Trabalho 1:

Leonardo Maximo Silva, 200022172

Sumário

1 Introdução

O trabalho foi composto por uma Lista de 14 exercícios subdivididos em duas partes. Na primeira, foram apresentados problemas de programação para a familiarização com a linguagem de programação Python. Por outro lado, na segunda parte, foram apresentados problemas numéricos a serem resolvidos por meio de códigos, de modo a aliar métodos numéricos com programação. A seguir, é descrita a solução para cada exercício proposto.

2 Exercício 1

Esse exercício constituiu da elaboração de uma função em Python que multiplique duas matrizes (A·B) construídas a partir dos arrays do numpy. Para construí-la, inicialmente, verificou-se se a matriz é multiplicável ou não, ou seja, se o número de colunas da matriz A era igual ao número de linhas da matriz B e, caso não fossem, uma mensagem de erro era retornada. Aplicou-se, então, um algoritmo de multiplicação de linha da matriz A por coluna da matriz B para se obter o resultado de sua multiplicação (matriz C). Um termo da matriz resultante era dado pela soma dos termos resultantes da multiplicação de uma determinada linha de A (índice fixo),cujo valor da coluna (índice móvel) se alterava conforme os termos resultantes iam sendo somados, com determinada coluna de B (índice fixo), cujo valor da linha (índice móvel) variava conforme se somavam os termos. Assim, realizava-se, primeiro, todos os termos correspondentes a uma linha da matriz resultante antes de seguir para a próxima linha e, iterativamente, obter as linhas de C até se terminar a multiplicação. A seguir, são apresentados alguns resultados obtidos com esse algoritmo.

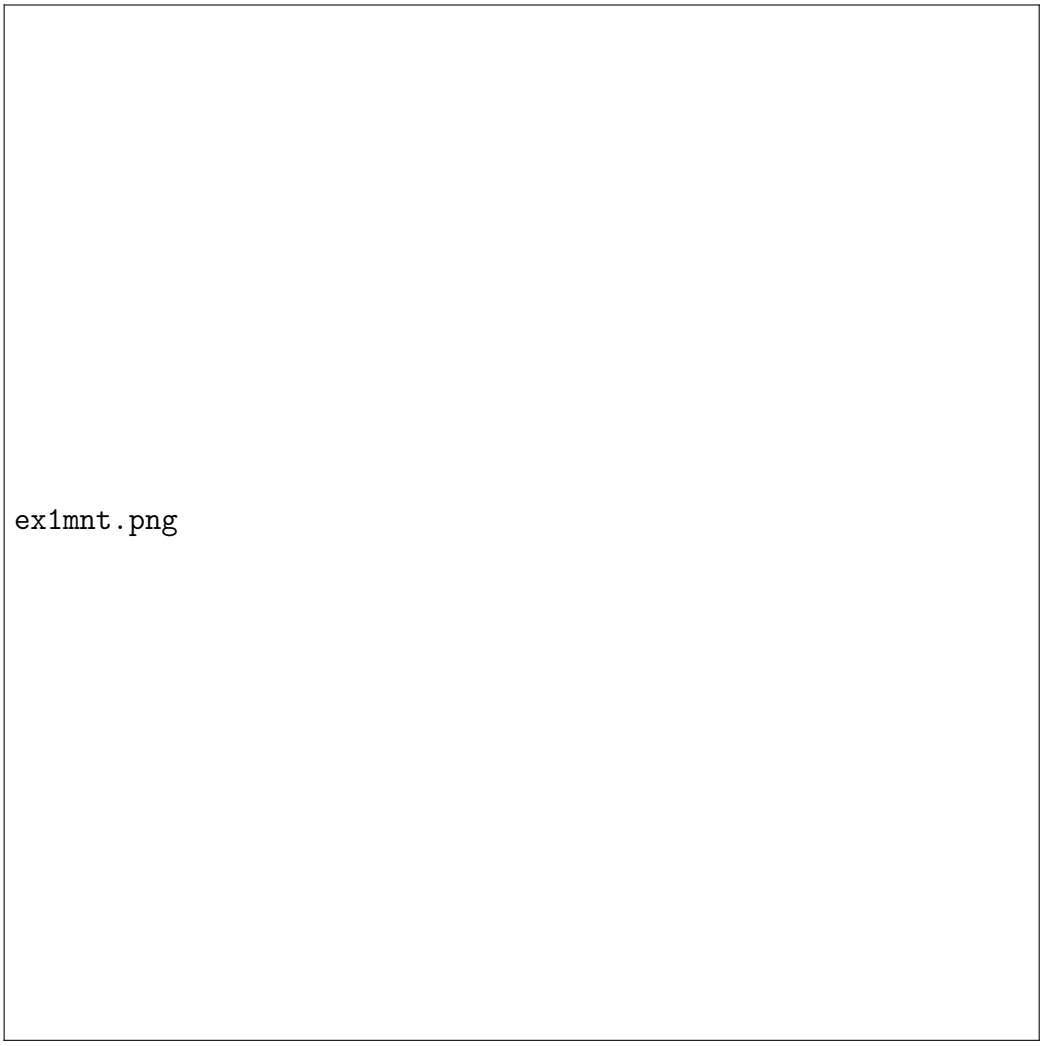

Figura 1: Primeiro Teste de Multiplicação de Matrizes

Figura 2: Segundo Teste de Multiplicação de Matrizes

3 Exercício 2

Para o exercício 2, programou-se um algoritmo que realizasse a média e o desvio padrão para uma matriz fornecida. Inicialmente, verificava-se se a matriz era válida, ou seja, se possuía comprimento maior que 0. Caso não cumprisse esse requisito, uma mensagem de erro era exibida. Caso o cumprisse, sua média era calculada a partir da acumulação em uma variável da

soma de todos os termos da Matriz, os quais eram, posteriormente, divididos pelo número de termos da matriz, dado pela multiplicação do número de suas linhas com o número de suas colunas. Para o cálculo do desvio padrão, utilizou-se da média calculada com o objetivo de se tomar, inicialmente, a acumulação da soma do quadrado da diferença de cada termo da matriz em relação à sua média. Após essa operação, o desvio padrão era dada pela raiz quadrada do resultado da divisão entre o termo calculado e o número de termos da matriz. A seguir, são apresentados alguns resultados obtidos com esse algoritmo.

