INTRODUÇÃO AOS MÉTODOS NUMÉRICOS

MAP3121

Autovalores e Autovetores de Matrizes Simétricas e Aplicação de Transformações de Householder e Método QR

Eduardo Tadashi Asato[1]

[1]Engenharia Elétrica, Escola Politécnica da USP, NUSP 10823810

Resumo

No EP1, o método QR para o cálculo de autovalores e autovetores de matrizes tridiagonais foi implementado. Agora, no EP2, esse método foi generalizado para matrizes simétricas, usando transformações de Householder, para tridiagonalizar a matriz, para o método QR poder ser aplicado. Neste EP, o método foi aplicado à dinâmica de uma treliça plana.

1 | INTRODUÇÃO

1.1 | Método QR

Do método QR, detalhado no relatório do EP1, são extraídas duas matrizes: Λ, que contém os autovalores da matriz A, na sua diagonal principal; e V, matriz com os autovetores nas colunas. Assim pode-se escrever o que está na equação (1).

$$A = V \Lambda V^T \tag{1}$$

1.2 | Transformações de Householder

Uma transformação de Householder é definida por $H_w : \mathbb{R}^n \to \mathbb{R}^n$, em que $w \in \mathbb{R}^n$, dada pela equação (2). Se aplicarmos essa transformação em um vetor $x \in \mathbb{R}^n$, obtemos a equação (3). É possível descobrir um w tal que $H_w x = \lambda y$, para tanto basta tomar $w = x + \alpha y$, em que $\alpha = \pm \frac{\|x\|}{\|y\|}$, como mostra a demonstração em (4). Isso é útil para eliminar os elementos da matriz fora das três diagonais centrais.

$$H_w = I - 2\frac{w w^T}{w \cdot w} \tag{2}$$

$$H_w x = x - 2\frac{w \cdot x}{w \cdot w} w \tag{3}$$

$$H_w x = x - 2\frac{w \cdot x}{w \cdot w} w = x - \frac{2\|x\|^2 + 2\frac{\|x\|}{\|y\|} x \cdot y}{\|x\|^2 + 2\frac{\|x\|}{\|y\|} x \cdot y + \|x\|^2} w = -\frac{\|x\|}{\|y\|} y = \lambda y \tag{4}$$

Então suponha que exista uma matriz simétrica $A \in M_n(\mathbb{R})$. Se for usado um $w_1 = (0, \tilde{w_1})$, em que $\tilde{w_1} = \tilde{a_1} + \delta \|\tilde{a_1}\| e_2$, é um vetor de dimensão $n-1$, em que $\tilde{a_1}$ é a primeira coluna de A sem o primeiro elemento desta coluna, e_2 é um vetor de zeros com apenas um um na primeira posição, e forem feitas multiplicações por H_{w_1} pela esquerda e pela direita, os elementos da matriz resultante, na primeira linha e coluna, fora das três diagonais centrais, são zerados, constituindo uma primeira otimização na implementação deste algoritmo, e uma primeira etapa da tridiagonalização da matriz. Aplicando isso, iterativamente, como mostra a equação (5), uma matriz simétrica tridiagonal T pode ser obtida.

Como a primeira posição de w_1 é zero, o elemento A_{11} não é modificado, constituindo outra otimização. Os zeros surgem por causa da propriedade descrita no parágrafo do início desta secção.

Quando $H_{w_1} A$ é multiplicada por H_{w_1} pela direita, o resultado é uma matriz simétrica, então mais uma otimização que pode ser implementada é calcular apenas os elementos do triângulo inferior ou superior da submatriz que abrange a partir das segundas linha e coluna.

$$T = H_{w_{n-2}} H_{w_{n-1}} \dots H_{w_2} H_{w_1} A H_{w_1} H_{w_2} \dots H_{w_{n-1}} H_{w_{n-2}} \tag{5}$$

Se juntarmos $H_{w_1} H_{w_2} \ldots H_{w_{n-1}} H_{w_{n-2}}$ em uma única matriz H^T, obtém-se o que está na equação matricial (6), pois H_w é ortogonal e simétrica.

$$T = HAH^T \tag{6}$$

1.3 | Tridiagonalizando Matrizes

Se o método QR for aplicado à matriz da equação (6), obtém-se o que está na equação (7).

$$A = H^T V \Lambda V^T H \tag{7}$$

Então os autovalores de A são os elementos da diagonal principal de Λ, ou seja, são os autovalores que são saída direta do método QR. Porém os autovetores são as colunas de $H^T V$. Essa multiplicação poderia ser feita, mas uma forma otimizada de fazer isto é usar H^T como a matriz onde as matrizes de transformações de Givens do método QR são acumuladas, ao invés de usar a matriz identidade. Fazendo assim, os autovetores passam a estar na matriz de saída do método QR.

1.4 | Treliças Planas

Uma treliça plana é uma das aplicações de cálculo de autovalores e autovetores de matrizes simétricas. A equação de movimento destas treliças é dada pela equação (8), em que **x** é um vetor com as posições dos nós móveis da treliça em relação à posição de equilíbrio, porém nas duas direções, então existem duas entradas nesse vetor por nó. M é uma matriz cujos elementos da diagonal principal são as massas dos nós, repetidas 2 vezes por nó. E K é a matriz de rigidez da treliça.

Para obter a matriz de rigidez, devemos primeiro obter a matriz de rigidez para cada barra $\{i, j\}$ da treliça, na equação (9), e inserir os valores na matriz K de acordo com a posição dos nós no vetor **x**.

As soluções dessa equação são da forma (10). Substituindo em (8), obtém-se (11). Se fizermos a troca de variáveis (12), o problema vira (13), em que $\tilde{K} = M^{-\frac{1}{2}} K M^{-\frac{1}{2}}$ é a matriz cujos autovalores são as frequências ω de vibração ao quadrado, e os autovetores são os **y**, que pode ser convertidos em **z** para obter os modos de vibração.

$$M\ddot{\mathbf{x}} + K\mathbf{x} = 0 \tag{8}$$

$$K^{\{i,j\}} = \frac{AE}{L^{\{i,j\}}} \begin{bmatrix} C^2 & CS & -C^2 & -CS \\ CS & S^2 & -CS & -S^2 \\ -C^2 & -CS & C^2 & CS \\ -CS & -S^2 & CS & S^2 \end{bmatrix} \tag{9}$$

$$\mathbf{x} = \mathbf{z}e^{i\omega t} \tag{10}$$

$$K\mathbf{z} = \omega^2 M\mathbf{z} \tag{11}$$

$$\mathbf{z} = \tilde{M}\mathbf{y} = M^{-\frac{1}{2}}\mathbf{y} \tag{12}$$

$$\tilde{K}\mathbf{y} = \omega^2\mathbf{y} \tag{13}$$

2 | IMPLEMENTAÇÃO

O programa foi implementado em Python, e se encontra divido em vários arquivos, como mostra a tabela a seguir:

nome do arquivo	descrição
main.py	Arquivo da rotina padrão do programa. É ela que chama as rotinas dos outros arquivos
teste_a.py	Arquivo com o teste a, que calcula os autovalores e autovetores de uma matriz pré-determinada, e faz estimativas de erro
teste_b.py	Arquivo com o teste b, que calcula os autovalores e autovetores de uma matriz configurável e com solução analítica, e faz estimativas de erro
teste_c.py	Arquivo com o teste c, que resolve uma treliça, calcula frequências e modos de vibração, e faz estimativas de erro
teste_d.py	Arquivo com o teste d, que calcula os autovalores e autovetores de uma matriz de um arquivo, e faz estimativas de erro
AutovalsAutovecs/functions.py	Arquivo com a rotina de cálculo dos autovalores e autovetores de matriz simétrica

2.1 | *Fuctions.py*

Este é o arquivo mais importante, pois é aqui que o algoritmo de tridiagonalização de matriz e cálculo de seus autovalores e autovetores está implementado.

2.1.1 | Método *metodo_QR()*

Um dos seus métodos, *metodo_QR(alfa, beta, epsilon, deslocamentos, V)*, cujo trecho se encontra na figura (1), é o que aplica o método QR na matriz tridiagonal dada pelos parâmetros *alfa* e *beta*, com *V* sendo a matriz onde

as transformações de Givens são acumuladas, é aqui que H^T deve ser inserida, constituindo uma das otimizações implementadas. A saída deste método é uma tupla *(Diagonal_principal, temp_V, k)*, em que *Diagonal_principal* é um vetor onde estão os autovalores; *temp_V* é a matriz com os autovetores, é daqui que $H^T V$ deve sair; e *k*, que é um inteiro com número de iterações utilizadas para aplicar o método QR, dada a tolerância *epsilon* e o booleano *deslocamentos*.

FIGURA 1 Trecho do código do método *metodo_QR()* de *Fuctions.py*

```
def metodo_QR(alfa, beta, epsilon, deslocamentos, V):
    # Este método faz o cálculo dos cossenos (forma estável)
    def __calcula_c(alfa, beta, i):
        if abs(alfa[i]) > abs(beta[i]):
            tau = -beta[i] / alfa[i]
            return 1 / math.sqrt(1 + tau * tau)
        else:
            tau = -alfa[i] / beta[i]
            return tau / math.sqrt(1 + tau * tau)

    # Este método faz o cálculo dos senos (forma estável)
    def __calcula_s(alfa, beta, i):
        if abs(alfa[i]) > abs(beta[i]):
            tau = -beta[i] / alfa[i]
            return tau / math.sqrt(1 + tau * tau)
        else:
            tau = -alfa[i] / beta[i]
            return 1 / math.sqrt(1 + tau * tau)

    # Este método faz o cálculo de mu
    # m: o tamanho-1 da submatriz onde o método é aplicado
    def __calcula_mu(alfa, beta, m, deslocamentos):
        # O if abaixo detecta se foram solicitados deslocamentos espectrais
        if deslocamentos:
            # Precisamos obter primeiramente dk.
            dk = (alfa[m - 1] - alfa[m]) / 2
            mu = alfa[m] + dk - np.sign(dk) * math.sqrt(dk ** 2 + beta[m - 1
            return mu
        else:
```

2.1.2 | Método *HouseHolder()*

Outro método é *HouseHolder(A, HT, n)*, que faz a tridiagonalização da matriz simétrica *A*, de tamanho *n*, e acumula as transformações de Householder a partir da matriz *HT*. E tem como saída a tupla *(A, HT)*, em que *A* é a matriz

tridiagonalizada, e *HT*, que é a matriz com as transformações de Householder acumuladas, é a matriz H^T da equação (7).

