

Escola Politécnica da Universidade de São Paulo
Engenharia Elétrica



MAP3121 - Métodos Numéricos e Aplicações

André Lucas Pierote Rodrigues - NUSP: 11356540 - Turma: 01

Leonardo Isao Komura - NUSP: 11261656 - Turma: 03

São Paulo, SP
2021

SUMÁRIO

1. Introdução	3
2. Informações gerais	3
3. Como executar o código	4
a. Operação 1	4
b. Operação 2	5
4. Código	4
a. Rotação de Givens	4
b. Algoritmo QR	6
c. Resolução do sistema massa-mola	10
5. Questões	12
6. Conclusão	21

1. Introdução

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

Assim sendo, esse relatório tem como objetivo descrever um programa feito para o cálculo dos autovalores e autovetores de uma matriz tridiagonal simétrica. Além disso, o código também possibilita o uso dos valores obtidos para um aplicação: a resolução de um sistema massa-mola específico.

2. Informações gerais

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

Como bibliotecas, foram utilizadas:

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

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

O código está contido inteiramente no arquivo **EP1.py** e há um arquivo chamado **LEIA-ME.txt** que apresenta instruções para sua execução.

3. Como executar o código

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

```
=====
EXERCÍCIO PROGRAMA 1 - MÉTODOS NUMÉRICOS (MAP3121)
=====
André Lucas Pierote Rodrigues Vasconcelos - NUSP: 11356540 - Engenharia Elétrica - Turma: 01
Leonardo Isao Komura - NUSP: 11261656 - Engenharia Elétrica - Turma: 03
-----
Qual operacao voce deseja realizar?
1. Teste do algoritmo QR (com e sem deslocamento espectral)
2. Resolucao de um sistema massa-mola
Operacao = -
```

Figura 1 - Interface do usuário inicial

3.a) Operação 1 - Algoritmo QR

A operação 1 realiza o algoritmo QR com e sem deslocamento espectral. Selecionada essa opção, serão pedidos:

- I. O tamanho da matriz;
- II. Se os elementos de cada diagonal são iguais entre si (exemplo: todos alphas iguais);
- III. Os valores dos elementos das diagonais principal (α 's) e secundárias (β 's e α 's)
- IV. O erro desejado ($\epsilon = 10^{-erro}$).

Dadas as entradas, o programa irá realizar cálculos e retornará as matrizes dos autovetores e os autovalores do algoritmo QR sem e com deslocamento espectral, respectivamente, e seus valores teóricos. Além disso, também são retornados o número de execuções que foram necessárias para atingir o erro desejado.

3.b) Operação 2 - Resolução sistema massa-mola

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

- I. Qual o número de massas será utilizada para o cálculo (5 ou 10);
- II. A massa das massas em quilogramas;
- III. O erro utilizado para as contas;
- IV. Qual constante elástica utilizar (vide questões b) e c) do enunciado);
- V. Se o usuário deseja digitar as posições iniciais de cada massa ou deseja utilizar as posições iniciais do modo de máxima frequência;
- VI. Tempo de execução da simulação para a montagem dos gráficos.

Dadas as entradas, o programa abrirá duas janelas que apresentam gráficos: uma das abas apresentará um gráfico único com todas as curvas do posicionamento das massas em função do tempo. A outra aba apresentará as mesmas curvas, porém, em gráficos separados.

4. Código

Como nosso objetivo é fazer um algoritmo QR funcionar devidamente, devemos começar fazendo a rotação de Givens.

É importante ressaltar que foram utilizados vetores para o armazenamento e manipulação dos alpha's, beta's e gama's.

4.a) Rotação de Givens

Definidos os vetores *alpha*, *beta* e *gama* (representando uma matriz A de entrada), as variáveis *ck* e *sk* (*cosθ* e *senθ*) serão calculadas conforme a forma equivalente, e mais estável, descrita no enunciado:

Obtidos $\alpha_k = \alpha_i$ e $\beta_k = \beta_j$:

Se $[\alpha_k] > [\beta_k]$:

$$\tau = -\beta_k/\alpha_k, c_k = 1/\sqrt{1 + \tau^2} \text{ e } s_k = c_k\tau .$$

Caso contrário:

$$\tau = -\alpha_k/\beta_k, s_k = 1/\sqrt{1 + \tau^2} \text{ e } c_k = s_k\tau .$$

Além disso, os novos α 's, β 's e

$$b_{i,k} = ca_{i,k} - sa_{j,k} \quad \text{e } b_{j,k} = sa_{i,k} + ca_{j,k} , \quad k = 1, \dots, n$$

```

# Rotacao de Givens
def givens(alpha, beta, gama, i, j, n):
    novoAlpha = alpha.copy()
    novoBeta = beta.copy()
    novoGama = gama.copy()
    ak = alpha[i-1]
    bk = beta[j-2]

    if (abs(ak) > abs(bk)):
        t = -bk/ak
        ck = 1/math.sqrt(1+(t**2))
        sk = ck*t
    else:
        t = -ak/bk
        sk = 1/math.sqrt(1+(t**2))
        ck = sk*t

    novoAlpha[i-1] = (ck*alpha[i-1]) - (sk*beta[j-2])
    novoGama[i-1] = (ck*gama[i-1]) - (sk*alpha[j-1])
    novoBeta[j-2] = (sk*alpha[i-1]) + (ck*beta[j-2])
    novoAlpha[j-1] = (sk*gama[i-1]) + (ck*alpha[j-1])
    if (j != n):
        novoGama[j-1] = (ck*gama[j-1])
    return novoAlpha, novoBeta, novoGama, ck, sk

```

Figura 2 - Código da Rotação de Givens

A função da Rotação de Givens retornará cinco vetores:

- novoAlpha: diagonal principal de $Q^*A = R$;
- novoBeta e novoGama: diagonais secundárias de $Q^*A = R$;
- ck e sk: vetores que armazenam cada ck e sk utilizados na rotação.