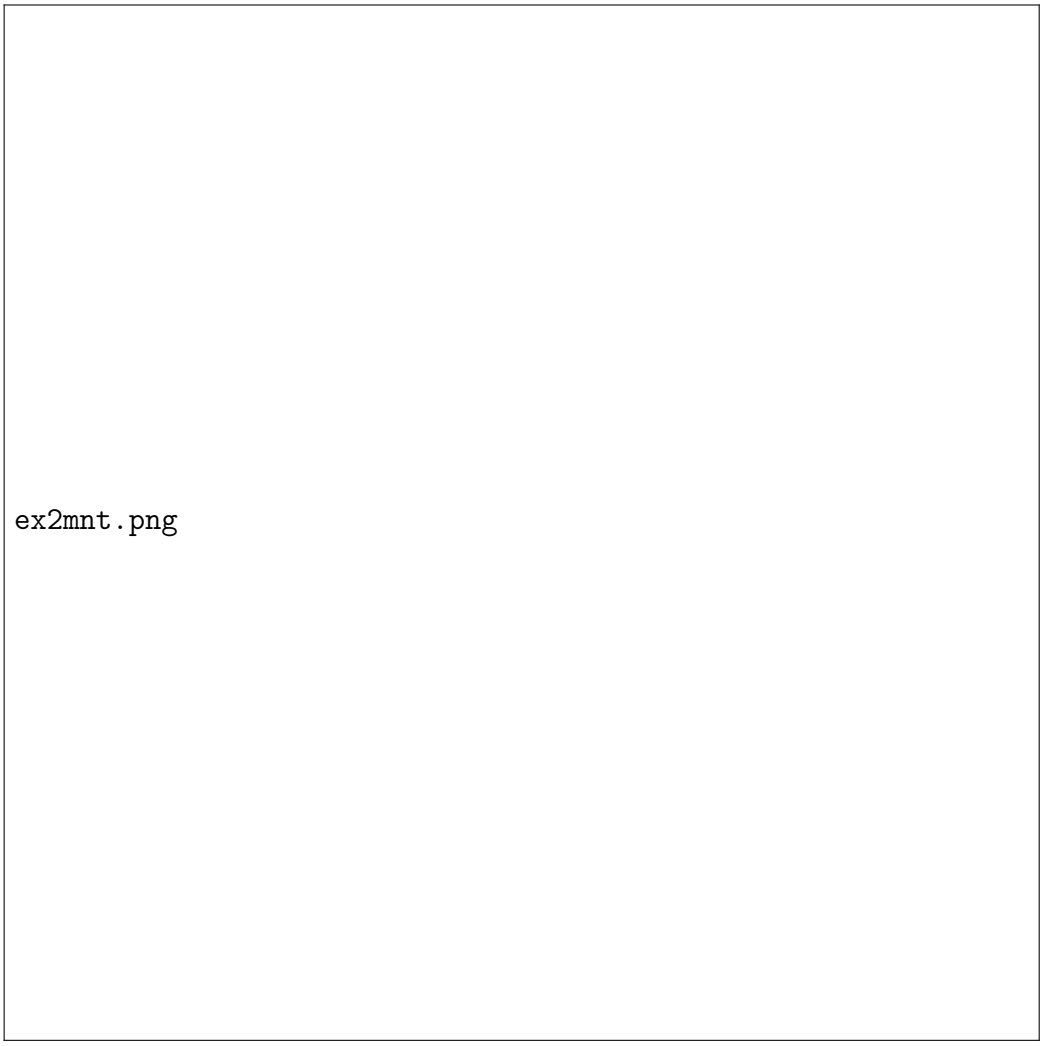

Figura 3: Primeiro Teste de obtenção da Média e Desvio Padrão de uma Matriz

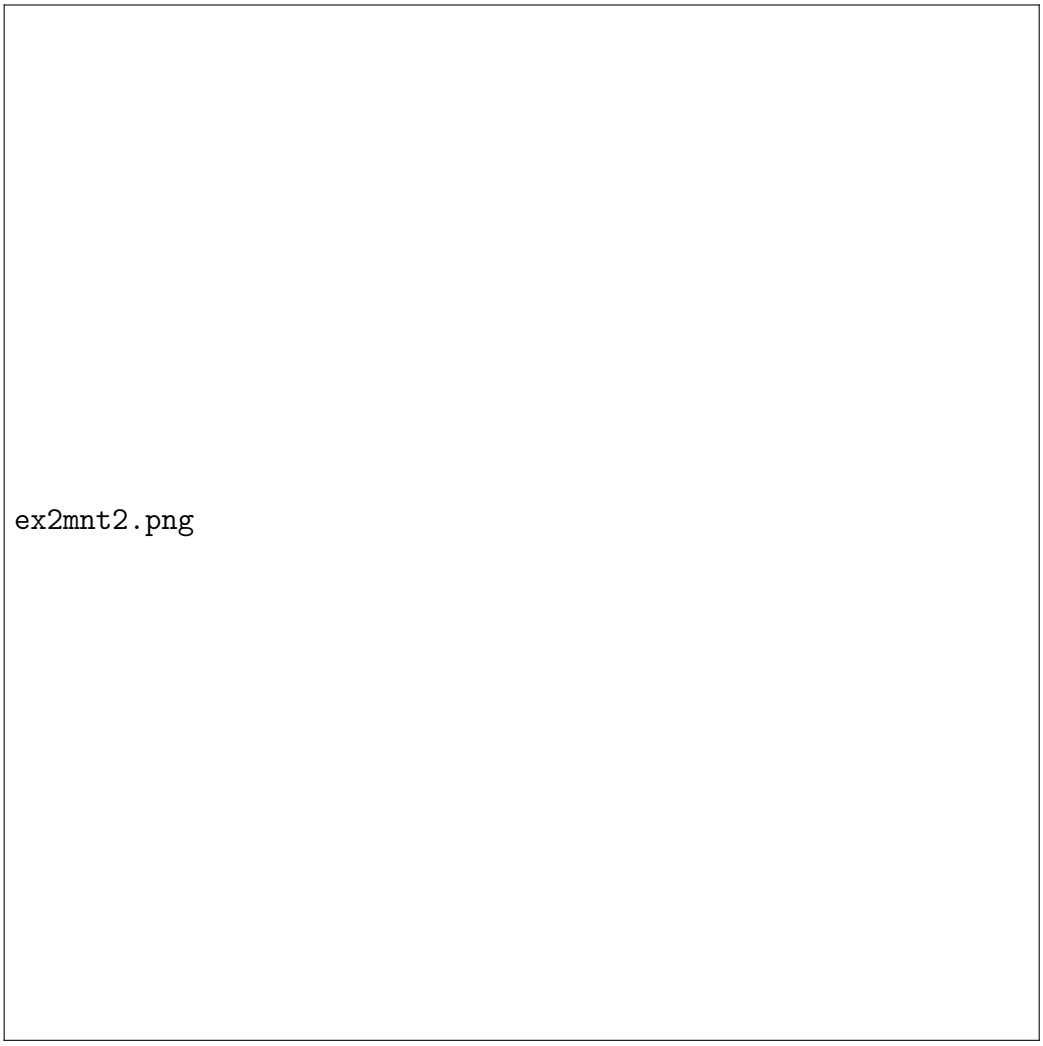

Figura 4: Segundo Teste de obtenção da Média e Desvio Padrão de uma Matriz

4 Exercício 3

Para o exercício 3, utilizou-se o mesmo código utilizado em 3 junto da função "np.random.uniform()"pertencente à biblioteca numpy para se gerar matrizes quadradas NXN que possuem valores aleatórios entre -2 e +2. Realizaram-se, então, testes com essas matrizes com $N \to \infty$, de modo a se

perceber que sua média se aproximava de zero, tendo em vista que, por estatística, quanto mais amostras aleatórias são tomadas, mais próximo a média do resultado estará do valor intermediário entre eles. Por exemplo, quanto mais uma moeda é jogada, mais a probabilidade de se tirar cara ou coroa aproxima-se de 50%, o que se mantém também no caso estudado. Já o desvio padrão aumentava, muito provavelmente, por se tomarem diversas amostras que, ainda quando tomada sua média se aproximem de zero, possuem uma boa probabilidade de que seus elementos sejam díspares o suficiente do valor de sua média para que seu desvio padrão aumente. A seguir, são apresentados alguns resultados obtidos com esse algoritmo.

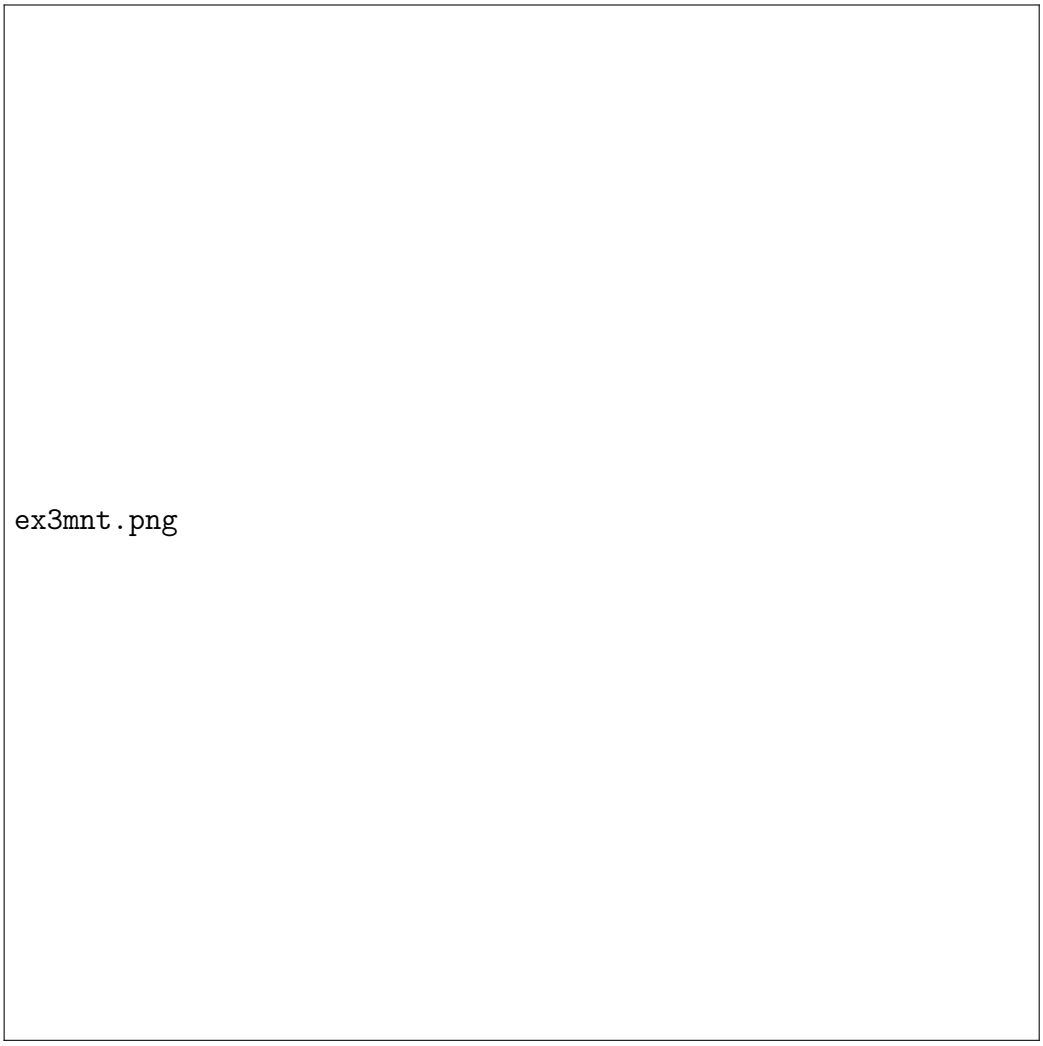

Figura 5: Primeiro Teste de obtenção de Média e Desvio Padrão de uma Matriz Aleatória

Figura 6: Segundo Teste de obtenção de Média e Desvio Padrão de uma Matriz Aleatória

Figura 7: Terceiro Teste de obtenção de Média e Desvio Padrão de uma Matriz Aleatória

5 Exercício 4

O Exercício 4 requisitou a confecção de um código que, a partir da função "input()"do Python, receba e exiba o nome e idade de uma pessoa. No código implementado, esses parâmetros eram requisitados como separados por espaço () e armazenados em um arquivo .txt nomeado de "amigos.txt".

Implementou-se, então, um algoritmo que, a partir do parâmetro .readlines() de arquivos, ler e, iterativamente, mostrar todos os nomes e idades das pessoas presentes no arquivo "amigos.txt". A seguir, são apresentados alguns resultados obtidos com esse algoritmo.

