ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Exercício-Programa 1

Autovalores e Autovetores de Matrizes Tridiagonais Simétricas - O Algoritmo QR

MAP3121 - Métodos Numéricos e Aplicações

Gabriel Boaventura Scholl - 10771218
William Simões Barbosa - 9837646

SÃO PAULO
2021

Introdução

O objetivo deste trabalho é servir de apoio para o Exercício-Programa 1 da disciplina MAP3121 - Métodos Numéricos e Aplicações, sendo um relatório que aborda diversos aspectos, tanto conceituais, quanto práticos, incluindo, ainda, os resultados obtidos e suas respectivas análises. Estão inclusos na pasta compactada, também, o arquivo "LEIA-ME.txt" e o código fonte "EP1.py".

1. Análise do problema estudado

O problema estudado é um problema que aborda os modos de vibração de sistemas massa-mola, que são problemas de extrema utilidade e importância, com aplicações em diversas áreas da engenharia e da ciência. Um exemplo clássico de aplicação é o cálculo estrutural do chassis de um carro. Nele são feitas, dentre outras, análises modais. São análises dos modos de vibração do chassis do carro, bem como de partes específicas dele, para inferir características importantes para tanto o conforto quanto a segurança do usuário do veículo.

Com tamanha importância e utilidade, o sistema massa-mola é essencial para o aprendizado do aluno das áreas STEM (*Science, Technology, Engineering and Mathematics*, no inglês, ou Ciência, Tecnologia, Engenharia e Matemática) e seu desenvolvimento, tanto para a área acadêmica quanto para as empresas. O sistema massa-mola, estudado durante o Exercício Programa 1 de Cálculo Numérico, é um tipo de sistema que pode ser modelado através de equações diferenciais. Mais especificamente, um sistema de equações diferenciais. Esse sistema, neste caso, pôde ser representado por uma equação matricial no qual a matriz de maior interesse é uma matriz tridiagonal simétrica. Sendo assim, pode ser descrita por seus autovalores e autovetores de forma a simplificar muitos cálculos e possibilitar o uso de técnicas para redução do tempo computacional, como será abordado neste EP.

Uma vez tendo a matriz A tridiagonal simétrica, é possível prosseguir com a sua diagonalização para subsequente resolução do sistema de equações diferenciais. Os autovetores dessa matriz são os modos de vibração do sistema, enquanto que os autovalores são as frequências de vibração de cada massa.

Na aplicação ao sistema massa-mola, para resolver as equações diferenciais foram usadas as condições iniciais de posição das massas, que nos possibilitou saber os termos a_i que multiplicam os termos de cosseno da solução geral $x(t) = a \cdot cos(\omega t) + b \cdot sen(\omega t)$. Para os termos de seno, utilizamos a condição de velocidade inicial nula para determinar que os b_i seriam todos nulos. Para determinar todos os coeficientes a_i, multiplicamos a transposta da matriz de autovetores pelo vetor X0 das posições iniciais. A partir disso, já podíamos calcular as posições das massas ao longo do tempo, apenas multiplicando a matriz de autovetores por um vetor composto por a_i vezes cosseno de ômega vezes t em cada linha. Os resultados disso serão vistos na seção 3 deste trabalho.

2. Transcrição do código

```
import matplotlib.pyplot as plt
import numpy as np # Usar apenas para aritmética de vetores, matrizes, leitura e escrita de dados.
import math

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

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

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

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

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

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

```
    return Qk

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

    return Q, R

def algoritmoQR(A0, V0,  n, erro):
     #Função que recebe a matriz A0 de dimensão n e um erro definido no enunciado, V0 a princípio corresponde à matriz identidade
    A = np.copy(A0)
    V = np.copy(V0)
    alfa, beta, gama = vetoresAlfaBetaGama(A)
    k = 0 #Calcular o número de passos
    while (abs(beta[0]) >= erro):
        Q, R = decomposicaoQR (A, n)
        A = R @ Q
        V = V @ Q
        alfa, beta, gama = vetoresAlfaBetaGama(A)
        k += 1
    print("Passos: ", k)

    return A, V

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

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

def inverteLista(lista):
  inversa = []
  for i in range(len(lista)-1,-1,-1):
    inversa.append(lista[i])
  return inversa

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

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

```
            else:
                sgn = -1
            mik = alfa[n-1] + dk - sgn * math.sqrt(dk**2 + beta[n-2]**2)

        Qk, Rk = decomposicaoQR (Ak - mik * I, n) #Realização da decomposição QR com a matriz A atual (2 =< dimensão <= n)
        Ak = Rk @ Qk + mik * I #Dados Qk e Rk da decomposição obtida, calcular A[k+1], usando mik
        #print("Ak: \n", Ak)
        Qk = arrumaDimensao(Qk, nOriginal) #Se dimensão de A for menor que n, preciso arrumar a dimensão de Qk para calcular V[k+1]
        Vk = Vk @ Qk #Cálculo de V[k+1] a partir de V[k] e da matriz Q
        alfa, beta, gama = vetoresAlfaBetaGama(Ak) #Obtendo os novos alfa, beta e gama.

        k += 1

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

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

    return autovalores, Vk

def devolveAutovalores (matrizA):
```

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

def modoMaiorFrequencia(Vk, autovalores): #Função que retorna o modo de maior frequência
    #Para tal, esta função identifica qual é o maior autovalor encontrado
    maiorAutoValor = autovalores[0]
    indice = 0
    for i in range(len(autovalores)):
        if (autovalores[i] > maiorAutoValor):
            maiorAutoValor = autovalores[i]
            indice = i

    listaAutovetorAssociado = []
    for i in range(len(Vk)):
        for j in range(len(Vk[0])):
            if (j == indice):
                listaAutovetorAssociado.append(Vk[i][j])

    return listaAutovetorAssociado

# A função ploarGraficos recebe uma matriz "x" da evolução do sistema no tempo, entre outras entradas, e
# retorna um gráfico disso.
def plotarGraficos(item, x, i, vetorInicial, passos):
    # Aqui, são feitas as colocações de escritas nos gráficos, como o título e outros.
    eixo = np.array([valor*0.025 for valor in range(passos+1)])
    plt.plot(eixo, x[i])
    title = f'{item}) Vetor inicial {vetorInicial} - Posição da massa número: {i+1}'
    plt.title(title)
    plt.xlabel("Tempo (s)")
    plt.ylabel("Posição da massa (m)")
      # name = r'G:\RL Desktop\Numerico\EP1\imgs\\'+str(item)+'.'+str(vetorInicial)+') Massa '+str(i+1)+'.jpg'
    # plt.savefig(name)
    plt.show() # Descomentar isso pra mostrar os gráficos
    plt.clf()
```

```
# Essa função plotarConjunto() faz a mesma coisa que a anterior (plotarGraficos()), mas com o gráfico conjunto
# de todas as massas.
def plotarConjuto(item, x, vetorInicial, passos):
    eixo = np.array([valor*0.025 for valor in range(passos+1)])
    plt.plot(eixo, x.T)
    title = f'{item}) Vetor inicial {vetorInicial} - Gráfico conjunto - Posição das massas'
    plt.title(title)
    plt.xlabel("Tempo (s)")
    plt.ylabel("Posição da massa (m)")
        # name = r'G:\RL Desktop\Numerico\EP1\imgs\\'+str(item)+'.'+str(vetorInicial)+')Conjunto.jpg'
    # plt.savefig(name)
    plt.show() # Descomentar isso pra mostrar os gráficos
    plt.clf()

def main():
    # Aqui o usuário escolhe o item do tópico 5 do enunciado do EP.
    item = input("Qual item quer verificar ('a', 'b' ou 'c')? ")

    if (item == 'a'): #Item 'a': testar a execução do algoritmo QR com e sem deslocamento espectral
        # Aqui escolhe se é com deslocamento ou não e logo abaixo, a dimensão da matriz.
        desloc = input("Com deslocamento espectral ('s' para Sim, 'n' para Não)? ")
        n = int(input("Digite a dimensão da matriz (4, 8, 16, 32): "))
        #Para um primeiro teste (o do enunciado), temos as diagonais com um só valor (2 para diagonal principal, -1 para as demais):
        #Mesmo os valores sendo definidos no enunciado, deixamos o usuário escolher o valor que quer verificar.
        valor_diagonal_principal = int(input("Digite o valor da diagonal principal: "))
        valor_subdiagonal = int(input("Digite o valor da subdiagonal: "))
        #Erro pre-definido no enunciado do item 'a'
        erro = 0.000001
        # V0 começa como a matriz identidade n x n.
        V0 = np.identity(n, dtype = float)

        #Aqui e no bloco "for" logo abaixo são colocados os valores das diagonais da matriz A0.
        A0 = valor_diagonal_principal * np.identity(n, dtype = float)
        # Para isso, ele percorre a matriz vendo se a posição é uma diagonal secundária e, caso seja,
        # atribui o valor que o usuário entrou nos inputs acima.
        for i in range(n):
```

```
            for j in range(n):
                if (i == j + 1):
                    A0[i][j] = valor_subdiagonal
                if (j == i + 1):
                    A0[i][j] = valor_subdiagonal


        # Então é aplicado o algoritmo QR, com ou sem deslocamento espectral, conforme escolhido acima (no
    # começo deste "if").
    if (desloc == 's'): #Se com deslocamento espectral
        autovalores, Vk = algoritmoQR_deslocamento(A0, V0, n, erro)
        print("Autovalores: ", autovalores)
        print("Autovetores: \n", Vk)
         # Abaixo é aplicado o caso sem deslocamento. Nos dois casos os resultados são mostrados para o usuário
    elif(desloc == 'n'):
        Ak, Vk = algoritmoQR(A0, V0, n, erro)
        print("Autovalores: ", devolveAutovalores(Ak))
        print("Autovetores: \n", Vk)

# Os itents b) e c) são tratados como uma situação parecida, por isso estão no mesmo elif.
elif (item == 'b' or item == 'c'):
    # Aqui não se pede o tamanho da matriz, ele é determinado pelo numero de massas.
    # Para um segundo teste (o do enunciado), temos:
        # Como é um sistema massa-mola, as diagonais estão determinadas a partir das constantes
    # elásticas e das massas. Abaixo são definidas para cada caso, b) ou c).
    if item == 'b':
        massa = 2 #kg
        numero_de_massas = 5 #portanto são 6 molas
        constantes_elasticas = [(40+2*i) for i in range(1, numero_de_massas+2, 1)]
    elif item == 'c':
        massa = 2 #kg
        numero_de_massas = 10 #portanto são 11 molas
        constantes_elasticas = [(40+2*(-1)**i) for i in range(1, numero_de_massas+2, 1)]
    # Os valores são, então, renomeados para melhor legibilidade das fórmulas.
    k = constantes_elasticas
    n = numero_de_massas
     # Daqui para baixo, são definidos o erro, V0 e A0 da mesma forma que no item a, com uma
    # diferença a ser comentada.
```

```
    # Erro usado.
    erro = 0.000001
    V0 = np.identity(n, dtype = float)

    A0 = np.identity(n, dtype = float)
    for i in range(n):
        for j in range(n):
            # A diferença se mostra aqui, com as definições das diagonais sendo funções dos "k" e das massas.
            if (i == j):
                A0[i][j] = (k[i]+k[i+1])/massa
            if (i == j + 1):
                A0[i][j] = -k[i]/massa
            if (j == i + 1):
                A0[i][j] = -k[j]/massa

     # É aplicado o algoritmo QR com deslocamento espectral, conforme pedido e os valores são impressos.
    autovalores, Vk = algoritmoQR_deslocamento(A0, V0, n, erro)

    print("Com deslocamento:")
    print("Autovalores (Frequências de vibração - Hz): \n", autovalores)
    print("Autovetores (Modos naturais de vibração): \n", Vk)
    print()

      # Daqui em diante, começa uma escolha de qual vetor X(0), de posições iniciais, o usuário quer.
     # É feito uma pergunta em forma de input e, a partir disso, o vetor é determinado, seja relativo
    # ao item b) ou ao item c).
    if item == 'b':
        X0 = []
        print("1 - X(0) = -2, -3, -1, -3, -1")
        print("2 - X(0) = 1, 10, -4, 3, -2")
        print("3 - X(0) correspondente ao modo de maior frequência")
        vetorInicial = int(input("Escolha o X0 digitando 1, 2 ou 3: "))
        if (vetorInicial == 1):
            X0 = np.array([-2, -3, -1, -3, -1])
        elif (vetorInicial == 2):
            X0 = np.array([1, 10, -4, 3, -2])
        elif (vetorInicial == 3):
            X0 = np.array(modoMaiorFrequencia(Vk, autovalores))
```

```
    elif item == 'c':
        X0 = []
        print("1 - X(0) = -2, -3, -1, -3, -1, -2, -3, -1, -3, -1")
        print("2 - X(0) = 1, 10, -4, 3, -2, 1, 10, -4, 3, -2")
        print("3 - X(0) correspondente ao modo de maior frequência")
        vetorInicial = int(input("Escolha o X0 digitando 1, 2, ou 3: "))
        if (vetorInicial == 1):
            X0 = np.array([-2, -3, -1, -3, -1, -2, -3, -1, -3, -1])
        elif (vetorInicial == 2):
            X0 = np.array([1, 10, -4, 3, -2, 1, 10, -4, 3, -2])
        elif (vetorInicial == 3):
            X0 = np.array(modoMaiorFrequencia(Vk, autovalores))

    # Com o valor das condições iniciais escolhido, agora prosseguimos com a simulação do sistema
    # massa-mola em intervalos de 0.025 segundos, que são 400 passos em 10 segundos.
    passos = 400 #Plotando o gráfico para 10s, teremos um valor a cada 0.025s
    x = np.zeros((n,passos+1), dtype = float)

        # A multiplicação de matrizes abaixo é feita com a notação do numpy. Isso foi considerado como
    # "aritmética de matrizes", permitida pelo enunciado.
    a = Vk.T @ X0.T
      # No loop "for" abaixo, varremos a matriz x, que vai ser a simulação ao longo de um eixo e as
        # massas ao longo de outro, calculando e colocando cada posição de cada massa ao longo do tempo.
    for i in range(n):
        for t in range(passos+1):
            for j in range(n):
                # O valor de ômega (representado por "w" aqui) é a raiz dos lâmbdas, que estão em autovalores[j].
                w = math.sqrt(float(autovalores[j]))
                x[i][t] += Vk[i][j] * a[j] * math.cos(w*t*0.025)

            # Isso aqui embaixo plota graficos individuais pra arquivos de imagem na pasta determinada
        # dentro da função plotarGraficos, ou apenas mostra para o usuário, dependendo da necessidade.
        plotarGraficos(item,x,i,vetorInicial,passos)
```

```
    # Esse é o plot do gráfico de todas as massas em conjunto, da mesma forma que a plotarGraficos().
    plotarConjuto(item,x,vetorInicial,passos)

