Escola Politécnica da Universidade de São Paulo - EPUSP

Relatório 2 - O Algoritmo QR

Turma: 03	
Nome:	**NUSP:**
Enzo Bustos Da Silva	11261531
Ana Alice Climaco Barbosa	11302348

São Paulo
2021

Apêndice

Relatório 2 - O Algoritmo QR

Enzo Bustos Da Silva, Ana Alice Climaco Barbosa

Escola Politécnica, Universidade de São Paulo, São Paulo, SP, Brazil
{enzobustos, anaclimacobarbosa}@usp.br

1 Introdução

Dentro da nossa graduação temos diversas matérias de Cálculo, Introdução a Computação e Física, o objetivo deste trabalho de Cálculo Numérico é criar um algoritmo que coloca em prática um pouco de tudo que já foi visto nessas matérias.

Durante o primeiro exercício programa visamos implementar um método que seria capaz de extrair autovalores e autovetores de matrizes simétricas tridiagonais, que são comuns no âmbito de engenharia e necessários para a solução de problemas, como o de sistemas com n massas e $n+1$ molas que além de de serem muito úteis para a simulação de outros fenômenos também foram o principal tema de resolução da primeira etapa deste projeto.

Agora neste segundo relatório, queremos ir um passo além e generalizar um pouco o algoritmo QR anterior através de transformações de Householder, com elas podemos transformar uma matriz qualquer simétrica em uma matriz tridiagonal simétrica e então usar nosso algoritmo anterior para calcular os autovalores e autovetores.

Como no último relatório fomos até tarefa C, este começará pelo item D.

2 Tarefa D

O primeiro exercício desta segunda parte consiste em uma série de testes que deveremos realizar em duas matrizes simétricas quaisquer para testar o algoritmo de tridiagonalização por Householder.

2.1 Matriz 1

A primeira matriz em que devemos realizar a bateria de testes está mostrada abaixo:

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

Temos os autovalores mostrados a seguir:

λ	Autovalores
$\lambda 1$	7.0
$\lambda 2$	2.0
$\lambda 3$	-2.0
$\lambda 4$	-1.0

Vale ressaltar que os autovalores obtidos correspondem aos mesmos valores destacados no enunciado, o que é um bom indicativo que está correndo tudo certo com nosso programa. Vamos agora aos autovetores (que estão na horizontal):

Λ	Posição 0	Posição 1	Posição 2	Posição 3
$\Lambda 1$	6.32456e-01	6.32456e-01	3.16228e-01	3.16228e-01
$\Lambda 2$	3.16228e-01	3.16228e-01	-6.32456e-01	-6.32456e-01
$\Lambda 3$	-7.07107e-01	7.07107e-01	2.02865e-16	6.27327e-17
$\Lambda 4$	7.52975e-18	4.74544e-17	-7.07107e-01	7.07107e-01

É pouco intuitivo ver dessa forma, então vamos realizar a normalização destes autovetores:

Λ	Posição 0	Posição 1	Posição 2	Posição 3
$\Lambda 1$	2.0	2.0	1.0	1.0
$\Lambda 2$	-1.0	-1.0	2.0	2.0
$\Lambda 3$	1.0	-1.0	-0.0	-0.0
$\Lambda 4$	-0.0	-0.0	1.0	-1.0

O proximo teste que precisavamos checar era se era possível verificar que $Av = \lambda v$ para cada um dos autovalores λ de A e seu autovetor correspondente:

```
====================== Comparação de Av = λv ======================

================= Para o autovetor: [2. 2. 1. 1.] =================
A @ v:
[14. 14.  7.  7.]
λ * v:
[14. 14.  7.  7.]
São iguais:  [ True  True  True  True]

================= Para o autovetor: [-1.  1. -0. -0.] =================
A @ v:
[ 2. -2.  0.  0.]
λ * v:
[ 2. -2.  0.  0.]
São iguais:  [ True  True  True  True]

================= Para o autovetor: [-1. -1.  2.  2.] =================
A @ v:
[-2. -2.  4.  4.]
λ * v:
[-2. -2.  4.  4.]
São iguais:  [ True  True  True  True]

================= Para o autovetor: [-0. -0.  1. -1.] =================
A @ v:
[ 0.  0. -1.  1.]
λ * v:
[ 0.  0. -1.  1.]
São iguais:  [ True  True  True  True]
```

Figura 1. Comparações realizadas pelo algoritmo de $Av = \lambda v$

Por último temos que checar a ortogonalidade da nossa matriz formada pelos autovetores de A, para isso basta checarmos se $\Lambda \cdot \Lambda^T = I$:

$$\Lambda \cdot \Lambda^T = \begin{bmatrix} 0.63246 & 0.31623 & -0.70711 & 0. \\ 0.63246 & 0.31623 & 0.70711 & 0. \\ 0.31623 & -0.63246 & 0. & -0.70711 \\ 0.31623 & -0.63246 & 0. & 0.70711 \end{bmatrix} \cdot \begin{bmatrix} 0.63246 & 0.63246 & 0.31623 & 0.31623 \\ 0.31623 & 0.31623 & -0.63246 & -0.63246 \\ -0.70711 & 0.70711 & 0. & 0. \\ 0. & 0. & -0.70711 & 0.70711 \end{bmatrix}$$

$$\Lambda \cdot \Lambda^T = \begin{bmatrix} 1.00000000e+00 & 2.22044605e-16 & -9.99354621e-18 & -3.90344168e-17 \\ 2.22044605e-16 & 1.00000000e+00 & 4.70714223e-16 & 2.72203131e-16 \\ -9.99354621e-18 & 4.70714223e-16 & 1.00000000e+00 & -7.77156117e-16 \\ -3.90344168e-17 & 2.72203131e-16 & -7.77156117e-16 & 1.00000000e+00 \end{bmatrix}$$

É um pouco difícil entender o que está acontecendo com todos esses algarismos, para facilitar um pouco vamos considerar todos os valores menores que $10^{-15} = 0$, dessa forma ficará mais fácil de comprovarmos o resultado:

$$\Lambda \cdot \Lambda^T = \begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

Certo, com isso conseguimos provar a ortogonalidade de Λ e também finalizamos a bateria de testes para a primeira matriz, vamos para o segundo caso.