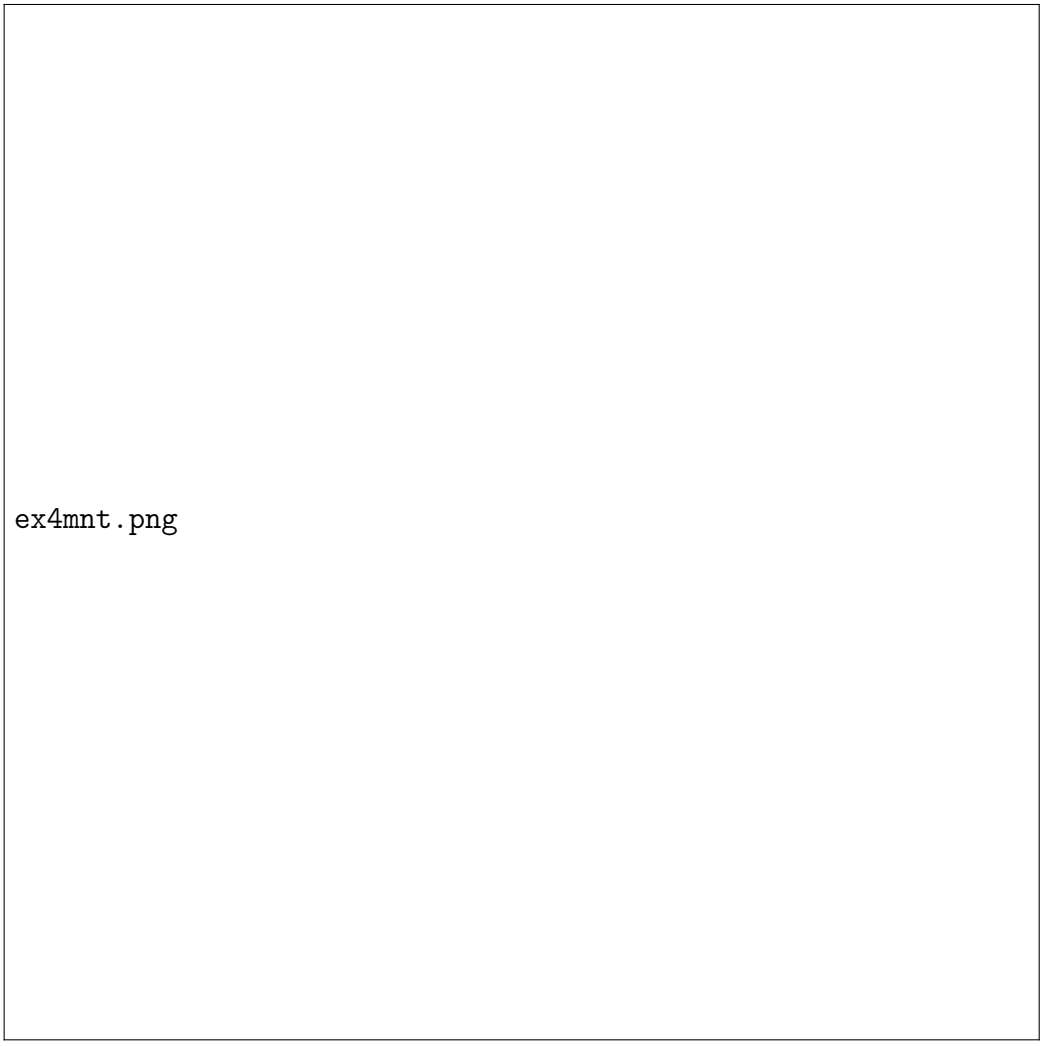

Figura 8: Primeiro Nome adicionado ao Arquivo

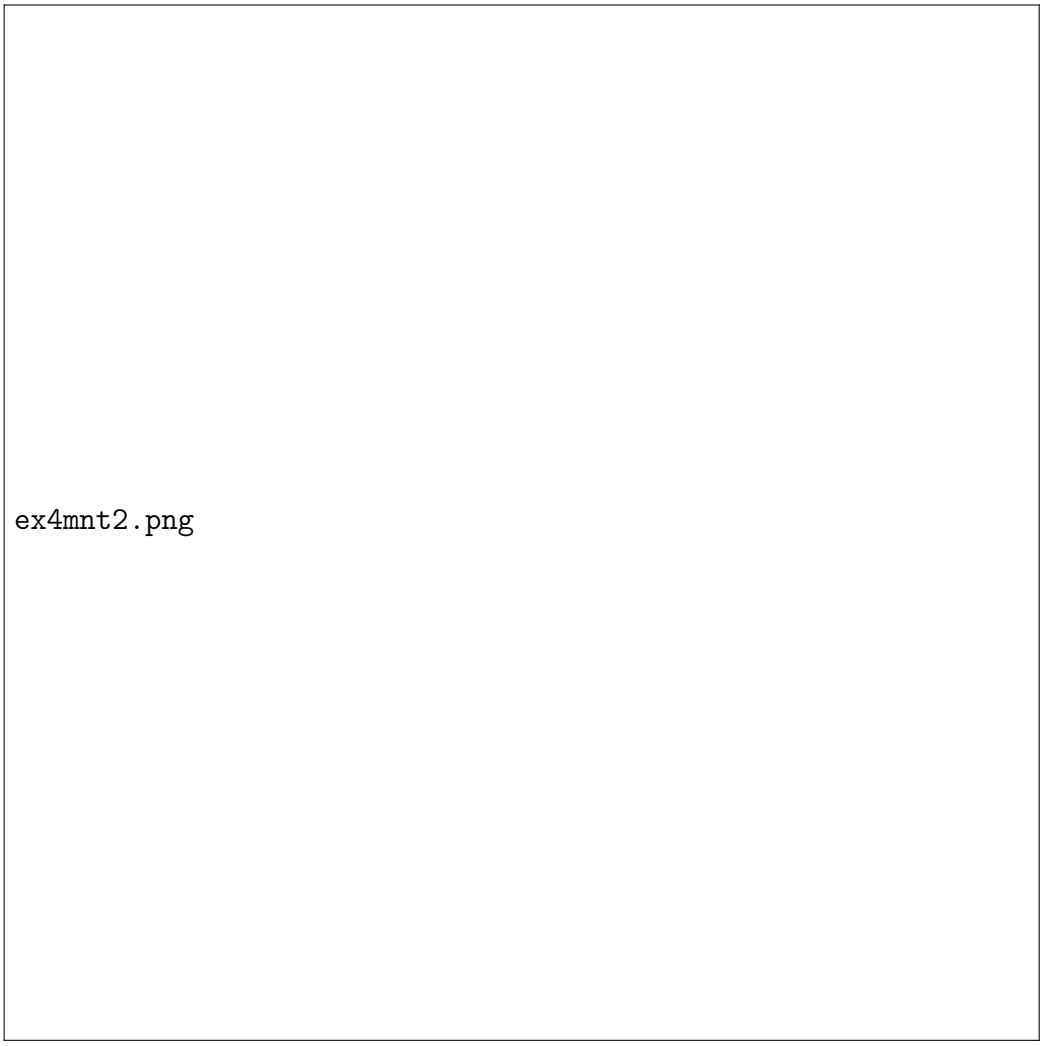

Figura 9: Segundo Nome adicionado ao Arquivo

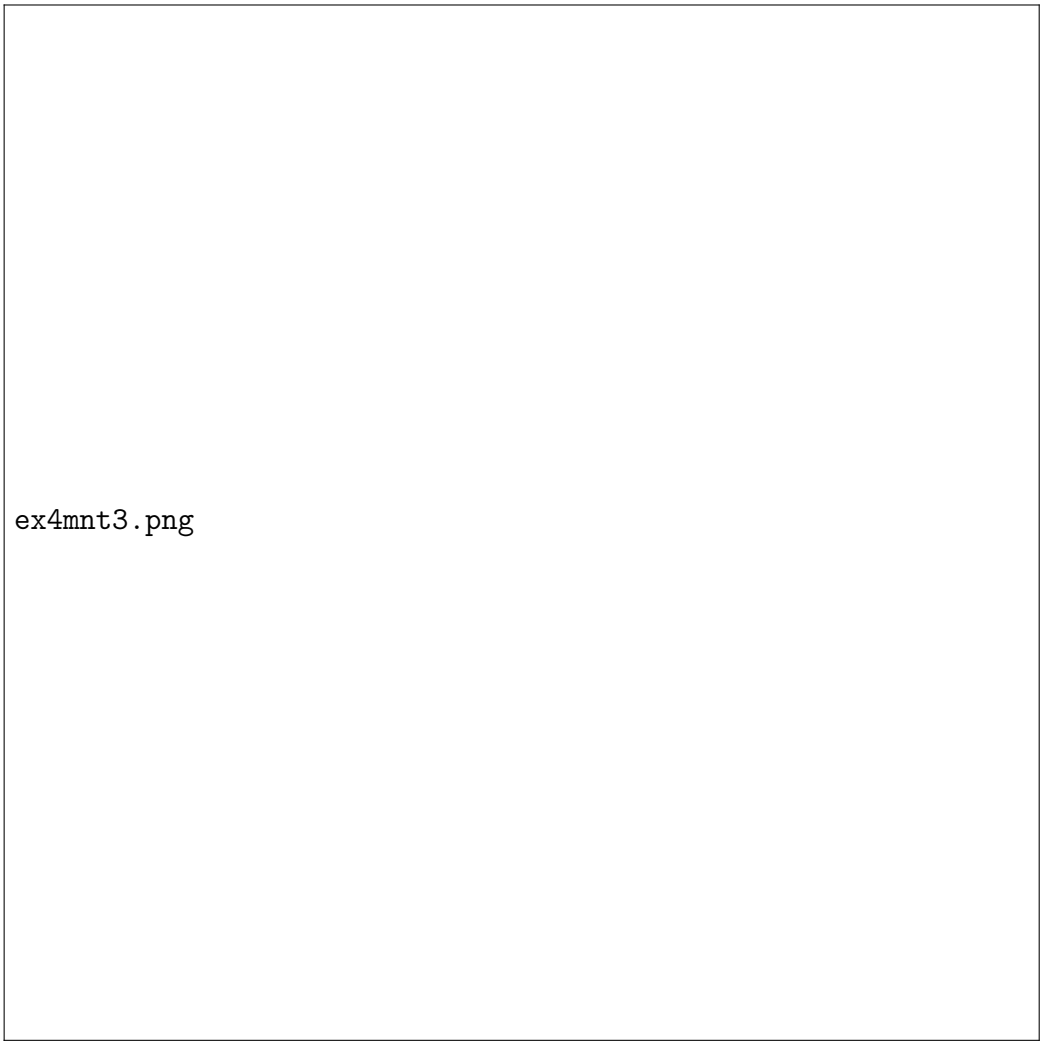

Figura 10: Terceiro Nome adicionado ao Arquivo

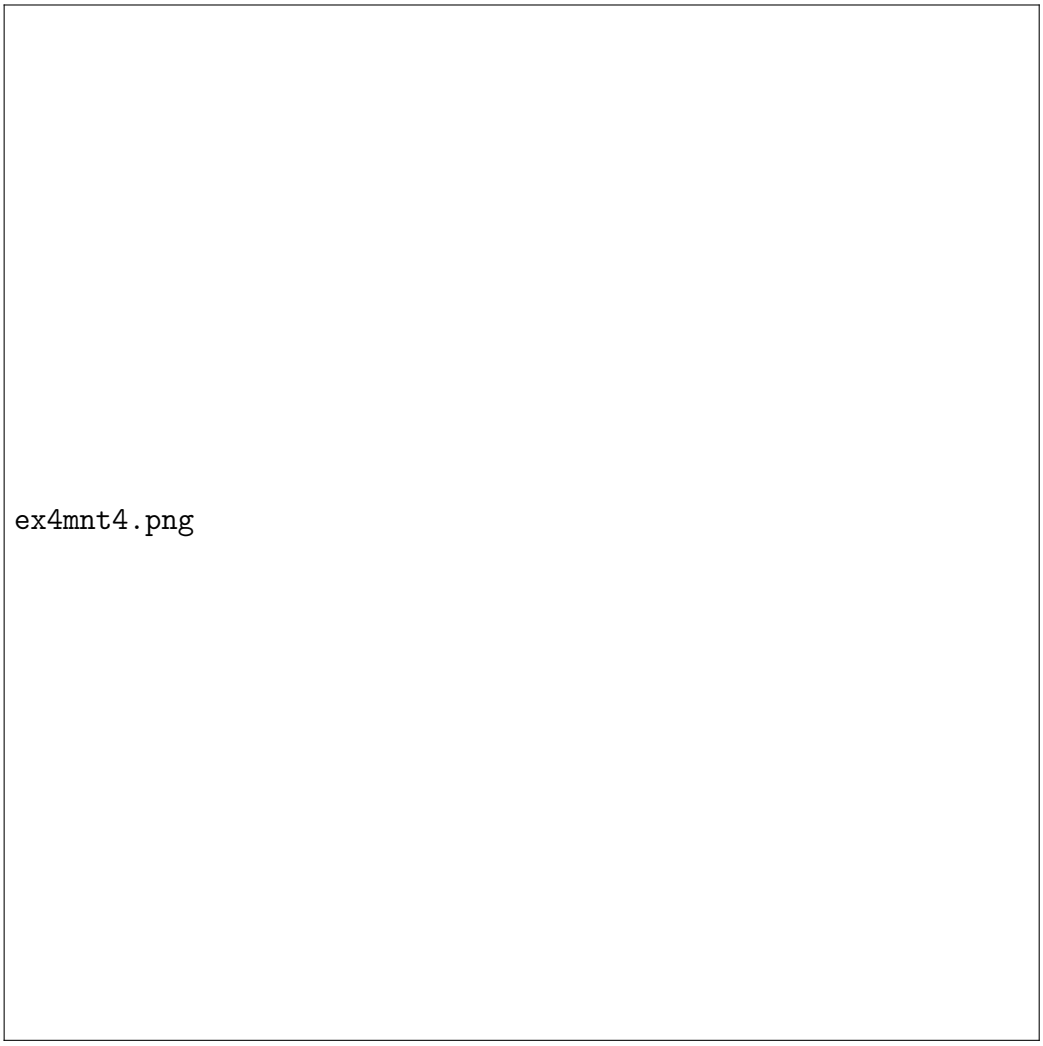

Figura 11: Quarto Nome adicionado ao Arquivo

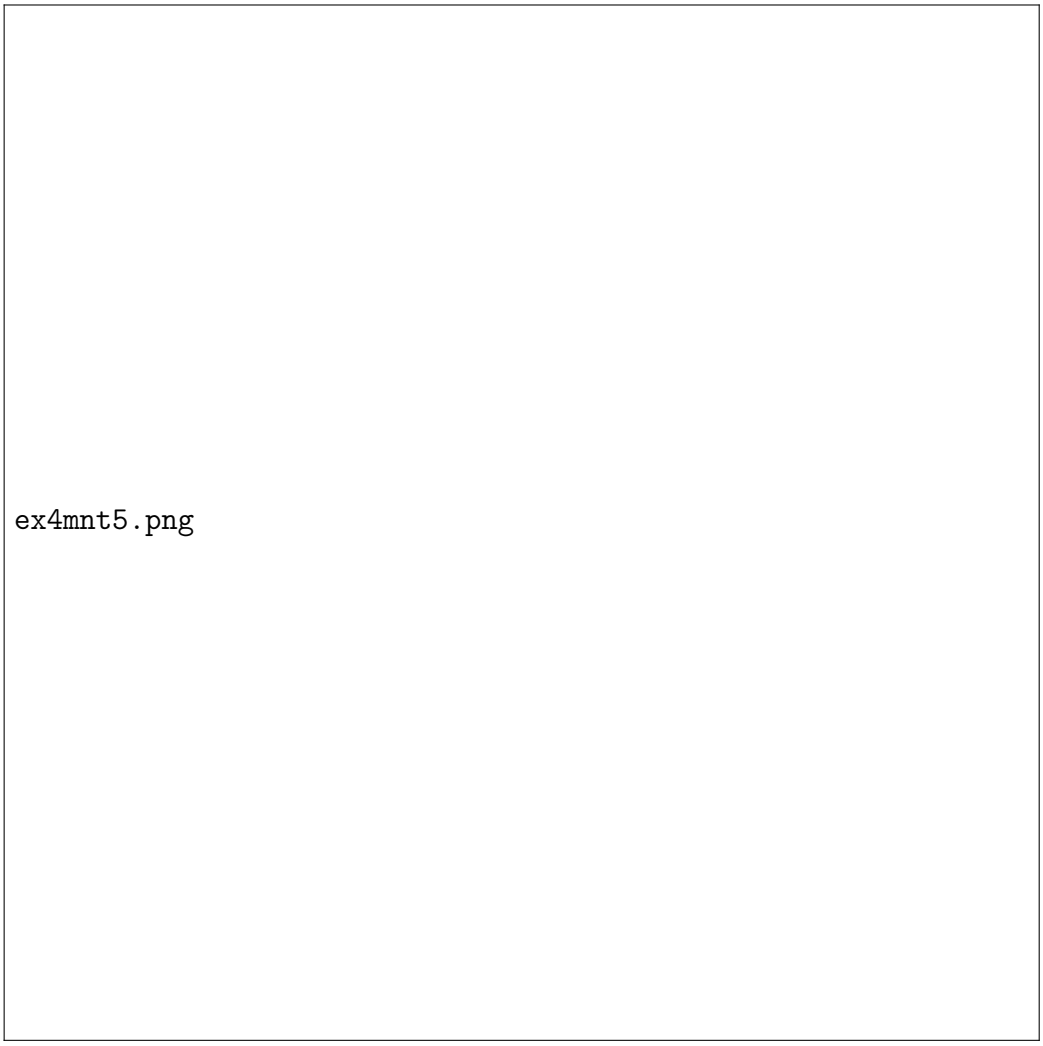

Figura 12: Exibição dos nomes adicionados ao Arquivo

6 Exercício 5

O exercício 5 requisitou a implementação do jogo Pedra, Papel e Tesoura tanto com trapaça quanto sem trapaça. No código implementado, inicialmente, era requisitado para o usuário escolher entre os modos "facil"e "dificil"sem a presença de acento nas palavras, de modo que o modo "facil"era o sem trapaça e o "dificil"era com trapaça. Após a escolha da dificuldade,

o jogador poderia escolher entre as opções de "pedra", "papel"ou "tesoura". Caso tenha requisitado o jogo no modo "facil", a máquina (jogador rival) sorteia, aleatoriamente a partir da função "random.randint()", 3 números entre 0, 1 e 2, sendo que 0 corresponde a "pedra", 1 corresponde a "tesoura"e 2 corresponde a "papel", sendo, posteriormente, mapeados do número inteiro à string correspondente. Caso tenha requisitado o jogo "difícil", a máquina, com base na escolha de seu adversário, sorteia 3 números entre 0,1 e 2 com peso de 60% para a escolha contrária à de seu adversário e de 20% para as demais escolhas a partir da função "random.choices()". Os valores escolhidos pelo jogador e pela máquina são, então, comparados até se obter um vencedor do duelo. Assim, caso haja empate, o jogo é jogado novamente até se obter um vencedor. Para os parâmetros que se requisita uma entrada do jogador, utilizou-se um laço "while"que se repete até que o jogador escolha parâmetros válidos, sendo que não há problemas em escrevê-los em letras minúsculas ou maiúsculas, já que, a partir do método de string ".lower()", é possível realizar a transformação das letras minúsculas em maiúsculas para se realizar a comparação. A seguir, são apresentados alguns resultados obtidos com esse algoritmo.