# Finalmente, com tudo organizado em funções, a main() é chamada para executar o código.
main()
```

3. Resultados e análises

3.1. Tarefa 1 - Implementação do algoritmo QR com deslocamento espectral

Diferente do Algoritmo QR sem deslocamento espectral, que também foi implementado neste Exercício-Programa, o Algoritmo QR com deslocamento espectral utiliza a heurística de Wilkinson para aumentar a taxa de convergência de nosso algoritmo.

Podemos mencionar, também, que a diminuição da dimensão de nossa matriz é uma etapa fundamental deste algoritmo, fazendo sua eficiência ser muito superior à eficiência do Algoritmo QR sem deslocamento. No item 3.2.3 mostraremos com mais detalhes as diferenças obtidas entre os dois algoritmos implementados utilizando matrizes de diversas dimensões (n = 4, 8, 16, 32).

Para a implementação em si do algoritmo, alguns cuidados tiveram de ser tomados com relação à dimensão da matriz que estávamos utilizando. Quando o último valor da diagonal secundária de nossa matriz ficava com módulo menor que o erro definido no enunciado, salvávamos o autovalor que se encontrava no último valor da diagonal principal, e realizávamos uma diminuição da dimensão da matriz, de nxn para n-1xn-1. Com isso, deste passo em diante, a decomposição QR nos retorna matrizes de dimensão n-1 (até que uma nova redução seja feita), e precisamos de matrizes de dimensão n para calcular nossa matriz de autovetores (Vk). Sendo assim, implementamos duas funções auxiliares (“reduzDimensao(An)” e “arrumaDimensao(Qk, nOriginal)”) para facilitar estas operações.

Para a implementação em si do algoritmo, seguimos o passo a passo que foi disponibilizado no enunciado do EP.

1: Sejam $A^{(0)} = A \in \mathbb{R}^{n \times n}$ uma matriz tridiagonal simétrica, $V^{(0)} = I$ e $\mu_0 = 0$. O algoritmo calcula a sua forma diagonal semelhante $A = V \Lambda V^T$.

2: $k = 0$

3: **para** $m = n, n-1, \ldots, 2$ **faça**

4: **repita**

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

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

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

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

9: $k = k + 1$

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

11: **fim do para**

12: $\Lambda = A^{(k)}$

13: $V = V^{(k)}$

Uma parte importante da implementação do nosso Exercício-Programa consistiu na implementação da fatoração QR, utilizada tanto na decomposição com deslocamento espectral, quanto na decomposição sem deslocamento espectral. Nesta, transformamos uma matriz tridiagonal simétrica em uma matriz diagonal superior através de rotações de Givens.

3.2. Tarefa 2 - Item a) do enunciado

3.2.1. Definindo a matriz

Neste item, a definição da matriz foi bastante simples. Prosseguimos com a implementação em código daquilo pedido no enunciado: que a diagonal principal tivesse um valor repetidas vezes a subdiagonal um outro valor, também repetido ao longo de seu comprimento. Para isso, foi apenas necessário percorrer a matriz, atribuindo o valor caso a posição da matriz fosse a desejada.

3.2.2. Cálculo dos autovetores e autovalores

Para o cálculo dos autovetores e dos autovalores, seguimos o passo a passo do Algoritmo QR explicado no enunciado deste Exercício-Programa. Após realizada a decomposição QR de nossa matriz A, utilizamos estas matrizes obtidas para calcular a nova matriz A(k+1). A cada decomposição QR, também atualizávamos os valores da matriz V(k+1).

Quando todos os valores da diagonal secundária de nossa matriz A estivessem zerados (na prática, quando estivessem menores que um erro estipulado), a diagonal principal da

matriz A continha todos os autovalores que buscávamos, então implementamos uma função auxiliar simples (“devolveAutovalores (matrizA)”), para obter uma lista ordenada com esses números.

Nesta última passagem, realizando o último produto matricial Vk * Qk, a matriz V(k+1) resultava na matriz de autovetores buscada. Nesta matriz, os autovetores associados aos autovalores da diagonal principal de A(k+1) se encontravam nas colunas de V(k+1).

3.2.3. Comparação do número de iterações

A decomposição QR com deslocamento espectral converge para a matriz buscada de forma muito mais rápida que a decomposição QR sem deslocamento espectral. Abaixo exporemos imagens da execução de nosso Exercício-Programa (EP) de forma a evidenciar essas diferenças nas eficiências dos dois algoritmos.

Para um primeiro teste, usando as entradas dadas no item a) do enunciado do EP (diagonal principal constante igual a 2, subdiagonal igual a -1 e erro igual a 10^-6), com uma matriz 4x4, nosso EP encontra uma aproximação para os autovalores e autovetores em 7 passos para a decomposição QR com deslocamento espectral, e em 45 passos sem deslocamento espectral, como mostra a figura abaixo.

```
C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? s
Digite a dimensão da matriz (4, 8, 16, 32): 4
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  7
Autovalores:  [3.6180339887498945, 2.6180339887498936, 1.3819660112501053, 0.3819660112501052]
Autovetores:
 [[ 0.37174803  0.60150096  0.60150096  0.37174803]
 [-0.60150096 -0.37174803  0.37174803  0.60150096]
 [ 0.60150096 -0.37174803 -0.37174803  0.60150096]
 [-0.37174803  0.60150096 -0.60150096  0.37174803]]

C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? n
Digite a dimensão da matriz (4, 8, 16, 32): 4
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  45
Autovalores:  [3.618033988749299, 2.618033988750492, 1.3819660112501073, 0.38196601125010493]
Autovetores:
 [[ 0.3717485   0.60150067  0.60150096  0.37174803]
 [-0.60150124 -0.37174757  0.37174803  0.60150096]
 [ 0.60150067 -0.3717485  -0.37174803  0.60150096]
 [-0.37174757  0.60150124 -0.60150096  0.37174803]]
```

Realizando novos testes com as mesmas entradas, exceto pela dimensão da matriz, que passa a ser 8x8, nosso algoritmo encontra uma aproximação para os autovalores e autovetores em 15 passos com deslocamento espectral, e em 143 passos sem o deslocamento espectral. As próximas duas imagens correspondem a esta rodada de testes.

```
C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? s
Digite a dimensão da matriz (4, 8, 16, 32): 8
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  15
Autovalores:  [3.879385241571816, 3.5320888862379554, 3.0, 2.3472963553338606, 1.6527036446661396, 1.0000000000000004, 0
.4679111137620438, 0.12061475842818327]
Autovetores:
 [[ 1.61229842e-01  3.03012985e-01  4.08248290e-01  4.64242827e-01
   4.64242827e-01  4.08248290e-01  3.03012985e-01  1.61229842e-01]
 [-3.03012985e-01 -4.64242827e-01 -4.08248290e-01 -1.61229842e-01
   1.61229842e-01  4.08248290e-01  4.64242827e-01  3.03012985e-01]
 [ 4.08248290e-01  4.08248290e-01 -2.19835146e-14 -4.08248290e-01
  -4.08248290e-01  2.52492169e-12  4.08248290e-01  4.08248291e-01]
 [-4.64242827e-01 -1.61229842e-01  4.08248290e-01  3.03012985e-01
  -3.03012985e-01 -4.08248290e-01  1.61229842e-01  4.64242827e-01]
 [ 4.64242827e-01 -1.61229842e-01 -4.08248290e-01  3.03012985e-01
   3.03012985e-01 -4.08248290e-01 -1.61229842e-01  4.64242827e-01]
 [-4.08248290e-01  4.08248290e-01 -2.29591194e-14 -4.08248290e-01
   4.08248290e-01  2.46083488e-12 -4.08248291e-01  4.08248290e-01]
 [ 3.03012985e-01 -4.64242827e-01  4.08248290e-01 -1.61229842e-01
  -1.61229842e-01  4.08248290e-01 -4.64242827e-01  3.03012985e-01]
 [-1.61229842e-01  3.03012985e-01 -4.08248290e-01  4.64242827e-01
  -4.64242827e-01  4.08248290e-01 -3.03012985e-01  1.61229842e-01]]
```

```
C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? n
Digite a dimensão da matriz (4, 8, 16, 32): 8
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  143
Autovalores:  [3.8793852415690715, 3.532088886240709, 3.000000000000002, 2.3472963553338646, 1.652703644666142, 1.000000
0000000004, 0.46791111376204475, 0.12061475842818327]
Autovetores:
 [[ 1.61230695e-01  3.03012531e-01  4.08248290e-01  4.64242827e-01
   4.64242827e-01  4.08248290e-01  3.03012985e-01  1.61229842e-01]
 [-3.03014292e-01 -4.64241974e-01 -4.08248290e-01 -1.61229842e-01
   1.61229842e-01  4.08248290e-01  4.64242827e-01  3.03012985e-01]
 [ 4.08249440e-01  4.08247141e-01 -1.00851278e-10 -4.08248290e-01
  -4.08248290e-01  2.76574180e-16  4.08248290e-01  4.08248290e-01]
 [-4.64243281e-01 -1.61228535e-01  4.08248291e-01  3.03012985e-01
  -3.03012985e-01 -4.08248290e-01  1.61229842e-01  4.64242827e-01]
 [ 4.64242373e-01 -1.61231149e-01 -4.08248290e-01  3.03012985e-01
   3.03012985e-01 -4.08248290e-01 -1.61229842e-01  4.64242827e-01]
 [-4.08247141e-01  4.08249440e-01 -1.00851827e-10 -4.08248290e-01
   4.08248290e-01 -6.76413033e-16 -4.08248290e-01  4.08248290e-01]
 [ 3.03011678e-01 -4.64243680e-01  4.08248291e-01 -1.61229842e-01
  -1.61229842e-01  4.08248290e-01 -4.64242827e-01  3.03012985e-01]
 [-1.61228989e-01  3.03013439e-01 -4.08248291e-01  4.64242827e-01
  -4.64242827e-01  4.08248290e-01 -3.03012985e-01  1.61229842e-01]]
```

Seguindo testes para matrizes de dimensão 16x16, com as mesmas entradas que os itens anteriores, encontramos uma aproximação para os autovalores e autovetores em 31 passos utilizando deslocamento espectral, e em 473 passos sem o deslocamento. As próximas duas imagens correspondem ao teste do item a) para matrizes 16x16.

```
C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? s
Digite a dimensão da matriz (4, 8, 16, 32): 16
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  31
Autovalores:  [3.9659461993678033, 3.8649444588087123, 3.700434271459227, 3.478017834441318, 3.2052692727585117, 2.89147
67115530773, 2.547325980144167, 2.1845367189266045, 1.8154632810733964, 1.4526740198558343, 1.1085232884469236, 0.794730
7272414876, 0.5219821655586818, 0.2995657285407677, 0.13505554119129204, 0.03405380063219633]
Autovetores:
 [[ 0.06302556  0.12390487  0.18056474  0.2310757   0.27371765  0.30703848
   0.32990347  0.341534    0.341534    0.32990347  0.30703848  0.27371765
   0.2310757   0.18056472  0.1239049   0.06302556]
 [-0.12390487 -0.2310757  -0.30703848 -0.341534   -0.32990347 -0.27371765
  -0.18056474 -0.06302556  0.06302556  0.18056474  0.27371765  0.32990347
   0.34153401  0.30703844  0.23107574  0.12390487]
 [ 0.18056474  0.30703848  0.341534    0.27371765  0.12390487 -0.06302556
  -0.2310757  -0.32990347 -0.32990347 -0.2310757  -0.06302556  0.12390487
   0.27371765  0.34153396  0.30703853  0.18056474]
 [-0.2310757  -0.341534   -0.27371765 -0.06302556  0.18056474  0.32990347
   0.30703848  0.12390487 -0.12390487 -0.30703848 -0.32990347 -0.18056474
   0.06302557  0.2737176   0.34153405  0.2310757 ]
 [ 0.27371765  0.32990347  0.12390487 -0.18056474 -0.341534   -0.2310757
   0.06302556  0.30703848  0.30703848  0.06302556 -0.2310757  -0.34153401
  -0.18056474  0.12390482  0.32990349  0.27371765]
 [-0.30703848 -0.27371765  0.06302556  0.32990347  0.2310757  -0.12390487
  -0.341534   -0.18056474  0.18056474  0.341534    0.12390487 -0.2310757
  -0.32990347 -0.06302561  0.27371764  0.30703848]
 [ 0.32990347  0.18056474 -0.2310757  -0.30703848  0.06302556  0.341534
   0.12390487 -0.27371765 -0.27371765  0.12390487  0.341534    0.06302556
  -0.30703847 -0.23107573  0.18056471  0.32990347]
 [-0.341534   -0.06302556  0.32990347  0.12390487 -0.30703848 -0.18056474
   0.27371765  0.2310757  -0.2310757  -0.27371765  0.18056474  0.30703848
  -0.12390487 -0.32990348  0.06302552  0.341534  ]
 [ 0.341534   -0.06302556 -0.32990347  0.12390487  0.30703848 -0.18056474
  -0.27371765  0.2310757   0.2310757  -0.27371765 -0.18056474  0.30703848
   0.12390487 -0.32990346 -0.06302562  0.341534  ]
 [-0.32990347  0.18056474  0.2310757  -0.30703848 -0.06302556  0.341534
  -0.12390487 -0.27371765  0.27371765  0.12390487 -0.341534    0.06302556
   0.30703847 -0.23107567 -0.18056478  0.32990347]
 [ 0.30703848 -0.27371765 -0.06302556  0.32990347 -0.2310757  -0.12390487
   0.341534   -0.18056474 -0.18056474  0.341534   -0.12390487 -0.2310757
```