2.2 Matriz 2

A segunda matriz que devemos usar para realizar nossos testes é da forma:

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

Para nossos testes foi pedido para que usassemos $n = 20$, dessa maneira obtemos a matriz mostrada abaixo:

```
========================= Matriz A =========================
[[20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [19 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [18 18 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [17 17 17 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [16 16 16 16 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [15 15 15 15 15 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [14 14 14 14 14 14 14 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [13 13 13 13 13 13 13 13 12 11 10  9  8  7  6  5  4  3  2  1]
 [12 12 12 12 12 12 12 12 12 11 10  9  8  7  6  5  4  3  2  1]
 [11 11 11 11 11 11 11 11 11 11 10  9  8  7  6  5  4  3  2  1]
 [10 10 10 10 10 10 10 10 10 10 10  9  8  7  6  5  4  3  2  1]
 [ 9  9  9  9  9  9  9  9  9  9  9  9  8  7  6  5  4  3  2  1]
 [ 8  8  8  8  8  8  8  8  8  8  8  8  8  7  6  5  4  3  2  1]
 [ 7  7  7  7  7  7  7  7  7  7  7  7  7  7  6  5  4  3  2  1]
 [ 6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  5  4  3  2  1]
 [ 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  4  3  2  1]
 [ 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  3  2  1]
 [ 3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  2  1]
 [ 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  1]
 [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]]
```

Figura 2. Matriz gerada usando $n = 20$

Vamos então começar explicitando os autovalores que foram obtidos:

λ	λCalculado	λAnalítico	Igual?
$\lambda 1$	170.40427	170.40427	True
$\lambda 2$	19.00810	19.00810	True
$\lambda 3$	6.89678	6.89678	True
$\lambda 4$	3.56048	3.56048	True
$\lambda 5$	2.18808	2.18808	True
$\lambda 6$	1.49399	1.49399	True
$\lambda 7$	1.09545	1.09545	True
$\lambda 8$	0.84612	0.84612	True
$\lambda 9$	0.68025	0.68025	True
$\lambda 10$	0.56477	0.56477	True
$\lambda 11$	0.48156	0.48156	True
$\lambda 12$	0.42003	0.42003	True
$\lambda 13$	0.37369	0.37369	True
$\lambda 14$	0.33836	0.33836	True
$\lambda 15$	0.31129	0.31129	True
$\lambda 16$	0.29061	0.29061	True
$\lambda 17$	0.27504	0.27504	True
$\lambda 18$	0.26369	0.26369	True
$\lambda 19$	0.25596	0.25596	True
$\lambda 20$	0.25147	0.25147	True

Agora vamos passar aos autovetores que foram obtido da matrix A:

Λ	Posição 0	Posição 1	Posição 2	Posição 3	Posição 4	Posição 5	Posição 6	Posição 7	Posição 8	Posição 9
$\Lambda 1$	0.31212	0.31029	0.30663	0.30118	0.29396	0.28502	0.27440	0.26217	0.24841	0.23318
$\Lambda 2$	0.31029	0.29396	0.26217	0.21659	0.15962	0.09424	0.02391	-0.04768	-0.11676	-0.17970
$\Lambda 3$	0.30663	0.26217	0.17970	0.07117	-0.04768	-0.15962	-0.24841	-0.30118	-0.31029	-0.27440
$\Lambda 4$	-0.30118	-0.21659	-0.07117	0.09424	0.23318	0.30663	0.29396	0.19873	0.04768	-0.11676
$\Lambda 5$	-0.29396	-0.15962	0.04768	0.23318	0.31212	0.24841	0.07117	-0.13859	-0.28502	-0.30118
$\Lambda 6$	0.28502	0.09424	-0.15962	-0.30663	-0.24841	-0.02391	0.21659	0.31212	0.19873	-0.04768
$\Lambda 7$	-0.27440	-0.02391	0.24841	0.29396	0.07117	-0.21659	-0.30663	-0.11676	0.17970	0.31212
$\Lambda 8$	-0.26217	0.04768	0.30118	0.19873	-0.13859	-0.31212	-0.11676	0.21659	0.29396	0.02391
$\Lambda 9$	0.24841	-0.11676	-0.31029	-0.04768	0.28502	0.19873	-0.17970	-0.29396	0.02391	0.30663
$\Lambda 10$	-0.23318	0.17970	0.27440	-0.11676	-0.30118	0.04768	0.31212	0.02391	-0.30663	-0.09424
$\Lambda 11$	0.21659	-0.23318	-0.19873	0.24841	0.17970	-0.26217	-0.15962	0.27440	0.13859	-0.28502
$\Lambda 12$	-0.19873	0.27440	0.09424	-0.31029	0.02391	0.30118	-0.13859	-0.24841	0.23318	0.15962
$\Lambda 13$	0.17970	-0.30118	0.02391	0.28502	-0.21659	-0.13859	0.31029	-0.07117	-0.26217	0.24841
$\Lambda 14$	-0.15962	0.31212	-0.13859	-0.17970	0.31029	-0.11676	-0.19873	0.30663	-0.09424	-0.21659
$\Lambda 15$	-0.13859	0.30663	-0.23318	-0.02391	0.26217	-0.29396	0.09424	0.17970	-0.31212	0.19873
$\Lambda 16$	0.11676	-0.28502	0.29396	-0.13859	-0.09424	0.27440	-0.30118	0.15962	0.07117	-0.26217
$\Lambda 17$	0.09424	-0.24841	0.31212	-0.26217	0.11676	0.07117	-0.23318	0.31029	-0.27440	0.13859
$\Lambda 18$	-0.07117	0.19873	-0.28502	0.31212	-0.27440	0.17970	-0.04768	-0.09424	0.21659	-0.29396
$\Lambda 19$	-0.02391	0.07117	-0.11676	0.15962	-0.19873	0.23318	-0.26217	0.28502	-0.30118	0.31029
$\Lambda 20$	-0.04768	0.13859	-0.21659	0.27440	-0.30663	0.31029	-0.28502	0.23318	-0.15962	0.07117