4.b) Algoritmo QR

O Algoritmo QR foi implementado em duas funções separadas:

→ Sem deslocamento espectral:

A função feita sem deslocamento espectral foi baseada no seguinte pseudocódigo mostrado no enunciado:

$$A^{(0)} = A; V^{(0)} = I_{n \times n}$$

$$k = 0$$

repita

$$A^{(k)} \rightarrow Q^{(k)} R^{(k)} \quad (\text{fatoração QR de } A^{(k)})$$

$$A^{(k+1)} = R^{(k)} Q^{(k)} \quad (\text{atualização da matriz})$$

$$V^{(k+1)} = V^{(k)} Q^{(k)} \quad (\text{atualização dos autosvetores})$$

até a convergência

→ Com deslocamento espectral

A função feita com o deslocamento espectral foi baseada no seguinte pseudocódigo mostrado no enunciado:

para $m = n, n-1, \dots, 2$ faça

repita

se $k > 0$ calcule μ_k pela heurística de Wilkinson

$$A^{(k)} - \mu_k I \rightarrow Q^{(k)} R^{(k)}$$

$$A^{(k+1)} = R^{(k)} Q^{(k)} + \mu_k I$$

$$V^{(k+1)} = V^{(k)} Q^{(k)}$$

$$k = k + 1$$

até que $|\beta_{m-1}^{(k)}| < \epsilon$

fim do para

Para a execução de ambas, foram criadas outras duas funções auxiliares:

❖ assist: calcula $A^{(n+1)} = R^{(n)} * (Q^{(n)})^T$

```

# Funcao que calcula o R*Qt
def assist(ck, sk, alpha, beta, gama, i):
    novoAlpha = alpha.copy()
    novoBeta = beta.copy()

    novoAlpha[i-1] = (ck*alpha[i-1]) - (sk*gama[i-1])
    novoBeta[i-1] = -sk*alpha[i]
    novoAlpha[i] = ck*alpha[i]
    return novoAlpha, novoBeta

```

Figura 3 - Código da função *assist*

❖ autovetor: calcula a matriz: $V^{(n+1)} = V^{(n)} * Q^T$

```

# Funca que calcula os autovetores (VQt)
def autovetor(V, ck, sk, i, n):
    ident = np.eye(n)
    Q = ident.copy()
    for k in range(n):
        Q[i][k] = (ck*ident[i][k]) - (sk*ident[i+1][k])
        Q[i+1][k] = (sk*ident[i][k]) + (ck*ident[i+1][k])

    Qt = np.transpose(Q)
    VQt = np.matmul(V, Qt)
    return VQt

```

Figura 4 - Código da função *autovetor*

Observação: O funcionamento da convergência para o deslocamento QR é baseado nos Beta's tornarem-se menores que o erro inserido pelo usuário. Assim, percorrendo a matriz de baixo para cima, quando o critério do erro é atingido, isso é, beta torna-se menor que o erro, a matriz a ser utilizada para o cálculo será a mesma sem a linha e coluna desse beta. (No nosso caso que usamos vetores, as posições vetoriais serão alteradas, não atualizando suas últimas posições progressivamente)

Assim sendo, os códigos dos algoritmos ficaram:

```
# Algoritmo QR
def QR(a, b, g, n, erro):
    c = np.zeros(n)
    s = np.zeros(n)
    k = 0
    alphaR = a.copy()
    betaR = b.copy()
    gamaR = g.copy()
    V = np.eye(n)
    VQ = np.eye(n)
    for m in range(n, 1, -1):
        while(abs(betaR[m-2]) > erro):
            for i in range(m-1):
                alpha_R, beta_R, gama_R, c[i], s[i] = givens(alphaR, betaR, gamaR, i+1, i+2, n)
                alphaR = alpha_R.copy()
                betaR = beta_R.copy()
                gamaR = gama_R.copy()

            alphaRQ = alphaR.copy()
            betaRQ = betaR.copy()
            gamaRQ = gamaR.copy()
            for i in range(m-1):
                alpha_RQ, beta_RQ = assist(c[i], s[i], alphaRQ, betaRQ, gamaRQ, i+1)
                alphaRQ = alpha_RQ.copy()
                betaRQ = beta_RQ.copy()
                gamaRQ = betaRQ.copy()

            alphaR = alphaRQ.copy()
            betaR = betaRQ.copy()
            gamaR = gamaRQ.copy()

            for i in range(n-1):
                VQ = autovetor(V, c[i], s[i], i, n)
                V = VQ.copy()

            k = k + 1
    return VQ, alphaR, k
```

Figura 5 - Algoritmo QR sem deslocamento espectral

```

# Algoritmo QR com deslocamento espectral
def QR_deslocamento(a, b, g, n, erro):
    c = np.zeros(n)
    s = np.zeros(n)
    k = 0
    alphaR = a.copy()
    betaR = b.copy()
    gamaR = g.copy()
    alphaRQ = a.copy()
    betaRQ = b.copy()
    gamaRQ = g.copy()
    V = np.eye(n)
    VQ = np.eye(n)
    mi_k = np.zeros(n)
    oldAlpha = a.copy()
    oldBeta = b.copy()
    for m in range(n, 1, -1):
        while(abs(betaR[m-2]) > erro):
            if (k > 0):
                for j in range(m):
                    dk = (oldAlpha[m-1] - alphaRQ[m-1])/2
                    if (dk >= 0):
                        mi_k[j] = alphaRQ[m-1] + dk - np.sqrt((dk**2) + oldBeta[m-2]**2)
                    else:
                        mi_k[j] = alphaRQ[m-1] + dk + np.sqrt((dk**2) + oldBeta[m-2]**2)

            oldAlpha = alphaRQ.copy()
            oldBeta = betaRQ.copy()
            alphaR = np.subtract(alphaRQ, mi_k)

            for i in range(m-1):
                alpha_R, beta_R, gama_R, c[i], s[i] = givens(alphaR, betaR, gamaR, i+1, i+2, n)
                alphaR = alpha_R.copy()
                betaR = beta_R.copy()
                gamaR = gama_R.copy()

            alphaRQ = alphaR.copy()
            betaRQ = betaR.copy()
            gamaRQ = gamaR.copy()
            for i in range(m-1):
                alpha_RQ, beta_RQ = assist(c[i], s[i], alphaRQ, betaRQ, gamaRQ, i+1)
                alphaRQ = alpha_RQ.copy()
                betaRQ = beta_RQ.copy()
            gamaRQ = betaRQ.copy()

            alphaRQ = np.add(alpha_RQ, mi_k)

            for i in range(n-1):
                VQ = autovetor(V, c[i], s[i], i, n)
                V = VQ.copy()

            betaR = betaRQ.copy()
            gamaR = gamaRQ.copy()

            k = k + 1

    return VQ, alphaRQ, k

