ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Exercício-Programa 2

Autovalores e Autovetores de Matrizes Reais Simétricas - O Algoritmo QR

MAP3121 - Métodos Numéricos e Aplicações

Gabriel Boaventura Scholl - 10771218
William Simões Barbosa - 9837646

SÃO PAULO

2021

Introdução

O objetivo deste trabalho é servir de apoio para o Exercício-Programa 2 da disciplina MAP3121 - Métodos Numéricos e Aplicações, sendo um relatório que aborda diversos aspectos, tanto conceituais, quanto práticos, incluindo, ainda, os resultados obtidos e suas respectivas análises. Estão inclusos na pasta compactada, também, o arquivo “LEIA-ME.txt” e o código fonte “EP2.py”.

1. Análise do problema estudado

Agora, nesta segunda parte do conjunto de Exercícios-Programa (EPs), vamos abordar o tópico mais aplicado ao mundo real, com uma aplicação em treliça plana, que traduz um pouco do que é trabalhar com estruturas em diversas áreas, incluindo a área de construção civil, que parece ser o caso estudado, mas que também se estende à análise modal de chassis de carro, máquinas na área de engenharia mecânica e até mesmo estruturas de diversas naturezas e tamanhos.

Desta vez, não vamos trabalhar com matrizes inicialmente tridiagonais, mas sim matrizes simétricas, que vêm dos cálculos de matrizes de rigidez, massa e afins. A partir dessas, se chegarmos a matrizes tridiagonais, podemos aplicar boa parte da teoria desenvolvida com o EP1, chegando a uma resolução do problema de treliças planas e estruturas afins. Para tanto, neste EP2 vamos estudar as transformações de Householder que permitem transformar matrizes simétricas em tridiagonais e, portanto, estabelecer esta ponte.

Finalmente, em cada caso, podem-se aplicar sucessivas multiplicações de matrizes, mas isso não é necessário ao fazer as transformações parte a parte, sendo também computacionalmente mais barato. Com a ligação de problemas mais complexos e a possibilidade de solução, veremos os resultados na seção 3 deste relatório.

A principal tarefa deste exercício-programa consistiu em implementar a transformação de Householder. A imagem abaixo contém algumas anotações que a dupla fez para entender o passo a passo que deveria ser implementado. Com esse passo a passo, não precisaríamos realizar multiplicação de matrizes, que tem um custo computacional muito elevado quando comparamos com os produtos escalares que realizamos.

a transformação H_{w_1} não altere a primeira linha de A e leve o vetor a_1, correspondente à primeira coluna de A, em um vetor com elementos nulos da posição 3 em diante

- isso é feito definindo

$$w_1 = \tilde{a}_1 + \delta \frac{\|\tilde{a}_1\|}{\|e_2\|} e_2 = \tilde{a}_1 + \delta \|\tilde{a}_1\| e_2 ,$$

o δ será escolhido como o sinal de $A_{2,1}$

• Após a aplicação de Householder H_{w_1} à matriz A, obteremos:

$$H_{w_1} A = \begin{bmatrix} x & x & x & \cdots & x \\ x & x & x & \cdots & x \\ 0 & x & x & \cdots & x \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & x & x & \cdots & x \end{bmatrix}$$

A é uma matriz simétrica

$$A = \begin{bmatrix} A_{1,1} & \alpha^T \\ \alpha & A_1 \end{bmatrix}$$

onde α representa a primeira coluna de A da segunda linha em diante e A_1 é a submatrix de A da segunda linha e coluna em diante

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

a_i e 'e' com tamanho $n-i$

$e = (1, 0, 0, \ldots, 0)$

Vejamos um exemplo:

$$A = \begin{bmatrix} 2 & -1 & 1 & 3 \\ -1 & 1 & 4 & 2 \\ 1 & 4 & 2 & -1 \\ 3 & 2 & -1 & 1 \end{bmatrix}$$

$$\bar{w}_1 = (-1-\sqrt{11}, 1, 3)^T = (-4{,}3166, 1, 3)^T$$

$$H_{w_1} \cdot A = \begin{bmatrix} 2 & -1 & 1 & 3 \\ 3{,}3166 & 2{,}7136 & -1{,}5076 & 0 \\ 0 & 3{,}16030 & 3{,}2758 & -0{,}53667 \\ 0 & 0{,}80906 & 2{,}8277 & 2{,}3900 \end{bmatrix}$$

$n = 4$
$n-2$ etapas
2 etapas

$$H_{\bar{w}_1} \bar{a}_1 = \bar{a}_1 - 2 \cdot \frac{\bar{w}_1 \cdot a_1}{\bar{w}_1 \cdot \bar{w}_1} \bar{w}_1$$

$\bar{a}_1 = (-1, 1, 3)$

$\bar{w}_1 = (-1-\sqrt{11}, 1, 3)$

$-(1+\sqrt{11})^2 = 1 + 2\sqrt{11} + 11$

$$H_{\bar{w}_1} \bar{a}_1 = (-1,1,3) - \frac{2 \cdot (1+\sqrt{11} + 1 + 9)}{(1 + 2\sqrt{11} + 11 + 1 + 9)} (-1-\sqrt{11}, 1, 3)$$

$$= (-1,1,3) - \frac{2 \cdot (11 + \sqrt{11})}{22 + 2\sqrt{11}} (-1-\sqrt{11}, 1, 3) \Rightarrow$$