O primeiro trecho deste método é o que está na figura (2), onde as matrizes de entrada são copiadas. Também é declarado a função interna *dot_prod(a,b,k)*, que faz o produto interno dos vetores *a* e *b*, mas só levando em conta o últimos *k* elementos destes dois vetores.

FIGURA 2 Trecho 0 do código do método *HouseHolder()* de *Fuctions.py*

```
def HouseHolder(A, HT, n):
    A = np.copy(A)
    HT = np.copy(HT)

    # Função interna que faz o produto interno dos vetores a e b,
    # utilizando os últimos k elementos dos vetores a e b
    def dot_prod(a,b,k):
        soma = 0
        a = np.squeeze(np.asarray(a))
        b = np.squeeze(np.asarray(b))
        size_a = np.max(a.shape)
        size_b = np.max(b.shape)
        for i in range(0,k):
            soma = soma + a[size_a-1-i]*b[size_b-1-i]
        return soma
```

Para entender o funcionamento desta implementação da tridiagonalização, considere a matriz de exemplo (14), aqui, $I = 0$. Calculamos w de acordo com o que está em 1.2, então obtemos $w_0 = (0, -1 - \sqrt{11}, 1, 3)$ (a partir daqui os índices começam em zero), mas, como os primeiros $I + 1$ elementos de w_I serão sempre zeros, só é necessário armazenar $\bar{w}_0 = (-1 - \sqrt{11}, 1, 3)$, por isso que o vetor *w* é instanciado com tamanho *n-1* na figura (3). Na mesma figura o *for* mais externo é aquele que roda em $n-2$ colunas da matriz, tridiagonalizando-a. O segundo *for* e a linha de código depois dele, na figura, fazem o cálculo do vetor $\bar{w}_I$, porém somente os últimos $n - 1 - I$ elementos desse vetor se referem a $\bar{w}_I$.

A próxima linha de código, faz o cálculo de *ww* que é a norma ao quadrado de $\bar{w}_I$, como esse cálculo será necessário repetidas vezes numa iteração *I* do for externo, então vale a pena fazer ele apenas uma vez e armazenar em um temporário.

Levando em conta novamente a matriz da equação (14), se fosse multiplicada por H_{w_0} pela esquerda, seria obtido o que está na equação (15). E, se fosse agora multiplicada, pela direita, novamente, seria obtido o que está na equação (16). No algoritmo implementado essas multiplicações não foram feitas calculando as matrizes H_{w_I}.

Percebe-se que o que está na equação (16), e isso vai valer para todas as iterações deste método, é uma matriz simétrica, e, entre essa equação e a (15), a coluna I não foi alterada, então é possível calcular a linha I e a coluna I de $A^{(I+1)}$ diretamente, a partir de $A^{(I)}$, foi exatamente isso que foi implementado no terceiro *for* da figura (3), essa é

uma das otimizações.

Dentro deste *for* existe um *if*, que insere zeros, que apareceriam caso a multiplicação de matrizes tivesse sido feita, essa é uma otimização. O *else* deste *for* calcula os elementos $(I+1, I)$ e $(I, I+1)$ da equação (16). Para isso, primeiro ele calcula o coeficiente que fica na frente de w na equação (3) (que foi usada, ao invés de calcular as matrizes H_w, uma das otimizações), usando x como sendo a linha I de $A^{(I)}$.

Desta forma o elemento A_{II} fica inalterado, mais uma otimização.

Entre uma iteração I e a próxima, a linha I e a coluna I permanecem inalteradas, então o algoritmo foi implementado de forma a apenas trabalhar com a submatriz de $A^{(I)}$, obtida considerando-se apenas as linhas a partir da linha I, contando ela, e a partir da coluna I, contando ela, outra otimização.

Dentro do escopo do for externo, existe mais outro for, o da figura (4), este for é o que calcula os valores que mudam entre as equações (14) e (15) na submatriz dos elementos da matriz $A^{(I)}$ a partir da linha e coluna I sem contar elas, ou seja, na submatriz dentro da submatriz. Isso é feito calculando, por exemplo, $H_{\bar{w}_0}\begin{bmatrix}1 & 4 & 2\end{bmatrix}^T = \begin{bmatrix}2.7136 & 3.6030 & 0.809068\end{bmatrix}^T$. Então, para cada coluna da submatriz, o coeficiente da equação (3) é calculado, e essa equação é aplicada para cada linha da submatriz de $A^{(I)}$ (no caso do algoritmo implementado foram usadas as colunas da submatriz de $A^{(I)}$, mas como a mesma é simétrica o resultado é o mesmo, fazer desta forma permitiu com que não fosse necessário armazenar, em temporários, os coeficientes para cada linha da submatriz de $A^{(I)}$, portanto, é mais uma otimização implementada).

Dentro do escopo do for externo, existe mais outro for, o da figura (5), este for é o que calcula os valores que mudam entre as equações (15) e (16) na submatriz dos elementos da matriz $A^{(I)}$ a partir da linha e coluna I sem contar elas. Dada uma linha i da submatriz, o coeficiente é calculado e armazenado em *prod*. O próximo *for* varia o j de i até $n-1$, ou seja, ele só vai atualizar os valores do triângulo superior da submatriz na linha i. O último *for* desta figura apenas reflete os valores do triângulo superior no inferior, para os valores da linha i, desta forma não é necessário nenhuma variável temporária adicional, e não se calculam valores que já se sabe o valor, constituindo mais duas otimizações do algoritmo.

O último *for* dentro deste escopo, está na figura (6), que é aquele que acumula, na matriz HT as matrizes de transformação de Householder, ou seja, realiza o que está na equações (17), (18) e (19). Aqui o *for* varia o *i* pela matriz inteira, pois quando multiplicamos as matrizes de transformações pela direita, todas as linhas da matriz vão mudar, mas o *for* interno a este, só varia j de *I+1* até o final da matriz. Aqui, as linhas de $HT^{(I+1)}$ estão sendo obtidas a partir das linhas de $HT^{(I)}$, ou seja, se observarmos as equações (18) e (19), o que está sendo feito no algoritmo implementado é $H_{w_1}\begin{bmatrix}0 & 0.30151 & 0.30151 & 0.90453\end{bmatrix}^T = \begin{bmatrix}0 & 0.30151 & 0.82288 & 0.48162\end{bmatrix}^T$, usando a equação (3), essa é mais uma otimização.

$$A = A^{(0)} = \begin{bmatrix} 2 & -1 & 1 & 3 \\ -1 & 1 & 4 & 2 \\ 1 & 4 & 2 & -1 \\ 3 & 2 & -1 & 1 \end{bmatrix} \tag{14}$$

$$H_{w_0}A^{(0)} = \begin{bmatrix} 2 & 1 & 1 & 3 \\ 3.3166 & 2.7136 & 1.5076 & 0 \\ 0 & 3.6030 & 3.2759 & 0.53667 \\ 0 & 0.809068 & 2.8277 & 2.3900 \end{bmatrix} \tag{15}$$

$$A^{(1)} = H_{w_0} A^{(0)} H_{w_0} = \begin{bmatrix} 2 & 3.3166 & 0 & 0 \\ 3.3166 & 1.2727 & 0.58407 & 2.7704 \\ 0 & 0.58407 & 4.2458 & 2.3733 \\ 0 & 2.7704 & 2.3733 & 1.0268 \end{bmatrix} \tag{16}$$

$$HT^{(0)} = I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

$$HT^{(1)} = HT^{(0)} H_{w_0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.30151 & 0.30151 & 0.90453 \\ 0 & 0.30151 & 0.93015 & 0.20955 \\ 0 & 0.90453 & 0.20955 & 0.37136 \end{bmatrix} \tag{18}$$

$$HT^{(2)} = HT^{(1)} H_{w_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.30151 & 0.82288 & 0.48162 \\ 0 & 0.30151 & 0.39692 & 0.86692 \\ 0 & 0.90453 & 0.40660 & 0.12843 \end{bmatrix} \tag{19}$$

2.1.3 | Método *AutovalsAutovecs()*

O código deste método está na figura (7). Este método tem como entrada *A*, a matriz simétrica cujos autovalores e autovetores se quer calcular; *epsilon*, a tolerância do método QR; e *deslocamentos*, booleano que indica se devem ser usados deslocamentos espectrais.

A primeira linha deste método apenas instancia uma matriz identidade, onde será armazenada *HT*. As transformações de Householder são aplicadas, e o resultado é armazenado na tupla *householder*. A matriz *A* é convertida numa forma que possa ser entrada do método QR. E, por fim, o método QR é aplicado com a matriz *HT* como uma das entradas, uma otimização que já foi explicada anteriormente.

2.2 | Teste A

Esse é o teste que faz o cálculo dos autovalores e autovetores da matriz em (20). O código deste teste começa com o trecho da figura (8), que faz a entrada do usuário dos parâmetros *epsilon*, que é a tolerância usada pelo método QR; *deslocamentos*, booleano que indica se o método QR deve usar deslocamentos espectrais. E, por fim, instancia a matriz *A*.