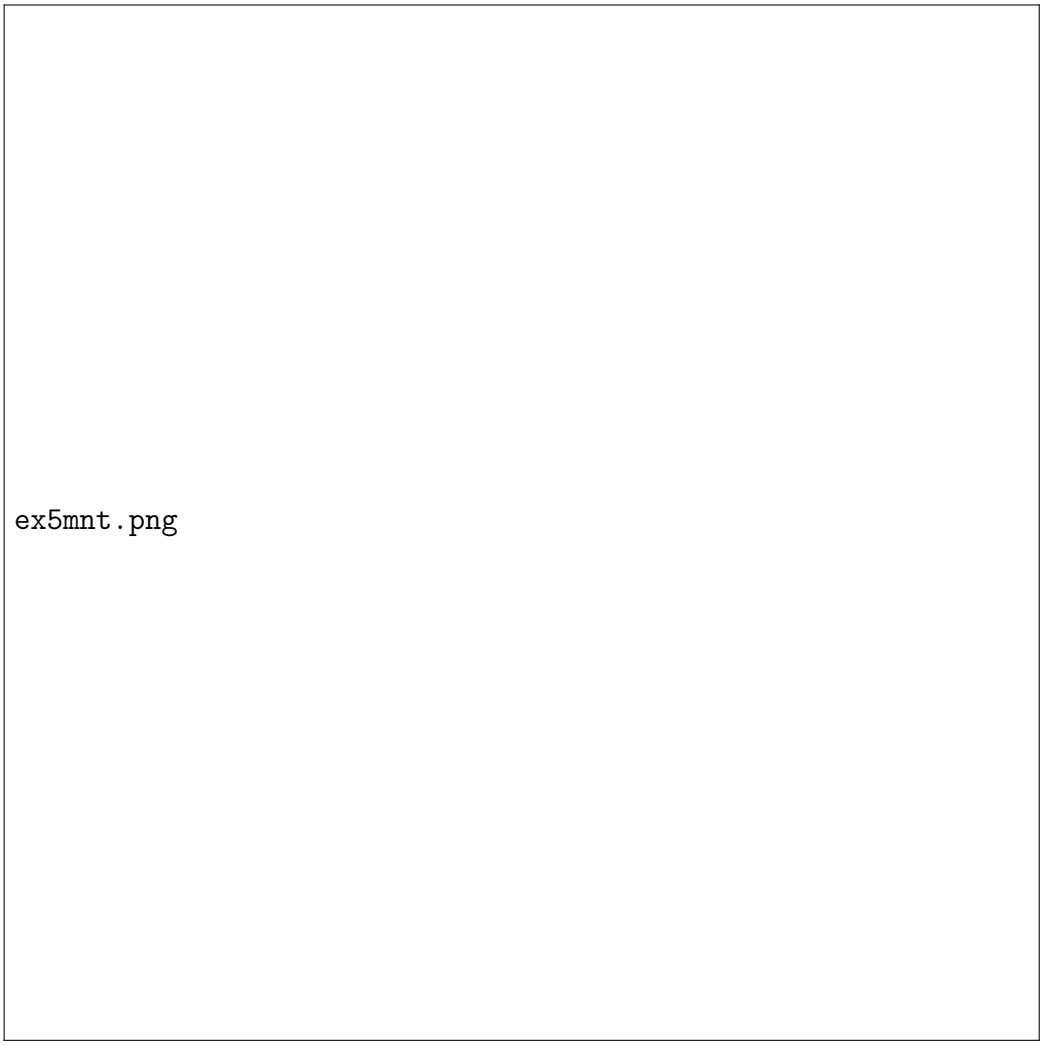

Figura 13: Primeiro Teste sem Trapaça

Figura 14: Segundo Teste sem Trapaça

Figura 15: Terceiro Teste sem Trapaça

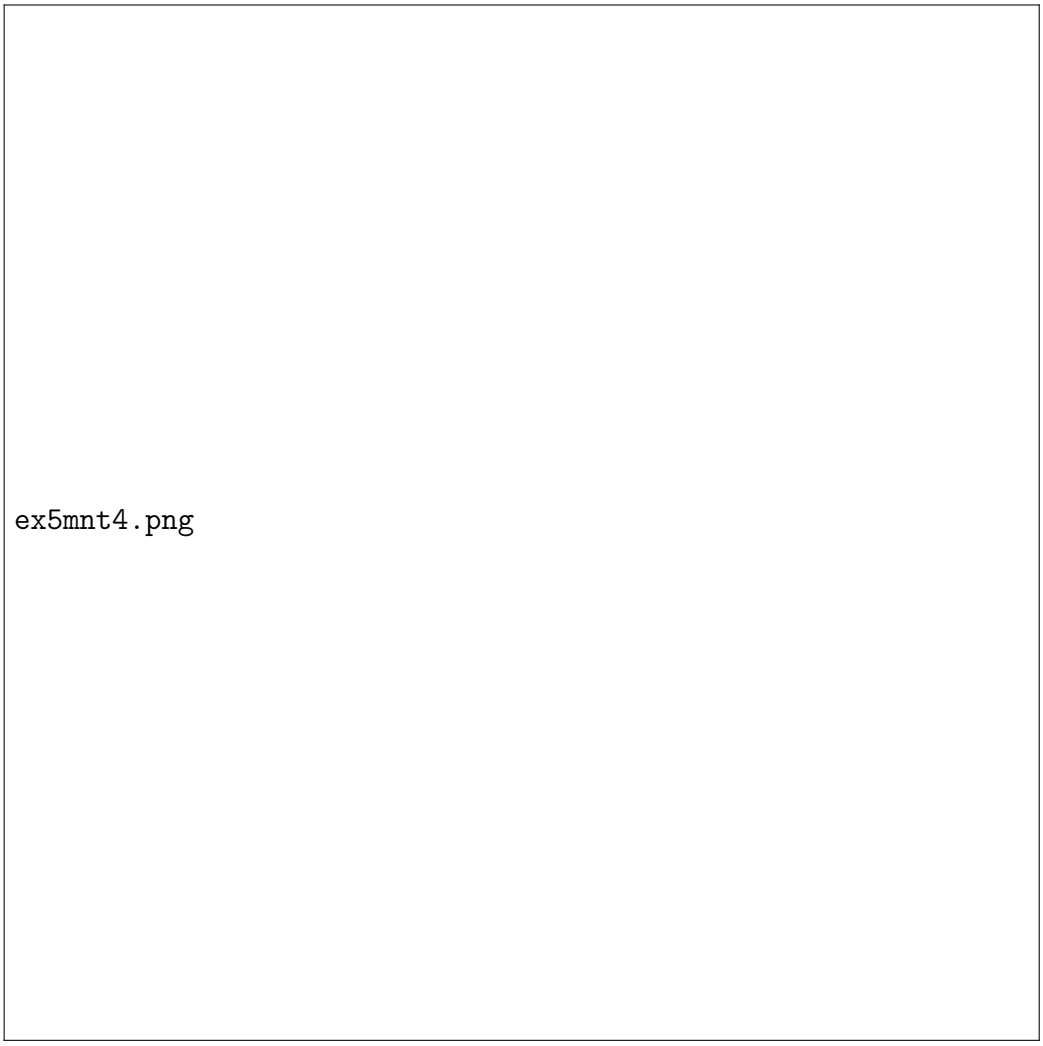

Figura 16: Primeiro Teste com Trapaça

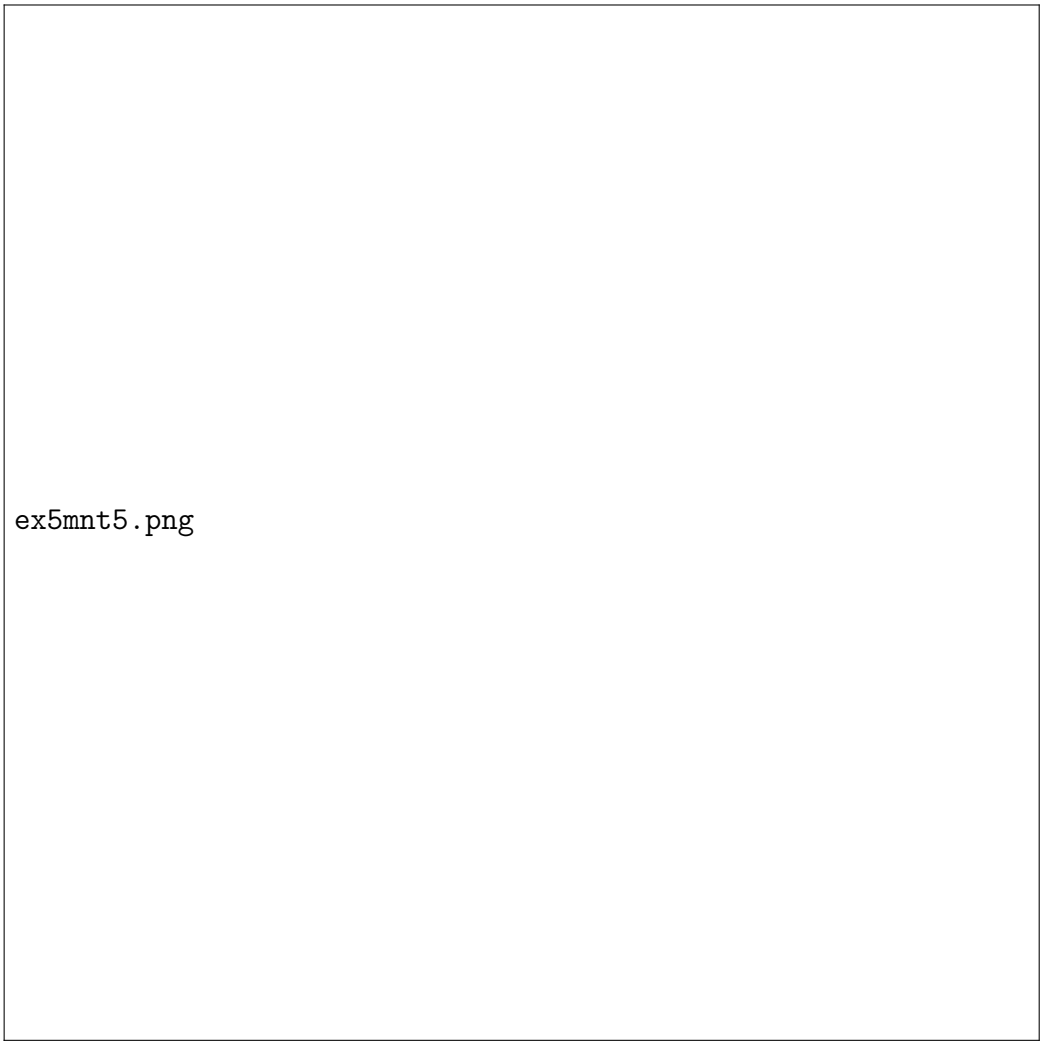

Figura 17: Segundo Teste com Trapaça

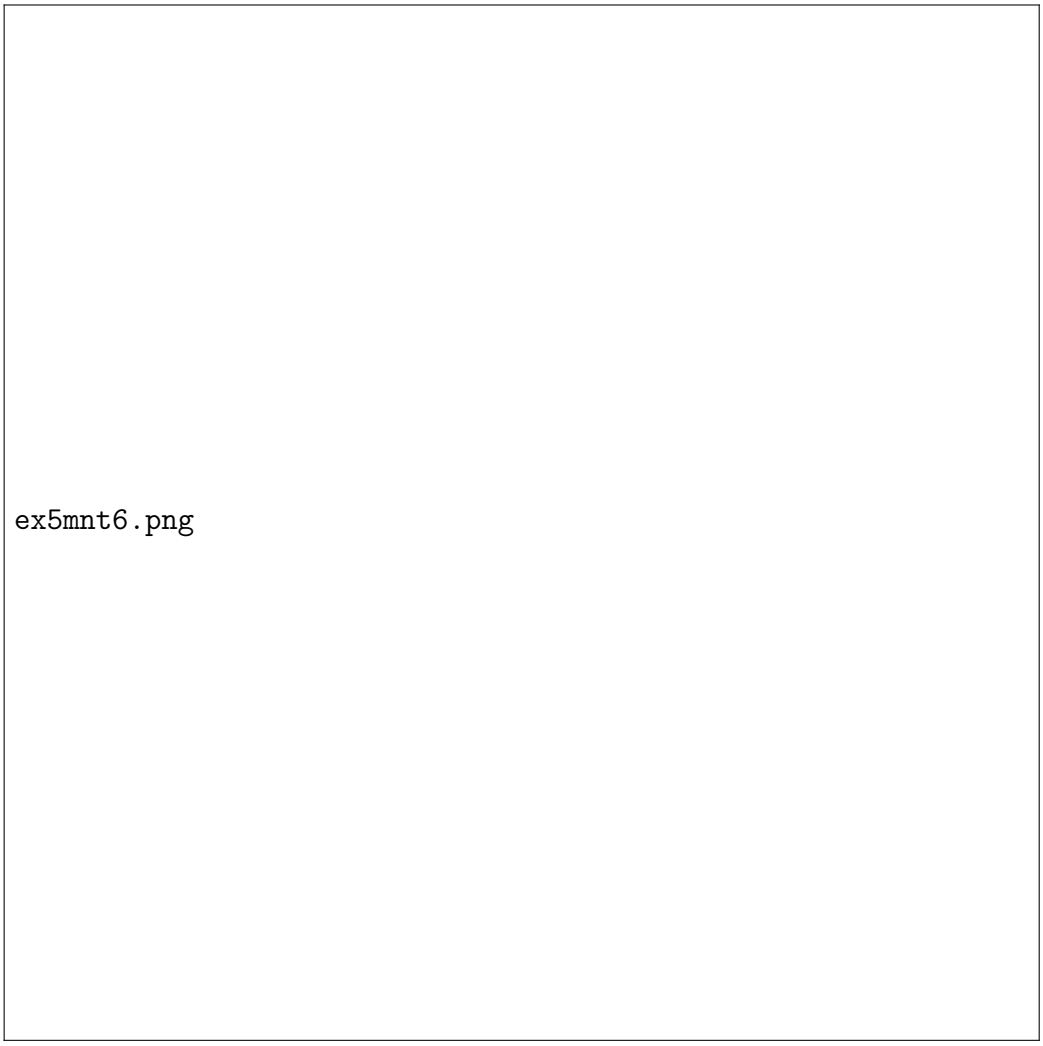

Figura 18: Terceiro Teste com Trapaça

7 Exercício 6

O exercício 6 requisitou a implementação de um gráfico da flutuação do dólar comercial nos últimos 20 dias. Esses valores foram fornecidos e postos nos códigos como a declaração de uma "array"da biblioteca numpy conhecida como "dolar", enquanto que os valores dos dias foram dados de 0 a 19 (20 dias incluindo o valor 0) a partir da transformação de uma lista com todos

os valores inteiros correspondentes em ordem crecente em uma "array"do numpy. Assim, com o auxílio da biblioteca "matplotlib", traçou-se o gráfico com título, nome da variável x e nome da variável y em que é possível verificar tanto a linha correspondente ao gráfico em azul quanto os valores em círculos vermelhos. A seguir, é possível verificar esse gráfico:

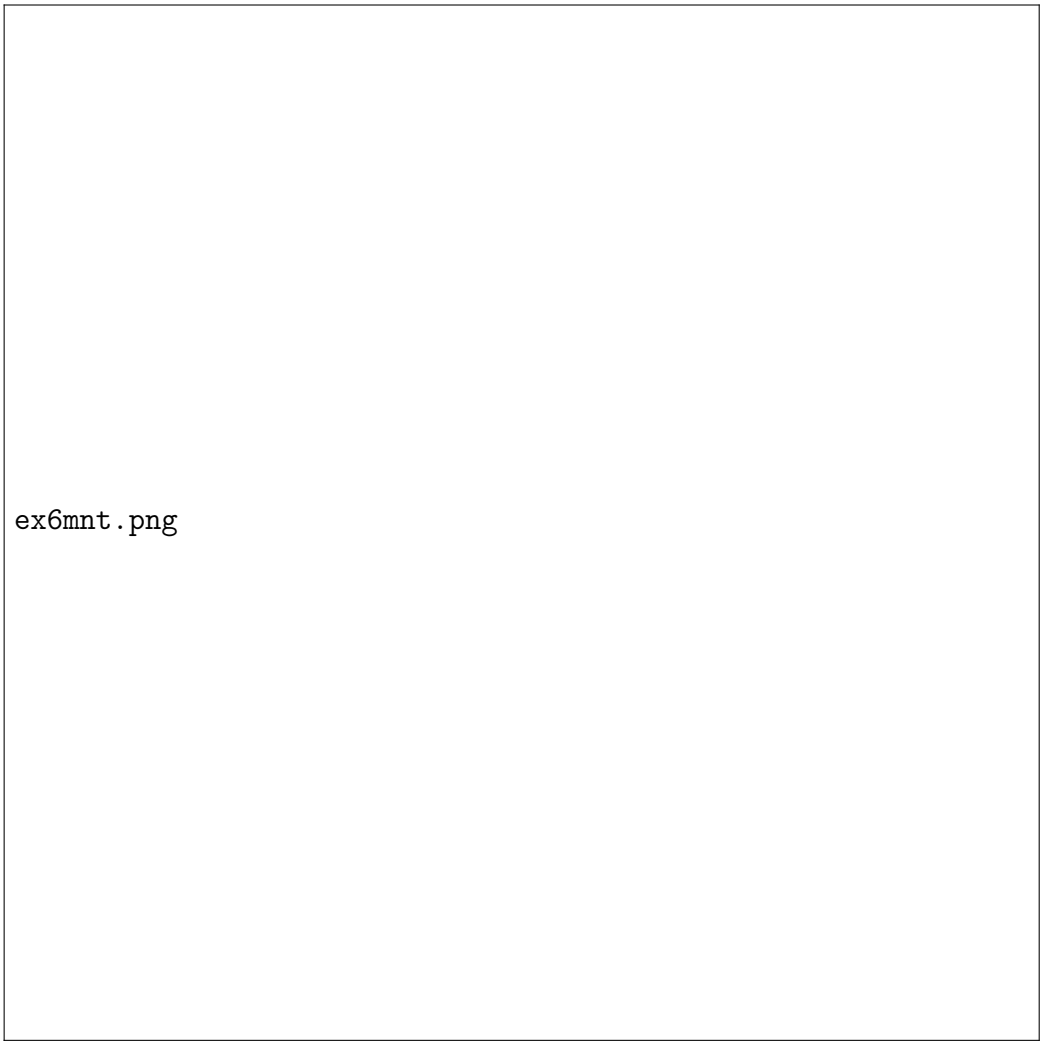

Figura 19: Gráfico obtido utilizando a biblioteca Matplotlib

8 Exercício 7