```
C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? n
Digite a dimensão da matriz (4, 8, 16, 32): 16
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  473
Autovalores:  [3.965946199357954, 3.8649444588185564, 3.7004342714592475, 3.478017834441324, 3.2052692727585144, 2.89147
67115530844, 2.54732598014417, 2.1845367189266014, 1.8154632810733953, 1.452674019855837, 1.1085232884469236, 0.79473072
7241488, 0.5219821655586825, 0.29956572854077157, 0.13505554119128835, 0.03405380063219639]
Autovetores:
 [[ 0.06302679  0.12390425  0.18056474  0.2310757   0.27371765  0.30703848
   0.32990347  0.341534    0.341534    0.32990347  0.30703848  0.27371765
   0.2310757   0.18056474  0.12390487  0.06302556]
 [-0.12390715 -0.23107448 -0.30703847 -0.341534   -0.32990347 -0.27371765
  -0.18056474 -0.06302556  0.06302556  0.18056474  0.27371765  0.32990347
   0.341534    0.30703848  0.2310757   0.12390487]
 [ 0.18056777  0.3070367   0.341534    0.27371765  0.12390487 -0.06302556
  -0.2310757  -0.32990347 -0.32990347 -0.2310757  -0.06302556  0.12390487
   0.27371765  0.341534    0.30703848  0.18056474]
 [-0.23107907 -0.34153173 -0.27371765 -0.06302556  0.18056474  0.32990347
   0.30703848  0.12390487 -0.12390487 -0.30703848 -0.32990347 -0.18056474
   0.06302556  0.27371765  0.341534    0.2310757 ]
 [ 0.27372091  0.32990077  0.12390487 -0.18056474 -0.341534   -0.2310757
   0.06302556  0.30703848  0.30703848  0.06302556 -0.2310757  -0.341534
  -0.18056474  0.12390487  0.32990347  0.27371765]
 [-0.30704118 -0.27371462  0.06302557  0.32990347  0.2310757  -0.12390487
  -0.341534   -0.18056474  0.18056474  0.341534    0.12390487 -0.2310757
  -0.32990347 -0.06302556  0.27371765  0.30703848]
 [ 0.32990526  0.18056149 -0.2310757  -0.30703848  0.06302556  0.341534
   0.12390487 -0.27371765 -0.27371765  0.12390487  0.341534    0.06302556
  -0.30703848 -0.2310757   0.18056474  0.32990347]
 [-0.34153463 -0.06302219  0.32990347  0.12390487 -0.30703848 -0.18056474
   0.27371765  0.2310757  -0.2310757  -0.27371765  0.18056474  0.30703848
  -0.12390487 -0.32990347  0.06302556  0.341534  ]
 [ 0.34153338 -0.06302894 -0.32990347  0.12390487  0.30703848 -0.18056474
  -0.27371765  0.2310757   0.2310757  -0.27371765 -0.18056474  0.30703848
   0.12390487 -0.32990347 -0.06302556  0.341534  ]
 [-0.32990169  0.180568    0.2310757  -0.30703848 -0.06302556  0.341534
  -0.12390487 -0.27371765  0.27371765  0.12390487 -0.341534    0.06302556
   0.30703848 -0.2310757  -0.18056474  0.32990347]
 [ 0.30703577 -0.27372068 -0.06302556  0.32990347 -0.2310757  -0.12390487
```

E, como teste final, utilizamos os mesmos valores de entrada dos itens anteriores e testamos a eficiência de nossos algoritmos para matrizes 32x32. Neste, nosso algoritmo com deslocamento espectral foi capaz de achar uma aproximação para os autovalores e autovetores em apenas 59 passos, enquanto que o algoritmo sem deslocamento espectral encontrou a mesma aproximação em 1600 passos. As próximas duas imagens mostram como os valores são dispostos para quem executa o EP.

```
C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? s
Digite a dimensão da matriz (4, 8, 16, 32): 32
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  59
Autovalores:  [3.9909438451461674, 3.9638573945254123, 3.918985947228994, 3.856735866032144, 3.777670897309846, 3.682507
065662361, 3.5721061894855746, 3.447468076210139, 3.3097214678905686, 3.1601138191423535, 2.999999999994304, 2.830830026
009508, 2.6541359266348423, 2.4715178710188535, 2.28462967654657, 2.095163831647465, 1.9048361683489397, 1.7153703234570
272, 1.5284821289811459, 1.3458640733651563, 1.1691699739961994, 0.9999999999951403, 0.8398861808624933, 0.6902785321094
302, 0.55253192378986, 0.4278938105144254, 0.31749293433763487, 0.22232910269006342, 0.1432641339584813, 0.0810140527797
0247, 0.009056154854600122, 0.03614260547458664]
Autovetores:
 [[ 0.02340118  0.04659044  0.06935776 ...  0.06935873  0.0234014
  -0.04659044]
 [-0.04659044 -0.09149697 -0.13309657 ...  0.13309836  0.04659086
  -0.09149697]
 [ 0.06935776  0.13309657  0.18605268 ...  0.18605503  0.06935835
  -0.13309657]
 ...
 [-0.06935776  0.13309657 -0.18605268 ...  0.18604948  0.06935839
   0.13309657]
 [ 0.04659044 -0.09149697  0.13309657 ...  0.13309414  0.04659089
   0.09149697]
 [-0.02340118  0.04659044 -0.06935776 ...  0.06935645  0.02340142
   0.04659044]]
```