Depois disto, o código do trecho da figura 9 é executado, no qual os autovalores e autovetores de *A* são calculados,

FIGURA 3 Trecho 1 do código do método *HouseHolder()* de *Fuctions.py*

```
w = np.zeros(n-1)   # Vetor onde o w da iteração atual é guardado

prod = 0 # Temporário onde resultados de produtos escalares são guardados

for I in range(0,n-2):
    # Extraindo a I-ésima coluna de A, da posição I+1 em diante
    for i in range(I+1, n):
        w[i-1] = A[i,I]

    # Fazendo bar_wI = bar_aI + delta*norm(bar_aI)*eI
    w[I] = w[I]+np.sign(w[I])*math.sqrt(dot_prod(w,w,n-I-1)) # aplicando o e da fórmula do w

    ww = dot_prod(w,w,n-I-1) # Produto escalar de w.w, mas só de um pedaço, pois deve agir como se o resto do vetor fosse 0

    # Este primeiro for faz o cálculo dos elementos de A(I+1) = Hw(I)*A(I)*Hw(I) que estão nas I-ésimas linha e coluna
    # Aplicando a otimização
    for i in range(I+1,n):
        if i>I+1:
            # Aplicando zeros na I-ésima linha e coluna
            A[i,I] = 0
            A[I,i] = 0
        else:
            # O else é ativado na primeira subiteração, ou seja,
            # quando i = I+1
            # Elemento não nulo da I-ésima linha e coluna

            # Neste caso prod guarda -2*w.x/w.w, em que x é o vetor coluna
            # com zeros no começo da I-ésima coluna de A
            prod = -2*dot_prod(w,A[:,I],n-I-1)/ww
            A[I+1,I] = A[I+1,I]+prod*w[I]
            A[I,I+1] = A[I,I+1]+prod*w[I]
```

FIGURA 4 Trecho 2 do código do método *HouseHolder()* de *Fuctions.py*

```
        # Este for faz a multiplicação da submatriz AI(I) por Hw(I) pela esquerda
        #
        # Multiplicar pela esquerda equivale a pegar as colunas de AI(I) e tratar
        # como se fossem os vetores x, e isso resulta nas colunas de Hw(I)*AI(I)
        # Então só é necessário usar uma posição de prod[]
        for j in range(I+1,n):
            # Este prod é o produto escalar da j-ésima coluna de AI(I) com w,
            # só levando em conta os n-I-1 elementos, ou seja, abaixo da parte
            # que já foi determinada no for anterior
            prod = -2*dot_prod(w, A[:,j], n-I-1)/ww

            # Este for aplica a transformação sobre as colunas de AI(I)
            for i in range(I+1,n):
                A[i,j]=A[i,j]+prod*w[i-1]
```

chamando o método *AutovalsAutovecs*, com os resultados na tupla *resultados*. Uma lista dos autovalores e autovetores é impressa no terminal.

FIGURA 5 Trecho 3 do código do método *HouseHolder()* de *Fuctions.py*

```
        # -> Multiplicação por HwI pela direita
        for i in range(I+1,n):
            prod = -2*dot_prod(w, A[i,:], n-I-1)/ww
            for j in range(i, n):
                A[i,j] = A[i,j]+prod*w[j-1]
            for j in range(I+1,i):
                A[i,j]=A[j,i]+0
```

FIGURA 6 Trecho 4 do código do método *HouseHolder()* de *Fuctions.py*

```
        # Multiplicação para encontrar HT
        for i in range(0,n):
            prod = -2*dot_prod(w,HT[i,:],n-I-1)/ww
            for j in range(I+1,n):
                HT[i,j]=HT[i,j]+prod*w[j-1]
    #print("Erro holdeholder = {0}".format(np.matmul(H, np.transpose(H))-np.identity(n)))
    return (A, HT)
```

FIGURA 7 Trecho do código do método *AutovalsAutovecs()* de *Fuctions.py*

```
def AutovalsAutovecs(A, epsilon, deslocamentos):
    HT = np.identity(A.shape[0]) # Matriz onde as transformações de Householder transpostas serão acumuladas
    householder = HouseHolder(A, HT, A.shape[0]) # Tridiagonalizando a matriz

    # Convertendo a matriz tridiagonal em uma forma compatível com o método QR
    alfa = np.zeros(A.shape[0])
    beta = np.zeros(A.shape[0]-1)

    for i in range(0, householder[0].shape[0]):
        alfa[i] = householder[0][i,i]
    for i in range(0, householder[0].shape[0]-1):
        beta[i] = householder[0][i+1,i]

    return metodo_QR(alfa, beta, epsilon, deslocamentos, householder[1]) # Aplicando o método QR
```

A função *EstimativasErroGerais*, no mesmo arquivo, cujo código se encontra na figura (10), é chamada, com os argumentos *A*; *resultados[0]*, vetor com os autovalores de *A*; e *resultados[1]*, matriz com os autovetores de *A*, ou seja, é a matriz $H^T V$.

$$A = \begin{bmatrix} 2 & 4 & 1 & 1 \\ 4 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 2 & 1 \end{bmatrix} \tag{20}$$

FIGURA 8 Trecho 0 do código do teste A

```
# Menu de seleção
print('#############')
print('Teste a selecionado')
print('#############')

epsilon = 0.1
deslocamentos = True

try:
    epsilon = float(input('epsilon = '))
    deslocamentos = input('Usar deslocamentos espectrais? (s,n)')
    if deslocamentos == 's':
        deslocamentos = True
    else:
        deslocamentos = False
except:
    print('epsilon deve ser número real, pe: 1e-6')
    return

print("\nOs autovalores e autovetores da matriz do item 4.1.a serão imp

A = np.matrix([[2,4,1,1],[4,2,1,1],[1,1,1,2],[1,1,2,1.0]])
```

Como as matrizes H, das transformações de Householder, e V, das transformações de Givens, são ortogonais e simétricas pela teoria, também se esperaria que as obtidas computacionalmente também fossem. É exatamente isso que a primeira estimativa de erro, na figura (10), tenta verificar. O erro obtido é o máximo entre as células do módulo da diferença entre as matrizes $H^T V$ e I.

A segunda estimativa de erro se baseia na equação (7), que pode ser reordenada para o que na equação (21). Então, basicamente, essa estimativa de erro verifica o quão próxima de zero está esta subtração de matrizes.

O *for* na rotina *EstimativasErroGerais* imprime no terminal uma lista com os autovalores obtidos fazendo $Av./v$ (notação de *matlab*), e compara com os autovalores obtidos pelo método QR. A diferença entre esses dois é acumulada

FIGURA 9 Trecho 1 do código do teste A

```
resultados = aa.AutovalsAutovecs(A, epsilon, deslocamentos)

print("\nForam necessárias k = {0} iterações no método QR\n".format(resultados[2]))

for i in range(0, A.shape[0]):
    # Configuração de impressão para ter mais dígitos
    #np.set_printoptions(formatter={'float': '{: 12.10f}'.format})

    # Impressão dos autovals/autovecs
    print('Autovalor: {0:12.10f}, Autovetor:'.format(resultados[0][i]), resultados[1][:, i])

print("\n")
print('############\n')
EstimativasErroGerais(A, resultados[0], resultados[1])

#Lambda = np.zeros((A.shape[0], A.shape[0]))
#for i in range(0, A.shape[0]):
#    Lambda[i,i] = resultados[0][i]

#erro = np.max(np.abs((np.matmul(np.matmul(resultados[1], Lambda), np.transpose(resultados[1])) - A)))
#print("\nEstimativa de erro H*L*HT-A = {0}\n".format(erro))
print('############')

return
```

num vetor *erros[]* com os erros para cada autovalor. Algo importante a se notar nesta estimativa de erro, é que, como um vetor é unidimensional, então tem n resultados de autovalor por autovalor, então, para obter um único resultado por autovalor, a média foi utilizada.

Para a última estimativa de erro geral, para cada autovalor, verificou-se o quão próxima de zero está $Av - \lambda v$.

$$A(H^T V) - (H^T V)\Lambda = 0 \tag{21}$$

2.3 | Teste B

Esse é o teste que faz o cálculo dos autovalores e autovetores da matriz em (22) e compara com os autovalores do resultado analítico, na equação (23). O código deste teste começa com o trecho da figura (11), que faz a entrada dos mesmos parâmetros do *teste A*, mas também tem uma entrada adicional: n, que é o tamanho da matriz A desejada.

O trecho da figura (12) faz a montagem da matriz A, de acordo com a equação (22).

Como essa matriz possui resultado analítico para os autovalores, uma estimativa de erro pode ser criada, cujo código está no trecho na figura (14). A função nesta figura, *EstimativasErroAnalitico(Lambda)*, recebe *Lambda*, vetor com os autovalores obtidos pelo método. Os autovalores reais são armazenados no vetor *autovalores_reais*, calculados pela fórmula em (23). O *for* nesta função compara os autovalores obtidos e reais, e libera uma estimativa de erro.