```

Figura 6 - Algoritmo QR com deslocamento espectral

4.c) Resolução do Sistema Massa-Mola

Para a resolução do sistema massa-mola é necessário, primeiramente, o processamento das entradas escolhidas pelo usuário:

Como procedimento inicial, será definida a matriz A, que armazena as constantes elásticas divididas pela massa. Após isso, essa matriz é dada como entrada para o algoritmo QR, com deslocamento espectral, na forma de vetores (cada vetor representa cada uma de suas três diagonais principais).

Obtidos os autovalores (Λ) e autovetores (V ou Q^T) da matriz A, o usuário escolherá se ele irá querer inserir as posições iniciais ou se deseja que o programa calcule-as para que o sistema fique no modo de maior frequência.

Caso o usuário deseje visualizar o modo de maior frequência, o programa irá checar qual o maior autovalor, pois, as frequências serão iguais as raízes quadradas desses. Então, obtém-se o autovetor relacionado a esse autovalor. Esse vetor será as posições iniciais das massas.

Antes de continuar a explicação do código, é necessário analisar a transformação da equação $X'' + AX = 0$ para $Y'' + \Lambda Y = 0$.

É notável que: $Y' = 0$ e, sendo:

$$y_i(t) = a_i * \cos(\omega_i * t) + b_i * \sin(\omega_i * t)$$

Temos: $0 = -a_i * \sin(\omega_i * 0) + b_i * \cos(\omega_i * 0)$, portanto, $b_i = 0$. Por conseguinte, podemos afirmar que o coeficiente a_i será igual à posição inicial no sistema em Y que é igual a em X.

Obtidos os valores das frequências (que são as raízes quadradas dos autovalores) e do coeficiente que multiplica o cosseno, o sistema sofre uma transformação para retornar à variável X: $X = Q * Y = V^T * Y$

Portanto, finalmente chega-se à equação de X, que será uma soma de cossenos multiplicado pelos valores armazenados nos autovetores.

Após feita essas ações pelo código, são gerados os pontos que serão utilizados para a montagem dos gráficos.

```

#Calculo das constantes elasticas
k = np.zeros(n+1)
if cte==1:
    for i in range(n+1):
        k[i] = (40 + 2*(i+1))
else:
    for i in range(n+1):
        k[i] = (40 + 2*(-1***(i+1)))

#Calculo da matriz A
Alpha = np.zeros(n)
Beta = np.zeros(n-1)
Gama = np.zeros(n-1)
for i in range(n):
    Alpha[i] = (k[i] + k[i+1])/m
for i in range(n-1):
    Beta[i] = -k[i+2]/m
Gama = Beta.copy()

#Calculo da matriz Qt e dos autovalores
Lambda = np.zeros(n)
Qt, Lambda, iter = QR_deslocamento(Alpha, Beta, Gama, n, epsilon)

print("\n  Voce deseja inserir as posicoes iniciais?")
maxfreq = input("  (Caso nao deseje, elas serao aquelas que geram maior frequencia) (s/n): ")

X_init = np.zeros(n)
Y_init = np.zeros(n)
if maxfreq == "s":
    print("\n  Digite as posicoes iniciais de cada massa: ")
    for i in range(5):
        X_init[i] = input("  Posicao X_0(%d) = " %(i+1))
    if n==10:
        for i in range(5, n, 1):
            X_init[i] = X_init[i-5]
#Obtencao do autovetor relacionado ao maior autovalor
else:
    maxvalue=max(Lambda)
    minvalue=min(Lambda)
    if abs(maxvalue) > abs(minvalue):
        maximo = maxvalue
    else:
        maximo = minvalue
    pos = np.where(Lambda == maximo)

    for i in range(n):
        X_init[i] = Qt[pos, i]

```

Figura 7 - Aplicação para o sistema massa-mola

```

Xinit = np.transpose(np.array([X_init]))      #Transpondo X(0)

Y_init = np.matmul(Qt, Xinit)                  #Calculando Y(0)
ay = Y_init.copy()                            #Calculando os coeficientes que multiplicam os cossenos

freqs = np.zeros(n)
for i in range(n):
    freqs[i] = np.sqrt(Lambda[i])            #Calculando as frequencias dos cossenos

```

Figura 8 - Aplicação para o sistema massa-mola

```

#Calculo da matriz X(t)
for i in range(n):
    for q in range(40*t):
        for j in range(n):
            X[i][q] = X[i][q] + (Q[i][j]*ay[j]*math.cos(freqs[j]*(q/40)))

```

Figura 9 - Posições das massas em cada momento analisado (a cada 0,025s)

5. Questões

- a) Nesse item, temos alpha, beta e gama definidos pelo usuário (que devem ser, respectivamente, 2, -1 e -1). Ademais, os autovalores serão do tipo

$$\lambda_j = 2\left(1 - \cos\left(\frac{j\pi}{n+1}\right)\right), j = 1, \dots, n$$

enquanto os autovetores teóricos devem ser os seguintes:

$$v_j = \left(\sin\left(\frac{j\pi}{n+1}\right), \sin\left(\frac{2j\pi}{n+1}\right), \dots, \sin\left(\frac{nj\pi}{n+1}\right)\right).$$

Abaixo, mostra-se como fazemos esses cálculos no programa:

```

# Resposta teorica
AutoVetores = np.zeros((n, n))
AutoValores = np.zeros(n)
for i in range(n):
    AutoValores[i] = 2*(1-math.cos((i+1)*math.pi/(n+1)))
    for j in range(n):
        AutoVetores[j][i] = math.sin((i+1)*(j+1)*math.pi/(n+1))

print("\nAutovetores teoricos = ")
print(np.matrix(AutoVetores))
print("Autovalores teoricos = ", AutoValores)

```

Figura 10 - Cálculo empírico dos autovalores e autovetores

Por fim, vemos se os valores empíricos obtidos da forma descrita no algoritmo QR batem com esses autovalores e autovetores esperados.