$$\Rightarrow H_{\bar{w}_1} \bar{a}_1 = (-1,1,3) - (-1-\sqrt{11}, 1, 3) =$$

$$\Rightarrow H_{\bar{w}_1} \bar{a}_1 = (\sqrt{11}, 0, 0) = (3{,}3166, 0, 0)$$

$\bar{a}_2 = (1, 4, 2)$; $\bar{a}_3 = (4, 2, -1)$; $\bar{a}_4 = (2, -1, 1)$

$H_{\bar{w}_1} \bar{a}_2 = (2{,}7136,\ 3{,}16030,\ 0{,}80906)$

$H_{\bar{w}_1} \bar{a}_3 = (-1{,}5076,\ 3{,}2758,\ 2{,}8276)$

$H_{\bar{w}_1} \bar{a}_4 = (0,\ -0{,}53667,\ 2{,}3900)$

$H_{w_1} A H_{w_1}$

2. Transcrição do código

```python
"""
EP2 - Numérico
Alunos:
Gabriel Boaventura Scholl, NUSP: 10771218
William Simões Barbosa, NUSP: 9837646
"""
import matplotlib.pyplot as plt
import numpy as np # Usar apenas para aritmetica de vetores, matrizes, leitura e escrita de dados.
import math

def vetoresAlfaBetaGama(matrizA): #Função que recebe uma matriz e retorna os valores das três diagonais (Alfa, Beta e Gama)
    alfa = []
    beta = []
    gama = []

    for i in range(len(matrizA)):
        for j in range(len(matrizA[0])):
            if (i == j):
                alfa.append(matrizA[i][j])
            elif (i == j + 1):
                beta.append(matrizA[i][j])
            elif (j == i + 1):
                gama.append(matrizA[i][j])
    return alfa, beta, gama

def rotacaoDeGivens(k, n, ck, sk):
    #Realizando as rotações de Givens para as Qk operações
    Qk = np.identity(n, dtype = float)
    Qk[0][0] = ck
    Qk[0][1] = -sk
    Qk[1][0] = sk
    Qk[1][1] = ck

    for i in range(k-1):
        Qk[i+2][i+2] = Qk[i][i]
        Qk[i][i] = 1

        Qk[i+1][i+2] = Qk[i][i+1]
        Qk[i][i+1] = 0

        Qk[i+2][i+1] = Qk[i+1][i]
```

```python
            Qk[i+1][i] = 0

    return Qk

def arrumaDimensao(Qk, nOriginal):
      #Função que recebe a matriz Qk e, como esta matriz pode ter dimensão menor que n na decomposição QR com deslocamento espectral,
     #se certifica que a matriz Q usada para o cálculo de V[k+1] sempre tem dimensão n.
      #Por isso carregamos o valor da dimensão original da matriz A: 'nOriginal'
    Qnova = np.identity(nOriginal, dtype = float)
    for i in range(len(Qk)):
        for j in range(len(Qk[0])):
            Qnova[i][j] = Qk[i][j]
    return Qnova

def decomposicaoQR(matrizA, n):
    #Dada uma matriz de dimensão n, obter sua decomposição Q, R
    I = np.identity(n, dtype = float)
    R = np.copy(matrizA)
    Q = np.copy(I)
    for k in range(1, n): #Transformações Qk, com 1 =< k <= n-1
        alfa, beta, gama = vetoresAlfaBetaGama (R)
        ck = alfa[k-1]/math.sqrt(alfa[k-1]**2 + beta[k-1]**2)
        sk = -beta[k-1]/math.sqrt(alfa[k-1]**2 + beta[k-1]**2)
        Qi = rotacaoDeGivens (k, n, ck, sk)
        R = Qi @ R
        Q = Q @ Qi.T

    return Q, R

def reduzDimensao(An):
      #Função responsável por diminuir a dimensão da matriz Ak caso encontre um autovalor.
    n = len(An)
    Anova = np.identity(n-1, dtype = float)
    for i in range(n-1):
        for j in range(n-1):
            Anova[i][j] = An[i][j]
    return Anova

def devolveAutovalores (matrizA):
```

```
    #Função usava para devolver os autovalores da matriz Ak após todas
as iterações
    autovalores = []
    for i in range(len(matrizA)):
        autovalores.append(matrizA[i][i])
    return autovalores

def algoritmoQR_deslocamento(A0, V0, n, erro):
     #Função que recebe a matriz A e o erro passado no enunciado, e
retorna uma lista de autovalores,
     #Retorna também a matriz de autovetores. Os autovetores estão nas
colunas da matriz.
      #A primeira coluna da matriz devolvida corresponde ao primeiro
autovetor, associado ao primeiro autovalor da lista de autovalores.

    Ak = np.copy(A0)
    Vk = np.copy(V0)
    alfa, beta, gama = vetoresAlfaBetaGama(Ak)
     listaAutoValores = [] #vou salvar todos os autovalores conforme for
encontrando
    mik = 0
     nOriginal = n #Vou salvar a dimensão original da matriz para obter
a matriz de autovetores
    m = n
    k = 0 #Calcular o número de passos
    #print("Ak: \n", Ak)
     while (m >= 2): #Vou realizando o processo até a dimensão da matriz
ser 2, sempre que beta[n-1] < erro, diminuir a dimensão
        I = np.identity(n, dtype = float)
        if (k > 0):
            #Cálculo de mik pela heurística de Wilkinson
            dk = (alfa[n-2]-alfa[n-1])/2
            if (dk >= 0):
                sgn = 1
            else:
                sgn = -1
                mik = alfa[n-1] + dk - sgn * math.sqrt(dk**2 +
beta[n-2]**2)

            Qk, Rk = decomposicaoQR (Ak - mik * I, n) #Realização da
decomposição QR com a matriz A atual (2 =< dimensão <= n)
         Ak = Rk @ Qk + mik * I #Dados Qk e Rk da decomposição obtida,
calcular A[k+1], usando mik
```

```
        #print("Ak: \n", Ak)
         Qk = arrumaDimensao(Qk, nOriginal) #Se dimensão de A for menor que n, preciso arrumar a dimensão de Qk para calcular V[k+1]
        Vk = Vk @ Qk #Cálculo de V[k+1] a partir de V[k] e da matriz Q
          alfa, beta, gama = vetoresAlfaBetaGama(Ak) #Obtendo os novos alfa, beta e gama.

        k += 1

           if (abs(beta[n-2]) < erro): #Condicional que verifica se o último valor de beta é menor que o erro pedido em enunciado.
             listaAutoValores.append(alfa[n-1]) #Se este último beta for menor, significa que o último alfa é uma aproximação para
                   #um autovalor. Então salvo este valor na lista de autovalores que será retornada.
            m -= 1
             n -= 1 #Caso este último beta seja menor que o erro dado, posso diminuir a dimensão da matriz Ak para aumentar a eficiência
             #do algoritmo. Este detalhe faz o número de passos reduzir consideravelmente.
             Ak = reduzDimensao(Ak) #Reduzindo a dimensão da matriz para continuar buscando os autovalores e autovetores.