FIGURA 10 Função *EstimativasErroGerais* no arquivo do teste A

```
def EstimativasErroGerais(A, Lambda, HV):
    n = A.shape[0]

    print("===Estimativas de Erro Gerais===\n")

    # Faz uma estimativa de erro = max(abs(H*V*VT*HT-I))
    erro = np.max(np.abs(np.matmul(HV, np.transpose(HV)) - np.identity(n)))
    print("(Verificação de ortogonalidade)")
    print('max(abs((HT*V)*(HT*V)T-I)) = {0}'.format(erro))

    # Faz uma estimativa de erro = max(abs(matmul(A,H*V)-matmul(H*V,L)))
    L = np.zeros((n, n))

    for i in range(0, n):
        L[i, i] = Lambda[i]

    erro = np.max(np.abs(np.matmul(A, HV) - np.matmul(HV, L)))
    print('\nmax(abs(matmul(A,HT*V)-matmul(HT*V,L))) = {0}\n'.format(erro))

    # Faz uma estimativa de erro comparando (A*v)/v, com os autovalores obtidos
    erros = np.zeros(n)
    for i in range(0, n):
        erros[i] = math.sqrt(math.pow(np.mean(np.divide(np.matmul(A, HV[:, i]), HV[:, i])) - Lambda[i], 2))
        print('Autovalor obtido fazendo matdiv(matmul(A,v),v): {0:12.10f},'
              '  Autovalor obtido pelo método: {1:12.10f}, erro: {2}'
              .format(np.mean(np.divide(np.matmul(A, HV[:, i]), HV[:, i])), Lambda[i], erros[i]))
    print('Erro máximo = {0}\n'.format(np.max(erros)))

    # Faz estimativa de erro verificando o quão próximo de zero está A*v-lambda*v
    print("\nEstimativa de erro fazendo max(abs(A*v-lambda*v)) para cada autovalor")
    for i in range(0, n):
        erros[i] = np.max(np.abs(A.dot(HV[:, i])-Lambda[i]*HV[:, i]))
        print('Autovalor: {0:12.10f}, Erro {1:12.10f}'
              .format(Lambda[i], erros[i]))
    print('Erro máximo = {0}\n'.format(np.max(erros)))
```

$$A = \begin{bmatrix} n & n-1 & n-2 & \dots & 2 & 1 \\ n-1 & n-1 & n-2 & \dots & 2 & 1 \\ n-2 & n-2 & n-2 & \dots & 2 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 2 & 2 & 2 & \dots & 2 & 1 \\ 1 & 1 & 1 & \dots & 1 & 1 \end{bmatrix} \tag{22}$$

$$i = 1, 2, \dots, n \qquad \lambda_i = \frac{1}{2} \frac{1}{1 - \cos\left(\frac{(2i-1)\pi}{2n+1}\right)} \tag{23}$$

FIGURA 11 Trecho 0 do código do teste B

```
def teste_b():
    # Menu de seleção
    print('############')
    print('Teste b selecionado')
    print('############')

    epsilon = 0.1
    deslocamentos = True
    n = 20

    try:
        epsilon = float(input('epsilon = '))
        n = int(input('n = '))
        deslocamentos = input('Usar deslocamentos espectrais? (s,n)')
        if deslocamentos == 's':
            deslocamentos = True
        else:
            deslocamentos = False
    except:
        print('epsilon deve ser número real, pe: 1e-6 e n deve ser inteiro')
        return
```

FIGURA 12 Trecho 1 do código do teste B

```
A = np.ones((n,n))
for k in range(0,n):
    for j in range(0,k+1):
        A[k,j] = n-k
    for i in range(0,k):
        A[i,k] = n-k

resultados = aa.AutovalsAutovecs(A, epsilon, deslocamentos)
```

FIGURA 13 Trecho 2 do código do teste B

```
resultados = aa.AutovalsAutovecs(A, epsilon, deslocamentos)

print("\nForam necessárias k = {0} iterações no método QR\n".format(resultados[2]))

for i in range(0, A.shape[0]):
    # Configuração de impressão para ter mais dígitos
    #np.set_printoptions(formatter={'float': '{: 12.10f}'.format})

    # Impressão dos autovals/autovecs
    print('Autovalor: {0:12.10f}, Autovetor:'.format(resultados[0][i]), resultados[1][:, i])

print("\n")
print('############\n')
EstimativasErroGerais(A, resultados[0], resultados[1])
print('############')
EstimativasErroAnalitico(resultados[0])
print('############')
```

FIGURA 14 Trecho 3 do código do teste B, estimativa de erro analítico

```
6   def EstimativasErroAnalitico(Lambda):
7       print("===Estimativas de Erro Analíticas===\n")
8
9       n = Lambda.shape[0]
10
11      # Calcula os autovalores reais, pela fórmula analítica, e ordena eles do maior para o menor
12      autovalores_reais = 1/2*(1-np.cos((2*np.arange(n, 0, -1)-1)*np.pi/(2*n+1)))**(-1)
13      autovalores_reais = np.flip(np.sort(autovalores_reais))
14
15      autovalores_obtidos = np.copy(np.flip(np.sort(Lambda)))
16
17      erros = np.zeros(n)
18
19      # Faz a comparação entre os autovalores reais e os obtidos pelo método
20      print('Autovalor obtido | Autovalor real | erro')
21      for i in range(0, n):
22          erros[i] = math.sqrt(math.pow(autovalores_obtidos[i] - autovalores_reais[i], 2))
23          print('{0:12.10f}       {1:12.10f}     {2}'.format(autovalores_obtidos[i], autovalores_reais[i],erros[i]))
24      print('-----------------------------')
25      print('Erro máx: ', np.max(erros))
26      print('\n')
```

2.4 | Teste C

Esse é o teste que faz o cálculo das frequências e modos de vibração de uma treliça cuja informação é entrada via arquivo no programa.

O trecho do código do teste está na figura (15), que apenas abre o arquivo de entrada, se este estiver desbloqueado pelo sistema, e faz outras entradas de usuário pelo terminal.

Como convenção para o formato deste arquivo onde a treliça está descrita: os dados na primeira linha são o

número total de nós da treliça, o número de nós móveis e número de barras; na segunda linha são a densidade ρ, a área A transversal das barras e E, o módulo de elasticidade das barras. A partir daí, nas linhas estão as descrições de cada barra, onde, para cada linha, estão os nós dos extremos da barra e o ângulo e o comprimento da mesma.

Assim, o código da figura (16) faz a leitura das duas primeiras linhas do arquivo e extrai as suas informações.

O código da figura (17) faz a leitura das linhas que descrevem cada barra da treliça. Algo a se notar é que este código converte os índices i e j para começarem em zero.

No código da figura (18), a linha 87 calcula o coeficiente da equação (9). Mas, a partir da linha 94 as contribuições da matriz de rigidez da barra da iteração atual são somadas na matriz K. Os ifs servem apenas para verificar se a posição onde a contribuição vai ser feita existe, desta forma as contribuições dos nós fixos não entram na matriz.

No código da figura (19), as contribuições de massa da barra atual são adicionadas às massas dos nós i e j, no vetor *m[]*.

No código da figura (20), as matrizes M, da equação (8), $\tilde{M}$, da equação (12), e $\tilde{K}$, da equação (13), são calculadas.

No código da figura (21), são calculados os autovalores e autovetores de $\tilde{K}$. As frequências de vibração são obtidas fazendo a raiz quadrada dos autovalores. E os modos naturais de vibração são inseridos nas colunas da matriz *z*, calculados usando a equação (12). Por fim, uma lista *list* é criada com as frequências e modos naturais, para ser ordenada.

No código da figura (22), aparece um seletor. Se a seleção 0 é selecionada as frequências e modos naturais são impressos na tela. Se a seleção 1 é selecionada, matrizes importantes do programa são impressas em arquivos.

No código da figura (23), algumas estimativas de erro são calculadas. A primeira compara as frequências obtidas pelo algoritmo com as obtidas isolando ω na equação (11) (EET1). A segunda estimativa calcula o valor de (8) para $t = 0$, e verifica o quão próximo de zero está este resultado (EET2).

2.5 | Teste D

A única diferença entre o Teste D e o A é que, no D, a matriz não é mais uma pré-definida no software, agora a entrada é uma matriz em um arquivo, cujo caminho é pedido na execução deste teste. O código que implementa a leitura do arquivo está na figura (24).

3 | RESULTADOS

Para todos os teste foi utilizado *epsilon*=1e − 6 e *deslocamentos=True*.

3.1 | Teste A

Após a execução do teste A, o que está na figura (25) foi impresso no terminal. Os autovalores e autovetores obtidos, assim como estimativas de erro por autovalor estão na tabela (1). As estimativas de erro estão na tabela (2).

Os autovalores obtidos são exatamente os esperados analiticamente, para a matriz (20). Para a estimativa de erro (EEG3), apareceram alguns valores esdrúxulos, mas isso foi porque algumas coordenadas dos autovetores eram nulas, ou quase nulas. Pela (EEG1), verificou-se que a matriz $H^T V$ é ortogonal.

TABELA 1 Resultados do teste A

Autovalor	Autovetor				EEG3[1]	EEG4[2]
7.0000000000	0.6324555377	0.6324555377	0.3162277547	0.3162277547	6.69e-08	5.64e-08
-2.0000000000	7.0710674840e-01	-7.0710681397e-01	6.5565252130e-08	6.5565252130e-08	2.00	2.62e-07
2.0000000000	0.316227828	0.3162276814	-0.6324555377	-0.6324555377	6.69e-08	3.50e-07
-1.0000000000	0.	0.	-0.7071067812	0.7071067812	*inf*	2.22e-16

TABELA 2 Resultados do teste A, estimativas de erro gerais

EEG1	4.440892098500626e-16
EEG2	3.49636611107762e-07
EEG3	*inf*
EEG4	3.496366109967397e-07

3.2 | Teste B

Após a execução do teste B, o que está na figura (26) foi impresso no terminal. Os autovalores, autovetores obtidos e estimativas de erro gerais por autovalor estão nas tabelas (3) e (4). Mas, neste caso, ainda existe a estimativa de erro analítico, comparando os autovalores obtidos com os reais, dados pela fórmula (23), na tabela (6). Todas as estimativas de erro estão na tabela (5).

3.3 | Teste C

Para a execução do teste C foi usado o arquivo *files/input-c*. Os resultados obtidos estão nas tabelas (7), (8), (9) e (10), onde também está os valores das estimativas de erro para treliça por frequência. Os valores de (EET1) e (EET2) foram *4.71550265501719e-09* e *1.1824071407318115e-05*, respectivamente.

Então as cinco menores frequências obtidas foram: 24.5925477697, 92.0124446460, 94.7033653738, 142.8096971065 e 150.8221265108, que foram idênticas às obtidas por outros colegas da disciplina.