Λ	Posição 10	Posição 11	Posição 12	Posição 13	Posição 14	Posição 15	Posição 16	Posição 17	Posição 18	Posição 19
$\Lambda 1$	0.21659	-0.19873	0.17970	-0.15962	-0.13859	0.11676	0.09424	-0.07117	-0.02391	-0.04768
$\Lambda 2$	-0.23318	0.27440	-0.30118	0.31212	0.30663	-0.28502	-0.24841	0.19873	0.07117	0.13859
$\Lambda 3$	-0.19873	0.09424	0.02391	-0.13859	-0.23318	0.29396	0.31212	-0.28502	-0.11676	-0.21659
$\Lambda 4$	0.24841	-0.31029	0.28502	-0.17970	-0.02391	-0.13859	-0.26217	0.31212	0.15962	0.27440
$\Lambda 5$	0.17970	0.02391	-0.21659	0.31029	0.26217	-0.09424	0.11676	-0.27440	-0.19873	-0.30663
$\Lambda 6$	-0.26217	0.30118	-0.13859	-0.11676	-0.29396	0.27440	0.07117	0.17970	0.23318	0.31029
$\Lambda 7$	-0.15962	-0.13859	0.31029	-0.19873	0.09424	-0.30118	-0.23318	-0.04768	-0.26217	-0.28502
$\Lambda 8$	0.27440	-0.24841	-0.07117	0.30663	0.17970	0.15962	0.31029	-0.09424	0.28502	0.23318
$\Lambda 9$	0.13859	0.23318	-0.26217	-0.09424	-0.31212	0.07117	-0.27440	0.21659	-0.30118	-0.15962
$\Lambda 10$	-0.28502	0.15962	0.24841	-0.21659	0.19873	-0.26217	0.13859	-0.29396	0.31029	0.07117
$\Lambda 11$	-0.11676	-0.29396	0.09424	0.30118	0.07117	0.30663	0.04768	0.31029	-0.31212	0.02391
$\Lambda 12$	0.29396	-0.04768	-0.31212	-0.07117	-0.28502	-0.17970	-0.21659	-0.26217	0.30663	-0.11676
$\Lambda 13$	0.09424	0.31212	0.11676	-0.23318	0.27440	-0.04768	0.30663	0.15962	-0.29396	0.19873
$\Lambda 14$	-0.30118	-0.07117	0.23318	0.29396	-0.04768	0.24841	-0.28502	-0.02391	0.27440	-0.26217
$\Lambda 15$	-0.07117	-0.28502	-0.27440	-0.04768	-0.21659	-0.31029	0.15962	-0.11676	-0.24841	0.30118
$\Lambda 16$	0.30663	0.17970	-0.04768	-0.24841	0.31029	0.19873	0.02391	0.23318	0.21659	-0.31212
$\Lambda 17$	0.04768	0.21659	0.30663	0.28502	-0.15962	0.02391	-0.19873	-0.30118	-0.17970	0.29396
$\Lambda 18$	-0.31029	-0.26217	-0.15962	-0.02391	-0.11676	-0.23318	0.30118	0.30663	0.13859	-0.24841
$\Lambda 19$	-0.02391	-0.11676	-0.19873	-0.26217	0.30118	0.31212	-0.29396	-0.24841	-0.09424	0.17970
$\Lambda 20$	0.31212	0.30663	0.29396	0.27440	-0.24841	-0.21659	0.17970	0.13859	0.04768	-0.09424

Infelizmente os vetores são muito grandes e a visualização no relatório acaba ficando truncada, de qualquer forma só com essa primeira parte já é possível perceber que a propriedade de $A \cdot \nu = \lambda \cdot \nu$, para uma visualização mais detalhada convido a abrir o Jupyter Notebook que acompanha este relatório.