```
C:\Users\Corleone\Desktop>EP1-Semi_Comentado.py
Qual item quer verificar ('a', 'b' ou 'c')? a
Com deslocamento espectral ('s' para Sim, 'n' para Não)? n
Digite a dimensão da matriz (4, 8, 16, 32): 32
Digite o valor da diagonal principal: 2
Digite o valor da subdiagonal: -1
Passos:  1600
Autovalores:  [3.990943845109297, 3.9638573945622864, 3.9189859472290296, 3.8567358660321682, 3.7776708973098683, 3.6825
0706566236, 3.5721061894855697, 3.4474680762101486, 3.3097214678905744, 3.1601138191423908, 2.999999999999979, 2.8308300
260037607, 2.6541359266348277, 2.4715178710188517, 2.284629676546579, 2.09516383164748, 1.9048361683525088, 1.7153703234
534272, 1.5284821289811445, 1.3458640733651608, 1.1691699739962254, 1.0000000000000007, 0.8398861808576105, 0.6902785321
094317, 0.5525319237898595, 0.42789381051442477, 0.31749293433763726, 0.2223291026901534, 0.14326413396785487, 0.0810140
5277100539, 0.03614260547458658, 0.009056154853830796]
Autovetores:
 [[ 0.0234029   0.04658958  0.06935776 ...  0.06935776  0.04659044
   0.02340118]
 [-0.04659381 -0.09149526 -0.13309656 ...  0.13309657  0.09149697
   0.04659044]
 [ 0.06936267  0.13309402  0.18605268 ...  0.18605268  0.13309657
   0.06935776]
 ...
 [-0.06935285  0.13309912 -0.18605269 ...  0.18605268 -0.13309657
   0.06935776]
 [ 0.04658706 -0.09149868  0.13309657 ...  0.13309657 -0.09149697
   0.04659044]
 [-0.02339946  0.0465913  -0.06935776 ...  0.06935776 -0.04659044
   0.02340118]]
```

Com isso, podemos montar a tabela abaixo de forma a facilitar a visualização e comparação entre as decomposições com e sem deslocamento espectral.

Dimensão n	4	8	16	32
Número de passos com deslocamento	7	15	31	59
Número de passos sem deslocamento	45	143	473	1600

Assim é possível observar uma diferença brutal no tempo de execução do programa, atingindo um tempo com duas ordens de grandeza menores, e isso com matrizes de dimensão 32. É fácil ver o impacto que isso teria em matrizes maiores, com dimensão desde uma ou até quatro ordens de grandeza maiores.

3.3. Tarefa 3 - Item b) do enunciado

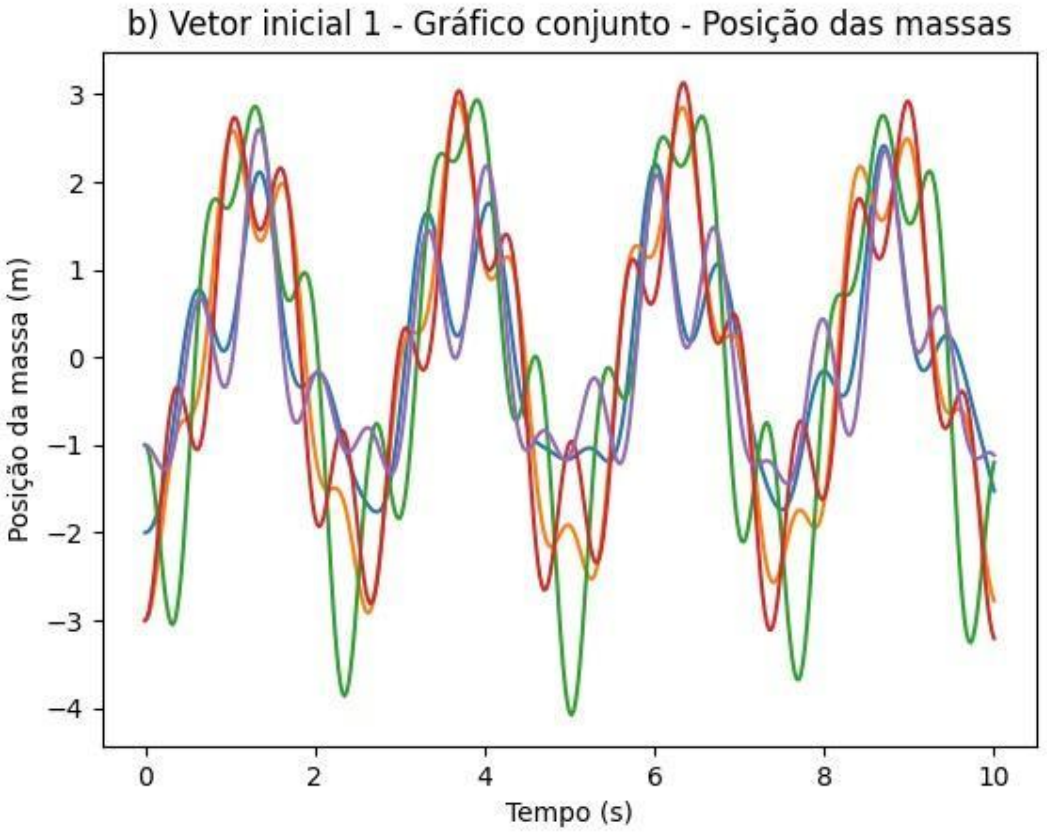
3.3.1. Definição do problema

O item b) do enunciado do Exercício Programa começa definindo o problema como um sistema massa-mola composto por 5 massas, com molas entre todas elas e as extremidades fixas. Cada massa tem 2 quilos e as constantes elásticas são 42, 44, 46, 48, 50 e 52.

Esse item foi modelado no código da seguinte maneira: as massas foram definidas como variáveis e as constantes elásticas como uma lista de valores. Desse modo, fica fácil prosseguir com sua colocação na matriz.

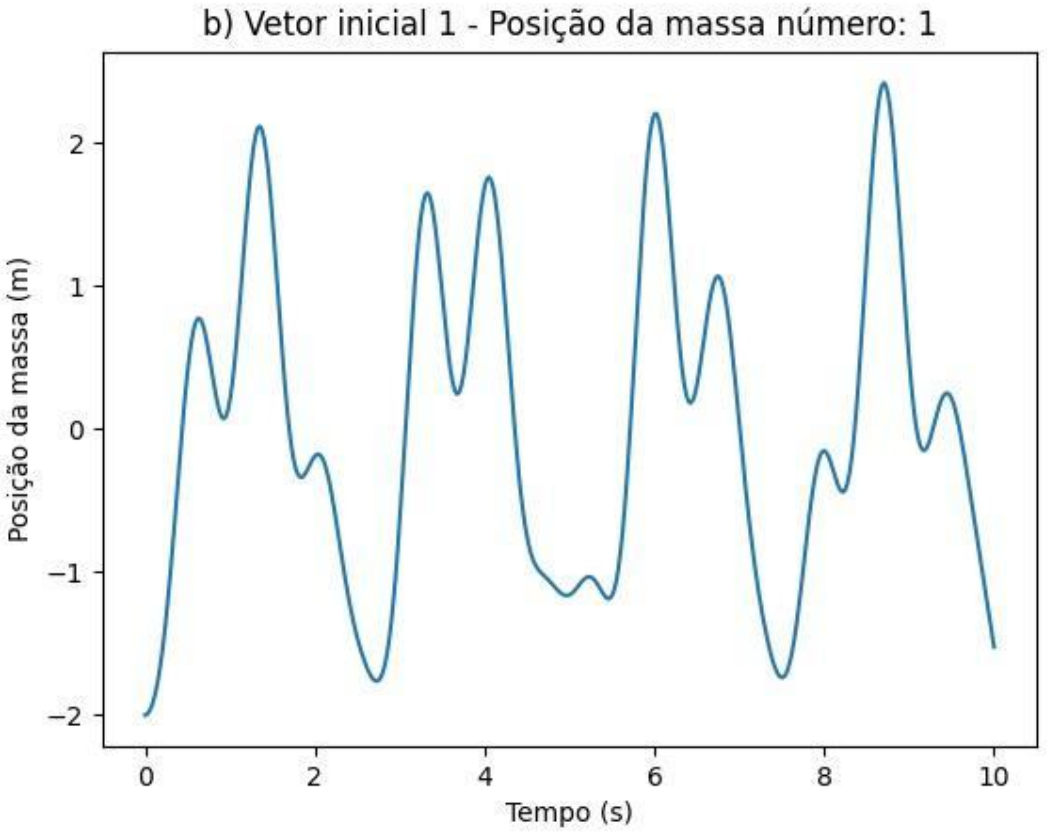
3.3.2. Caso 1 - X(0) = -2, -3, -1, -3, -1

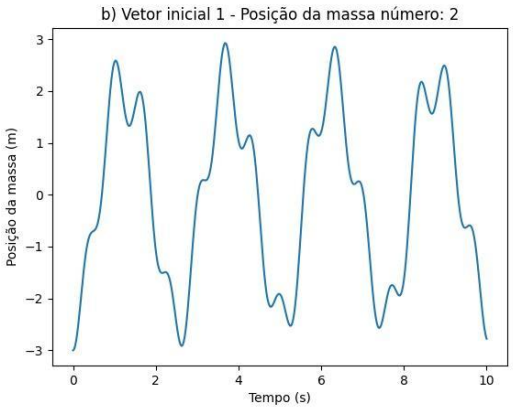
Neste caso, as condições iniciais foram dadas na forma de deslocamentos iniciais das massas e suas respectivas velocidades iniciais nulas. Abaixo, podemos ver os gráficos da solução. Eles mostram a evolução da posição de cada massa ao longo de 10 segundos. Primeiro, vamos apresentar o gráfico conjunto.

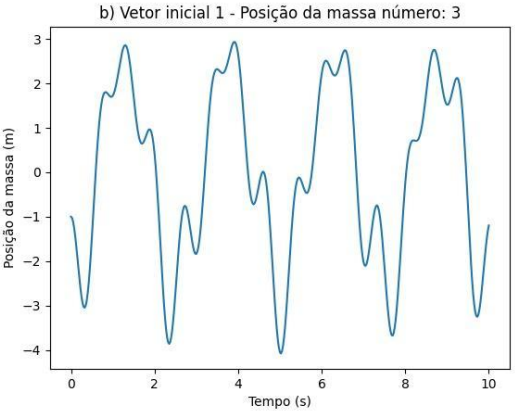


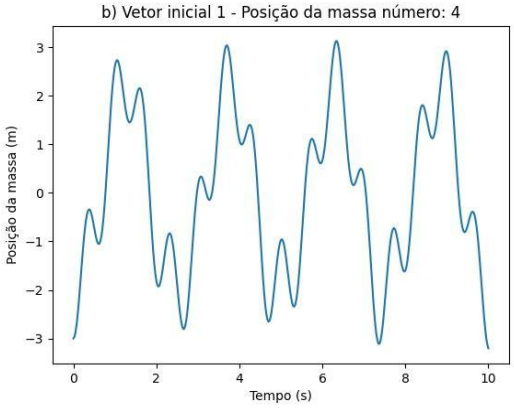
Neste gráfico, podemos observar algumas peculiaridades. Primeiro, vemos todas as 5 molas componentes do sistema indicadas, cada qual com uma cor diferente. Ao longo dos 10 segundos, é possível notar que o comportamento de cada mola é uma sobreposição de ondas, algumas com maior amplitude, outras com maior frequência e assim por diante.