            alfa, beta, gama = vetoresAlfaBetaGama(Ak) #Atualizando as listas de alfa, beta e gama para a dimensão n atual
    #print("Passos: ", k)
     autovalores = devolveAutovalores(Ak) #Pegando o último autovalor que faltava, todos os demais já estão na lista 'listaAutoValores'
    for i in range(len(listaAutoValores)-1, -1, -1):
             autovalores.append(listaAutoValores[i]) #Aqui simplesmente invertemos os autovalores da lista 'listaAutovalores'
                                                   #Dessa forma, a ordem de autovalores corresponde à ordem de autovetores da matriz Vk

    return autovalores, Vk

def devolve_e(d): #Devolve o vetor 'e' de dimensão 'd'
    e = [1]
    for i in range(d-1):
        e.append(0)
    return e

def norma(vetor):
```

```python
    # Calcula a norma de um vetor.
    norma = 0
    for el in vetor:
        norma += el**2
    return math.sqrt(norma)

def devolve_wi(matrizA, n, i): #Dada a matriz A, esta função retorna o
vetor wi
    wi = []
    ai = []
     for linha in range(i, n): #Se i for igual a 1, vou calcular w1, e
pra isso preciso do vetor a1, que está na coluna 0 da matriz A
        ai.append(matrizA[linha][i-1])
    #print(ai)
    delta = 1
    if (ai[0] < 0):
        delta = delta * (-1)
    e = devolve_e(n-i)

    for it in range(len(ai)): #it = iterador
        wi.append(ai[it] + delta*e[it]*norma(ai))
    #print("w%d: "%i, wi)
    return wi

def devolve_aj(matrizA, n, i, j):
    aj = []
    for coluna in range(i, n):
        aj.append(matrizA[j][coluna])
    return aj

def devolve_ai(matrizA, n, i, j): #Dada a matriz A, esta função retorna
o vetor ai da coluna j, de dimensão n - i
    ai = []
    for linha in range(i, n):
        ai.append(matrizA[linha][j])
    return ai

def somaVetores (vetorA, vetorB): #Retorna a soma de dois vetores A e B
quaisquer
    vetorSoma = []
    for it in range(len(vetorA)):
        vetorSoma.append(vetorA[it]+vetorB[it])
    return vetorSoma
```

```python
def produtoPorEscalar (escalar, vetor): #Retorna o produto de um vetor A qualquer por um escalar
    novoVetor = []
    for it in range(len(vetor)):
        novoVetor.append(escalar*vetor[it])
    return novoVetor

def produtoEscalar (vetorA, vetorB): #Retorna o produto escalar entre dois vetores A e B quaisquer
    escalar = 0
    for it in range(len(vetorA)):
        escalar += vetorA[it]*vetorB[it]
    return escalar

def aplicaColunaNaMatriz(matriz, resultadoColuna, i, j): #Substituindo a coluna calculado na matriz
    for linha in range(i, len(matriz)):
        matriz[linha][j] = resultadoColuna[linha-i]

def aplicaLinhaNaMatriz(matriz, resultadoLinha, i, j):
    for coluna in range(i, len(matriz)):
        matriz[j][coluna] = resultadoLinha[coluna-i]

def multiplica_H_esquerda (matriz, n, wi, i): #Primeiro acha o valor de ai (vetor de cada coluna i da matriz)
     #Em posse do vetor ai, realiza a transformação de Householder de ai no vetor wi
    for j in range(i-1, n):
        aj = devolve_ai(matriz, n, i, j)
        #print("a%d"%j, aj)
        escalarWX = produtoEscalar(wi, aj)
        escalarWW = produtoEscalar(wi, wi)
        coeficiente = -2 * (escalarWX/escalarWW)
        multiploW = produtoPorEscalar(coeficiente, wi)
        resultadoColuna = somaVetores(aj, multiploW)
        aplicaColunaNaMatriz(matriz, resultadoColuna, i, j)

def multiplica_H_direita (matriz, n, wi, i): #Primeiro acha o valor de aj (vetor de cada linha j da matriz, exceto a 1a)
     #Com o vetor aj em mãos, aplicar a transformação de Householder de aj no vetor wi
```

```
    #Nesta tranformação, não há a necessidade de pegar a1 (vetor da primeira linha), pois a matriz é simétrica
    for it in range(i, n):
            matriz[i-1][it] = matriz[it][i-1] #Pra não aplicar a transformação na primeira linha, pois a matriz é simétrica

    for j in range(i, n):
        aj = devolve_aj(matriz, n, i, j)
        #print("a%d"%j, aj)
        escalarWX = produtoEscalar(wi, aj)
        escalarWW = produtoEscalar(wi, wi)
        coeficiente = -2 * (escalarWX/escalarWW)
        multiploW = produtoPorEscalar(coeficiente, wi)
        resultadoLinha = somaVetores(aj, multiploW)
        aplicaLinhaNaMatriz(matriz, resultadoLinha, i, j)

def achaH_transposto (matriz, n, wi, i): #Aplicando a transformação de householder para achar a matriz H transposta

    for j in range(n):
        aj = devolve_aj(matriz, n, i, j)
        #print("a%d"%j, aj)
        escalarWX = produtoEscalar(wi, aj)
        escalarWW = produtoEscalar(wi, wi)
        coeficiente = -2 * (escalarWX/escalarWW)
        multiploW = produtoPorEscalar(coeficiente, wi)
        resultadoLinha = somaVetores(aj, multiploW)
        aplicaLinhaNaMatriz(matriz, resultadoLinha, i, j)

def devolve_TridiagonalSimetrica(matrizA, n): #Aplicando a transformação de Householder
    matrizT = np.copy(matrizA)
    HT = np.identity(n, dtype = float) #Inicializando H transposto como a identidade

    for i in range(1, n-1): #Número de etapas até obtermos a matriz tridiagonal simétrica (n-2)
        wi = devolve_wi(matrizT, n, i)
        #print("w%d: "%i, wi)
        multiplica_H_esquerda (matrizT, n, wi, i)
        #print(matrizT)
        multiplica_H_direita (matrizT, n, wi, i)
        #print(matrizT)
```

```
        achaH_transposto (HT, n, wi, i)
        #print(I)

    return matrizT, HT

def devolve_autovetores(matriz, coluna):
    lista_autovetores = []
    for i in range(len(matriz)):
        lista_autovetores.append(matriz[i][coluna])
    return lista_autovetores

def rotinaDeFormacao(barra, K, M, ro, area, elasticidade):
    # Esta função coloca as contribuições nas matrizes K e M.