Λ	$A \cdot \nu$	$\lambda \cdot \nu$	Iguais?
$\Lambda 0$	[53.18629, 52.87417, 52.25177, 51.32273, 50.09...	[53.18629, 52.87417, 52.25177, 51.32273, 50.09...	True
$\Lambda 1$	[5.89796, 5.58767, 4.98342, 4.117, 3.03399, 1....	[5.89796, 5.58767, 4.98342, 4.117, 3.03399, 1....	True
$\Lambda 2$	[2.11479, 1.80816, 1.23935, 0.49084, -0.32884,...	[2.11479, 1.80816, 1.23935, 0.49084, -0.32884,...	True
$\Lambda 3$	[-1.07235, -0.77117, -0.2534, 0.33555, 0.83025...	[-1.07235, -0.77117, -0.2534, 0.33555, 0.83025...	True
$\Lambda 4$	[-0.64321, -0.34925, 0.10433, 0.51023, 0.68294...	[-0.64321, -0.34925, 0.10433, 0.51023, 0.68294...	True
$\Lambda 5$	[0.42581, 0.1408, -0.23846, -0.45811, -0.37112...	[0.42581, 0.1408, -0.23846, -0.45811, -0.37112...	True
$\Lambda 6$	[-0.30059, -0.02619, 0.27212, 0.32202, 0.07796...	[-0.30059, -0.02619, 0.27212, 0.32202, 0.07796...	True
$\Lambda 7$	[-0.22183, 0.04034, 0.25484, 0.16815, -0.11727...	[-0.22183, 0.04034, 0.25484, 0.16815, -0.11727...	True
$\Lambda 8$	[0.16898, -0.07943, -0.21107, -0.03243, 0.1938...	[0.16898, -0.07943, -0.21107, -0.03243, 0.1938...	True
$\Lambda 9$	[-0.1317, 0.10149, 0.15497, -0.06594, -0.1701,...	[-0.1317, 0.10149, 0.15497, -0.06594, -0.1701,...	True
$\Lambda 10$	[0.1043, -0.11229, -0.0957, 0.11962, 0.08654, ...	[0.1043, -0.11229, -0.0957, 0.11962, 0.08654, ...	True
$\Lambda 11$	[-0.08347, 0.11526, 0.03958, -0.13033, 0.01004...	[-0.08347, 0.11526, 0.03958, -0.13033, 0.01004...	True
$\Lambda 12$	[0.06715, -0.11255, 0.00893, 0.10651, -0.08094...	[0.06715, -0.11255, 0.00893, 0.10651, -0.08094...	True
$\Lambda 13$	[-0.05401, 0.10561, -0.04689, -0.0608, 0.10499...	[-0.05401, 0.10561, -0.04689, -0.0608, 0.10499...	True
$\Lambda 14$	[-0.04314, 0.09545, -0.07259, -0.00744, 0.0816...	[-0.04314, 0.09545, -0.07259, -0.00744, 0.0816...	True
$\Lambda 15$	[0.03393, -0.08283, 0.08543, -0.04028, -0.0273...	[0.03393, -0.08283, 0.08543, -0.04028, -0.0273...	True
$\Lambda 16$	[0.02592, -0.06832, 0.08584, -0.07211, 0.03211...	[0.02592, -0.06832, 0.08584, -0.07211, 0.03211...	True
$\Lambda 17$	[-0.01877, 0.0524, -0.07516, 0.0823, -0.07236,...	[-0.01877, 0.0524, -0.07516, 0.0823, -0.07236,...	True
$\Lambda 18$	[-0.00601, 0.0179, -0.02936, 0.04014, -0.04998...	[-0.00601, 0.0179, -0.02936, 0.04014, -0.04998...	True
$\Lambda 19$	[-0.0122, 0.03548, -0.05544, 0.07024, -0.07849...	[-0.0122, 0.03548, -0.05544, 0.07024, -0.07849...	True