[1] Resultado por autovalor de Estimativa de Erro Geral 3 (EEG3)

[2] Resultado por autovalor de Estimativa de Erro Geral 4 (EEG4)

[3] Resultado por freq. de Estimativa de Erro de Treliça 1 (EET1)

[4] Resultado por freq. de Estimativa de Erro de Treliça 2 (EET2)

TABELA 3 Resultados do teste B, parte 1 da tabela

Autovalor	Autovetor[0:10]										EEG3'
170.4042675054	0.3121	0.3103	0.3066	0.3012	0.294	0.285	0.2744	0.2622	0.2484	0.2332	8.526512829121202e-14
19.008099491	0.3103	0.294	0.2622	0.2166	0.1596	0.0942	0.0239	-0.0477	-0.1168	-0.1797	4.2368952790639014e-10
6.8967848927	-0.3066	-0.2622	-0.1797	-0.0712	0.0477	0.1596	0.2484	0.3012	0.3103	0.2744	8.474874135799837e-10
3.5604828077	0.3012	0.2166	0.0712	-0.0942	-0.2332	-0.3066	-0.294	-0.1987	-0.0477	0.1168	4.440892098500626e-16
2.1880801951	-0.294	-0.1596	0.0477	0.2332	0.3121	0.2484	0.0712	-0.1386	-0.285	-0.3012	4.440892098500626e-15
1.4939898291	0.285	0.0942	-0.1596	-0.3066	-0.2484	-0.0239	0.2166	0.3121	0.1987	-0.0477	9.325873406851315e-15
1.0954523501	-0.2744	-0.0239	0.2484	0.294	0.0712	-0.2166	-0.3066	-0.1168	0.1797	0.3121	1.1546319456101628e-14
0.846121955	0.2622	-0.0477	-0.3012	-0.1987	0.1386	0.3121	0.1168	-0.2166	-0.294	-0.0239	3.849143226375417 7e-13
0.6802549888	-0.2484	0.1168	0.3103	0.0477	-0.285	-0.1987	0.1797	0.294	-0.0239	-0.3066	9.224399022400576e-12
0.5647697268	0.2332	-0.1797	-0.2744	0.1168	0.3012	-0.0477	-0.3121	-0.0239	0.3066	0.0942	7.56843476779067 7e-11
0.4815551224	-0.2166	0.2332	0.1987	-0.2484	-0.1797	0.2622	0.1596	-0.2744	-0.1386	0.285	2.0206198381167 44e-9
0.4200300188	0.1987	-0.2744	-0.0942	0.3103	-0.0239	-0.3012	0.1386	0.2484	-0.2332	-0.1596	1.6350185094715641e-9
0.3736873638	-0.1797	0.3012	-0.0239	-0.285	0.2166	0.1386	-0.3103	0.0712	0.2622	-0.2484	1.380604519596318e-9
0.3383591353	-0.1596	0.3121	-0.1386	-0.1797	0.3103	-0.1168	-0.1987	0.3066	-0.0942	-0.2166	5.491824961456615e-8
0.311288812	0.1386	-0.3066	0.2332	0.0239	-0.2622	0.294	-0.0942	-0.1797	0.3121	-0.1987	8.59629201155343e-10
0.2906095465	0.1168	-0.285	0.294	-0.1386	-0.0942	0.2744	-0.3012	0.1596	0.0712	-0.2622	6.362403170978581e-8
0.2750381895	0.0942	-0.2484	0.3121	-0.2622	0.1168	0.0712	-0.2332	0.3103	-0.2744	0.1386	4.860007013496315e-8
0.263690055	-0.0712	0.1987	-0.285	0.3121	-0.2744	0.1797	-0.0477	-0.0942	0.2166	-0.294	3.133206055716542e-8
0.2514735819	0.0239	-0.0712	0.1168	-0.1596	0.1987	-0.2332	0.2622	-0.285	0.3012	-0.3103	1.681994106582803e-8
0.255964433	-0.0477	0.1386	-0.2166	0.2744	-0.3066	0.3103	-0.285	0.2332	-0.1596	0.0712	4.585109519839037e-8

TABELA 4 Resultados do teste B, parte 2 da tabela

Autovalor	Autovetor[11:20]										EEG4'
170.4042675054	0.2166	0.1987	0.1797	0.1596	0.1386	0.1168	0.0942	0.0712	0.0477	0.0239	7.105427357601002e-14
19.008099491	-0.2332	-0.2744	-0.3012	-0.3121	-0.3066	-0.285	-0.2484	-0.1987	-0.1386	-0.0712	4.4085002315341626e-10
6.8967848927	0.1987	0.0942	-0.0239	-0.1386	-0.2332	-0.294	-0.3121	-0.285	-0.2166	-0.1168	4.4086312378510684e-10
3.5604828077	0.2484	0.3103	0.285	0.1797	0.0239	-0.1386	-0.2622	-0.3121	-0.2744	-0.1596	6.439293542825908e-15
2.1880801951	-0.1797	0.0239	0.2166	0.3103	0.2622	0.0942	-0.1168	-0.2744	-0.3066	-0.1987	5.10702591327572e-15
1.4939898291	-0.2622	-0.3012	-0.1386	0.1168	0.294	0.2744	0.0712	-0.1797	-0.3103	-0.2332	9.242606680004428e-15
1.0954523501	0.1596	-0.1386	-0.3103	-0.1987	0.0942	0.3012	0.2332	-0.0477	-0.285	-0.2622	1.1157741397482823e-14
0.846121955	0.2744	0.2484	-0.0712	-0.3066	-0.1797	0.1596	0.3103	0.0942	-0.2332	-0.285	3.1874503036988244e-13
0.6802549888	-0.1386	0.2332	0.2622	-0.0942	-0.3121	-0.0712	0.2744	0.2166	-0.1596	-0.3012	8.543887819456586e-12
0.5647697268	-0.285	-0.1596	0.2484	0.2166	-0.1987	-0.2622	0.1386	0.294	-0.0712	-0.3103	1.1335196670181347e-10
0.4815551224	0.1168	-0.294	-0.0942	0.3012	0.0712	-0.3066	-0.0477	0.3103	0.0239	-0.3121	8.852558686101 02e-10
0.4200300188	0.294	0.0477	-0.3121	0.0712	0.285	-0.1797	-0.2166	0.2622	0.1168	-0.3066	4.493602656641116e-9
0.3736873638	-0.0942	0.3121	-0.1168	-0.2332	0.2744	0.0477	-0.3066	0.1596	0.1987	-0.294	1.553261638753689e-8
0.3383591353	0.3012	-0.0712	-0.2332	0.294	-0.0477	-0.2484	0.285	-0.0239	-0.2622	0.2744	3.826017842822882e-8
0.311288812	-0.0712	0.285	-0.2744	0.0477	0.2166	-0.3103	0.1596	0.1168	-0.3012	0.2484	6.665583199283454e-8
0.2906095465	0.3066	-0.1797	-0.0477	0.2484	-0.3103	0.1987	0.0239	-0.2332	0.3121	-0.2166	6.020679924056971e-8
0.2750381895	0.0477	-0.2166	0.3066	-0.285	0.1596	0.0239	-0.1987	0.3012	-0.294	0.1797	2.3009008032204292e-8
0.263690055	0.3103	-0.2622	0.1596	-0.0239	-0.1168	0.2332	-0.3012	0.3066	-0.2484	0.1386	1.4937993425556684e-8
0.2514735819	0.3121	-0.3066	0.294	-0.2744	0.2484	-0.2166	0.1797	-0.1386	0.0942	-0.0477	1.990108046412 42e-8
0.255964433	0.0239	-0.1168	0.1987	-0.2622	0.3012	-0.3121	0.294	-0.2484	0.1797	-0.0942	2.767805612913926e-8

TABELA 5 Resultados do teste B, estimativas de erro

EEG1	6.661338147750939e-16
EEG2	6.665583193732338e-08
EEG3	6.362403170978581e-08
EEG4	6.665583199283454e-08
Analítica	1.3358203432289883e-12

TABELA 6 Resultados do teste B, estimativas de erro analítico

Autovalor obtido	Autovalor real	Erro
170.4042675054	170.4042675054	1.3358203432289883e-12
19.0080994910	19.0080994910	3.197442310920451e-14
6.8967848927	6.8967848927	3.552713678800501e-15
3.5604828077	3.5604828077	2.220446049250313e-15
2.1880801951	2.1880801951	4.440892098500626e-16
1.4939898291	1.4939898291	8.881784197001252e-16
1.0954523501	1.0954523501	2.220446049250313e-15
0.8461219550	0.8461219550	1.4432899320127035e-15
0.6802549888	0.6802549888	2.220446049250313e-16
0.5647697268	0.5647697268	7.771561172376096e-16
0.4815551224	0.4815551224	1.5543122344752192e-15
0.4200300188	0.4200300188	4.829470157119431e-15
0.3736873638	0.3736873638	4.2243986086987206e-14
0.3383591353	0.3383591353	2.6872948311051914e-13
0.3112888120	0.3112888120	4.954925358902074e-13
0.2906095465	0.2906095465	5.44286837822483e-13
0.2750381895	0.2750381895	2.5623947408348613e-13
0.2636900550	0.2636900550	1.98507876802978e-13
0.2559644330	0.2559644330	4.2482684037281615e-13
0.2514735819	0.2514735819	6.272204977619822e-13

