0.26218   0.11676   -0.30664 ]
Lambda =  0.42003001883383645
A*v    =    [ 0.08347   -0.11526   -0.03958  ...  0.11012   0.04904   -0.12880 ]
Lambda*v =  [ 0.08347   -0.11526   -0.03958  ...  0.11012   0.04904   -0.12880 ]


v =  [ -0.17970   0.30119   -0.02391  ...  0.15961   0.19872   -0.29396 ]
Lambda =  0.3736873637702624
A*v    =    [ -0.06715   0.11255   -0.00894  ...  0.05965   0.07426   -0.10985 ]
Lambda*v =  [ -0.06715   0.11255   -0.00894  ...  0.05965   0.07426   -0.10985 ]


v =  [ -0.15961   0.31211   -0.13859  ...  -0.02391   -0.26218   0.27441 ]
Lambda =  0.33835913531541667
A*v    =    [ -0.05401   0.10561   -0.04689  ...  -0.00809   -0.08871   0.09285 ]
Lambda*v =  [ -0.05401   0.10561   -0.04689  ...  -0.00809   -0.08871   0.09285 ]


v =  [ 0.13859   -0.30663   0.23318  ...  0.11676   -0.30118   0.24841 ]
Lambda =  0.31128881200989345
A*v    =    [ 0.04314   -0.09545   0.07259  ...  0.03635   -0.09375   0.07733 ]
Lambda*v =  [ 0.04314   -0.09545   0.07259  ...  0.03635   -0.09375   0.07733 ]


v =  [ 0.11676   -0.28502   0.29396  ...  -0.23318   0.31212   -0.21659 ]
Lambda =  0.2906095464647803
A*v    =    [ 0.03393   -0.08283   0.08543  ...  -0.06777   0.09070   -0.06294 ]
Lambda*v =  [ 0.03393   -0.08283   0.08543  ...  -0.06777   0.09070   -0.06294 ]


v =  [ -0.09424   0.24841   -0.31212  ...  -0.30118   0.29396   -0.17970 ]
Lambda =  0.2750381894851846
A*v    =    [ -0.02592   0.06832   -0.08584  ...  -0.08284   0.08085   -0.04942 ]
Lambda*v =  [ -0.02592   0.06832   -0.08585  ...  -0.08284   0.08085   -0.04942 ]


v =  [ 0.07117   -0.19873   0.28502  ...  -0.30664   0.24841   -0.13860 ]
Lambda =  0.26369005499954823
A*v    =    [ 0.01877   -0.05240   0.07516  ...  -0.08086   0.06550   -0.03655 ]
Lambda*v =  [ 0.01877   -0.05240   0.07516  ...  -0.08086   0.06550   -0.03655 ]


v =  [ -0.04768   0.13860   -0.21659  ...  -0.24840   0.17970   -0.09424 ]
Lambda =  0.25596443303682964
A*v    =    [ -0.01220   0.03548   -0.05544  ...  -0.06358   0.04600   -0.02412 ]
Lambda*v =  [ -0.01220   0.03548   -0.05544  ...  -0.06358   0.04600   -0.02412 ]
```

```
   v =  [ -0.04768   0.13860   -0.21659  ...  -0.24840   0.17970   -0.09424 ]
   Lambda =  0.25596443303682964
   A*v    =    [ -0.01220   0.03548   -0.05544  ...  -0.06358   0.04600   -0.02412 ]
   Lambda*v =  [ -0.01220   0.03548   -0.05544  ...  -0.06358   0.04600   -0.02412 ]


   v =  [ 0.02391   -0.07116   0.11675  ...  -0.13860   0.09425   -0.04768 ]
   Lambda =  0.251473581911361
   A*v    =    [ 0.00601   -0.01790   0.02936  ...  -0.03486   0.02370   -0.01199 ]
   Lambda*v =  [ 0.00601   -0.01790   0.02936  ...  -0.03486   0.02370   -0.01199 ]


   Verificacao se V é ortogonal (V * Vt = I) (erro absoluto de 1e-5):
   Matriz V*Vt:
[[ 1.00000   0.00000   0.00000  ...  -0.00000   -0.00000   0.00000 ]
 [ 0.00000   1.00000   -0.00000  ...  0.00000   0.00000   0.00000 ]
 [ 0.00000   -0.00000   1.00000  ...  -0.00000   -0.00000   -0.00000 ]
 ...
 [ -0.00000   0.00000   -0.00000  ...  1.00000   0.00000   0.00000 ]
 [ -0.00000   0.00000   -0.00000  ...  0.00000   1.00000   0.00000 ]
 [ 0.00000   0.00000   -0.00000  ...  0.00000   0.00000   1.00000 ]]
   Ela é ortogonal
```

c)