Para fechar essa seção temos o teste de ortogonalidade para Λ, pelo fato da matriz ser muito grande aqui dentro do relatório ficará apenas a matriz já arredondada, considerando $10^{-12} = 0$, assim temos:

```
orthogonal = Λ.T @ Λ
orthogonal[abs(orthogonal) < 1e-12] = 0
with np.printoptions(threshold=np.inf):
    print(orthogonal)
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

Figura 3. Matriz identidade obtida por $\Lambda \cdot \Lambda^T$, observe que o primeiro valor que está com o T no código, pois como falado no primeiro relatório os autovetores estão nas linhas para facilitar prints e outras manipulações

3 Tarefa E

A tarefa final deste Exercício Programa, traz a tona um estudo de caso em que o programa desenvolvido pode ser usado para poder calcular as frequências e os modos de vibração de uma estrutura de treliças, a estrutura em questão pode ser conferida na imagem abaixo:

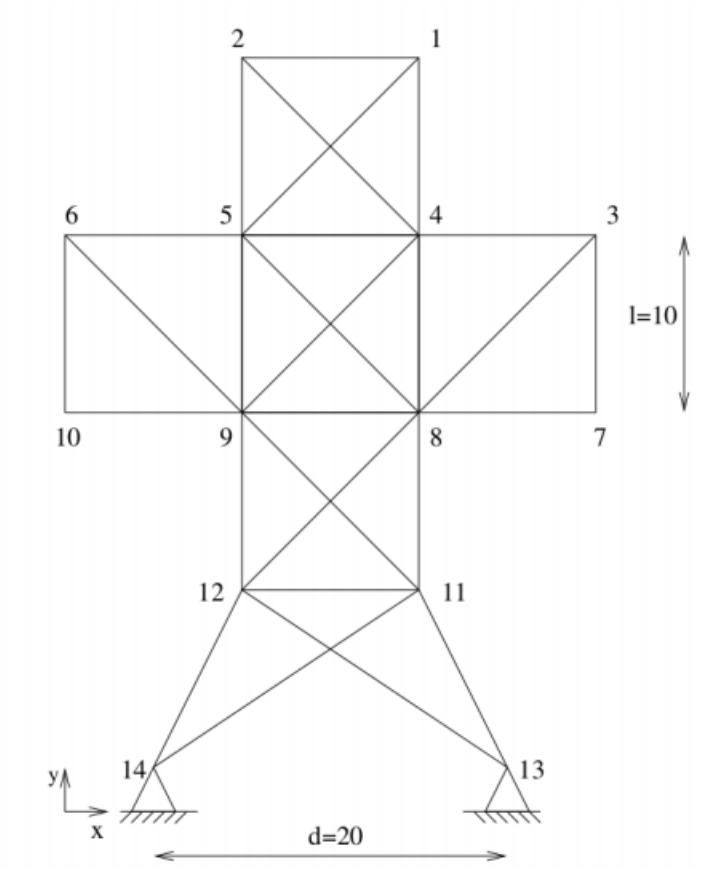

Figura 4. Estrutura de treliças que será desenvolvida nessa tarefa

Dentro do nosso estudo de caso consideramos não só que os nós fixos apresentam deslocamentos nulos e não contribuem para a energia do sistema, mas também que a massa de cada barra está concentrada estritamente nos seu extremos.

Vale ressaltar que para a execução do programa de forma correta recebemos um arquivo que continua as principais informações da estrutura, bem como in-

formações sobre os materiais que a compunham. Entre essas informações a que nossa estrutura é composta por 28 barras que estão conectadas por 12 nós com os pontos 13 e 14 sendo pontos fixos do nosso sistema.

Vamos então começar os cálculos, precisamos então primeiramente usar as fórmular fornecidas no enunciado para determinar a matriz de massa M, isso pode ser feito através de:

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

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

Observe que m_i é a força peso que cada um dos nós está sujeita e pode ser calculada pela fórmula $m_i = 0.5(A\rho)L$, o valor de L é basicamente o único que muda nessa fórmula dado que ele se relaciona com quantos comprimentos de barra um nó pode "enxergar".

Essas informações de constantes podem ser lidas no arquivo 'input-c.txt' que foi fornecido, além dessas informações também existem dados importantes sobre cada uma das barra, tudo isso pode ser conferido na tabela a seguir:

Extremo_1	Extremo_2	Theta	Comprimento
1	2	0.00000	10.00000
1	4	90.00000	10.00000
1	5	45.00000	14.14214
2	5	90.00000	10.00000
2	4	135.00000	14.14214
3	4	0.00000	10.00000
3	7	90.00000	10.00000
3	8	45.00000	14.14214
4	5	0.00000	10.00000
4	8	90.00000	10.00000
4	9	45.00000	14.14214
5	6	0.00000	10.00000
5	9	90.00000	10.00000
5	8	135.00000	14.14214
6	9	135.00000	14.14214
6	10	90.00000	10.00000
7	8	0.00000	10.00000
8	9	0.00000	10.00000
8	11	90.00000	10.00000
8	12	45.00000	14.14214
9	10	0.00000	10.00000
9	11	135.00000	14.14214
9	12	90.00000	10.00000
11	12	0.00000	10.00000
11	13	116.56505	11.18034
11	14	33.69007	18.02776
12	13	146.30993	18.02776
12	14	63.43495	11.18034

Com esses valores conseguimos então chegar na Matriz de massa M, como ela é uma matriz diagonal abaixo serão explicitados apenas os valores da diagonal principal:

Índice	Massa
1	13315.43289
2	13315.43289
3	13315.43289
4	13315.43289
5	13315.43289
6	13315.43289
7	26630.86579
8	26630.86579
9	26630.86579
10	26630.86579
11	13315.43289
12	13315.43289
13	7800.00000
14	7800.00000
15	32146.29868
16	32146.29868
17	32146.29868
18	32146.29868
19	7800.00000
20	7800.00000
21	24706.59044
22	24706.59044
23	24706.59044
24	24706.59044

Outra matriz importante que podemos obter a partir das informações básicas de cada uma das barras é a matriz de rigidez total K explicitada abaixo:

Posição 0	Posição 1	Posição 2	Posição 3	Posição 4	Posição 5	Posição 6	Posição 7
2.707e+09	7.071e+08	-2.000e+09	0	0	0	-7.499e-24	-1.225e-07
7.071e+08	2.707e+09	0	0	0	0	-1.225e-07	-2.000e+09
-2.000e+09	0	2.707e+09	-7.071e+08	0	0	-7.071e+08	7.071e+08
0	0	-7.071e+08	2.707e+09	0	0	7.071e+08	-7.071e+08
0	0	0	0	2.707e+09	7.071e+08	-2.000e+09	0
0	0	0	0	7.071e+08	2.707e+09	0	0
-7.499e-24	-1.225e-07	-7.071e+08	7.071e+08	-2.000e+09	0	5.414e+09	2.384e-07
-1.225e-07	-2.000e+09	7.071e+08	-7.071e+08	0	0	2.384e-07	5.414e+09
-7.071e+08	-7.071e+08	-7.499e-24	-1.225e-07	0	0	-2.000e+09	0
-7.071e+08	-7.071e+08	-1.225e-07	-2.000e+09	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	-7.499e-24	-1.225e-07	0	0
0	0	0	0	-1.225e-07	-2.000e+09	0	0
0	0	0	0	-7.071e+08	-7.071e+08	-7.499e-24	-1.225e-07
0	0	0	0	-7.071e+08	-7.071e+08	-1.225e-07	-2.000e+09
0	0	0	0	0	0	-7.071e+08	-7.071e+08
0	0	0	0	0	0	-7.071e+08	-7.071e+08
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Posição 8	Posição 9	Posição 10	Posição 11	Posição 12	Posição 13	Posição 14	Posição 15
-7.071e+08	-7.071e+08	0	0	0	0	0	0
-7.071e+08	-7.071e+08	0	0	0	0	0	0
-7.499e-24	-1.225e-07	0	0	0	0	0	0
-1.225e-07	-2.000e+09	0	0	0	0	0	0
0	0	0	0	-7.499e-24	-1.225e-07	-7.071e+08	-7.071e+08
0	0	0	0	-1.225e-07	-2.000e+09	-7.071e+08	-7.071e+08
-2.000e+09	0	0	0	0	0	-7.499e-24	-1.225e-07
0	0	0	0	0	0	-1.225e-07	-2.000e+09
5.414e+09	2.384e-07	-2.000e+09	0	0	0	-7.071e+08	7.071e+08
2.384e-07	5.414e+09	0	0	0	0	7.071e+08	-7.071e+08
-2.000e+09	0	2.707e+09	-7.071e+08	0	0	0	0
0	0	-7.071e+08	2.707e+09	0	0	0	0
0	0	0	0	2.000e+09	1.225e-07	-2.000e+09	0
0	0	0	0	1.225e-07	2.000e+09	0	0
-7.071e+08	7.071e+08	0	0	-2.000e+09	0	6.121e+09	7.071e+08
7.071e+08	-7.071e+08	0	0	0	0	7.071e+08	6.121e+09
-7.499e-24	-1.225e-07	-7.071e+08	7.071e+08	0	0	-2.000e+09	0
-1.225e-07	-2.000e+09	7.071e+08	-7.071e+08	0	0	0	0
0	0	-7.499e-24	-1.225e-07	0	0	0	0
0	0	-1.225e-07	-2.000e+09	0	0	0	0
0	0	0	0	0	0	-7.499e-24	-1.225e-07
0	0	0	0	0	0	-1.225e-07	-2.000e+09