TABELA 7 Resultados do teste C, parte 1

ω [rad/s]	Modo de vibração[0:5]					
24.5925477697	0.0034957235	-0.0006120484	0.0034957235	0.0006120484	0.0022690781	-0.0018130252
92.012444646	-0.0000062619	-0.0021861658	0.0000062619	-0.0021861658	0.0004729349	-0.0029851787
94.7033653738	-0.0007788999	0.0008083593	-0.0007788999	-0.0008083593	0.000640509	0.002837993
142.8096971065	0.003868795	-0.0019304009	0.003868795	0.0019304009	-0.0014881393	0.0026809534
150.8221265108	-0.0000898215	0.0019225429	0.0000898215	0.0019225429	0.0015051225	-0.0033876959
230.4611731646	-0.0007485096	0.0003656766	-0.0007485096	-0.0003656766	-0.0001220276	0.0006771622
287.2743983926	0.0001059467	-0.001564398	0.0001059467	0.001564398	-0.001840995	-0.0014064423
290.1346041148	0.0000228834	0.0025007637	-0.0000228834	0.0025007637	-0.002910344	-0.00025851
351.3220442713	0.0001401173	-0.0017087942	-0.0001401173	-0.0017087942	-0.0003895104	0.0004008576
382.7624227356	0.0005437123	-0.0008978069	-0.0005437123	-0.0008978069	-0.0033775975	-0.0007440779
387.8165281411	0.0018465153	0.0003828269	0.0018465153	-0.0003828269	0.000143563	-0.0004758856
422.9254861044	0.0022214713	0.0032473718	0.0022214713	-0.0032473718	-0.0021492381	-0.000113793
461.8703154513	0.0009321586	-0.0017578973	-0.0009321586	-0.0017578973	-0.0012561896	-0.0001620319
463.3723681767	0.0002109223	0.0017308938	0.0002109223	-0.0017308938	-0.0020751688	-0.0000371186
522.2472012372	0.0027851275	-0.0025663031	-0.0027851275	-0.0025663031	-0.0002352704	0.0000188598
534.9361880397	0.0002606818	0.0036429967	0.0002606818	-0.0036429967	-0.0001692521	0.0000589843
549.8234742845	0.0008479572	0.001732856	0.0008479572	-0.001732856	0.0038614272	-0.0003532573
559.7843134376	-0.0034130668	0.0002149628	0.0034130668	0.0002149628	-0.0025908442	0.0002859073
583.5226458801	-0.0000577567	0.0005024664	-0.0000577567	-0.0005024664	-0.0008245646	0.0005594669
592.9947336346	0.001292775	-0.000846855	-0.001292775	-0.000846855	-0.0016336844	-0.0001385487
624.4300841547	-0.0003468984	-0.00054253	0.0003468984	-0.00054253	-0.000947472	0.0016885028
657.5101803707	-0.0034552774	-0.0025352285	0.0034552774	-0.0025352285	-0.0000092976	-0.0020992635
665.5276300465	0.0000722167	0.0000139899	0.0000722167	-0.0000139899	0.0013443959	0.0039878492
678.0759905052	0.0017947419	0.001390632	-0.0017947419	0.001390632	-0.0017660163	-0.0029993288

TABELA 8 Resultados do teste C, parte 2

ω [rad/s]	Modo de vibração[6:11]					
24.5925477697	0.0022483549	-0.0005955082	0.0022483549	0.0005955082	0.0022690781	0.0018130252
92.012444646	0.0001760825	-0.0020507693	-0.0001760825	-0.0020507693	-0.0004729349	-0.0029851787
94.7033653738	0.000874589	0.0007135821	0.000874589	-0.0007135821	0.000640509	-0.002837993
142.8096971065	-0.0006903867	-0.0011429772	-0.0006903867	0.0011429772	-0.0014881393	-0.0026809534
150.8221265108	0.0004343333	0.0017974229	-0.0004343333	0.0017974229	-0.0015051225	-0.0033876959
230.4611731646	0.0003374827	-0.0000283074	0.0003374827	0.0000283074	-0.0001220276	-0.0006771622
287.2743983926	-0.0022697437	-0.0006466462	-0.0022697437	0.0006466462	-0.001840995	0.0014064423
290.1346041148	-0.0015505127	0.0010663089	0.0015505127	0.0010663089	0.002910344	-0.00025851
351.3220442713	0.0006320219	-0.0004696984	-0.0006320219	-0.0004696984	0.0003895104	0.0004008576
382.7624227356	-0.0018007531	-0.00057917	0.0018007531	-0.00057917	0.0033775975	-0.0007440779
387.8165281411	-0.001151882	0.0018484643	-0.001151882	-0.0018484643	0.000143563	0.0004758856
422.9254861044	0.0000121708	0.0020256896	0.0000121708	-0.0020256896	-0.0021492381	0.000113793
461.8703154513	-0.0005044492	0.0001983426	0.0005044492	0.0001983426	0.0012561896	-0.0001620319
463.3723681767	0.0006470921	-0.0004419124	0.0006470921	0.0004419124	-0.0020751688	0.0000371186
522.2472012372	-0.000088769	0.0015807827	0.000088769	0.0015807827	0.0002352704	0.0000188598
534.9361880397	-0.0003018475	-0.0028008217	-0.0003018475	0.0028008217	-0.0001692521	-0.0000589843
549.8234742845	-0.0022944868	-0.0000481504	-0.0022944868	0.0000481504	0.0038614272	0.0003532573
559.7843134376	0.0018375747	-0.0005278872	-0.0018375747	-0.0005278872	0.0025908442	0.0002859073
583.5226458801	0.0000475112	-0.0007675279	0.0000475112	0.0007675279	-0.0008245646	-0.0005594669
592.9947336346	0.0023782205	0.0015767636	-0.0023782205	0.0015767636	0.0016336844	-0.0001385487
624.4300841547	0.0009638286	0.0006591126	-0.0009638286	0.0006591126	0.000947472	0.0016885028
657.5101803707	-0.0008655596	0.001727185	0.0008655596	0.001727185	0.0000092976	-0.0020992635
665.5276300465	-0.0003304372	0.0001856939	-0.0003304372	-0.0001856939	0.0013443959	-0.0039878492
678.0759905052	0.0012394279	-0.000961818	-0.0012394279	-0.000961818	0.0017660163	-0.0029993288

TABELA 9 Resultados do teste C, parte 3

ω [rad/s]	Modo de vibração[12:17]					
24.5925477697	0.0009983716	-0.0018173117	0.0009960167	-0.0005071916	0.0009960167	0.0005071916
92.012444646	-0.0000041661	-0.0030871105	-0.0000040285	-0.0017439886	0.0000040285	-0.0017439886
94.7033653738	0.0024842072	0.0029408585	0.0023973145	0.0003109346	0.0023973145	-0.0003109346
142.8096971065	-0.0005565432	0.0029126203	-0.0005122764	0.0000202225	-0.0005122764	-0.0000202225
150.8221265108	-0.0006892737	-0.0037174914	-0.0006281251	0.0011294791	0.0006281251	0.0011294791
230.4611731646	0.0006684086	0.0008540733	0.0005299557	-0.0011524663	0.0005299557	0.0011524663
287.2743983926	0.0026447801	-0.002073951	0.0017935479	-0.000967308	0.0017935479	0.000967308
290.1346041148	-0.001351096	-0.0003848562	-0.0009075385	-0.0014941783	0.0009075385	-0.0014941783
351.3220442713	-0.0050080371	0.0007729104	-0.0025973383	0.0006246718	0.0025973383	0.0006246718
382.7624227356	0.0005632143	-0.0017359752	0.0002414063	0.0004952306	-0.0002414063	0.0004952306
387.8165281411	0.0015149696	-0.0011510572	0.0006263392	0.0022988117	0.0006263392	-0.0022988117
422.9254861044	-0.0009100743	-0.0003762713	-0.0002752271	-0.0008621256	-0.0002752271	0.0008621256
461.8703154513	0.0000324732	-0.0009642711	0.0000054567	0.0014962918	-0.0000054567	0.0014962918
463.3723681767	0.001223119	-0.0002282599	0.0001988982	-0.0016204767	0.0001988982	0.0016204767
522.2472012372	0.0022589833	-0.0002960986	-0.0001438845	0.0007214472	0.0001438845	0.0007214472
534.9361880397	0.001950478	-0.0005084357	-0.0002262774	0.0014030743	-0.0002262774	-0.0014030743
549.8234742845	-0.0013410385	0.0019735837	0.0002400363	-0.0013021853	0.0002400363	0.0013021853
559.7843134376	0.0028216095	-0.0012873021	-0.000626674	0.0010843601	0.000626674	0.0010843601
583.5226458801	-0.0061945706	-0.0017059787	0.0020314775	0.0005238351	0.0020314775	-0.0005238351
592.9947336346	0.0010889262	0.0003730377	-0.0004044345	-0.0018973502	0.0004044345	-0.0018973502
624.4300841547	-0.00436845	-0.0032430019	0.002274479	0.0000172939	-0.002274479	0.0000172939
657.5101803707	-0.0009736139	0.0030599431	0.0006679445	-0.0002789397	-0.0006679445	-0.0002789397
665.5276300465	0.0007670192	-0.0054822172	-0.0005579416	-0.0005858721	-0.0005579416	0.0005858721
678.0759905052	-0.0014566269	0.0037814476	0.001155352	0.0008691245	-0.001155352	0.0008691245