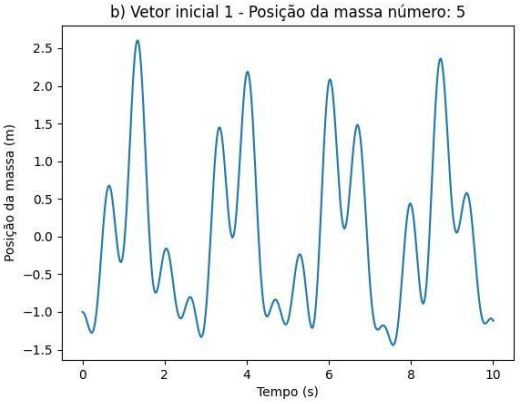
Aqui, ainda, é possível notar que, apesar de cada massa se movimentar em alta frequência em uma amplitude pequena e própria, todas as massas parecem demonstrar um comportamento de rebanho, no caso se movimentando em conjunto com uma frequência menor que suas frequências individuais, mas com uma amplitude maior. A seguir, vejamos os comportamentos das massas individualmente.


b) Vetor inicial 1 - Posição da massa número: 1
Posição da massa (m)
Tempo (s)

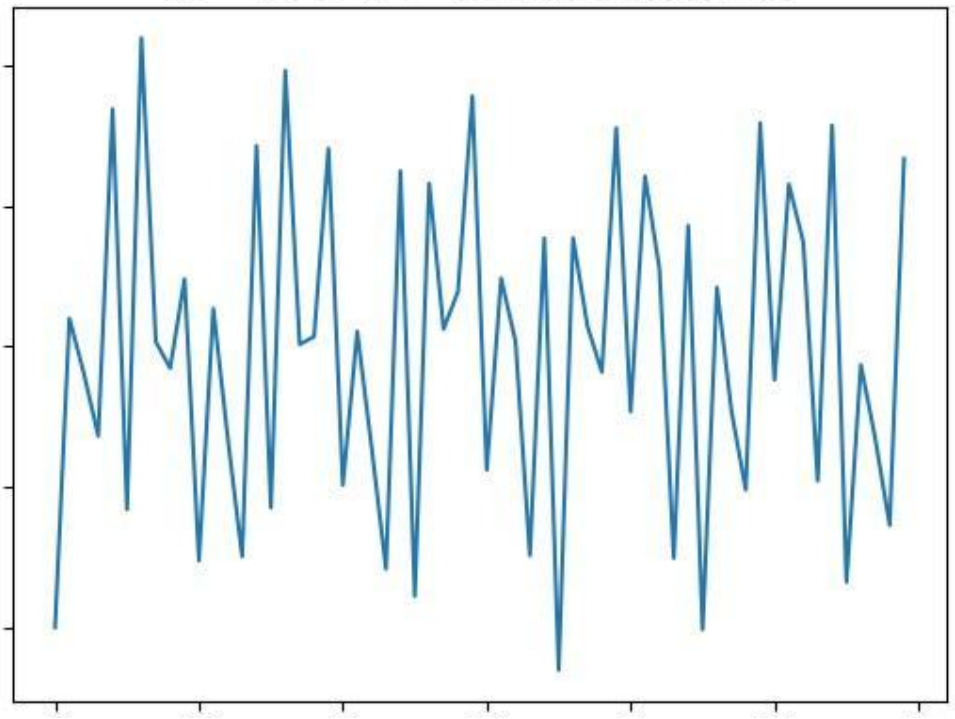

b) Vetor inicial 1 - Posição da massa número: 2
Posição da massa (m)
Tempo (s)


b) Vetor inicial 1 - Posição da massa número: 3
Posição da massa (m)
Tempo (s)


b) Vetor inicial 1 - Posição da massa número: 4
Posição da massa (m)
Tempo (s)

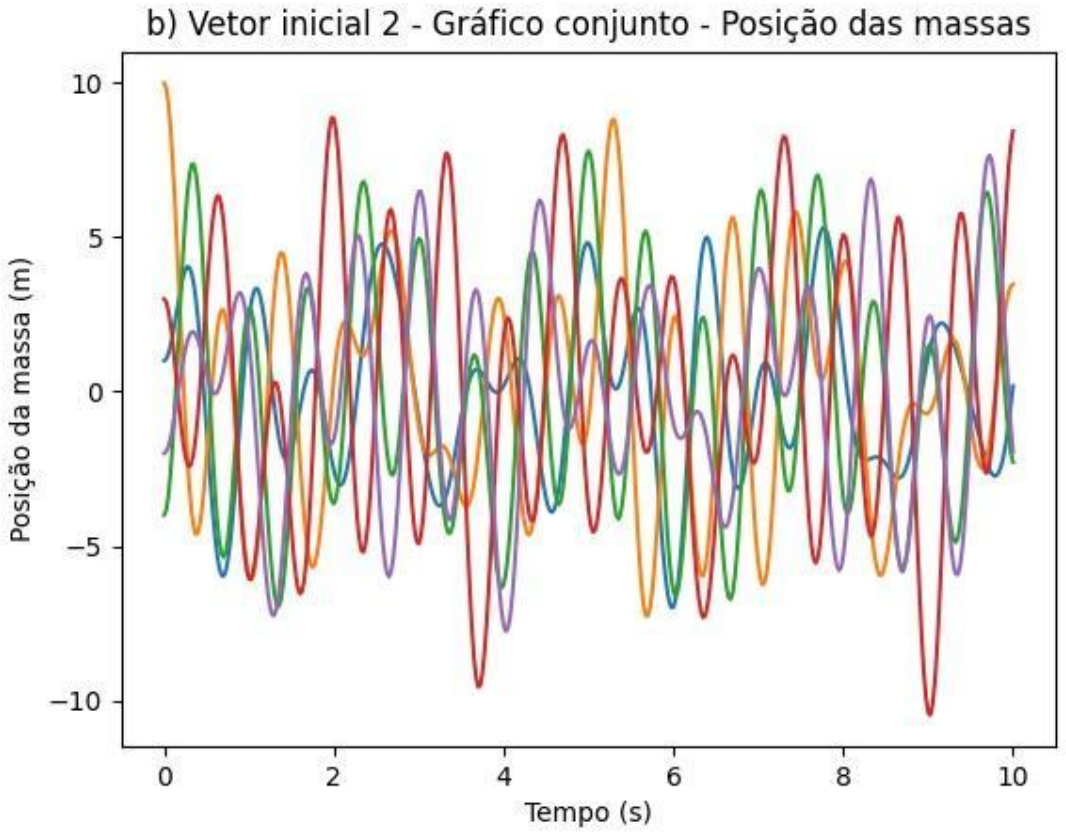

b) Vetor inicial 1 - Posição da massa número: 5
Posição da massa (m)
Tempo (s)

Com os gráficos individuais, é possível observar esse comportamento de sobreposição de ondas, com algumas de alta frequência e outras de baixa frequência, porém maior amplitude. Além disso, é bastante importante observar que a resolução dos gráficos está boa, com uma amostra a cada 0.025s. Isso se vê, por exemplo, pela clara observação das curvaturas nos pontos de máximo e mínimo. Esse fato é de grande importância, uma vez que caso a resolução fosse muito baixa, não seria possível observar alguns comportamentos ondulatórios de mais alta frequência, ficando com aparência de picos discretos e aparentemente sem padrão, como observado no teste abaixo.

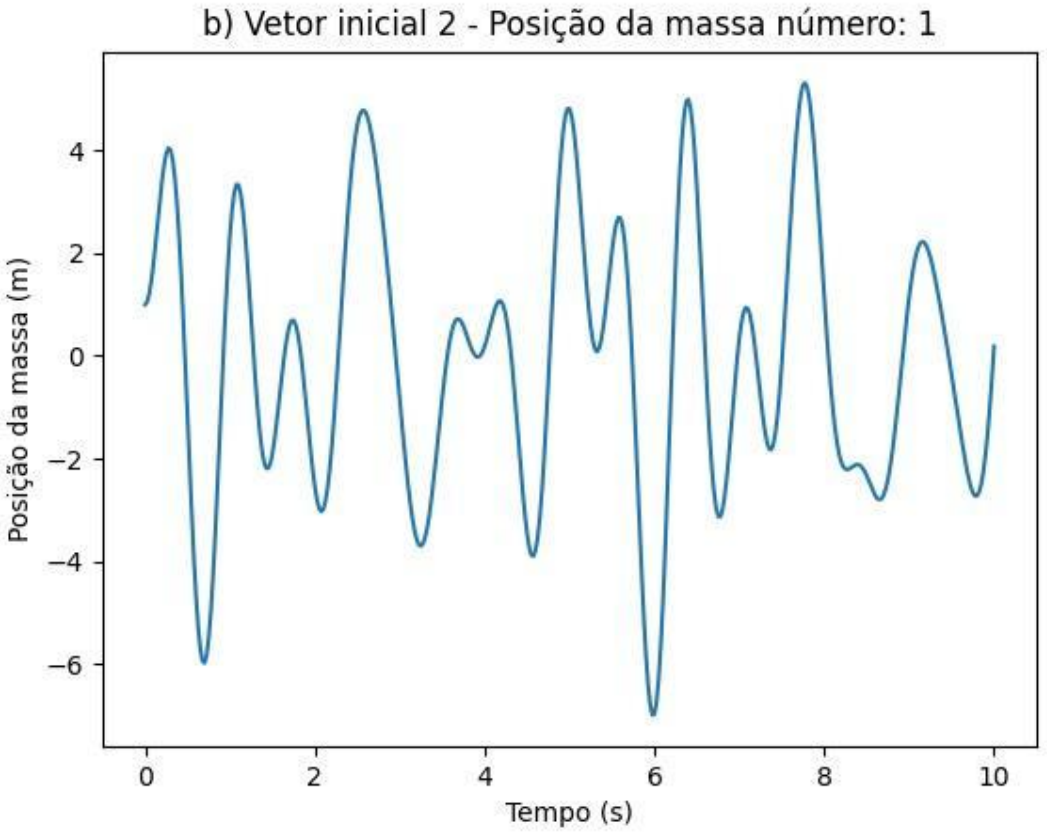


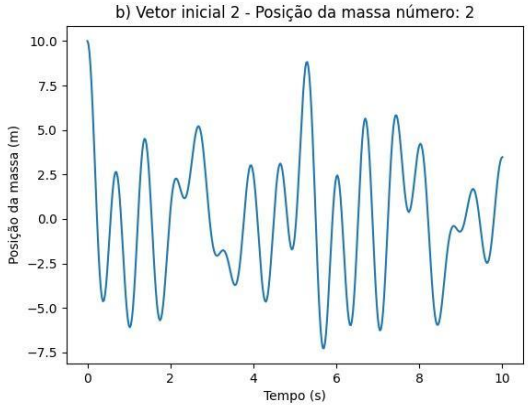
3.3.3. Caso 2 - X(0) = 1, 10, −4, 3, −2

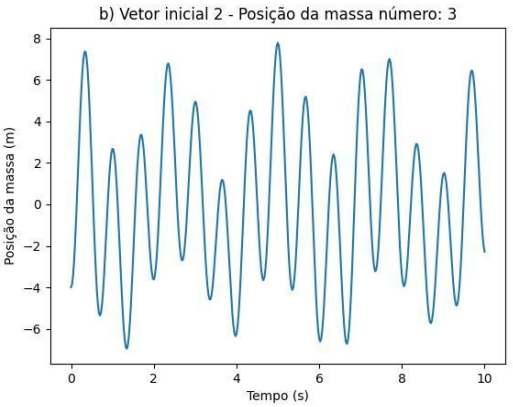
Neste caso, o vetor de posições iniciais foi outro. Diferentemente da situação anterior, onde as posições iniciais estavam mais próximas entre si e todas de um mesmo lado da origem, agora elas estão mais espalhadas. Isso pode ter sido a causa de o gráfico conjunto demonstrar uma característica praticamente caótica, como vemos abaixo.

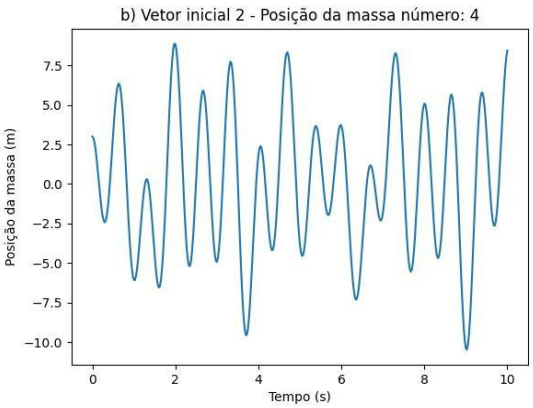


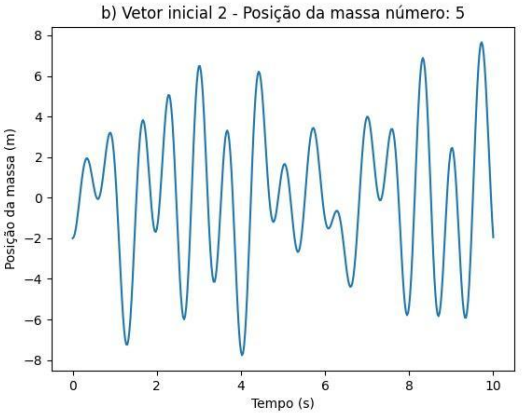
Nos gráficos individuais, pode-se observar que as frequências dominantes são mais altas, sendo essas deslocadas em alguns momentos por frequências mais baixas.


b) Vetor inicial 2 - Posição da massa número: 1
Posição da massa (m)
Tempo (s)


b) Vetor inicial 2 - Posição da massa número: 2
Posição da massa (m)
Tempo (s)


b) Vetor inicial 2 - Posição da massa número: 3
Posição da massa (m)
Tempo (s)

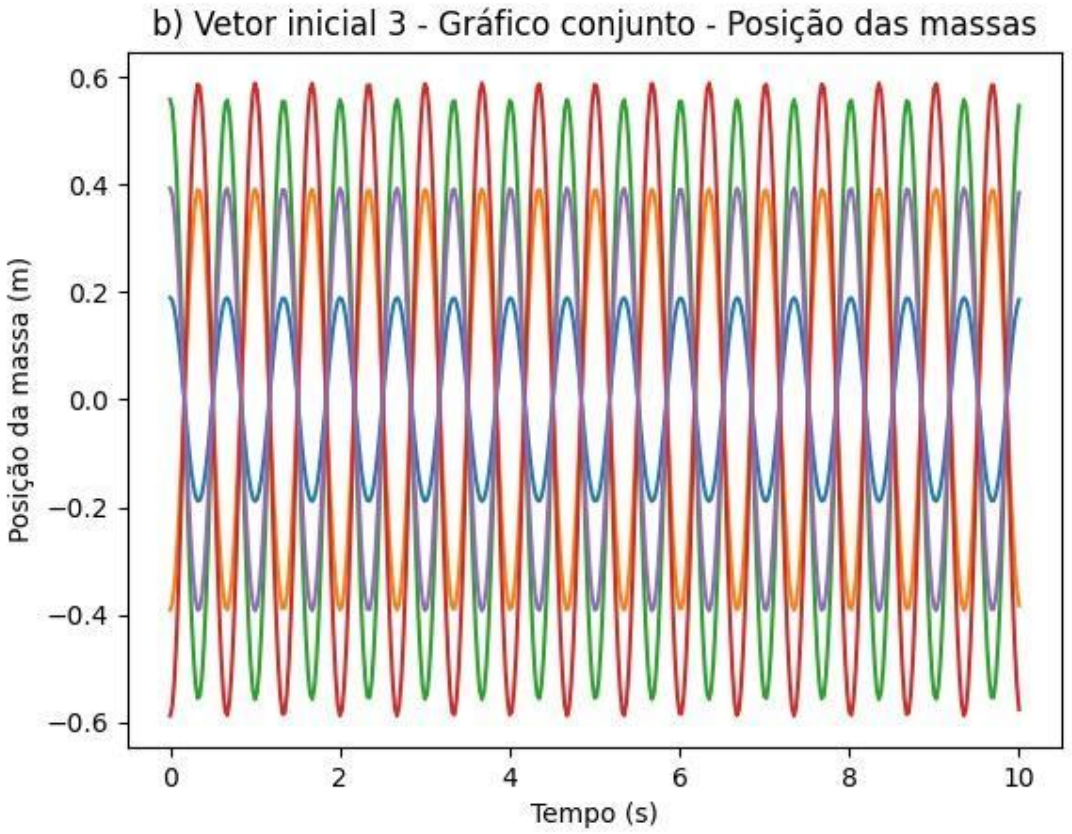

b) Vetor inicial 2 - Posição da massa número: 4
Posição da massa (m)
Tempo (s)


b) Vetor inicial 2 - Posição da massa número: 5
Posição da massa (m)
Tempo (s)

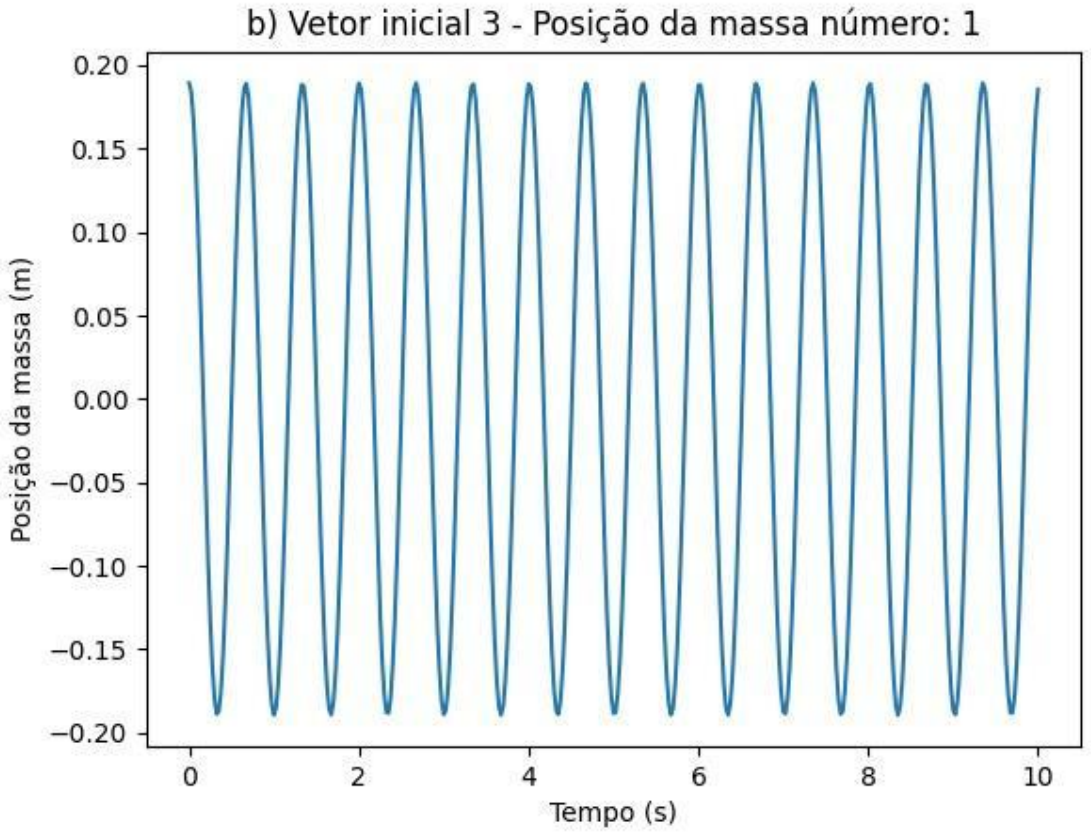
Novamente, observa-se a boa resolução do eixo x dos gráficos.

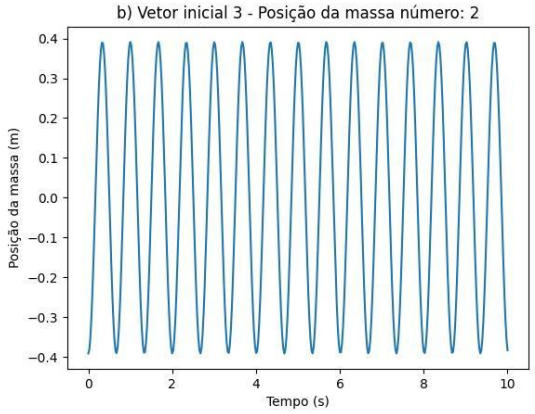
3.3.4. Caso 3 - X(0) do modo de maior frequência

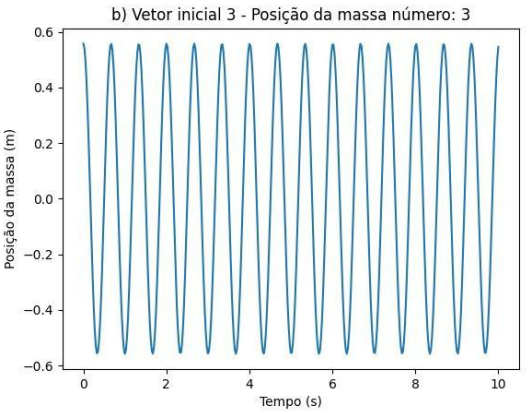