Exemplo da execução com uma matriz 4x4:

```

Sem deslocamento:
Avtovetor =
[[ 0.371748   0.601495   0.601496   0.371765 ]
 [ -0.601501  -0.371751   0.371728   0.601511 ]
 [ 0.601501   -0.371745   -0.371768   0.601491 ]
 [ -0.371748   0.601506   -0.601506   0.371732 ]]
Avtovvalor = [ 3.618034   2.618034   1.381966   0.381966 ]
Numero de iteracoes: 45

Com deslocamento:
Avtovetor =
[[ 0.371748   0.601391   0.601431   0.372039 ]
 [ -0.601501  -0.371816   0.371389   0.601681 ]
 [ 0.601501   -0.371680   -0.372107   0.601321 ]
 [ -0.371748   0.601611   -0.601570   0.371457 ]]
Avtovvalor = [ 3.618034   2.618034   1.381966   0.381966 ]
Numero de iteracoes: 12

Avtovetores teoricos =
[[ 0.587785   0.951057   0.951057   0.587785 ]
 [ 0.951057   0.587785   -0.587785   -0.951057 ]
 [ 0.951057   -0.587785   -0.587785   0.951057 ]
 [ 0.587785   -0.951057   0.951057   -0.587785 ]]
Avtovalores teoricos = [ 0.381966   1.381966   2.618034   3.618034 ]

```

Figura 11 - Execução dos algoritmos QR numa matriz 4x4 com os dados pedidos

Ao analisar o resultado obtido por meio dessa execução, pode-se notar que os valores obtidos pelo algoritmo QR sem deslocamento espectral é igual ao atingido pelo algoritmo que o utilizou.

Ademais, pode-se notar que os autovalores obtidos são iguais aos esperados, porém, escritos na ordem contrária.

Além disso, é notável que os autovetores, apesar de numericamente diferentes, possuem a mesma proporção entre seus elementos, ou seja, cada linha dos autovetores calculados é igual ao valor esperado multiplicado por uma constante qualquer.

Esse comportamento se repete para qualquer tamanho de matriz, ou seja, os autovalores calculados serão idênticos aos teóricos, e os autovetores obtidos serão múltiplos dos teóricos.

OBS: Não foram colocadas print's das execuções com 8, 16 e 32 por serem muito grandes para serem postas no relatório. Além do que, devido ao alto número de elementos em cada matriz, os dados tornam-se confusos.

n	Número de iterações SEM DESLOCAMENTO ESPECTRAL	Número de iterações COM DESLOCAMENTO ESPECTRAL
4	45	12
8	143	25
16	473	46
32	1600	82

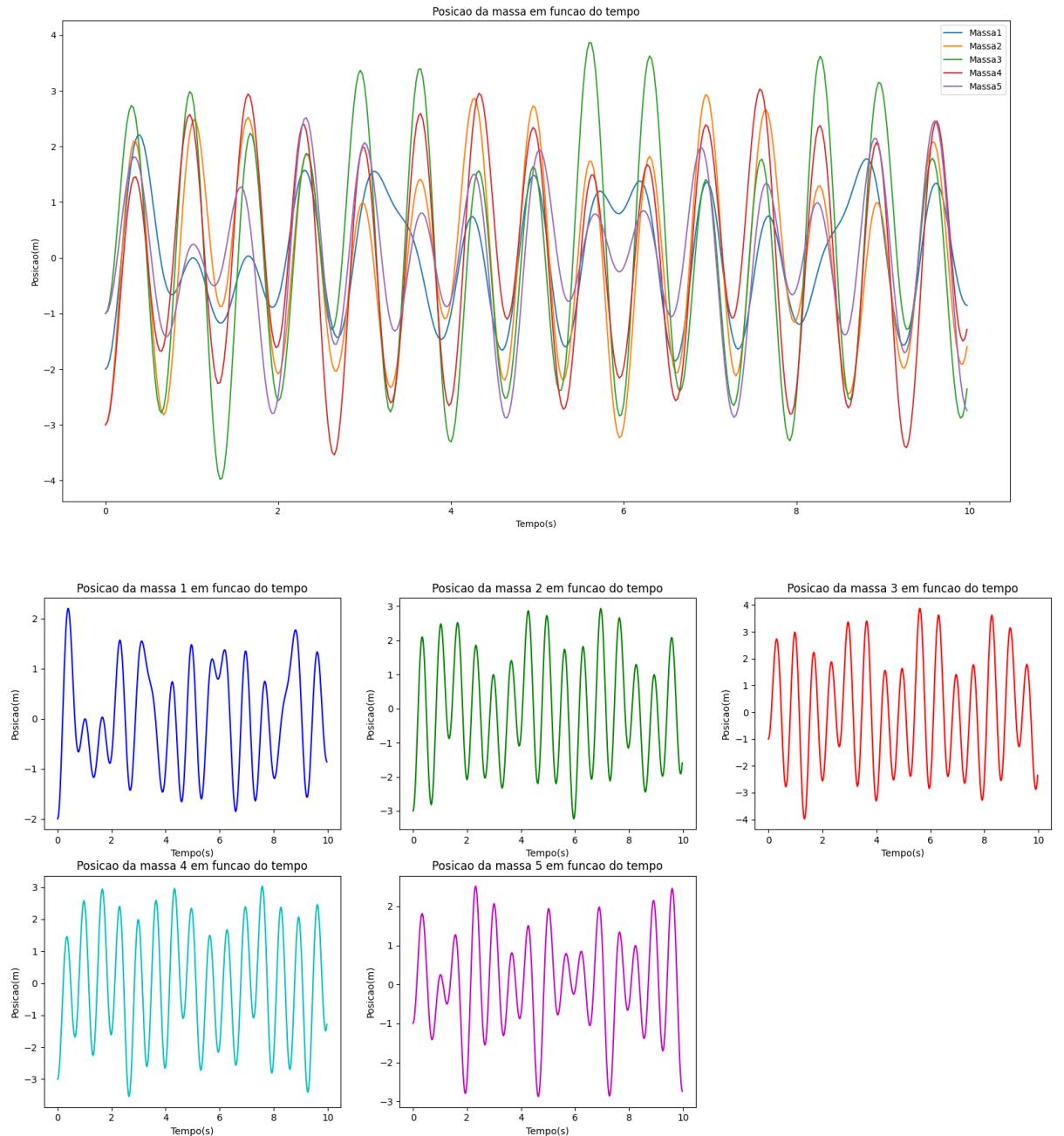
Ao observar a tabela, é notável a grande diferença de iterações necessárias para que o resultado final seja atingido, entre o algoritmo que utiliza o deslocamento espectral em relação ao que não o utiliza. No entanto, essa discrepância é esperada, pois, o deslocamento espectral acelera a convergência de maneira significativa, dado que ela será proporcional a $|(\lambda_j - \mu_k)/(\lambda_{j-1} - \mu_k)|$ e, ao subtrair o μ_k , obtido pela heurística de Wilkinson, dos α s, a convergência irá acelerar muito.