    # Primeiro definimos os valores.
    # print(f'barra: {barra}')
    i = barra[0]
    j = barra[1]
    angulo = barra[2]
    angulorad = math.radians(angulo)
    comprimento = barra[3]
    c = math.cos(angulorad)
    s = math.sin(angulorad)
    senosEcossenos = np.array([[c**2,c*s,-c**2,-c*s],
                               [c*s,s**2,-c*s,-s**2],
                               [-c**2,-c*s,c**2,c*s],
                               [-c*s,-s**2,c*s,s**2]])

     # Então adicionamos as contribuições nas posições correspondentes
(matriz de rigidez total).
    index1 = -1
    index2 = -1
    for a in [2*i-1, 2*i, 2*j-1, 2*j]:
        index1 += 1
        index2 = -1
        for b in [2*i-1, 2*i, 2*j-1, 2*j]:
            index2 += 1
            a, b = int(a), int(b)
            # print(f'Calculating K[{a-1}][{b-1}]')
            if a < 25 and b < 25:
                                                       K[a-1][b-1]    +=
(area*elasticidade/comprimento)*(senosEcossenos[index1][index2])
            else:
                pass
```

```
    # Adiciona as contribuições às massas dos nós extremos da barra.
    for (c,d) in [(2*i-1,2*i-1),(2*i,2*i),(2*j-1,2*j-1),(2*j,2*j)]:
        c, d = int(c), int(d)
        # print(f'Calculating M[{c-1}][{d-1}]')
        if c < 25 and d < 25:
            M[c-1][d-1] += ro*comprimento*area/2
    return K, M

def matrixInverter(M):
     # Só funciona se a matriz for diagonal, que é o caso da matriz M de
massas.
    M_inv = np.zeros_like(M)
    for i in range(len(M)):
        for j in range(len(M[0])):
            if (M[i][j] != 0):
                M_inv[i][j] = 1/M[i][j]

    return M_inv

def devolveIndices(frequencias):
    cincoMenoresFrequencias = []
      ordenaFrequencias  =  []  #criando  uma  cópia  para  não  bagunçar  os
indices de 'frequencias', tbm estou salvando os índices
    for i in range(len(frequencias)):
        ordenaFrequencias.append([frequencias[i], i])
        #ordenaFrequencias.append(i)

    for i in range(len(frequencias)):
        for j in range(i, len(frequencias)):
            if (ordenaFrequencias[j][0] < ordenaFrequencias[i][0]):
                tempValor = ordenaFrequencias[j][0]
                tempIndice = ordenaFrequencias[j][1]
                ordenaFrequencias[j][0] = ordenaFrequencias[i][0]
                ordenaFrequencias[j][1] = ordenaFrequencias[i][1]
                ordenaFrequencias[i][0] = tempValor
                ordenaFrequencias[i][1] = tempIndice

    print("Menores frequências:\n")
     for i in range(5): #Quero as 5 menores frequencias, de acordo com o
enunciado
                cincoMenoresFrequencias.append([ordenaFrequencias[i][0],
ordenaFrequencias[i][1]])
```

```
        print(cincoMenoresFrequencias[i][0])
    print("Índices da menores frequências, na ordem:\n")
     for i in range(5): #Quero as 5 menores frequencias, de acordo com o
enunciado
        print(cincoMenoresFrequencias[i][1])

    return cincoMenoresFrequencias

def devolveModoDeVibracao(modosDeVibracao, indice):
    modoAtual = []
    for i in range(len(modosDeVibracao)):
        modoAtual.append(modosDeVibracao[i][indice])

    return modoAtual

def main():
    print("1 - Teste da aplicação da transformação de Householder")
    print("2 - Teste da treliça")
     decisao_inicial1 = int(input("Digite o número de teste que gostaria
de fazer: "))

     if(decisao_inicial1 == 1): #Teste da aplicação da transformação de
Householder
        n = 0 #Inicializando a dimensão n da matriz
        matrizA = [] #Inicializando a matriz A

        print("\n1 - Usar para teste o arquivo 'input-a' ou 'input-b'")
          print("2 - Digitar manualmente os valores de uma matriz real
simétrica A")
           decisao_inicial2 = int(input("Digite o número do teste que
gostaria de fazer: "))

         if (decisao_inicial2 == 1): #Usar os arquivos prontos 'input-a'
ou 'input-b'
            arquivo = ""
            print("\n1 - Usar o arquivo 'input-a'")
            print("2 - Usar o arquivo 'input-b'")
              decisao_inicial3 = int(input("Digite 1 para 'input-a' ou 2
para 'input-b': "))

            if (decisao_inicial3 == 1): #Testes com o arquivo 'input-a'
                arquivo = open("input-a", "r")
```

```
                elif (decisao_inicial3 == 2): #Testes com o arquivo
'input-b'
                arquivo = open("input-b", "r")

               conteudo = arquivo.readlines() #Extraindo todo conteúdo do
arquivo escolhido
            arquivo.close()
            n = int(conteudo[0])
            #print("Número de linhas: ", n)
               for i in range(1, n+1): #Extraindo os valores em si da
matriz A
                linha = conteudo[i].split()
                for j in range(n):
                    linha[j] = float(linha[j])
                matrizA.append(linha)
            print("\nMatriz A:")
            print(np.array(matrizA)) #Matriz A OK. Falta transformar em
ARRAY DE NUMPY

        elif (decisao_inicial2 == 2): #Digitar manualmente os valores
de uma matriz real simétrica A
             n = int(input("\nDigite o valor da dimensão n da matriz A
simétrica: "))
            print("\nEntre com os valores reais da matriz A")
            for i in range(n):
                print("Linha ", i+1)
                linha = []
                for j in range(n):
                        print("Digite o valor da coluna %d: " %(j+1),
end="")
                    valor = float(input())
                    linha.append(valor)
                matrizA.append(linha)
            print("\nMatriz A:")
            print(np.array(matrizA)) #Inicialização da matriz A OK.

                      print("\n#########################   RESULTADOS
##########################\n")