0	0	0	0	0	0	-7.071e+08	-7.071e+08
0	0	0	0	0	0	-7.071e+08	-7.071e+08

16	17	18	19	20	21	22	23
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
-7.071e+08	-7.071e+08	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
-7.071e+08	-7.071e+08	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
-7.499e-24	-1.225e-07	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
-1.225e-07	-2.000e+09	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
-7.071e+08	7.071e+08	-7.499e-24	-1.225e-07	0.000e+00	0.000e+00	0.000e+00	0.000e+00
7.071e+08	-7.071e+08	-1.225e-07	-2.000e+09	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000e+00
-2.000e+09	0.000e+00	0.000e+00	0.000e+00	-7.499e-24	-1.225e-07	-7.071e+08	-7.071e+08
0.000e+00	0.000e+00	0.000e+00	0.000e+00	-1.225e-07	-2.000e+09	-7.071e+08	-7.071e+08
6.121e+09	-7.071e+08	-2.000e+09	0.000e+00	-7.071e+08	7.071e+08	-7.499e-24	-1.225e-07
-7.071e+08	6.121e+09	0.000e+00	0.000e+00	7.071e+08	-7.071e+08	-1.225e-07	-2.000e+09
-2.000e+09	0.000e+00	2.000e+09	1.225e-07	0.000e+00	0.000e+00	0.000e+00	0.000e+00
0.000e+00	0.000e+00	1.225e-07	2.000e+09	0.000e+00	0.000e+00	0.000e+00	0.000e+00
-7.071e+08	7.071e+08	0.000e+00	0.000e+00	3.833e+09	-9.106e+08	-2.000e+09	0.000e+00
7.071e+08	-7.071e+08	0.000e+00	0.000e+00	-9.106e+08	4.480e+09	0.000e+00	0.000e+00
-7.499e-24	-1.225e-07	0.000e+00	0.000e+00	-2.000e+09	0.000e+00	3.833e+09	9.106e+08
-1.225e-07	-2.000e+09	0.000e+00	0.000e+00	0.000e+00	0.000e+00	9.106e+08	4.480e+09

Bom, agora com essas duas matrizes podemos resolver nosso sistema, com a matriz M e a matriz K podemos criar uma matriz $\tilde{K}$ que é simétrica e, portanto conseguimos aplicar o algoritmo de Householder para chegarmos em uma matriz tridiagonal simétrica que pode ser resolvida pelo algoritmo QR (levemente modificado) do último exercício programa, fazemos isso por meio da relação:

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

Dessa forma conseguimos obter as frequências e os modos de vibração, que serão explicitados abaixo, vamos começar pelas frequências de vibração, temos que $w = \sqrt{\lambda}$, então conseguimos obter facilmente esse valor diretamente dos autovalores, dessa forma:

λ	ω [rad/s]	f [Hz]
604.793	**24.593**	**3.914**
8466.290	**92.012**	**14.644**
8968.727	**94.703**	**15.073**
20394.610	**142.810**	**22.729**
22747.314	**150.822**	**24.004**
53112.352	230.461	36.679
82526.580	287.274	45.721
84178.089	290.135	46.176
123427.179	351.322	55.915
146507.072	382.762	60.919
150401.659	387.817	61.723
178865.967	422.925	67.311
213324.188	461.870	73.509
214713.952	463.372	73.748
272742.139	522.247	83.118
286156.725	534.936	85.138
302305.853	549.823	87.507
313358.478	559.784	89.092
340498.678	583.523	92.871
351642.754	592.995	94.378
389912.930	624.430	99.381
432319.637	657.510	104.646
442927.026	665.528	105.922
459787.049	678.076	107.919

As frequências destacadas em negrito correspondem aos modos que requerem menos energia e, portanto, são os mais importantes na prática.

Por último temos os nossos modos de vibração que correspondem ao autovetores associados às menores frequências de vibração, os modos de vibração são calculados através da relação mostrada abaixo:

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

Desse modo temos os valores mostrados na tabela abaixo, atente-se que o Modo 5 está relacionado a menor frequência de vibração:

Modo 1	Modo 2	Modo 3	Modo 4	Modo 5
-4.000000e-08	0.000000e+00	-0.000000e+00	1.000000e-08	-0.000000e+00
-3.800000e-07	-0.000000e+00	-0.000000e+00	9.000000e-08	-1.000000e-08
1.041000e-05	0.000000e+00	4.000000e-08	-2.440000e-06	2.100000e-07
-1.787000e-05	-1.000000e-08	-1.400000e-07	6.020000e-06	-6.800000e-07
1.216900e-04	1.900000e-07	2.100000e-06	-5.978000e-05	8.510000e-06
-5.609320e-03	-1.176000e-05	-1.269100e-04	3.184030e-03	-4.912700e-04
-2.593800e-04	-1.370000e-06	-1.357000e-05	2.383000e-04	-4.629000e-05
1.290520e-03	1.131000e-05	1.074600e-04	-1.591950e-03	3.463400e-04
-7.600000e-06	-2.390000e-06	-2.031000e-05	1.708000e-04	-5.526000e-05
1.662700e-03	4.183000e-05	3.748700e-04	-4.178810e-03	1.105280e-03
-3.150000e-04	1.081500e-04	7.042600e-04	2.609100e-04	1.199720e-03
-1.924830e-03	4.583200e-04	3.005930e-03	6.852500e-04	5.192520e-03
1.823900e-04	6.310000e-06	3.375000e-05	9.901000e-05	3.739000e-05
2.150510e-03	5.116000e-05	2.248500e-04	8.156300e-04	1.369700e-04
7.418040e-03	1.753600e-04	7.477600e-04	2.707260e-03	4.039800e-04
-3.157650e-03	-8.825000e-05	-3.077900e-04	-9.421700e-04	-3.888000e-05
-8.400000e-05	2.840000e-06	3.190000e-06	-1.704000e-05	-4.330000e-06
-9.210700e-04	4.763200e-04	1.005940e-03	-2.909800e-04	-6.582000e-04
2.426800e-04	-2.262240e-03	-4.903330e-03	4.272500e-04	3.042520e-03
2.360000e-06	1.000000e-07	-7.000000e-08	3.800000e-07	7.000000e-08
-1.016200e-04	4.520000e-06	1.000000e-07	-1.759000e-05	-2.400000e-06
4.221640e-03	-1.664900e-04	-1.682000e-05	7.385400e-04	1.070900e-04
6.225000e-05	-5.119000e-05	1.856000e-05	9.730000e-06	-5.120000e-06
1.303900e-04	5.902130e-03	-2.239160e-03	8.763000e-05	7.737500e-04

4 O Código

O intuito dessa seção é detalhar um pouco melhor a implementação em código de cada uma das funções que regem esse Exercício Programa, muitas das funções são as mesmas do Exercício passado, então não vale a pena repetir.

Sobre as novas funções vamos comentar um pouco mais das que implementam o algoritmo das transformações de Householder.

Primeiro vamos passar pelas funções auxiliares que fundamentam a base do algoritmo principal que trabalhamos no nosso trabalho. Começaremos por uma função que recebe uma matriz A e nos dá o vetor coluna $\tilde{a}$ associado a essa matrix, localizado na primeira coluna excluindo-se a primeira linha