- b) Nesse item foi pedido uma aplicação do sistema massa-mola com 5 massa de 2kg com constante elástica:

$$k_i = (40 + 2i) \text{ N/m}, i = 1, \dots, 6.$$

É esperado que o gráfico da posição das massas em função do tempo apresentará um comportamento oscilatório, pois a posição $x(t)$ da massa é representada por uma soma de cossenos, o que também causa que as oscilações de cada massa sejam diferentes e sem sintonia. No entanto, quando as posições iniciais são, especificamente, um múltiplo de algum autovetor da matriz A, as massas passam a oscilar sincronizadamente, ou seja, com a mesma frequência, porém, com diferentes amplitudes de deslocamento.

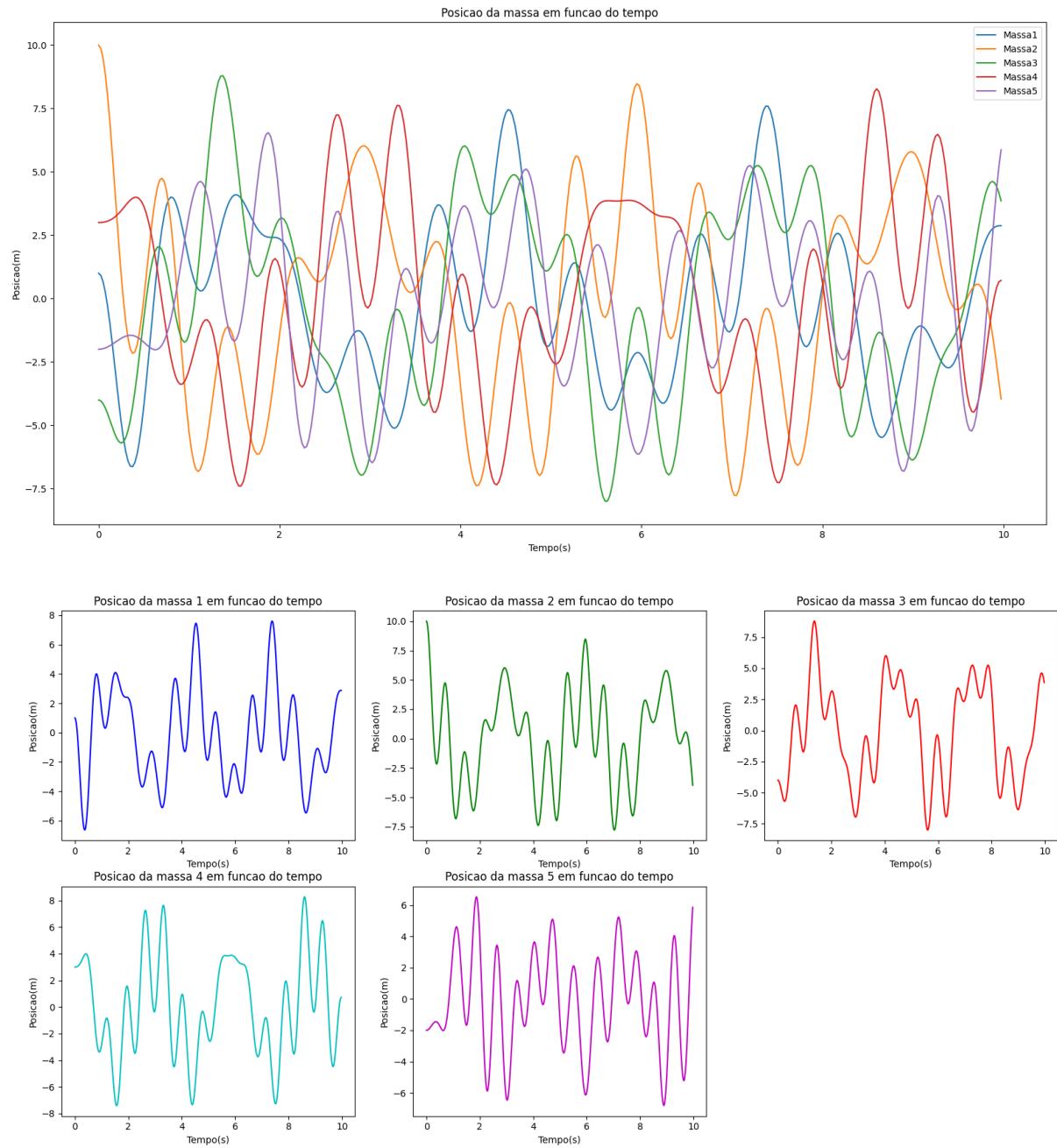
Caso 1: $X(0) = -2, -3, -1, -3, -1$



Podemos ver que as posições iniciais coincidem com aquelas dadas no enunciado, além de um comportamento oscilatório próprio da posição em função do tempo em uma soma de cossenos.