        T, HT = devolve_TridiagonalSimetrica(matrizA, n)
        print("\nA matriz tridiagonal simétrica é: \n", T)
        print("\nA matriz H transposta é: \n", HT)
        print()
```

```
        #Aplicando o Algoritmo QR com deslocamento
        erro = 0.000001
        autovalores, Vk = algoritmoQR_deslocamento(T, HT, n, erro)
        print("\nAutovalores encontrados: \n", autovalores)
        print("\nAutovetores encontrados: \n", Vk)
        print("\n########################## VERIFICAÇÕES PEDIDAS ##########################\n")

        print("Verificando se A * v = lambda * v, para cada autovalor lambda e seu autovetor v correspondente\n")
        for i in range(len(autovalores)):
            print("Verificação para o %d° autovalor:"%(i+1))
            autovetor = devolve_autovetores(Vk, i)
            autovalor = autovalores[i]
            produtoAV = np.array(matrizA) @ autovetor
            produtolambdaV = produtoPorEscalar(autovalor, autovetor)
            print("\nProduto A * V:     ", produtoAV)
            print("Produto lambda * v: ", produtolambdaV, "\n")

        print("\nVerificando se a matriz formada pelos autovetores é ortogonal\n")
        print("Inversa da matriz formada pelos autovetores:\n", np.linalg.inv(Vk))
        print("Transposta da matriz formada pelos autovetores:\n", Vk.T)


    elif(decisao_inicial1 == 2): #Teste da treliça
        # Extraindo as constantes do arquivo input-c.
        arquivo = open("input-c", "r")
        conteudo = arquivo.readlines() #Extraindo todo conteúdo do arquivo escolhido
        arquivo.close()
        numNos, nosLivres, numBarras = conteudo[0].split()
        ro, area, elasticidade = conteudo[1].split()
        area = float(area)
        elasticidade = int(elasticidade) * 10**9
        ro = int(ro)
        numNos, nosLivres, numBarras = int(numNos), int(nosLivres), int(numBarras)
        # print(f'numNos: {numNos}')
        # print(f'nosLivres: {nosLivres}')
        # print(f'numBarras: {numBarras}')
```

```
        # print(f'ro: {ro}')
        # print(f'area: {area}')
        # print(f'elasticidade: {elasticidade}')

        # Extraindo as informações das barras do arquivo input-c.
        counter = 0
        for el in conteudo: counter += 1
        # print(f'counter: {counter}')
        barras = []
           for i in range(2, counter): #Extraindo os valores em si da
matriz A ##############################MUDEI AQUI era counter -1
            linha = conteudo[i].split()
            for j in range(4):
                if j < 2:
                    linha[j] = int(linha[j])
                else:
                    linha[j] = float(linha[j])
            barras.append(linha)
        barras = np.array(barras)
        # print(f'barras:\n{barras}')

         # Procedendo para a definição e montagem das matrizes K e M, em
especial.
           K = np.zeros((nosLivres*2,nosLivres*2)) # Matriz de rigidez
##################MUDEI AQUI era 24, tentando generalizar
            M = np.zeros((nosLivres*2,nosLivres*2)) # Matriz de massas
##################MUDEI AQUI era 24, tentando generalizar

        # Forma as matrizes de rigidez e de massas
        for barra in barras:
                    K, M = rotinaDeFormacao(barra, K, M, ro, area,
elasticidade)
         #print(f'K (matriz de rigidez total):\n{K}') #SE DESEJAR VER A
MATRIZ DE RIGIDEZ TOTAL, DESCOMENTAR ESTA LINHA
        # print(f'M:\n{M}')

        # Faz o calculo de K_til, crucial para a resolução.
                      M_inv = np.zeros((nosLivres*2, nosLivres*2))
##################MUDEI AQUI era 24, tentando generalizar
        for i in range(len(M)):
            for j in range(len(M[0])):
                M_inv[i][j] = math.sqrt(M[i][j])
        M_inv = matrixInverter(M_inv)
```

```
        K_til = M_inv @ K @ M_inv
        # print(f'K_til:\n{K_til}')

           # Como K_til é simétrica, podemos triagonalizá-la aplicando
Householder.
        T, HT = devolve_TridiagonalSimetrica(K_til, len(K_til))

         # E agora achar os autovalores e autovetores com o algoritmo QR
do EP1.
            autovalores, autovetores = algoritmoQR_deslocamento(T, HT,
len(K_til), 10**(-15))
        # print(f'autovalores (ref. a frequências):\n{autovalores}')
                          #   for   i   in   range(len(autovalores)):
print(f'{math.sqrt(autovalores[i])}')
        #print("\nOs autovalores encontrados são:\n", autovalores)
           indicesModos = sorted(range(len(autovalores)), key = lambda
sub: autovalores[sub])[:5]

        frequencias = []
        for av in autovalores:
            frequencias.append(math.sqrt(av))

            #print("\nAs frequências de vibração encontradas são:\n",
frequencias)
           #print("Os índices dos menores modos de vibração são: \n",
indicesModos)
        #print(f'\nModos principais:\n')
        # for i in indicesModos: print(f'{autovetores[i]}')
        # print(f'autovetores:\n{autovetores}')

        # Cálculo dos modos de vibração da treliça.
        y = autovetores
        z = M_inv @ y
        modosDeVibracao = z
           print("\n5 menores frequências e seus respectivos modos de
vibração: \n")
        #print(modosDeVibracao)
        #print()
        for coluna in indicesModos:
            #print("Indice: ", coluna)
            print(f'\nFrequências (Hz):')
            print(f'{frequencias[coluna]}')
```

```
            print(f'\nModo de vibração:')
            print(devolveModoDeVibracao(modosDeVibracao, coluna))

        print()
        verMatriz = int(input("Se deseja ver a matriz de rididez total,
digite 1, se não, digite 0: "))
        if (verMatriz == 1):
            print("\nMatriz de rigidez total:\n")
            print(K)


main()
```