```
def get_truncated_a(A):
    return A[1:, :1].reshape(A.shape[0] - 1)
```

A segunda função também é auxiliar e tem como objetivo usar o vetor $\tilde{a}$ calculado anteriormente para devolver o vetor $\tilde{w}$

```
def get_truncated_w(a):
    e = np.zeros(a.shape)
    e[0] = 1
    return a + sgn(a[0]) * np.linalg.norm(a) * e
```

A próxima função é talvez a alma das transformações de Householder, essa função recebe um vetor x e devolve o vetor y que foi refletivo como se o vetor w atuasse como um espelho.

```
def apply_Hw_transformation(x, w):
    return x - 2 * (np.dot(w, x)/np.dot(w, w)) * w
```

As próximas 3 funções são super auxiliares e servem apenas para atualização de matrizes durante as rotinas do algoritmo.

A primeira visa atualizar a matriz A com os vetores coluna transformados por H_w, essa função faz o mostrado na segunda matriz da página 3 do enunciado.

```
def update_columns(A, columns):
    for i in range(len(columns)):
        A[1:, i] = columns[i]
    return A
```

Seguindo, temos uma função que reflete o conteúdo da primeira coluna na primeira linha, sendo basicamente a função que imite as marcações em verde da página 3 do enunciado do EP.

```
def mirror_first_col(A):
    A[:1, 1:] = A[1:, :1].reshape(A[:1, 1:].shape)
    return A
```

Finalmente temos uma função para atualizar as linhas e assim terminar uma iteração do algoritmo.

```
def update_rows(A, rows):
    for i in range(len(rows)):
        A[1:, i + 1] = rows[i]
    return A
```

Certo, visto agora todas as funções auxiliares podemos comentar um pouco sobre a função que implementa todas essas subrotinas nas transformações de Householder, como indicado pelo enunciado, não calculamos cada uma das matrizes H_w, utilizamos apenas as transformações$H_w x$, o que de fato economiza espaço computacional e também contas efetivas que precisamos realizar, já que temos apenas $O(n)$ operações, enquanto que cálculos com matrizes cheias teíamos $O(n^2)$ operações, o que é bem mais ineficiente.

Guardamos então cada um dos vetores $\tilde{w}$ para aplicarmos as mesmas transformações que fizemos em A em uma segunda matriz que inicia sendo a Identidade, desse modo também conseguimos obter o valor de H^T a partir da transformação $H_w x$ de cada um dos vetores $w_{n-2}, w_{n-1}, \ldots, w_2, w_1$ que foram guardados.