Obs: As cores das molas nos gráficos conjunto e separado das mesmas não devem ser confundidas.

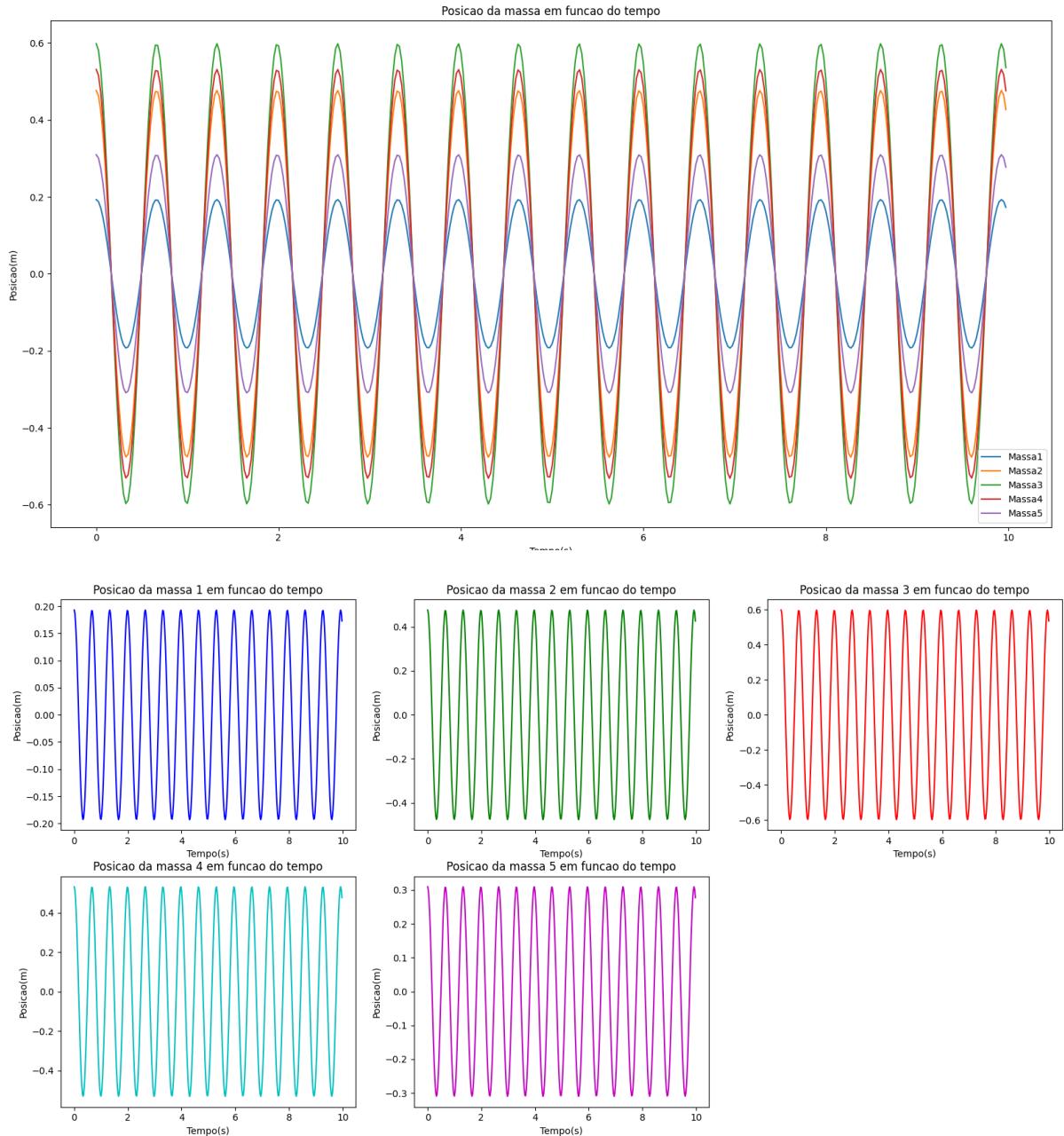
Caso 2: $X(0) = 1, 10, -4, 3, -2$



Podemos ver que as posições iniciais coincidem com aquelas dadas no enunciado, além de um comportamento oscilatório próprio da posição em função do tempo em uma soma de cossenos.

Obs: As cores das molas nos gráficos conjunto e separado das mesmas não devem ser confundidas.