TABELA 10 Resultados do teste C, parte 4

ω[rad/s]	Modo de vibração[17:23]						EET1[-3]	EET2[-4]
24.5925477697	0.0009983716	0.0018173117	0.0000418157	-0.0002960973	0.0000418157	0.0002960973	1.4651391211373266e-11	1.21e-8
92.012444646	0.0000041661	-0.0030871105	0.000042118	-0.0010972486	-0.000042118	-0.0010972486	2.59865373664069992e-9	5.72e-8
94.7033653738	0.0024842072	-0.0029408585	0.0011566254	-0.0000563436	0.0011566254	0.0000563436	1.106457148125628e-10	1.193e-7
142.8096971065	-0.0005565432	-0.0029126203	-0.000231133	0.0000447487	-0.000231133	-0.0000447487	1.261923898709938e-11	1.07e-8
150.8221265108	0.0006892737	-0.0037174914	0.0000500037	0.0006787439	-0.0000500037	0.0006787439	1.0697931429604068e-10	1.126e-7
230.4611731646	0.0006684086	-0.0008540733	-0.003771083	-0.0016729497	-0.003771083	0.0016729497	4.5369574763753917e-10	0.0000018794
287.2743983926	0.0026447801	0.002073951	0.0006514934	-0.0007889861	0.0006514934	0.0007889861	3.8176040106918663e-10	0.000002885
290.1346041148	0.001351096	-0.0003848562	-0.0000235071	-0.0019618515	0.0000235071	-0.0019618515	1.1652900866465643e-11	1.63e-8
351.3220442713	0.0050080371	0.0007729104	0.000590943	0.0002745168	-0.000590943	0.0002745168	2.4442670110147446e-12	4.47e-8
382.7624227356	-0.0005632143	-0.0017359752	0.000864541	0.0026732542	-0.000864541	0.0026732542	1.7621459846850485e-12	1.07e-8
387.8165281411	0.0015149696	0.0011510572	-0.0017125816	0.0012698818	-0.0017125816	-0.0012698818	9.987388693843968e-11	5.253e-7
422.9254861044	-0.0009100743	0.0003762713	0.0008995021	-0.0016715054	0.0008995021	0.0016715054	3.211653165635653e-11	4.61e-7
461.8703154513	-0.0000324732	-0.0009642711	-0.0035571	-0.0010307891	0.0035571	-0.0010307891	2.8830982046201825e-10	1.434e-7
463.3723681767	0.001223119	0.0002282599	-0.0005924373	0.0033626697	-0.0005924373	-0.0033626697	8.765255188336596e-11	4.116e-7
522.2472012372	-0.0022589833	-0.0002960986	0.0021441351	-0.0016838455	-0.0021441351	-0.0016838455	2.2168933355715126e-11	2.05e-8
534.9361880397	0.001950478	0.0005084357	-0.0000281037	-0.0007521719	-0.0000281037	0.0007521719	1.15960574476048335e-11	2.51e-8
549.8234742845	-0.0013410385	-0.0019735837	0.0000348044	0.0006093607	0.0000348044	-0.0006093607	7.73070496506989e-12	2.79e-8
559.7843134376	-0.0028216095	-0.0012873021	0.0006159162	-0.0009358591	-0.0006159162	-0.0009358591	5.684341886080801e-13	2.98e-8
583.5226458801	-0.0061945706	0.0017059787	-0.0003113082	0.0002651123	-0.0003113082	-0.0002651123	1.4335910236695781e-10	2.116e-7
592.9947336346	-0.0010889262	0.0003730377	-0.0010537935	0.0015164879	0.0010537935	0.0015164879	1.55750967678613996e-11	2.189e-7
624.4300841547	0.00436845	-0.0032430019	0.0005255971	-0.0004140058	-0.0005255971	-0.0004140058	4.169692147115711e-9	0.000009004
657.5101803707	0.0009736139	0.0030599431	0.0000495367	0.0000383292	-0.0000495367	0.0000383292	1.2734062693198211e-9	0.000001166
665.5276300465	0.0007670192	0.0054822172	0.0000843538	0.0000442677	0.0000843538	-0.0000442677	4.71550265501719e-9	0.0000118241
678.0759905052	0.0014566269	0.0037814476	0.000342386	-0.000506023	-0.000342386	-0.000506023	2.2737367544323206e-13	2.98e-8

FIGURA 15 Trecho 0 do Teste C

```
def teste_c():
    print('\n#############')
    print('Teste c selecionado')
    print('#############\n')

    epsilon = 0.1
    deslocamentos = True
    f = 0 # Arquivo será inserido nessa variável

    n_total = 0 # Número de nós da treliça
    n_moveis = 0 # Número de nós móveis da treliça
    n_barras = 0 # Número de barras da treliça

    rho = 0 # Densidade rho [kg/m³]
    A = 0 # Área transversal das barras [m²]
    E = 0 # Módulo de elasticidade em Pa

    K = 0 # Matrix de rigidez total
    m = 0 # Vetor de massas dos nós
    M = 0 # Matriz onde as massas dos nós estarão na diagonal principal

    # Fazendo leitura das entradas de usuário
    try:
        nome_arquivo = str(input('Digite o nome do arquivo: '))
        with open(nome_arquivo) as f:
            f = f.read().split()
    except:
        print('Arquivo não existe ou está indisponível')
        return
    try:
        epsilon = float(input('epsilon = '))
        deslocamentos = input('Usar deslocamentos espectrais? (s,n)')
        if deslocamentos == 's':
            deslocamentos = True
        else:
            deslocamentos = False
    except:
        print('epsilon deve ser número real, pe: 1e-6')
        return
```

FIGURA 16 Trecho 1 do Teste C

```
## Faz a leitura das duas primeiras linhas do arquivo
try:
    n_total = int(f.pop(0)) # Número de nós da treliça
    n_moveis = int(f.pop(0)) # Número de nós móveis da treliça
    n_barras = int(f.pop(0)) # Número de barras da treliça

    rho = float(f.pop(0)) # Densidade rho [kg/m³]
    A = float(f.pop(0)) # Área transversal das barras [m²]
    E = float(f.pop(0)) # Módulo de elasticidade em GPa
    E = E*1e9 # Módulo de elasticidade em Pa

    K = np.zeros((n_moveis*2, n_moveis*2)) # Matrix de rigidez total
    m = np.zeros(n_moveis) # Vetor de massas dos nós
    M = np.zeros((n_moveis*2, n_moveis*2)) # Matriz onde as massas dos nós estarão na diagonal principal

except:
    print('Erro na leitura dos parâmetros das 2 primeiras linhas do arquivo')
    return
```

FIGURA 17 Trecho 2 do Teste C

```
71      # Uma iteração é realizada para cada barra
72      for barra in range(0, n_barras):
73          try:
74              # Extraindo cada linha do arquivo correspondente às barras
75              i = int(f[barra*4]) # Extrai o nó i
76              j = int(f[barra*4+1]) # Extrai o nó j
77              if i > n_total or j > n_total: # Se for detectado i ou j inválido, levantar exceçã
78                  raise
79              i = i-1 # Faz o índice começar em 0
80              j = j-1 # Faz o índice começar em 0
81              theta = float(f[barra*4+2]) # Extrai o ângulo da barra
82              L = float(f[barra*4+3]) # Extrai o comprimento da barra
83          except:
84              print('Erro na leitura dos parâmetros das barras da treliça')
85              return
```

FIGURA 18 Trecho 3 do Teste C

```
87          alfa = A*E/L # Coeficiente da equação
88          C = math.cos(theta*math.pi/180) # Cosseno da equação
89          S = math.sin(theta*math.pi/180) # Seno da equação
90
91          # Fazendo o cálculo de cada elemento da matriz da equação
92          # E somando à matriz K na posição correta
93          n = n_moveis*2 # Dimensão das matrizes K e M
94          if 0 <= 2*i < n:
95              K[2*i, 2*i] = K[2*i, 2*i]+alfa*C**2
96              if 0 <= 2*i+1 < n:
97                  K[2*i, 2*i+1] = K[2*i, 2*i+1]+alfa*C*S
98                  K[2*i+1, 2*i] = K[2*i+1, 2*i]+alfa*C*S
99              if 0 <= 2*j < n:
100                 K[2*i, 2*j] = K[2*i, 2*j]-alfa*C**2
101                 K[2*j, 2*i] = K[2*j, 2*i]-alfa*C**2
102             if 0 <= 2*j+1 < n:
103                 K[2*i, 2*j+1] = K[2*i, 2*j+1]-alfa*C*S
104                 K[2*j+1, 2*i] = K[2*j+1, 2*i]-alfa*C*S
105         if 0 <= 2*i+1 < n:
106             K[2*i+1, 2*i+1] = K[2*i+1, 2*i+1]+alfa*S**2
107             if 0 <= 2*j < n:
108                 K[2*i+1, 2*j] = K[2*i+1, 2*j]-alfa*C*S
109                 K[2*j, 2*i+1] = K[2*j, 2*i+1]-alfa*C*S
110             if 0 <= 2*j+1 < n:
111                 K[2*i+1, 2*j+1] = K[2*i+1, 2*j+1]-alfa*S**2
112                 K[2*j+1, 2*i+1] = K[2*j+1, 2*i+1]-alfa*S**2
113         if 0 <= 2*j < n:
114             K[2*j, 2*j] = K[2*j, 2*j]+alfa*C**2
115             if 1 <= 2*j+1 < n:
116                 K[2*j, 2*j+1] = K[2*j, 2*j+1]+alfa*C*S
117                 K[2*j+1, 2*j] = K[2*j+1, 2*j]+alfa*C*S
118         if 0 <= 2*j+1 < n:
119             K[2*j+1, 2*j+1] = K[2*j+1, 2*j+1]+alfa*S**2
```

FIGURA 19 Trecho 4 do Teste C

```
121         # Nestes dois ifs, as massas de cada nó são acumuladas no vetor m
122         if 0 <= i < n_moveis:
123             m[i] = m[i] + 1/2*rho*A*L
124         if 0 <= j < n_moveis:
125             m[j] = m[j] + 1/2*rho*A*L
```

FIGURA 20 Trecho 5 do Teste C

```
128     # Agora a matriz K e o vetor m estão completamente montados
129
130     # Agora serão montadas as matrizes M, tilde_M e tilde_K
131
132     tilde_M = np.zeros((n_moveis*2, n_moveis*2)) # Instancia tilde_M = M^(-1/2)
133
134     # Este for monta as matrizes M e tilde_M
135     for no in range(0, n_moveis):
136         M[no*2,no*2] = m[no]
137         M[no*2+1, no*2+1] = m[no]
138
139         tilde_M[no*2,no*2] = 1/math.sqrt(m[no])
140         tilde_M[no*2+1, no*2+1] = tilde_M[no*2,no*2]
141
142     # Esta multiplicação de matrizes monta a matriz tilde_K
143     tilde_K = np.matmul(np.matmul(tilde_M, K), tilde_M) # M^(-1/2)*K*M^(-1/2)
```