3. Resultados e análises

3.1. Testes - Aplicação da transformação de Householder

Após implementar o algoritmo responsável por transformar a matriz simétrica A em uma matriz tridiagonal, para então utilizarmos esta matriz como parâmetro das funções do EP1 (Algoritmo QR) e obter seus autovalores e autovetores, realizamos uma série de testes com algumas matrizes já fornecidas pelos professores.

De forma a não poluir o relatório, apresentaremos os resultados relativos à matriz de dimensão 4 apenas (os resultados da matriz 20x20 podem ser verificados compilando nosso EP).

```
Matriz A:
[[2. 4. 1. 1.]
 [4. 2. 1. 1.]
 [1. 1. 1. 2.]
 [1. 1. 2. 1.]]

########################### RESULTADOS ###########################


A matriz tridiagonal simétrica é:
 [[ 2.00000000e+00 -4.24264069e+00 -2.22044605e-16 -2.22044605e-16]
 [-4.24264069e+00  3.00000000e+00  1.41421356e+00  0.00000000e+00]
 [-2.22044605e-16  1.41421356e+00  2.00000000e+00 -4.44089210e-16]
 [-2.22044605e-16  0.00000000e+00 -1.11022302e-16 -1.00000000e+00]]

A matriz H transposta é:
 [[ 1.          0.          0.          0.        ]
 [ 0.         -0.94280904  0.33333333  0.        ]
 [ 0.         -0.23570226 -0.66666667 -0.70710678]
 [ 0.         -0.23570226 -0.66666667  0.70710678]]


Autovalores encontrados:
 [6.999999999999999, -1.999999999999956, 1.999999999999958, -1.0000000000000002]

Autovetores encontrados:
 [[ 6.32455538e-01 -7.07106748e-01  3.16227828e-01 -2.37904934e-17]
 [ 6.32455538e-01  7.07106814e-01  3.16227681e-01 -5.28677631e-18]
 [ 3.16227755e-01 -6.55652520e-08 -6.32455538e-01  7.07106781e-01]
 [ 3.16227755e-01 -6.55652520e-08 -6.32455538e-01 -7.07106781e-01]]
```

A imagem acima mostra qual a matriz A que está sendo tridiagonalizada. Após tridiagonalizar e imprimir o resultado, também estamos imprimindo a matriz H transposta (necessária para o cálculo dos autovetores).

Com estas duas matrizes em mãos (tridiagonalizada e H transposta), nosso EP2 chama a função feita no EP1 responsável por encontrar os autovalores e os autovetores de uma matriz tridiagonal simétrica. Estes valores também estão registrados na figura acima.