Caso 3: Condição de máxima frequêcia



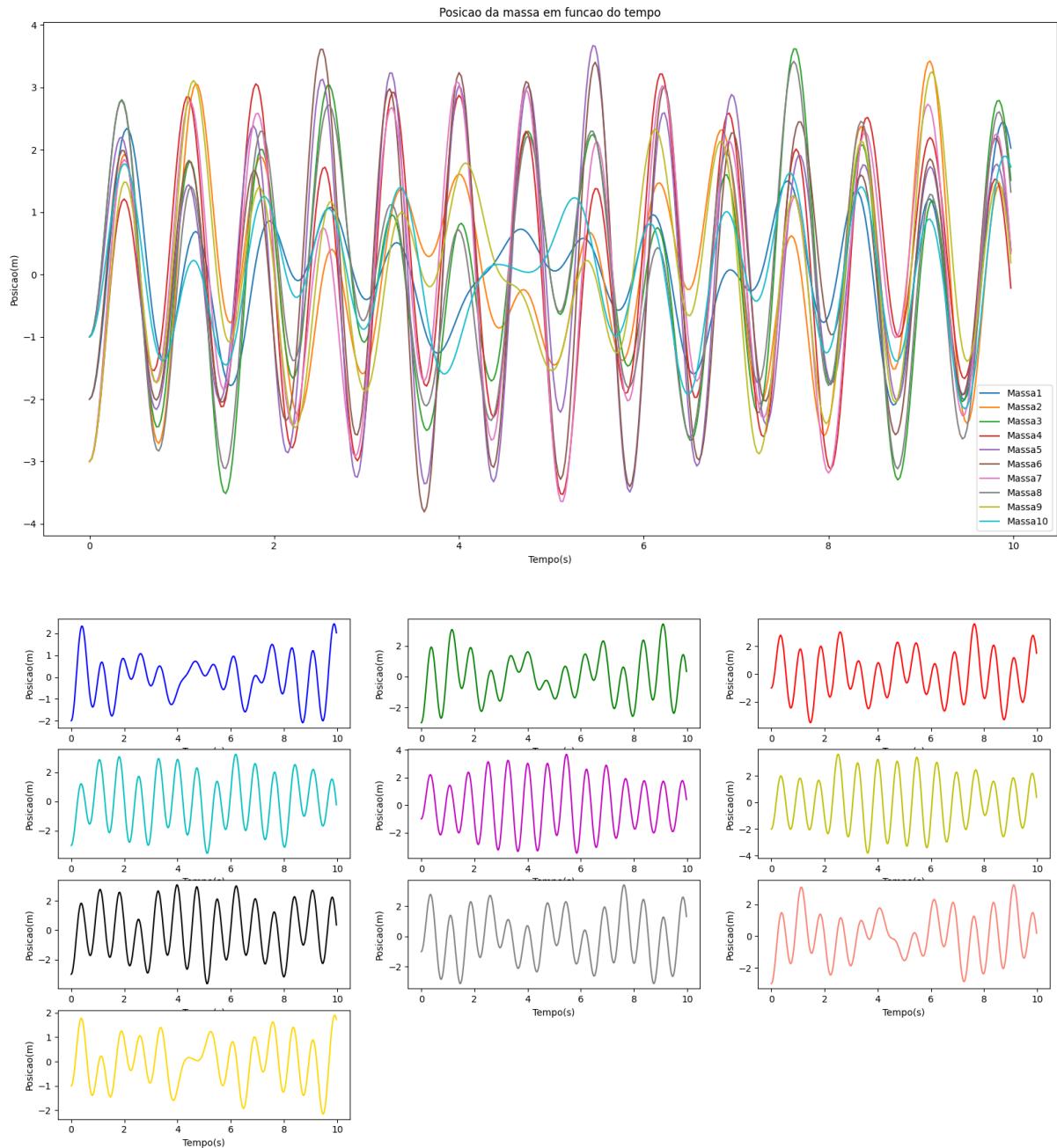
Nesse caso, percebemos que as frequências são iguais para todas as massas, o que de fato era esperado no caso de máxima frequência.

Obs: As cores das molas nos gráficos conjunto e separado das mesmas não devem ser confundidas.

c) Neste último item, temos a mesma ideia empregada no item “b”, com pequenas divergências apenas na quantidade de molas (que passa a ser 10, e não mais 5) e no valor da constante elástica:

$$k_i = (40 + 2(-1)^i) \text{ N/m}, i = 1, \dots, 11.$$

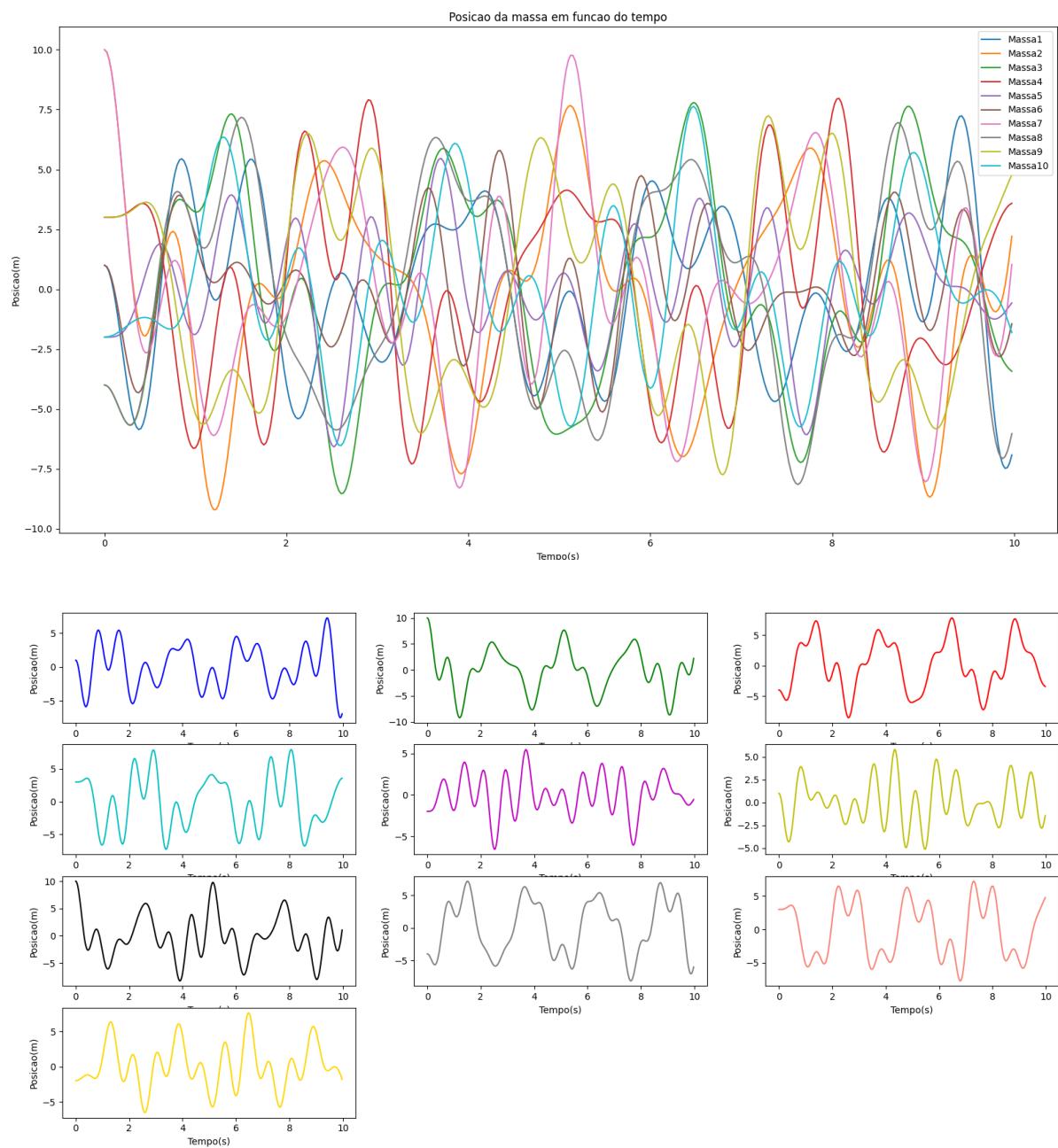
Caso 1: $X(0) = -2, -3, -1, -3, -1$



Podemos ver que as posições iniciais coincidem com aquelas dadas no enunciado, além de um comportamento oscilatório próprio da posição em função do tempo em uma soma de cossenos.