```
def Householder(A, zero_threshold=1e-150, set_initial_T=False, T_initial=None):
    M = A.copy().astype(float)

    if set_initial_T and T_initial.any() != None:
        T = T_initial.copy().astype(float)
    else:
        T = np.identity(A.shape[0]).astype(float)

    W = []

    for i in range(A.shape[0], 2, -1):
        a = get_truncated_a(M)
        w = get_truncated_w(a)
        W.append(w)

        subcols = [M[1:, j].reshape(M.shape[0] - 1) for j in range(M.shape[0])]
        subcols = [apply_Hw_transformation(col, w) for col in subcols]

        M = update_columns(M, subcols)

        M = mirror_first_col(M)

        #updates T matrice
        T[A.shape[0] - i : A.shape[0] - i + 1, A.shape[0] - i :] = M[0, :]
        T[A.shape[0] - i :, A.shape[0] - i : A.shape[0] - i + 1] = M[:, 0].reshape(i, 1)
```

```
        subrows = [M[j, 1:].reshape(M.shape[0] - 1) for j in range(1, M.shape[0])]
        subrows = [apply_Hw_transformation(row, w) for row in subrows]

        M = update_rows(M, subrows)

        M = M[1:, 1:]

    T[-2:, -2:] = M
    T[abs(T) < zero_threshold] = 0

    H = np.identity(A.shape[0]).astype(float)
    for w in W:
        I = np.identity(A.shape[0]).astype(float)
        for i in range(len(w)):
            I[:, - (i + 1)] = apply_Hw_transformation(I[:, - (i + 1)],
                              np.pad(w, (A.shape[0] - len(w), 0)))

        H = H @ I

    H[abs(H) < zero_threshold] = 0

    return T, H
```

As próximas 3 funções são especificamente feitas para interagir com as requisições da Tarefa E.

A primeira delas é uma função auxiliar para calcular a matrizes com cossesnos e senos que forma $K^{\{i,j\}}$

```
def create_cos_and_sin_matrix(theta):
    theta_rad = np.deg2rad(theta)
    C = np.cos(theta_rad)
    S = np.sin(theta_rad)
    M = np.identity(4)

    M[0, 0] = np.multiply(C,  C)
    M[0, 1] = np.multiply(C,  S)
    M[0, 2] = np.multiply(C, -C)
    M[0, 3] = np.multiply(C, -S)
    M[1, 0] = np.multiply( C, S)
    M[1, 1] = np.multiply( S, S)
    M[1, 2] = np.multiply(-C, S)
    M[1, 3] = np.multiply(-S, S)
    M[2, 0] = np.multiply(C, -C)
    M[2, 1] = np.multiply(C, -S)
    M[2, 2] = np.multiply(C,  C)
```

```
    M[2, 3] = np.multiply(C,  S)
    M[3, 0] = np.multiply(-C, S)
    M[3, 1] = np.multiply(-S, S)
    M[3, 2] = np.multiply( C, S)
    M[3, 3] = np.multiply( S, S)

    return M
```

Essa próxima função é responsável por criar a matriz K de rigidez total do sistema

```
def create_global_K(trellis, free_nodes):
    K_global = np.zeros((free_nodes * 2, free_nodes * 2))

    for k in range(len(trellis)):
        i = trellis["Extreme_1"][k] - 1
        j = trellis["Extreme_2"][k] - 1

        if j <= free_nodes - 1:
            K_global[2 * i    , 2 * i    ] += trellis["K_ij"][k][0, 0]
            K_global[2 * i    , 2 * i + 1] += trellis["K_ij"][k][0, 1]
            K_global[2 * i    , 2 * j    ] += trellis["K_ij"][k][0, 2]
            K_global[2 * i    , 2 * j + 1] += trellis["K_ij"][k][0, 3]
            K_global[2 * i + 1, 2 * i    ] += trellis["K_ij"][k][1, 0]
            K_global[2 * i + 1, 2 * i + 1] += trellis["K_ij"][k][1, 1]
            K_global[2 * i + 1, 2 * j    ] += trellis["K_ij"][k][1, 2]
            K_global[2 * i + 1, 2 * j + 1] += trellis["K_ij"][k][1, 3]
            K_global[2 * j    , 2 * i    ] += trellis["K_ij"][k][2, 0]
            K_global[2 * j    , 2 * i + 1] += trellis["K_ij"][k][2, 1]
            K_global[2 * j    , 2 * j    ] += trellis["K_ij"][k][2, 2]
            K_global[2 * j    , 2 * j + 1] += trellis["K_ij"][k][2, 3]
            K_global[2 * j + 1, 2 * i    ] += trellis["K_ij"][k][3, 0]
            K_global[2 * j + 1, 2 * i + 1] += trellis["K_ij"][k][3, 1]
            K_global[2 * j + 1, 2 * j    ] += trellis["K_ij"][k][3, 2]
            K_global[2 * j + 1, 2 * j + 1] += trellis["K_ij"][k][3, 3]

        else:
            K_global[2 * i    , 2 * i    ] += trellis["K_ij"][k][0, 0]
            K_global[2 * i    , 2 * i + 1] += trellis["K_ij"][k][0, 1]
            K_global[2 * i + 1, 2 * i    ] += trellis["K_ij"][k][1, 0]
            K_global[2 * i + 1, 2 * i + 1] += trellis["K_ij"][k][1, 1]

    return K_global
```

Por último, temos uma função para auxiliar na criação da matriz de massa M.

```
def create_mass_matrix(trellis, density, transversal_section, free_nodes):
    M = np.zeros((free_nodes * 2, free_nodes * 2))
    constant = 0.5 * density * transversal_section

    for k in range(free_nodes):
        L_seen = trellis.loc[(trellis["Extreme_1"] == k + 1)|
                             (trellis["Extreme_2"] == k + 1), "Length"].sum()

        M[2 * k    , 2 * k    ] = constant * L_seen
        M[2 * k + 1, 2 * k + 1] = constant * L_seen

    return M
```