FIGURA 21 Trecho 6 do Teste C

```
145     # Aqui o método QR com Householder é aplicado
146     resultados = aa.AutovalsAutovecs(tilde_K, epsilon, deslocamentos)
147
148     # Calcula as frequências
149     w = np.zeros(n_moveis*2)
150     for i in range(n_moveis*2):
151         w[i] = math.sqrt(resultados[0][i]) # w = sqrt(lambda)
152
153     # Calcula os modos de vibração
154     z = np.zeros((n_moveis*2, n_moveis*2))
155     for j in range(n_moveis*2):
156         z[:,j] = tilde_M.dot(resultados[1][:,j]) # z = M^(-1/2)*y
157
158     # Monta uma lista com as frequencias e modos naturais
159     list = []
160     for i in range(0, n_moveis*2):
161         list.append([w[i], z[:, i]])
162
163     list = sorted(list,key=lambda x: x[0]) # Ordena essa lista da menor freq. para a maior
164
165     print("\n====================RESULTADOS DO TESTE C====================")
166
167     print("\nForam necessárias k = {0} iterações do método QR\n".format(resultados[2]))
```

FIGURA 22 Trecho 7 do Teste C

```
    print("\n====================RESULTADOS DO TESTE C====================")

    print("\nForam necessárias k = {0} iterações do método QR\n".format(resultados[2]))

    # Seletor de opções:
    def seletor():
        print('Selecione uma opção:')
        print('(0) Para imprimir as frequências e modos naturais no terminal')
        print('(1) Dump de matrizes e vetores do programa em arquivos')
        print('(2) Estimativas de erro')
        print('(qualquer outra) Para sair do teste')
        selecao = input('= ')

        if selecao == '0':
            # Esta seleção faz a apenas a impressão no terminal das freqs. e modos naturais
            print('\n')
            for i in range(0, n_moveis*2):
                # Configuração de impressão para ter mais dígitos
                np.set_printoptions(formatter={'float': '{: 12.10f}'.format})

                # Impressão dos autovals/autovecs
                print('freq: {0:12.10f},\nmodo:'.format(list[i][0]), list[i][1])
                print('\n')
            seletor()
        elif selecao == '1':
            # Esta seleção imprime as matrizes em arquivos
            print('\n')
            try:
                np.savetxt('m.txt', m)
                np.savetxt('K.txt', K)
                np.savetxt('tilde_K.txt', tilde_K)
                np.savetxt('M.txt', M)
                np.savetxt('tilde_M.txt', tilde_M)
                np.savetxt('w.txt', w)
                np.savetxt('z.txt', z)
                print('Arquivos criados com sucesso')
            except:
                print('Arquivos não foram criados com sucesso')
            seletor()
        elif selecao == '2':
```

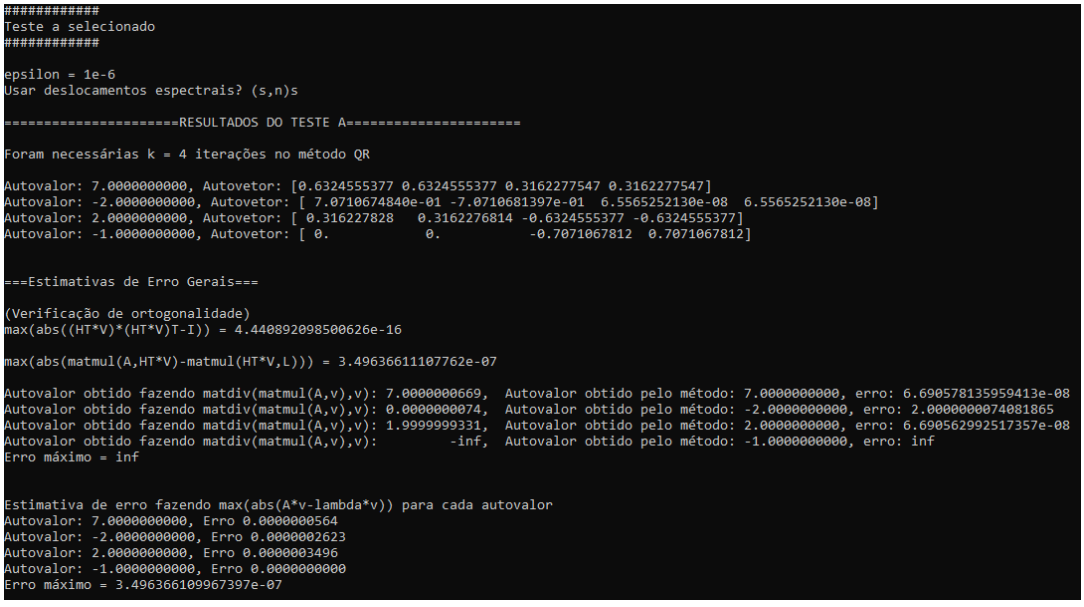
FIGURA 23 Trecho 8 do Teste C

```
204         elif selecao == '2':
205             # Esta seleção faz estimativas de erro
206             print('\n')
207             print('Estimativa de erro comparando sqrt((K*z)/(M*z)) com as frequências obtidas')
208             erros = np.zeros(n)
209             for i in range(0, n_moveis*2):
210                 erros[i] = np.abs(np.sqrt(np.max(np.divide(K.dot(z[:,i]), M.dot(z[:,i])))) - w[i])
211                 print('Freq. obtida fazendo sqrt((K*z)/(M*z)): {0:12.10f},'
212                       '  Freq. obtida pelo método: {1:12.10f}, erro: {2}'
213                       .format(np.sqrt(np.max(np.divide(K.dot(z[:,i]), M.dot(z[:,i])))), w[i], erros[i]))
214             print('Erro máximo = {0}\n'.format(np.max(erros)))
215 
216             print('\n')
217             print('Estimativa de erro calculando M*x\" + K*x, para t=0s')
218 
219             erros = np.zeros(n)
220             for i in range(0, n_moveis*2):
221                 erros[i] = np.max(np.abs(-(w[i]**2)*M.dot(z[:,i])+K.dot(z[:,i])))
222                 print('freq.: {0:12.10f}, |-w^2*M*z + K*z|: {1:12.10f}'
223                       .format(w[i],
224                       np.max(np.abs(-(w[i]**2)*M.dot(z[:,i])+K.dot(z[:,i]))))
225                     )
226             print('Erro máximo = {0}\n'.format(np.max(erros)))
227             print('\n')
228             seletor()
229         else:
230             return
231     seletor()
232     return
```

FIGURA 24 Trecho do Teste D

```
17
18      try:
19          nome_arquivo = str(input('Digite o nome do arquivo: '))
20          with open(nome_arquivo) as f:
21              f = f.read().split()
22      except:
23          print('Arquivo não existe ou está indisponível')
24          return
25
26      try:
27          epsilon = float(input('epsilon = '))
28          deslocamentos = input('Usar deslocamentos espectrais? (s,n)')
29          if deslocamentos == 's':
30              deslocamentos = True
31          else:
32              deslocamentos = False
33      except:
34          print('epsilon deve ser número real, pe: 1e-6')
35          return
36
37      try:
38          n = int(f.pop(0)) # Número de linhas da matriz quadrada
39          A = np.zeros((n,n))
40      except:
41          print('Erro na leitura do número de linhas da matriz')
42          return
43      try:
44          for i in range(0,n):
45              for j in range(0,n):
46                  A[i,j] = float(f[j+i*n])
47      except:
48          print('Arquivo mal formatado')
49          return
50
```

FIGURA 25 Resultados do teste A



```
############
Teste a selecionado
############

epsilon = 1e-6
Usar deslocamentos espectrais? (s,n)s

=====================RESULTADOS DO TESTE A=====================

Foram necessárias k = 4 iterações no método QR

Autovalor: 7.0000000000, Autovetor: [0.6324555377 0.6324555377 0.3162277547 0.3162277547]
Autovalor: -2.0000000000, Autovetor: [ 7.0710674840e-01 -7.0710681397e-01  6.5565252130e-08  6.5565252130e-08]
Autovalor: 2.0000000000, Autovetor: [ 0.316227828   0.3162276814 -0.6324555377 -0.6324555377]
Autovalor: -1.0000000000, Autovetor: [ 0.           0.          -0.7071067812  0.7071067812]


===Estimativas de Erro Gerais===

(Verificação de ortogonalidade)
max(abs((HT*V)*(HT*V)T-I)) = 4.440892098500626e-16

max(abs(matmul(A,HT*V)-matmul(HT*V,L))) = 3.49636611107762e-07

Autovalor obtido fazendo matdiv(matmul(A,v),v): 7.0000000669,  Autovalor obtido pelo método: 7.0000000000, erro: 6.690578135959413e-08
Autovalor obtido fazendo matdiv(matmul(A,v),v): 0.0000000074,  Autovalor obtido pelo método: -2.0000000000, erro: 2.0000000074081865
Autovalor obtido fazendo matdiv(matmul(A,v),v): 1.9999999331,  Autovalor obtido pelo método: 2.0000000000, erro: 6.690562992517357e-08
Autovalor obtido fazendo matdiv(matmul(A,v),v):         -inf,  Autovalor obtido pelo método: -1.0000000000, erro: inf
Erro máximo = inf


Estimativa de erro fazendo max(abs(A*v-lambda*v)) para cada autovalor
Autovalor: 7.0000000000, Erro 0.0000000564
Autovalor: -2.0000000000, Erro 0.0000002623
Autovalor: 2.0000000000, Erro 0.0000003496
Autovalor: -1.0000000000, Erro 0.0000000000
Erro máximo = 3.496366109967397e-07
```

FIGURA 26 Resultados do teste B