Realizamos também os produtos pedidos no enunciado, isto é A * (autovetor) = (lambda) * (autovetor). Dessa forma fomos capazes de verificar se a produto matricial A * (autovetor), para cada um dos autovetores, é igual ao produto do escalar (lambda), que é o autovalor, multiplicado pelo devido autovetor. Estes resultados estão mostrados abaixo.

```
######################### VERIFICAÇÕES PEDIDAS #########################

Verificando se A * v = lambda * v, para cada autovalor lambda e seu autovetor v correspondente

Verificação para o 1º autovalor:

Produto A * V:     [4.42718874 4.42718874 2.21359434 2.21359434]
Produto lambda * v:  [4.427188763729668, 4.427188763729658, 2.213594283129999, 2.213594283129999]

Verificação para o 2º autovalor:

Produto A * V:     [ 1.41421363e+00 -1.41421350e+00 -1.31130505e-07 -1.31130505e-07]
Produto lambda * v:  [1.4142134968078055, -1.4142136279383075, 1.311305040541907e-07, 1.311305040541907e-07]

Verificação para o 3º autovalor:

Produto A * V:     [ 0.63245531  0.6324556  -1.2649111  -1.2649111 ]
Produto lambda * v:  [0.6324556560740573, 0.6324553628573375, -1.2649110753512987, -1.2649110753512987]

Verificação para o 4º autovalor:

Produto A * V:     [ 1.11022302e-16  0.00000000e+00 -7.07106781e-01  7.07106781e-01]
Produto lambda * v:  [2.379049338482479e-17, 5.2867763077388504e-18, -0.7071067811865477, 0.7071067811865476]
```

Podemos observar que os valores possuem um erro menor que 10^(-6).

Realizamos também a verificação final para checar se a matriz formada pelos autovetores é ortogonal, os resultados estão expressos abaixo. Podemos concluir que ela é ortogonal, já que a inversa desta matriz é igual à sua transposta.

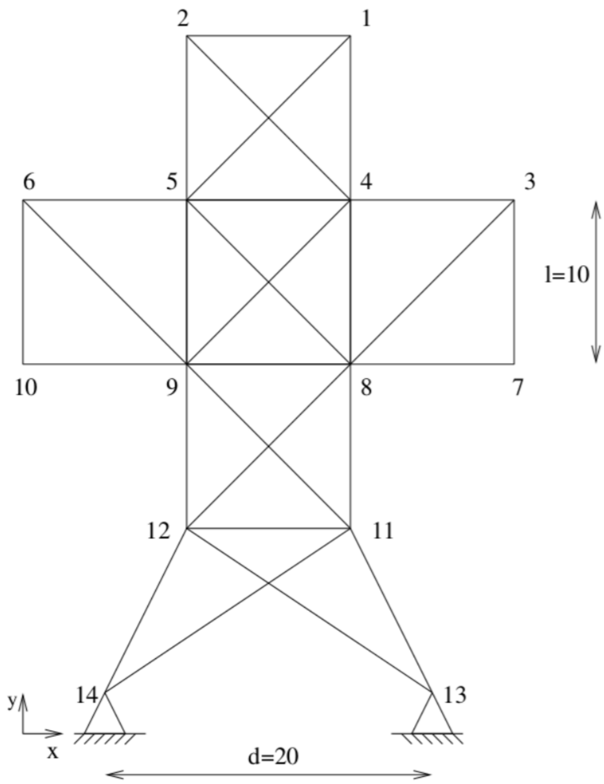
```
Verificando se a matriz formada pelos autovetores é ortogonal

Inversa da matriz formada pelos autovetores:
 [[ 6.32455538e-01  6.32455538e-01  3.16227755e-01  3.16227755e-01]
 [-7.07106748e-01  7.07106814e-01 -6.55652520e-08 -6.55652520e-08]
 [ 3.16227828e-01  3.16227681e-01 -6.32455538e-01 -6.32455538e-01]
 [-0.00000000e+00 -0.00000000e+00  7.07106781e-01 -7.07106781e-01]]
Transposta da matriz formada pelos autovetores:
 [[ 6.32455538e-01  6.32455538e-01  3.16227755e-01  3.16227755e-01]
 [-7.07106748e-01  7.07106814e-01 -6.55652520e-08 -6.55652520e-08]
 [ 3.16227828e-01  3.16227681e-01 -6.32455538e-01 -6.32455538e-01]
 [-2.37904934e-17 -5.28677631e-18  7.07106781e-01 -7.07106781e-01]]
```

3.2. Aplicação: Treliças Planas

3.2.1. Definição do problema

Agora vamos aplicar o conhecimento desenvolvido no cálculo de frequências de vibração de uma treliça plana, bem como seus principais modos de vibração. A estrutura é formada por barras interligadas por nós, cada qual podendo ter um material, módulo de elasticidade, densidade, e áreas transversais diferentes, mas neste caso foram todas padronizadas em forma de um arquivo de entrada “input-c”. A treliça é mostrada abaixo.



3.2.2. Solução do problema

Para resolver o problema, primeiro foram tratados os dados de entrada do arquivo input-c. Para tanto, leu-se o arquivo e colocaram-se os dados em variáveis adequadas. Foram anotados o número de nós da estrutura, o número de nós livres, o número total de barra e parâmetros como a densidade, a área e a elasticidade. Em seguida, foram extraídos os dados de cada barra no arquivo de entrada.

Com isso em mãos, agora é possível começar a solução. Organizou-se o problema, procedendo com a montagem de cada uma das matrizes. Para tal, dentro de um mesmo loop foram montadas, simultaneamente, a matriz de rigidez K e a matriz de massas M. Este efeito foi alcançado com as fórmulas e métodos descritos no enunciado.