Já na parte 3 do item b) é pedido o X(0) do modo de maior frequência. Esse é dado por algum múltiplo dos autovetores já calculados anteriormente, ou seja, fica fácil implementar simplesmente copiando eles da variável utilizada para guardá-los. Em um sistema do tipo massa-mola é esperado que, em um sistema com as mesmas massas e constantes elásticas, quando a frequência for mais alta, a amplitude venha a diminuir, o que é descrito pela física. Segue o gráfico do caso 3.

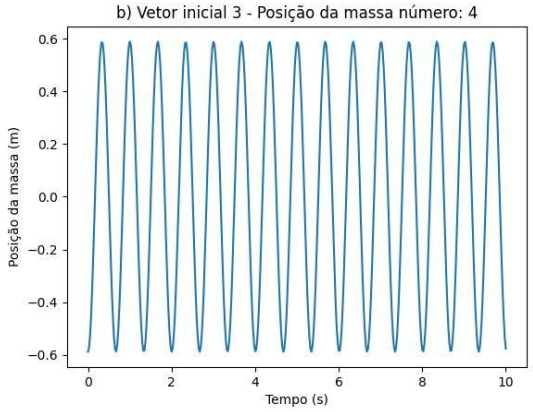


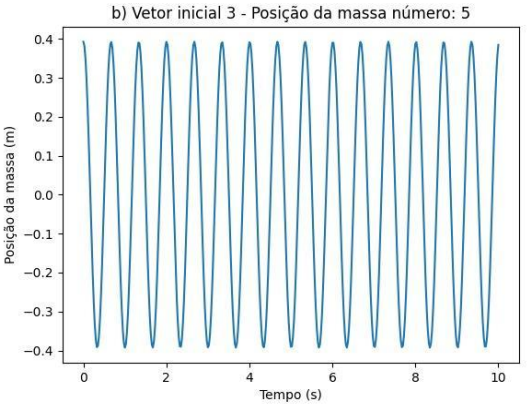
Esse é o gráfico conjunto de todas as molas. Com ele é fácil notar que, enquanto algumas massas vão para um lado, outras massas vão para o outro lado. Na primeira oscilação, as duas massas que começaram no fundo vão para o topo, enquanto as 3 massas que começaram no topo, vão para o fundo, todas sincronizadas. Assim elas seguem trocando de extremo ao longo do tempo da simulação. A seguir vamos ver os gráficos individuais, que confirmam isso.


b) Vetor inicial 3 - Posição da massa número: 1
Posição da massa (m)
Tempo (s)


b) Vetor inicial 3 - Posição da massa número: 2
Posição da massa (m)
Tempo (s)


b) Vetor inicial 3 - Posição da massa número: 3
Posição da massa (m)
Tempo (s)


b) Vetor inicial 3 - Posição da massa número: 4
Posição da massa (m)
Tempo (s)


b) Vetor inicial 3 - Posição da massa número: 5
Posição da massa (m)
Tempo (s)

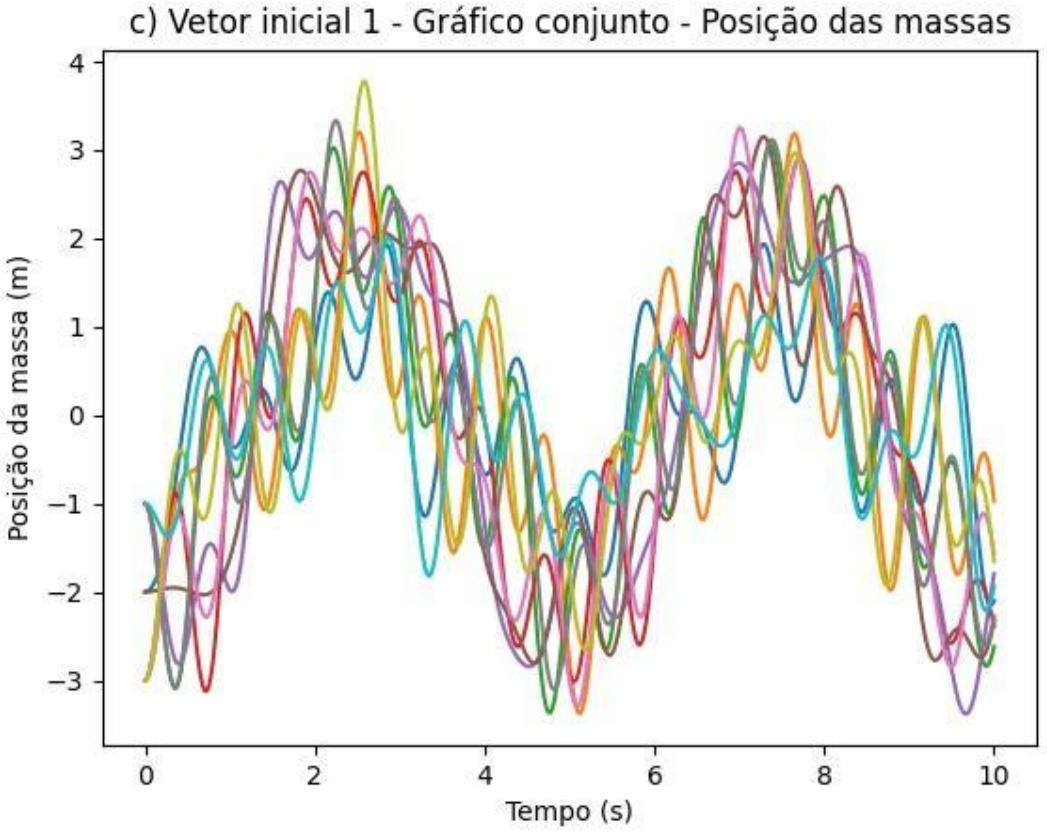
3.4. Tarefa 3 - Item c) do enunciado

3.4.1. Definição do problema

No item c), o enunciado pede praticamente a mesma coisa do item anterior, porém com algumas mudanças chave. A primeira delas é que agora o sistema conta com um total de 10 massas, e portanto, 11 molas. Com essa configuração, as massas continuam tendo 2kg cada, mas as constantes elásticas mudam, passando a ser alternadas entre 38 e 42 N/m. As velocidades iniciais nulas continuam valendo, mas as posições iniciais das 5 últimas massas passam a ser uma cópia das 5 primeiras.

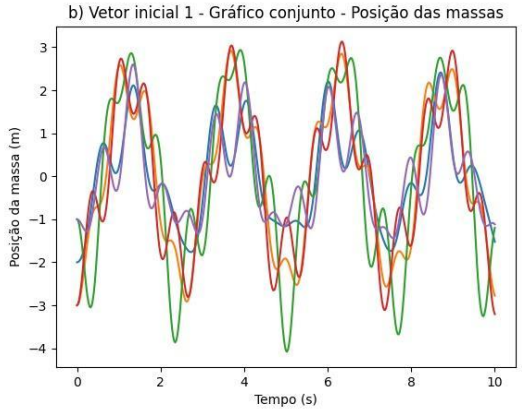
3.4.2. Caso 4 - X(0) = -2, -3, -1, -3, -1, -2, -3, -1, -3, -1

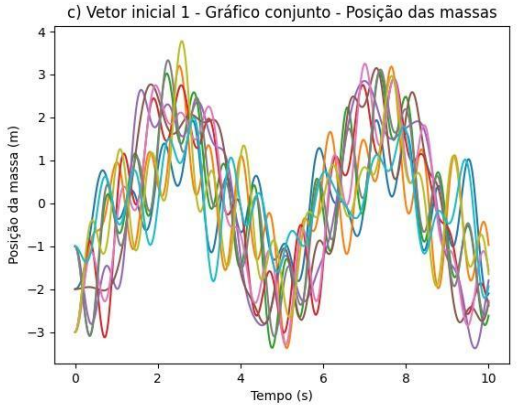
Depois de implementar o "setup" do item c), são repetidos os testes e *outputs* do item b). Vamos checar como ficou o caso do vetor de posições iniciais cujas posições ficam todas mais próximas entre si e do mesmo lado da origem. Abaixo está o gráfico conjunto.



Nele, podemos observar dois padrões: o primeiro é que ele está bastante parecido com o do item b), e o segundo é que ele também apresenta ciclicidade de baixa frequência. Enquanto as massas se movimentam em alta frequência individualmente, em grupo elas

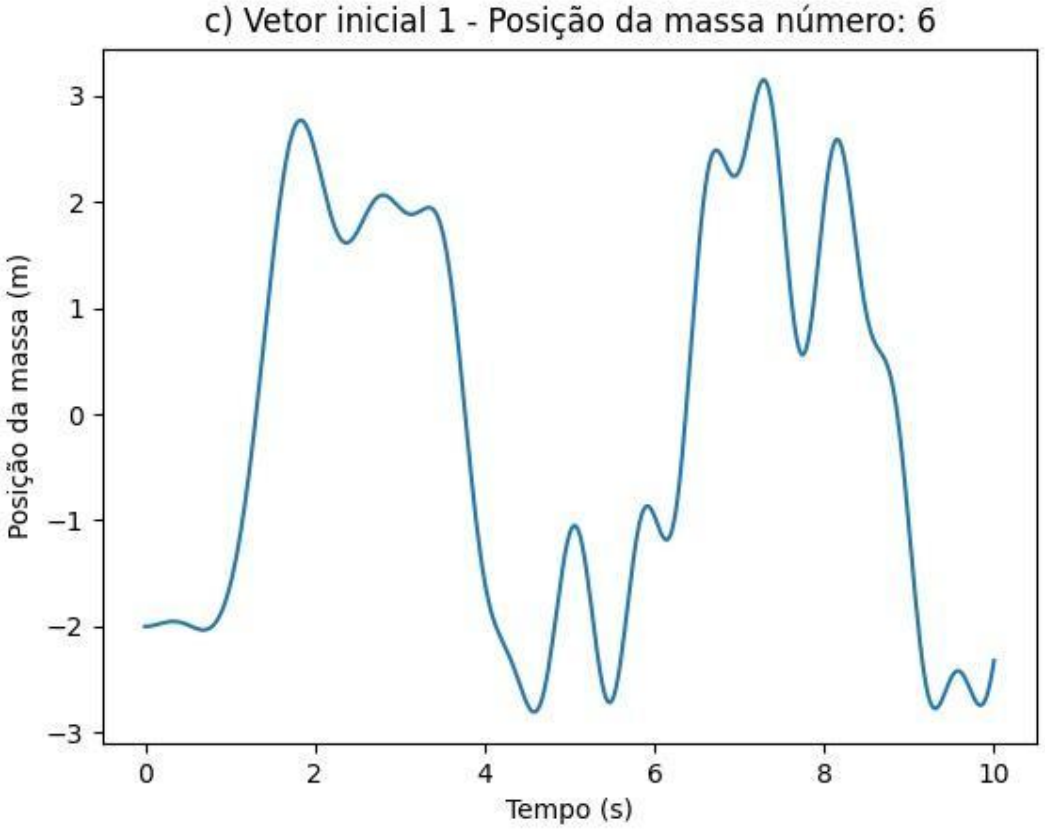
seguem um movimento mais amplo. Uma peculiaridade observada é que com o dobro de massas no sistema, o comportamento cíclico ocorre na metade da frequência que ocorreu quando tinha apenas 5 massas (item b). Isso pode ser observado na comparação abaixo, onde o item b) está à esquerda e o item c) à direita.

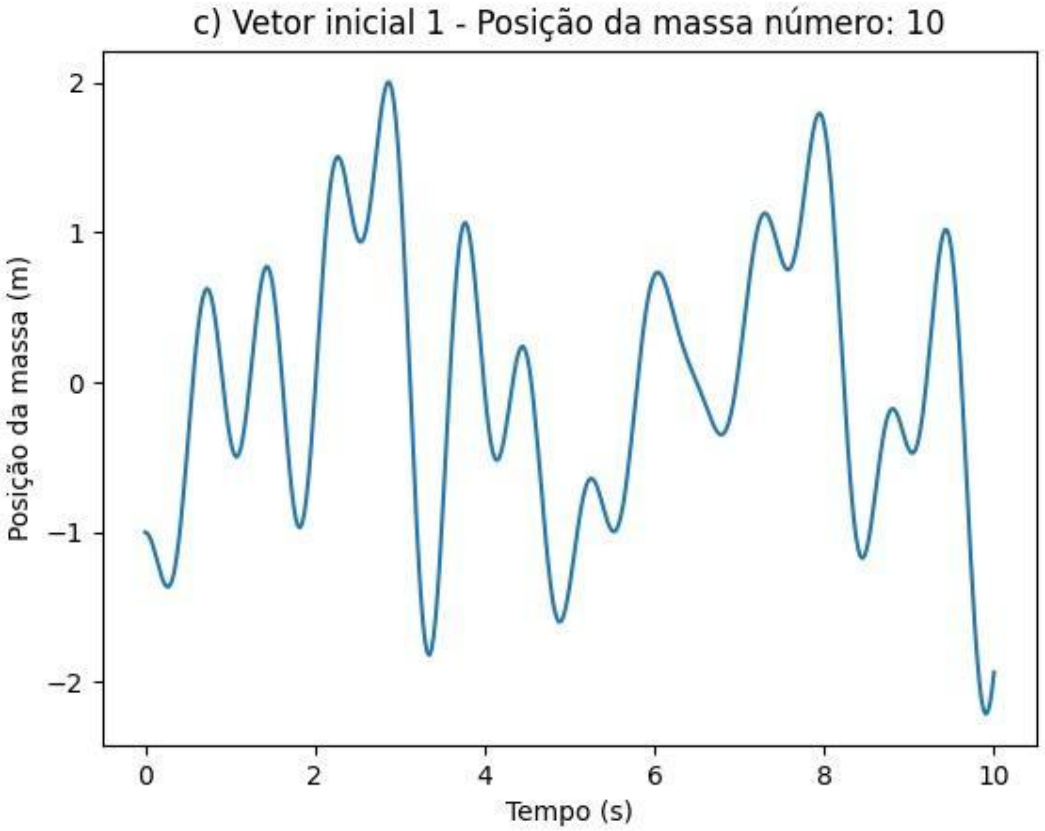


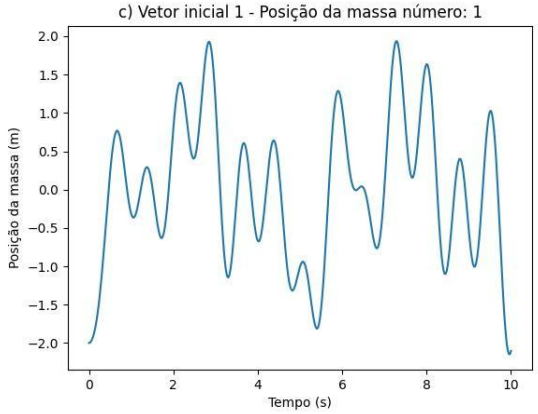


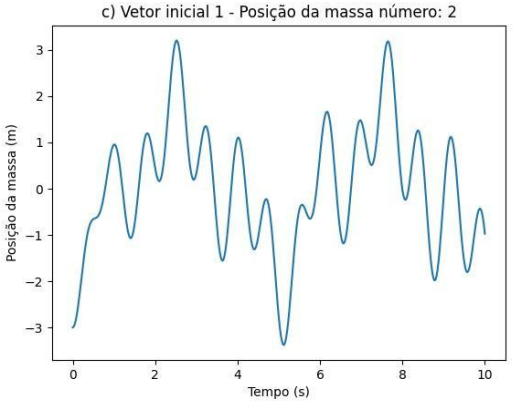
Acima, observa-se claramente que com menos massas, ocorrem 4 ondas conjuntas, enquanto que apenas ocorrem duas no caso de 10 massas. Além disso, os gráficos individuais das massas demonstram ambos os comportamentos oscilatórios, de alta e baixa frequência, com suas particularidades vindas dos posicionamentos iniciais das massas e das constantes elásticas das molas. Abaixo, apresentam-se.

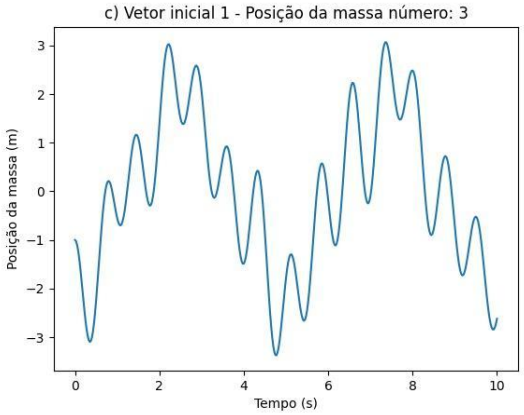