. O exercício 7 requisitou a obtenção da raiz $x^* \simeq 8.51$ para o Método do Ponto Fixo da função $f(x) = 2cosh(x/4) - x = 0$. A partir do algoritmo para o Método do Ponto Fixo mostrado em [Ros22], obteve-se a raiz de $x \simeq 2.38$, conforme previsto no roteiro, tendo em vista que o Método do Ponto Fixo sempre converge para a menor raiz de um número. Realizou-se, então, uma conversão na função fornecida de modo a tornar possível a convergência para esse método. Assim, a partir da soma de -x em ambos os termos da equação, obteve-se que:

$$(-x) + 2cosh(x/4) - x = 0 + (-x)$$

$$2cosh(x/4) - 2x = -x$$

$$x = 2x - 2cosh(x/4)$$

$$x = 2 \cdot (x - cosh(x/4))$$

Aplicando a função $f(x) = 2 \cdot (x - cosh(x/4))$ no algoritmo utilizado para a determinação da raiz superior de f(x) a partir do Método do Ponto Fixo, obteve-se o valor desejado. De fato, ao se tomar a derivada da nova função aplicada, percebe-se que converge segundo o método do ponto fixo tal que $|cosh(x/4)| < |x|$. Logo, matematicamente, obtém-se que:

$$\frac{dg(x)}{dx} \leq \rho < 1$$

$$\frac{dg(2 \cdot (x - cosh(x/4)))}{dx} < 1$$

$$2 \cdot (1 - \frac{1}{4} \cdot senh(x/4)) < 1$$

Que é verdadeiro caso $|cosh(x/4)| < |x|$. A seguir, é apresentado o resultado obtido com esse algoritmo.