Obs: As cores das curvas nos gráficos conjunto e separado das mesmas não são condizentes.

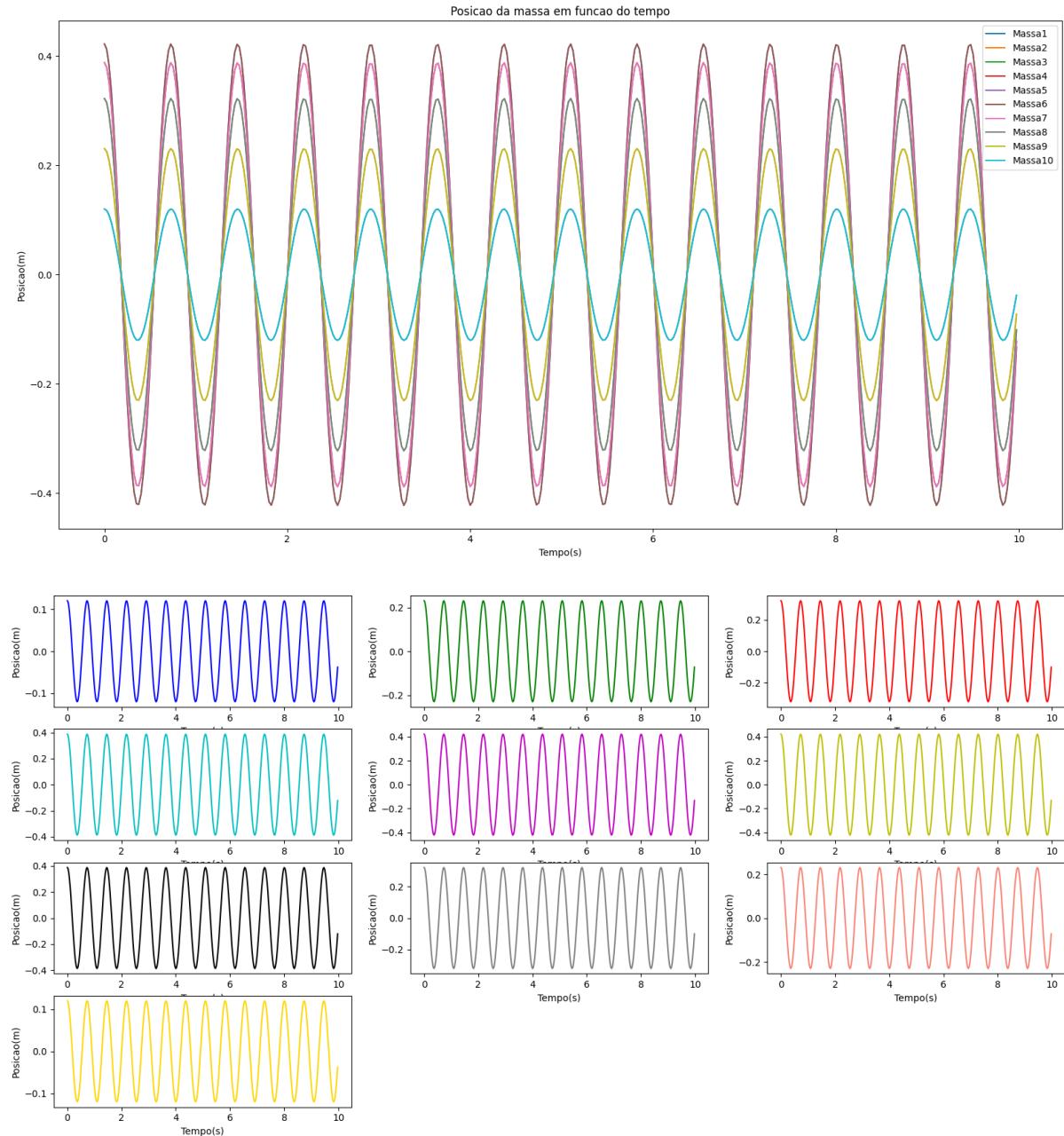
Caso 2: $X(0) = 1, 10, -4, 3, -2$



Podemos ver que as posições iniciais coincidem com aquelas dadas no enunciado, além de um comportamento oscilatório próprio da posição em função do tempo em uma soma de cossenos.

Obs: As cores das curvas nos gráficos conjunto e separado das mesmas não são condizentes.

Caso 3: Condição de máxima frequência



Obs: As cores das curvas nos gráficos conjunto e separado das mesmas não são condizentes.

6. Conclusão

Pode-se concluir que os resultados condizem de forma geral com o que havia sido esperado. Primeiramente, na letra “a”, os autovalores foram iguais ao esperado, além de que os autovetores encontrados foram uma combinação linear dos autovetores resultantes. Ademais, o fato de o número de iterações sem deslocamento ser bem maior do que com deslocamento, corrobora com a teoria de que este último é bem mais eficaz.

Nos itens seguintes, que já dizem respeito a aplicações diretas do algoritmo QR em um sistema massa-mola, também percebemos que o código teve êxito, uma vez que para as posições dadas houve um comportamento oscilatório e com $X(0)$ igual em cada massa e, no caso de máxima frequência, todas as massas tiveram frequências de oscilação iguais.