Nós gostaríamos de chamar a atenção para as massas de número 6 e 10. Enquanto algumas massas mostram um comportamento bastante imprevisível, quase com aparência caótica (que é o caso da massa 10), outras massas demonstram comportamentos mais parecido com a simples sobreposição de duas ondas: uma de alta frequência e uma de baixa, e ainda outras são um híbrido dessas duas situações (caso da massa 6), que aparenta conter uma onda grande, mas na primeira quase perde a amplitude de oscilações de mais alta frequência e depois volta a demonstrá-las.

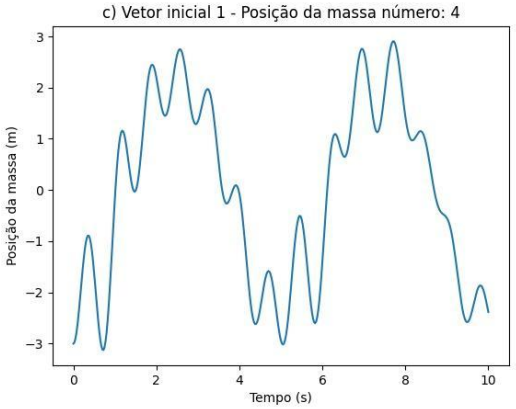

c) Vetor inicial 1 - Posição da massa número: 6
Posição da massa (m)
Tempo (s)
3
2
1
0
−1
−2
−3
0
2
4
6
8
10

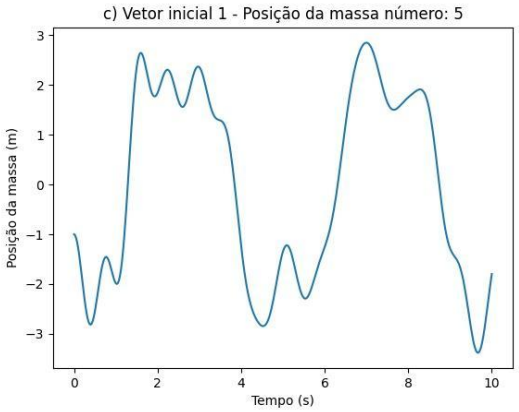

c) Vetor inicial 1 - Posição da massa número: 10
Posição da massa (m)
Tempo (s)
2
1
0
−1
−2
0
2
4
6
8
10

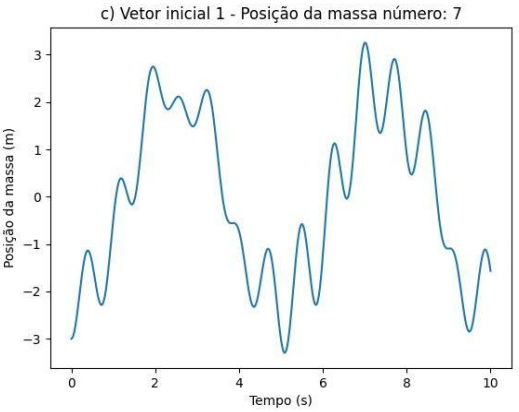

c) Vetor inicial 1 - Posição da massa número: 1
Posição da massa (m)
Tempo (s)

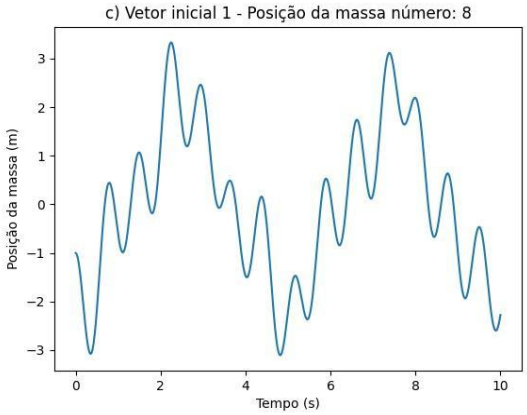

c) Vetor inicial 1 - Posição da massa número: 2
Posição da massa (m)
Tempo (s)

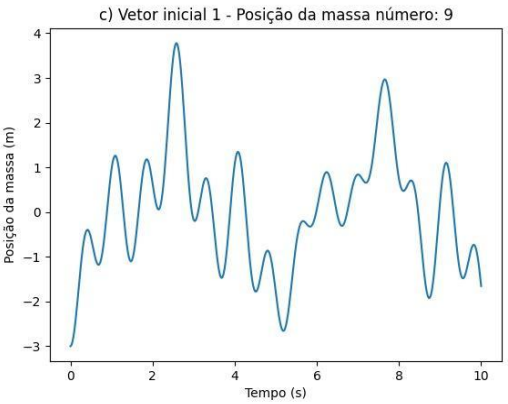

c) Vetor inicial 1 - Posição da massa número: 3
Posição da massa (m)
Tempo (s)


c) Vetor inicial 1 - Posição da massa número: 4
Posição da massa (m)
Tempo (s)


c) Vetor inicial 1 - Posição da massa número: 5
Posição da massa (m)
Tempo (s)

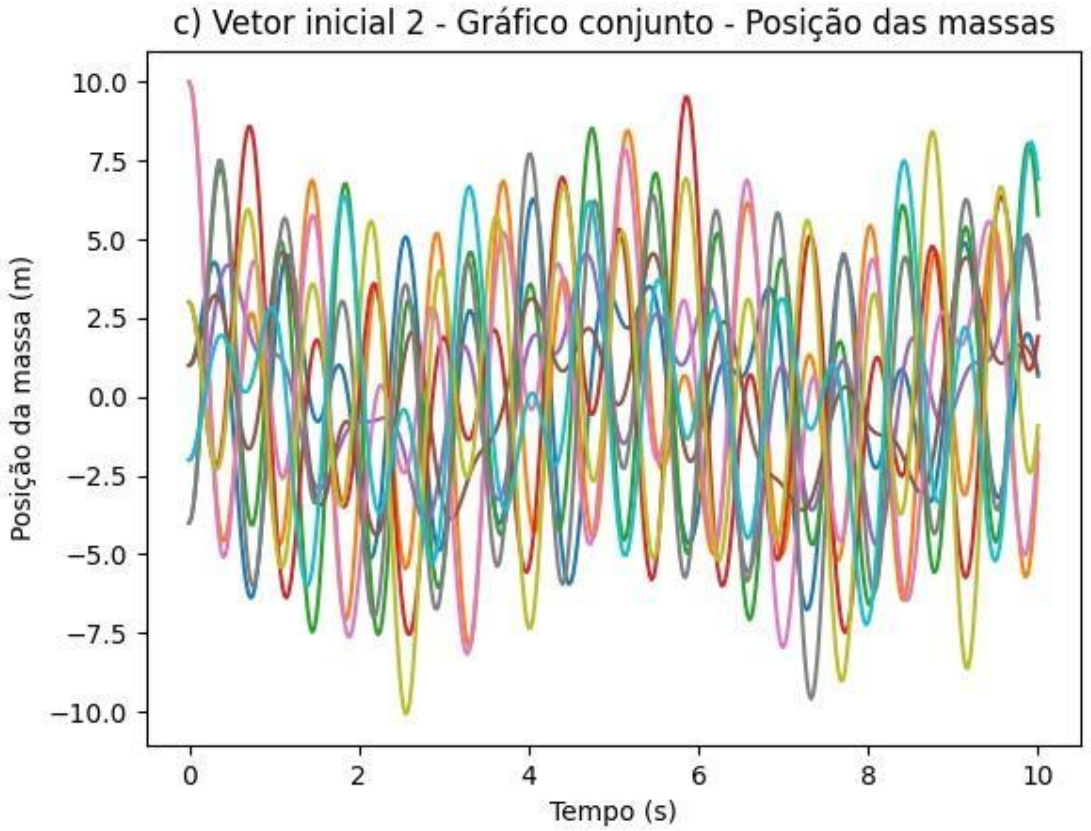

c) Vetor inicial 1 - Posição da massa número: 7
Posição da massa (m)
Tempo (s)


c) Vetor inicial 1 - Posição da massa número: 8
Posição da massa (m)
Tempo (s)


c) Vetor inicial 1 - Posição da massa número: 9
Posição da massa (m)
Tempo (s)

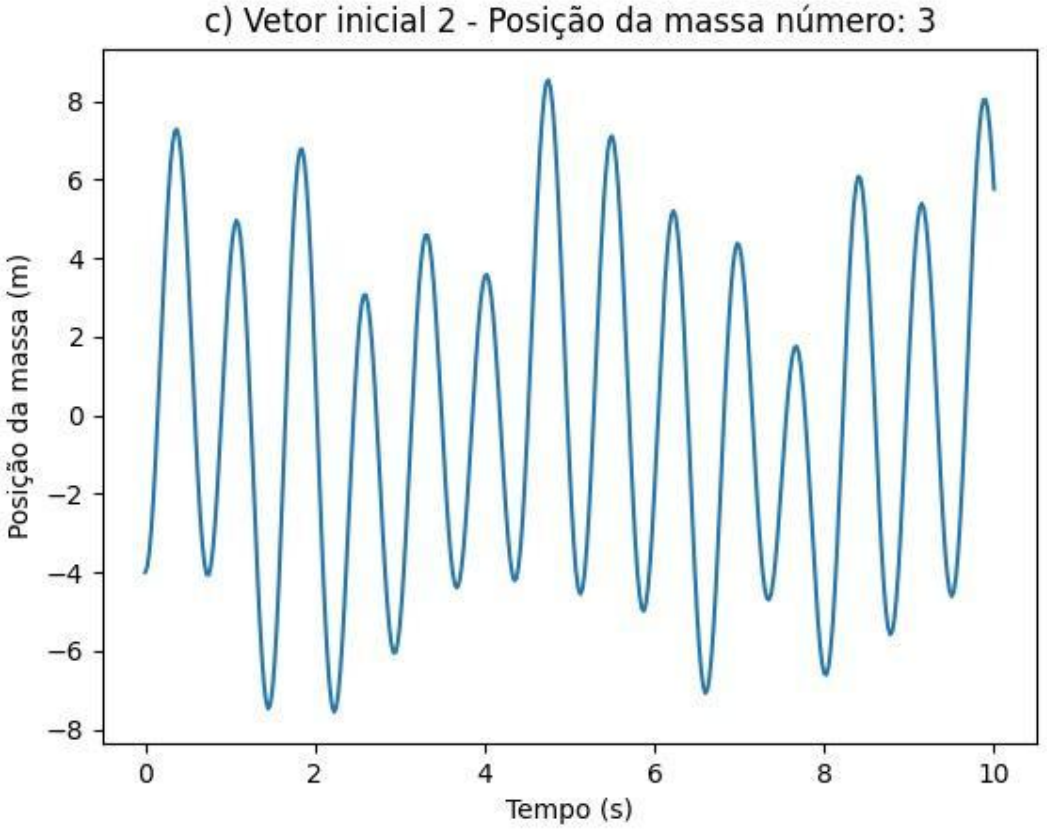
3.4.3. Caso 5 - X(0) = 1, 10, −4, 3, −2, 1, 10, −4, 3, −2

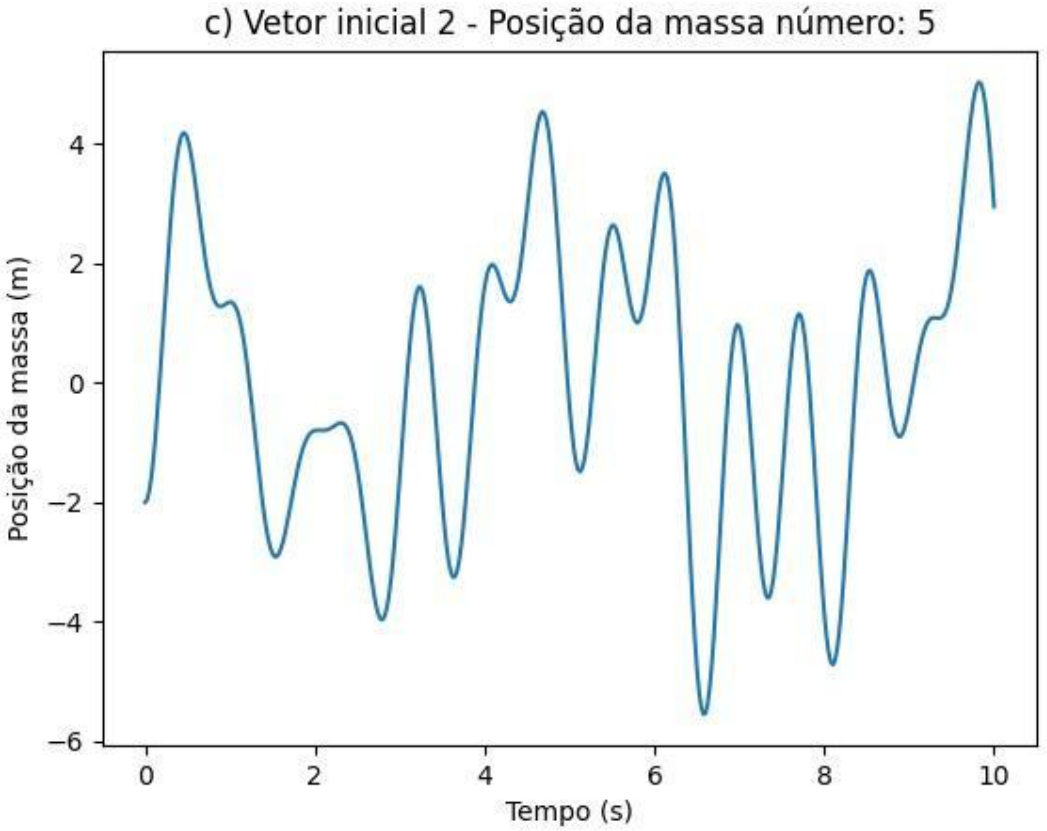
Como no item b), este caso é uma variação do anterior, mas com condições de posição inicial diferentes. Essas posições são, agora, parecidas com as do caso 2, contendo massas em posições mais espalhadas, dos dois lados da origem. Já no primeiro gráfico, as posições de todas as massas mostradas em conjunto ao longo do tempo, é possível notar dois padrões interessantes. Vamos comentar depois da imagem.

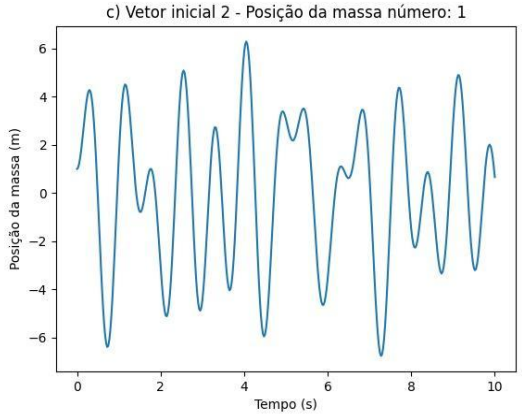


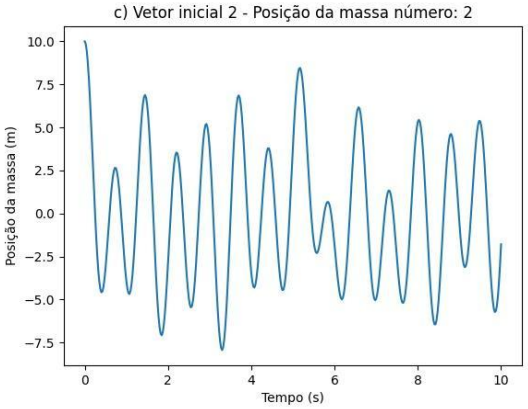
Nela, vemos um comportamento quase caótico das massas, bem como no caso anterior. Porém, agora, diferentemente do caso 2, é possível identificar mais claramente um padrão de onda longa, que todas as massas parecem seguir, por volta de dois comprimentos de onda durante os 10 segundos da simulação.

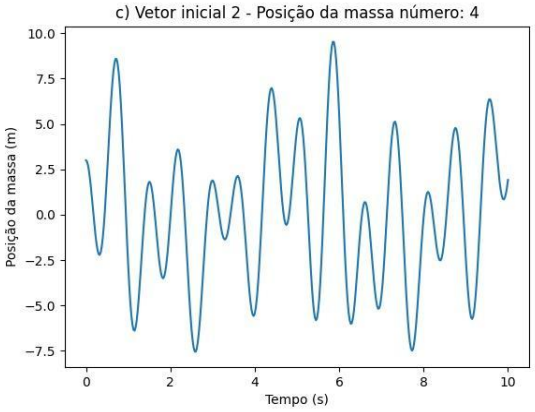
Quanto aos casos individualmente analisados das massas, podemos ver, novamente, algumas sendo mais estáveis em seu comportamento ondulatório e outras apresentando uma composição de curso aparentemente mais complexo, mas que talvez não seja mais complexo que as aparentemente simples, que podem ter suas ondas secundárias mais fracas. Vejamos abaixo.

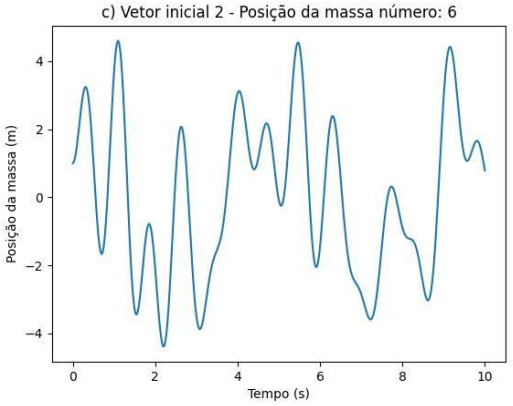

c) Vetor inicial 2 - Posição da massa número: 3
Posição da massa (m)
Tempo (s)
8
6
4
2
0
−2
−4
−6
−8
0
2
4
6
8
10

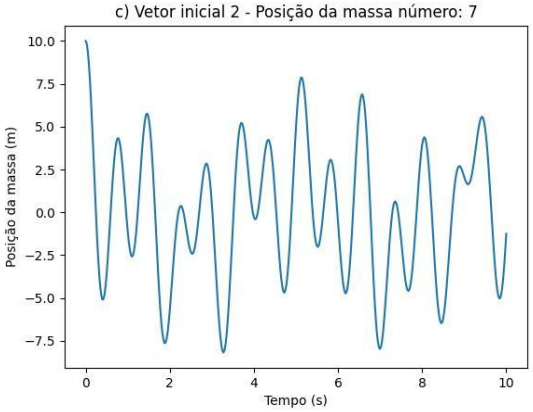

c) Vetor inicial 2 - Posição da massa número: 5
Posição da massa (m)
Tempo (s)
4
2
0
−2
−4
−6
0
2
4
6
8
10

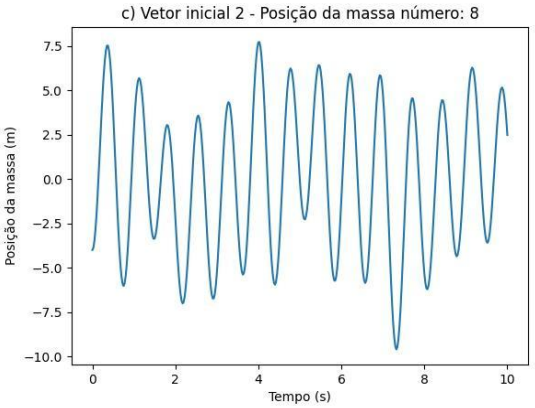