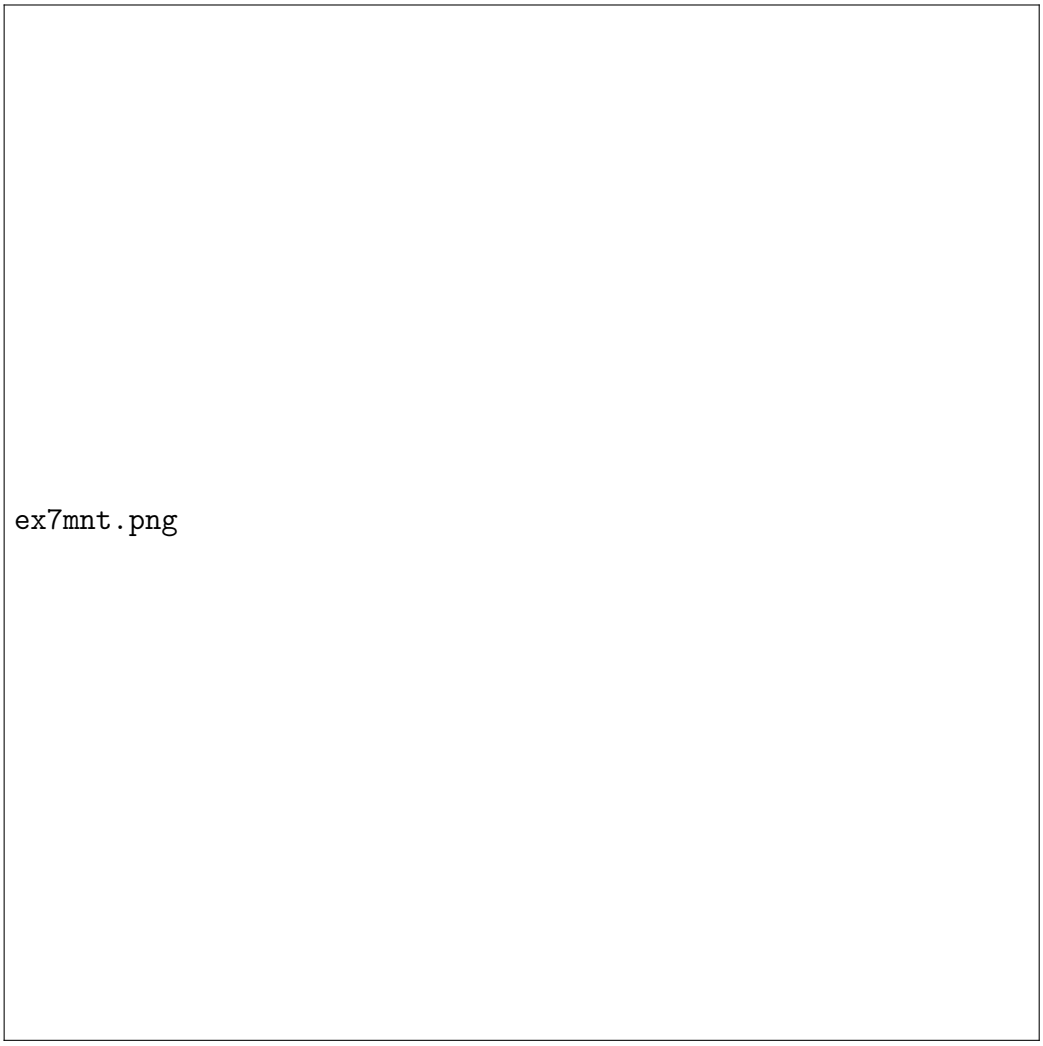

Figura 20: Aproximação da raiz de f(x) utilizando o Método do Ponto Fixo

9 Exercício 8

O exercício 8 requisitou a obtenção da mesma raiz presente em 8, porém, a partir do uso do Método da Secante. Utilizando o algoritmo presente em [Ros22], foi possível aplicar o Método e obter as duas raízes desejadas a depender dos valores inciais de x, de modo a se obter que, para x próximos à sua raiz de $x^* \simeq 2.38$, obtém-se essa raiz e, para x próximos à sua raiz de

$x^* \simeq 8.51$, obtém-se essa segunda raiz. A seguir, são apresentados alguns resultados obtidos com esse algoritmo.

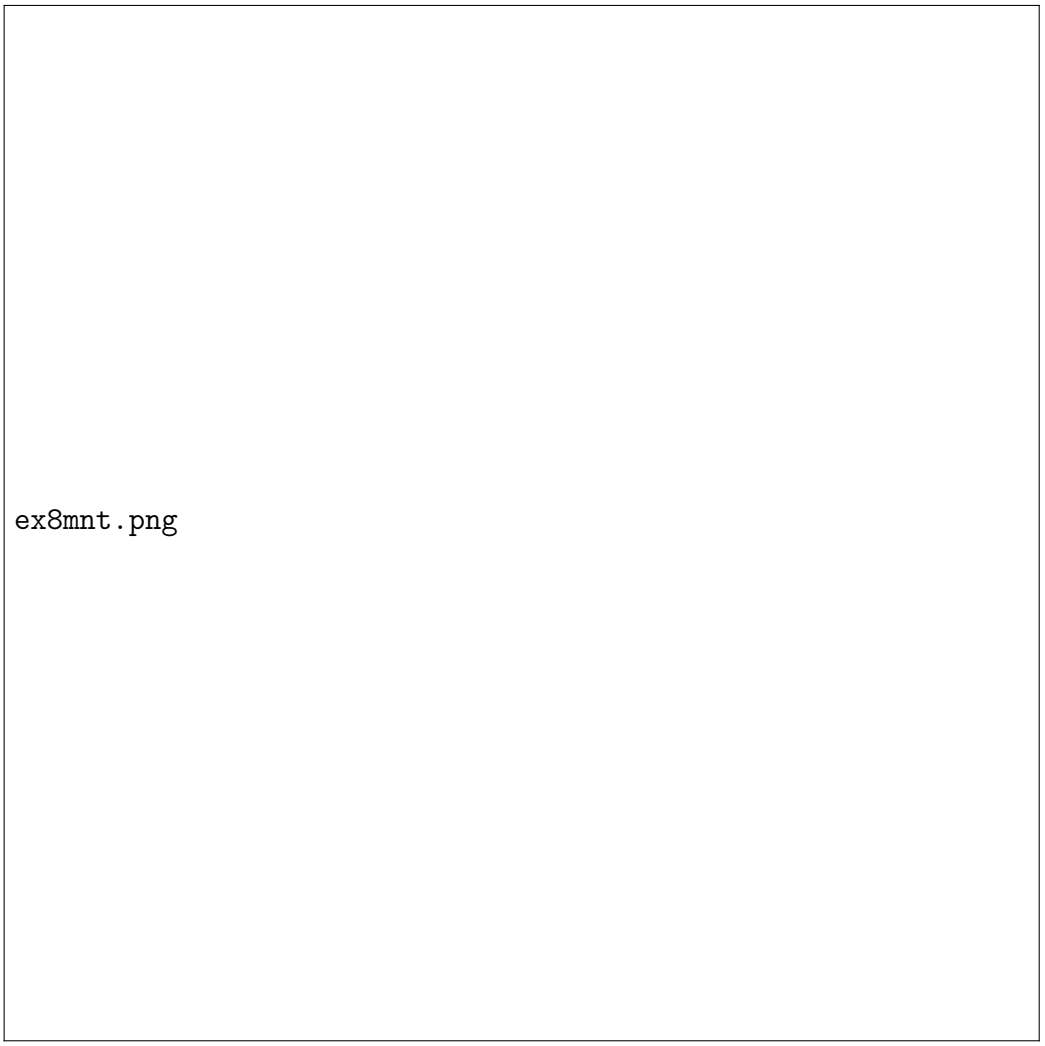

Figura 21: Aproximação da maior raiz de f(x) utilizando o Método da Secante

Figura 22: Aproximação da menor raiz de f(x) utilizando o Método da Secante

10 Exercício 9

O exercício 9 consistiu da Comparação entre 4 Métodos Numéricos para a obtenção de raízes de funções: Método da Bissecção, Método do Ponto Fixo, Método de Newton e Método da Secante. Para realizar essa operação, tomou-se um total de 50 iterações, as quais poderiam ser interrompidas caso

o erro atingisse o valor limítrofe de 0. A partir do Gráfico obtido, cujo eixo y está em escala logarítmica e para o qual, a linha azul corresponde ao Método da Bissecção, a linha laranja ao do Ponto Fixo, a linha verde ao de Newton e a linha vermelha ao da secante, percebe-se que os dois primeiros métodos expostos são lineares, enquanto os dois últimos são não lineares.

Para os Métodos da Bissecção e do Ponto Fixo, seu erro não atingiu o valor de 0 durante as iterações realizadas, de modo a se perceber que são métodos lentos, que necessitam de muitas iterações para se atingir a tolerância adequada. É notável, entretanto, que o Erro para o Método da Bissecção decresce muito mais rapidamente que para o Método do Ponto Fixo, muito provavelmente, devido ao fato de seu erro sempre cair por um fator de 2 a cada iteração, o que não é necessariamente verdade para o Método do Ponto Fixo que, ainda que comece com um erro relativamente menor, decresce mais lentamente que para o Método da Bissecção.

Para os Métodos de Newton e da Secante, percebe-se que o Método de Newton converge mais rapidamente que o da Secante, ainda que os dois apresentem um número de iterações próximas antes de convergirem. É interessante notar, entretanto, que o Método de Newton é mais trabalhoso de se aplicar na prática que o método da Secante, tendo em vista que, de acordo com [Ros22], esse método depende de que a derivada da função desejada seja conhecida. E, justamente por necessitar dessa derivada, o Método de Newton é relativamente mais rápido que o da Secante, o qual apenas a estima, tendo em vista que, por possuir essa derivada exata, é possível realizar aproximações mais acuradas a cada iteração, o que é mais trabalhoso para o Método da Secante por apenas estimá-la, ou seja, não possuir o valor exato da variação da função em certo instante.