```
   APLICACAO PARA TRELIÇAS PLANAS
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   Número de nós:  14
   Número de nós não fixos:  12
   Número de barras:  28
   Densidade de massa (rho):  7800.0
   Seção transversal (A):  0.1
   Módulo de elasticidade (E):  200000000000.0


   5 menores frequencias (rad/s):  [ 24.59255   92.01244   94.70337   142.80970   150.82213 ]


   Matriz dos modos de vibração:
   (Cada coluna refere-se a uma das 5 menores frequência em ordem crescente da esquerda para direita)
   (Cada linha par (0, 2, 4, ...) refere-se ao deslocamento horizontal, enquanto cada linha ímpar (1, 3, 5, ...) refere-se ao vertical)


   Z =
[[ 1.79474190e-03 -3.45527741e-03  7.22166927e-05 -3.46898371e-04 -1.29277496e-03]
 [-2.74111749e-03  4.71210155e-03 -1.03046416e-04  3.85478734e-04  1.14193695e-03]
 [-3.22124381e-03  3.88101143e-03 -9.91800381e-05  8.28706871e-05 -5.17362873e-04]
 [ 2.91977626e-03 -5.79538292e-04  4.81704018e-05  3.84948829e-04  1.85331807e-03]
 [-4.19094995e-03 -2.40453390e-03 -1.71278900e-05 -7.55755321e-04 -2.03766548e-03]
 [ 4.52162917e-03  3.49065585e-03 -3.87575105e-06  5.85224753e-04  6.54049400e-04]
 [ 1.78966582e-03  1.72812279e-03 -2.79816998e-05 -3.58882914e-04 -2.14433421e-03]
 [-6.39219187e-04 -7.37946789e-04  5.83809153e-05  1.02775054e-03  2.81267317e-03]
 [ 2.98527531e-04  4.15408852e-04 -1.28711633e-04 -1.81835364e-03 -2.91807062e-03]
 [ 1.43107094e-04  2.32130733e-04 -2.46726603e-04 -2.35701965e-03 -1.59443259e-03]
 [ 8.64626241e-05  1.58270373e-04 -1.04938406e-03 -5.33028887e-03  2.33547379e-03]
 [-5.12218123e-05 -7.89592768e-05  1.36684804e-03  4.19024341e-03 -2.71668142e-03]
 [-3.72788566e-05  8.58926606e-06  2.73955877e-03  2.34904799e-03 -2.07958844e-03]
 [-4.93078722e-05  2.14646381e-04  8.35484621e-03 -1.71087840e-03  5.20805834e-04]
 [-1.69446343e-05  8.57629320e-05  3.13868913e-03 -9.37350887e-04  4.96267636e-04]
 [ 5.11248977e-06 -3.08136956e-05 -1.04998121e-03  4.82179568e-04 -4.35065424e-04]
 [ 1.34009124e-06 -8.90689926e-06 -2.91900715e-04  1.66063457e-04 -1.85770013e-04]
 [-1.24447323e-07  9.80694456e-07  2.99479771e-05 -2.70474884e-05  8.01496651e-05]
 [ 1.23395929e-07 -1.10295925e-06 -3.20043932e-05  3.90930237e-05 -1.70760015e-04]
 [-3.69737236e-08  3.79374215e-07  1.04054237e-05 -1.78708160e-05  1.21687587e-04]
 [ 7.92390042e-09 -9.48215699e-08 -2.44376556e-06  6.01523945e-06 -5.97751975e-05]
 [-6.51246583e-10  8.72010714e-09  2.14852474e-07 -6.78789573e-07  8.51189263e-06]
 [-1.10601562e-10  1.60757248e-09  3.83422043e-08 -1.44188943e-07  2.09657925e-06]
 [-7.47182076e-12  1.15831682e-10  2.69345176e-09 -1.15795697e-08  1.87787269e-07]]
```

6. Conclusão

Pode-se concluir que os resultados obtidos por meio do programa são condizentes com o esperado.

Nos itens a) e b) é possível dizer que estão corretos, por baterem com os dados fornecidos pelo enunciado.

No item c) é possível notar que a magnitude dos valores resultantes estão boas, a frequência é alta e o modo de vibração é baixo, o que significa, vibrações rápidas, porém, bem curtas.