c) Vetor inicial 2 - Posição da massa número: 1
Posição da massa (m)
Tempo (s)

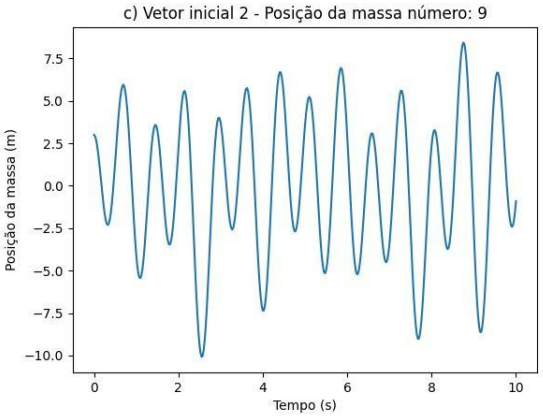

c) Vetor inicial 2 - Posição da massa número: 2
Posição da massa (m)
Tempo (s)

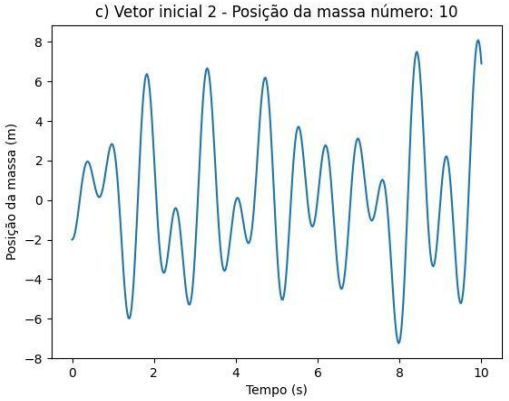

c) Vetor inicial 2 - Posição da massa número: 4
Posição da massa (m)
Tempo (s)


c) Vetor inicial 2 - Posição da massa número: 6
Posição da massa (m)
Tempo (s)


c) Vetor inicial 2 - Posição da massa número: 7
Posição da massa (m)
Tempo (s)

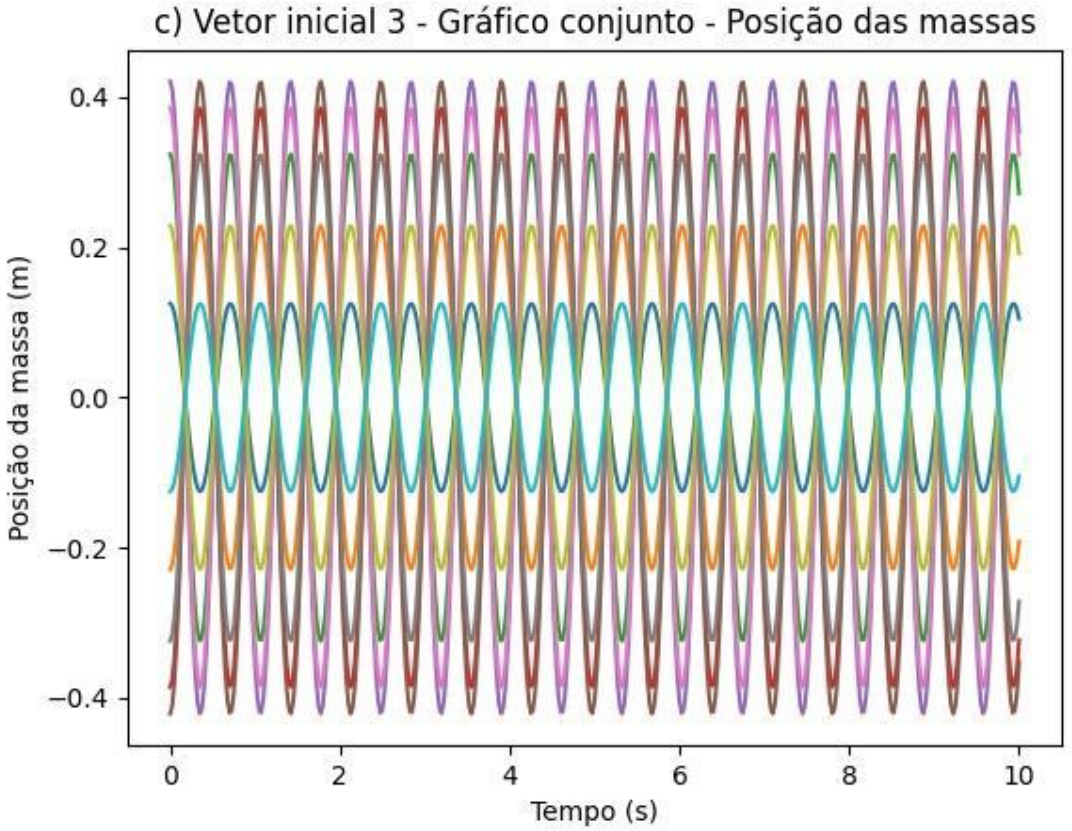

c) Vetor inicial 2 - Posição da massa número: 8
Posição da massa (m)
Tempo (s)


c) Vetor inicial 2 - Posição da massa número: 9
Posição da massa (m)
Tempo (s)

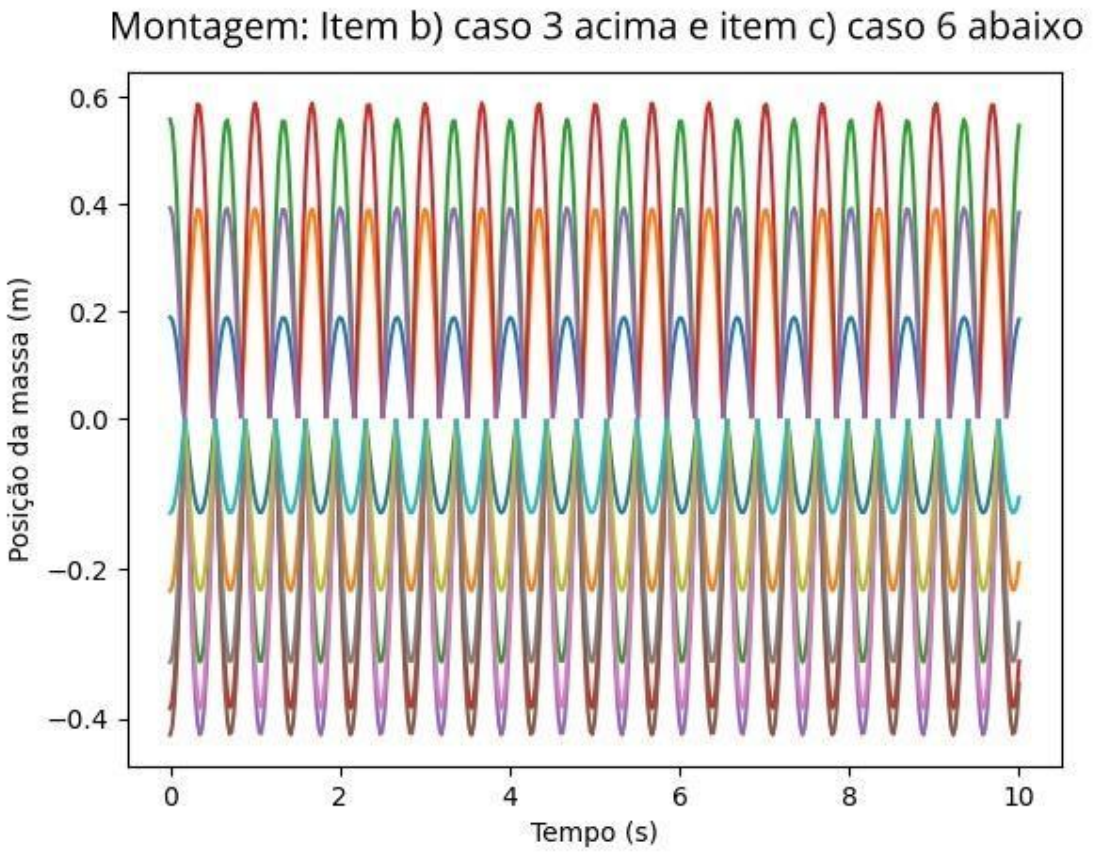

c) Vetor inicial 2 - Posição da massa número: 10
Posição da massa (m)
Tempo (s)

3.4.4. Caso 6 - X(0) do modo de maior frequência

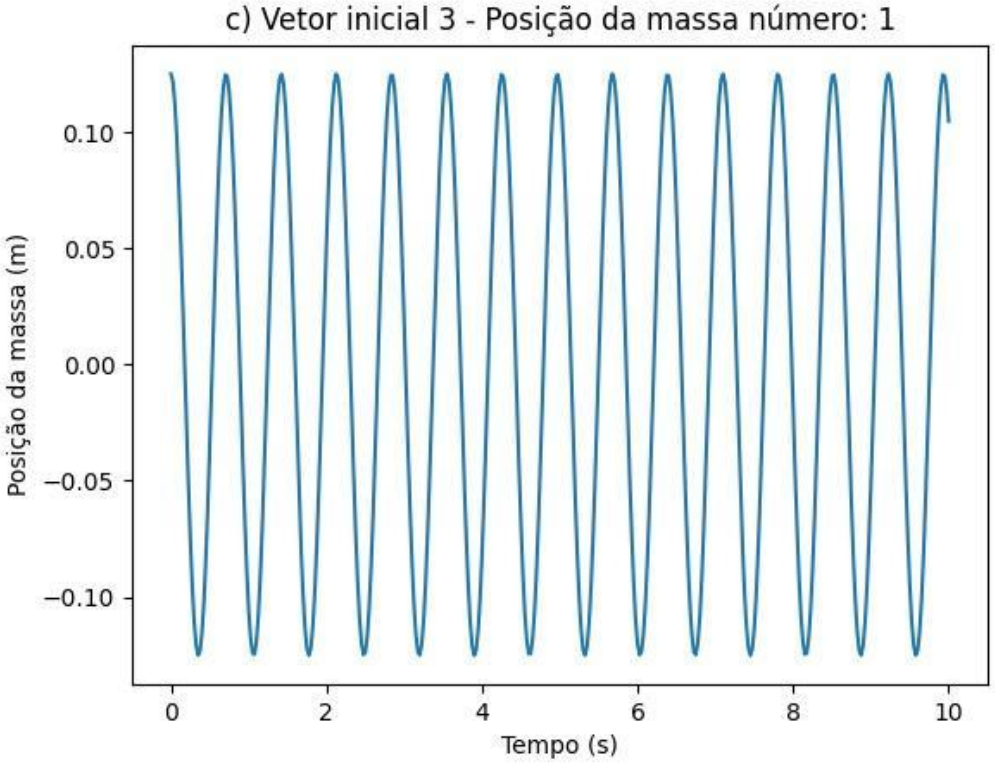
Por fim, vamos analisar o caso do modo de maior frequência. Aqui, foi considerado o caso em que as dez massas apresentam maior frequência de vibração dentro do sistema massa-mola. Esse caso é um tanto parecido com o caso 3 (do item b), em que as massas parecem se mover em conjunto para frente e para trás, alternando em grupo entre aquelas que estavam lá vindo e as que estavam cá indo. Vejamos no gráfico abaixo.

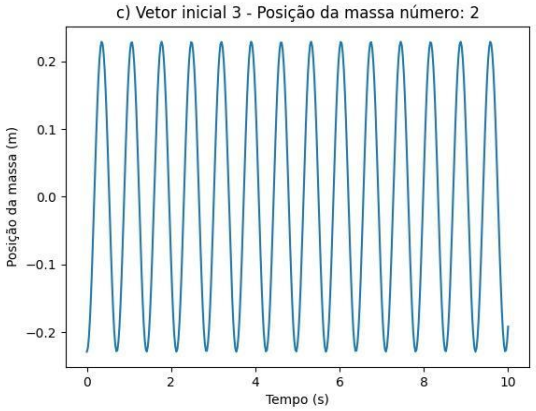


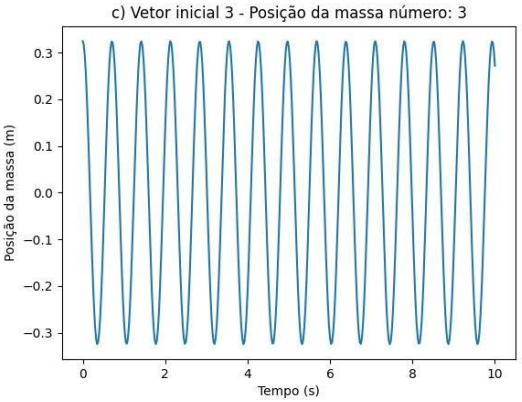
Uma curiosidade é que as frequências, nesse caso, não representam a metade da frequência do mesmo caso do item b), e mesmo parecendo serem as mesmas, a montagem abaixo mostra que não são. A frequência do caso 6 é levemente menor que a do caso 3. Isso já era esperado, dado às diferenças tanto nas condições iniciais, quanto nas constantes de elasticidade das molas. Esses dois valores geram matrizes diferentes e delas são tirados autovalores e autovetores também diferentes, ainda que um tanto parecidos. Daí essa diferença observada abaixo. A montagem conta com o caso 3 (do item b) acima da origem e o caso 6 (do item c) abaixo da linha do zero.

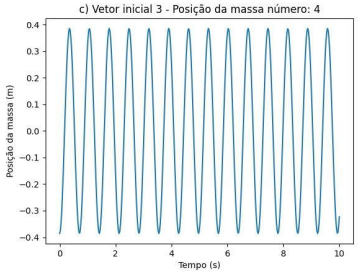


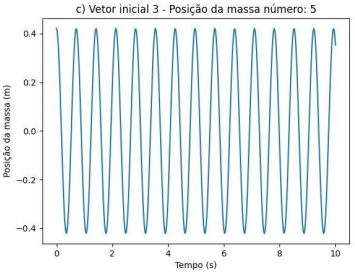
Finalmente, os gráficos de comportamento individual de cada massa demonstram um movimento ondulatório bastante comportado para todas as massas. Podem ser observados abaixo.

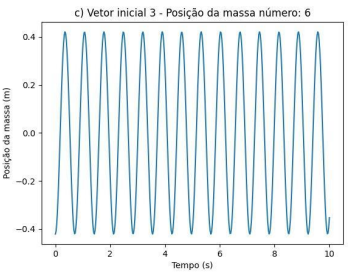

c) Vetor inicial 3 - Posição da massa número: 1
Posição da massa (m)
Tempo (s)

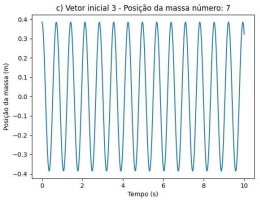

c) Vetor inicial 3 - Posição da massa número: 2
Posição da massa (m)
Tempo (s)

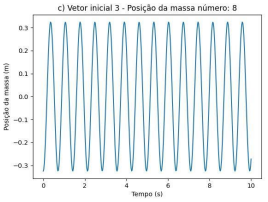

c) Vetor inicial 3 - Posição da massa número: 3
Posição da massa (m)
Tempo (s)

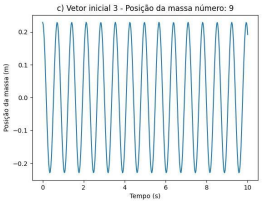

c) Vetor inicial 3 - Posição da massa número: 4
Posição da massa (m)
Tempo (s)

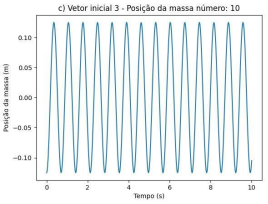

c) Vetor inicial 3 - Posição da massa número: 5
Posição da massa (m)
Tempo (s)


c) Vetor inicial 3 - Posição da massa número: 6
Posição da massa (m)
Tempo (s)


c) Vetor inicial 3 - Posição da massa número: 7
Posição da massa (m)
Tempo (s)


c) Vetor inicial 3 - Posição da massa número: 8
Posição da massa (m)
Tempo (s)


c) Vetor inicial 3 - Posição da massa número: 9
Posição da massa (m)
Tempo (s)


c) Vetor inicial 3 - Posição da massa número: 10
Posição da massa (m)
Tempo (s)