De posse dessas informações e matrizes, agora é questão de transformar a matriz K na matriz K_til a partir da inversa de M. Feito isso, partimos para a triagonalização e aplicação do algoritmo QR com deslocamento para achar os autovalores e respectivos autovetores. Os primeiros são referentes às frequências, e os segundos são os modos de vibração. Pegando as 5 menores frequências, temos o pedido de impressão do EP (pede-se apenas para calcular as frequências e os modos de vibração e não printá-los).

Abaixo podemos ver as 5 primeiras frequências e seus respectivos modos de vibração para o teste disponibilizado pelos professores:

```
5 menores frequências e seus respectivos modos de vibração:


Frequências (Hz):
24.59254776972408

Modo de vibração:
[0.0034957234757986228, -0.0006120483507039197, 0.0034957234757986267, 0.0006120483507039248, 0.0022690781036322923, -0.001813025
1734179044, 0.0022483548566249777, -0.0005955082307912311, 0.0022483548566249764, 0.0005955082307912352, 0.0022690781036322923, 0
.0018130251734179014, 0.0009983715663215312, -0.0018173116560307658, 0.0009960167130160932, -0.0005071916280142904, 0.00099601671
30160956, 0.0005071916280142951, 0.0009983715663215318, 0.0018173116560307623, 4.181565299668371e-05, -0.000296097335851637, 4.18
1565299668313e-05, 0.00029609733585163914]

Frequências (Hz):
92.01244464604126

Modo de vibração:
[6.261856301945498e-06, 0.002186165783380751, -6.261856302031636e-06, 0.00218616578338066, -0.00047293490413368567, 0.00298517867
02397073, -0.00017608247392349158, 0.0020507692616560822, 0.00017608247392358631, 0.0020507692616560016, 0.0004729349041337532, 0
.002985178670239377, 4.166105955572546e-06, 0.0030871105244266012, 4.028547257420346e-06, 0.0017439886157908904, -4.0285472571483
38e-06, 0.0017439886157908531, -4.166105955290771e-06, 0.003087110524426258, -4.2117962341510804e-05, 0.001097248590341578, 4.211
796234164423e-05, 0.0010972485903415834]

Frequências (Hz):
94.70336537381635

Modo de vibração:
[-0.000778899923910793, 0.0008083592771068742, -0.0007788999239107927, -0.0008083592771071325, 0.0006405090017071284, 0.002837993
0144523434, 0.0008745889619211643, 0.0007135820556578032, 0.0008745889619211463, -0.0007135820556580446, 0.0006405090017070804, -
0.0028379930144526683, 0.0024842071832332805, 0.0029408584705871333, 0.002397314492683092, 0.00031093455250483205, 0.002397314492
6830903, -0.0003109345525050356, 0.002484207183233278, -0.0029408584705874715, 0.0011566254013408248, -5.6343617196554063e-05, 0.
0011566254013408216, 5.6343617196424996e-05]

Frequências (Hz):
142.80969710649003
```

```
Frequências (Hz):
142.80969710649003

Modo de vibração:
[0.003868794968343746, -0.0019304008734020986, 0.0038687949683437455, 0.0019304008734020478, -0.0014881392983057701, 0.0026809534
301750248, -0.0006903866809850476, -0.0011429772027278212, -0.0006903866809850384, 0.001142977202727771, -0.0014881392983057328,
-0.002680953430174943, -0.0005565432338917722, 0.002912620267807109, -0.0005122763541936824, 2.022249765213124e-05, -0.0005122763
541936992, -2.02224976521639e-05, -0.0005565432338917922, -0.0029126202678070147, -0.0002311329925527663, 4.4748724265315e-05, -0
.0002311329925527644, -4.474872426533426e-05]

Frequências (Hz):
150.8221265108163

Modo de vibração:
[8.982146202549605e-05, -0.0019225428544940716, -8.982146202558895e-05, -0.0019225428544941154, -0.0015051224601160137, 0.0033876
959228772057, -0.00043433329002901254, -0.0017974229315753378, 0.00043433329002903157, -0.0017974229315753625, 0.0015051224601160
546, 0.0033876959228772885, 0.0006892736810391287, 0.003717491403170376, 0.0006281250945224571, -0.0011294791302261927, -0.000628
1250945224523, -0.0011294791302261886, -0.0006892736810391208, 0.003717491403170464, -5.000369792287652e-05, -0.00067874387316497
58, 5.000369792287227e-05, -0.000678743873164972]

Se deseja ver a matriz de rididez total, digite 1, se não, digite 0:
```

Se o usuário desejar ver a matriz de rigidez total calculada pelo grupo, basta digitar 1 nesta pergunta final. As primeiras linhas desta matriz estão representadas abaixo:

```
[[ 2.70710678e+09  7.07106781e+08 -2.00000000e+09  0.00000000e+00
   0.00000000e+00  0.00000000e+00 -7.49879891e-24 -1.22464680e-07
  -7.07106781e+08 -7.07106781e+08  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 7.07106781e+08  2.70710678e+09  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00 -1.22464680e-07 -2.00000000e+09
  -7.07106781e+08 -7.07106781e+08  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-2.00000000e+09  0.00000000e+00  2.70710678e+09 -7.07106781e+08
   0.00000000e+00  0.00000000e+00 -7.07106781e+08  7.07106781e+08
  -7.49879891e-24 -1.22464680e-07  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00 -7.07106781e+08  2.70710678e+09
   0.00000000e+00  0.00000000e+00  7.07106781e+08 -7.07106781e+08
  -1.22464680e-07 -2.00000000e+09  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   2.70710678e+09  7.07106781e+08 -2.00000000e+09  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -7.49879891e-24 -1.22464680e-07 -7.07106781e+08 -7.07106781e+08
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
```