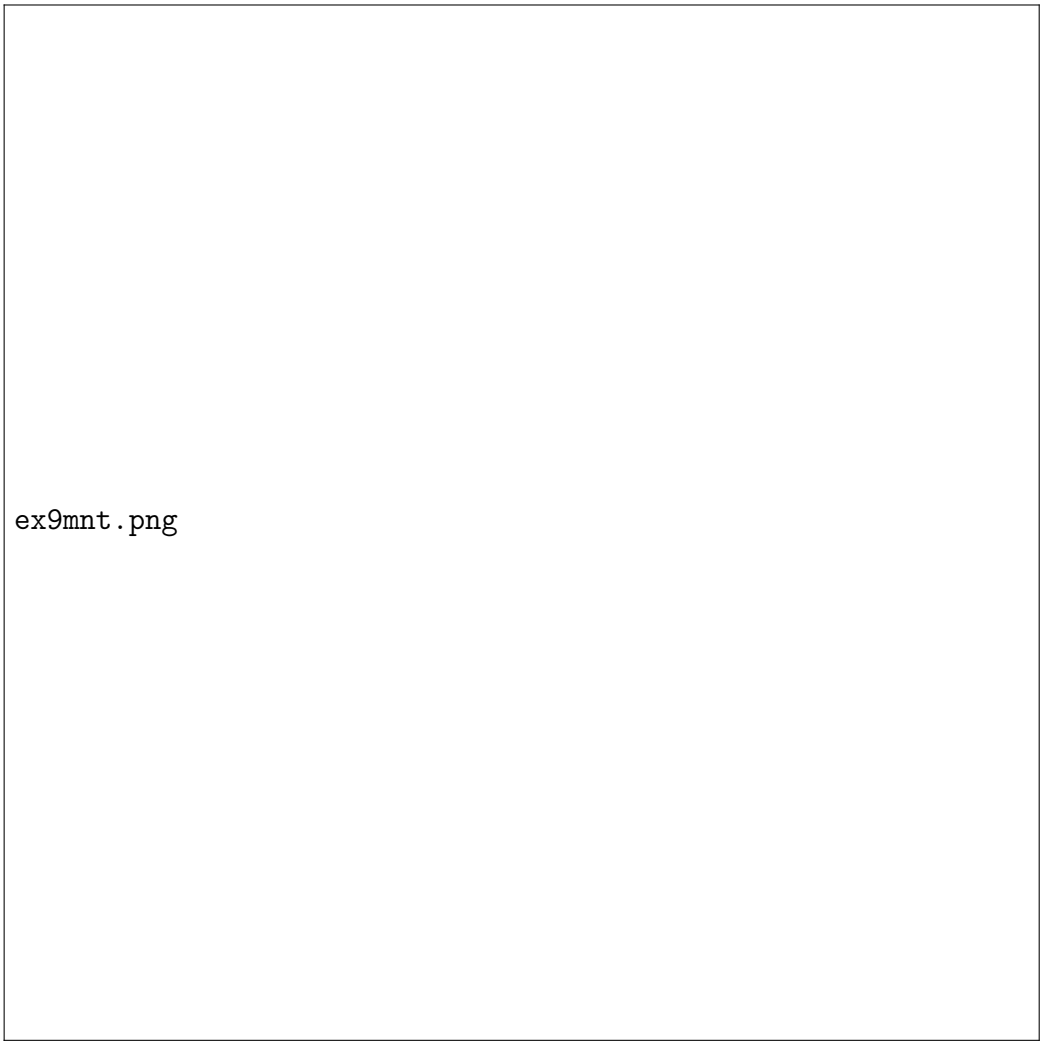

Figura 23: Linha Azul: Método da Bissecção; Linha Laranja: Método do Ponto Fixo; Linha Verde: Método de Newton; Linha Vermelha: Método da Secante

11 Exercício 10

Utilizando o mesmo algoritmo utilizado para o Método do Ponto Fixo que para o 8, percebeu-se que, para se encontrar f(x) tal que se possa escrever uma expressão iterativa que convirja para $\sqrt{2}$, bastava tomar $f(x) = x + 1 = \sqrt{2}$

tal que $f(x)^2 = (x+1)^2 = 2$ seja verdadeira. Dessa forma, isolando-se os termos, obteve-se que:

$$f(x)^2 = (x+1)^2 = 2$$

$$x^2 + 2x + 1 = 2$$

$$2x = 2 - 1 - x^2$$

$$x = \frac{1 - x^2}{2}$$

Assim, após aplicado o algoritmo do programa, obteve-se um termo x tal que, ao ser somado com 1, obtenha-se o valor desejado de $\sqrt{2}$. seguir, é apresentado o resultado obtido.

Figura 24: Método do Ponto Fixo para a obtenção de $\sqrt{2}$

12 Exercício 11

Utilizando o mesmo algoritmo utilizado para o Método da Bissecção em 10, adicionou-se a condicional de $f(a) \cdot f(b) < 0$ antes de aplicá-lo, de modo a se exibir uma mensagem de erro caso o algoritmo não possa ser aplicado. Após adicionada essa condicional, o algoritmo foi convertido em uma função que recebe três parâmetro, "a", "b"e "atol", em que "a"e "b"são os valores

limítrofes em que se encontra a raiz desejada e "atol"é a tolerância desejada. Esse algoritmo leva em consideração que a função a ser verificada está implementada ou importada no código principal como uma função, de modo a poder ser chamada diretamente dentro da função que realiza o cálculo de sua raiz pelo Método da Bissecção. Caso todas as iterações possíveis sejam realizadas e a tolerância desejada não foi encontrada, o programa retorna uma mensagem de erro. O Algoritmo implementado leva em consideração que $b > a$, de modo que, caso $a > b$, os valores dos dois limites são trocados a partir das atribuições do Python. A seguir são apresentados alguns resultados obtidos.

Figura 25: Método da Bissecção para Valores que Convergem

Figura 26: Método da Bissecção para Valores que não Convergem

13 Exercício 12

Para o exercício 12, foi requisitado que se provasse o porquê o Método de Newton apresenta taxa de convergência quadrática (não linear). Logo, de acordo com [UFR20], tomando uma função f(x) tal que sua primeira e segunda derivadas sejam contínuas e que $f(x^*) = 0$ e $f^{'}(x^*) \geq 0$, pode-se definir uma função g(x) tal que:

$$g(x) = x - \frac{f(x)}{f'(x)}$$

Tomando sua expansão em Série de Taylor, para $x = x^*$ obtém-se que:

$$g(x) = g(x^*) + g^{'}(x^*)(x - x^*) + \frac{g^{''}(x^*)}{2} \cdot (x - x^*)^2 + O((x - x^*)^3)$$

Como $g(x^*) = x^*$ e $g^{'}(x^*) = 1 - \frac{f^{'}(x^*) \cdot f^{'}(x^*) - f(x^*)f^{''}(x^*)}{(f^{'}(x^*))^2} = 0$, tem-se que:

$$g(x) = x^* + \frac{g^{''}(x^*)}{2} \cdot (x - x^*)^2 + O((x - x^*)^3)$$

Como g(x) é aproximado por x_{k+1}, obtém-se que:

$$x_{k+1} = x^* + \frac{g^{''}(x^*)}{2} \cdot (x - x^*)^2 + O((x - x^*)^3)$$

$$x_{k+1} - x^* = \frac{g^{''}(x^*)}{2} \cdot (x - x^*)^2 + O((x - x^*)^3)$$

Considerando $O((x - x^*)^3)$ como sendo tão pequeno que, na prática, é irrelevante e aproximando $\frac{g^{''}(x^*)}{2}$ como uma constante, obtém-se que:

$$x_{k+1} - x^* = \rho \cdot (x - x^*)^2$$

E, logo, o Método de Newton apresenta taxa de convergência quadrática.

14 Exercício 13

No Exercício 13, foi requisitado que se utilize um polinômio de Taylor para aproximar em torno de 45° para aproximar $cos(41°)$ com uma precisão de 10^{-6}. Para essa série, seu Erro pode ser aproximado como sendo da mesma ordem que o maior algarismo da potência correspondente à ordem da Série. Dessa forma, para $\Delta x = 0.1$, seu erro para uma Expansão em Série de Taylor de Segunda Ordem seria aproximadamente igual a $Erro = \frac{(\Delta X)^2}{2} = \frac{0.01}{2} \simeq 0.005$, o que equivaleria a um erro a partir da terceira casa decimal. Implementou-se, então, um código que verifica a ordem da Série de Taylor

segundo a precisão desejada. Obteve-se, assim, uma Ordem de 4, conforme pode ser verificado pelos resultados abaixo

Figura 27: Ordem da Série de Taylor para a Tolerância Desejada

15 Exercício 14

O Exercício 4 requisitou o desenvolvimento de um método numérico para se calcular o Fator de Atrito de Darcy a partir dos parâmetros fornecidos.

Para isso, escolheu-se o método da secante de modo que a primeira hipótese tomada fosse o número 1.0 e a segunda fosse dada pelo valor aproximado de f dado por Generaux. Obteve-se, assim, o seguinte gráfico em escala logarítmica (logXlog), o qual é correspondente ao gráfico esperado teoricamente para o Diagrama de Moody com o valor de convergência aproximado de $2 \cdot 10^{-2}$, segundo pode-se verificar pelo gráfico abaixo.

Figura 28: Diagrama de Moody para o parâmetro 0.01

A Código Exercício 1

```
import numpy as np

def mult_matrix(A,B):

    nl_a = len(A)
    if nl_a > 0: #verifica se a matriz e valida
        nc_a = len(A[0])
    nl_b = len(B)
    if nl_b > 0: #verifica se a matriz e valida
        nc_b = len(B[0])
    C = np.array(nl_a*[nc_b*[0]]) #Matriz Resposta
    aux_c = 0
    aux_l = 0
    if nc_a == nl_b: #Verifica se a Matriz e Multiplicavel
        while aux_l < nl_a:
            aux = 0
            for j in range(nc_a):
                aux += A[aux_l][j]*B[j][aux_c] #Atribui um elemento
                    da Matriz

            C[aux_l][aux_c] = aux #Atribuicao de um Termo de uma
                linha dessa Matriz

            aux_c+=1
            if aux_c == nc_b: #Condicao de mudana para a proxima
                Linha da Matriz
                aux_l+=1
                i = 0
                aux_c = 0
    else:
        print('Nao e possivel realizar a multiplicacao')
    return C


A = np.array([[1,2,3],[4,5,6]])

B = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
print('A:')
print(A)
print('B:')
print(B)
print('C:')
print(mult_matrix(A,B))
```

B Código Exercício 2

```
import numpy as np

def mediaedesvio(N):
    n = len(N)
    if n > 0: #Verifica se a Matriz fornecida  valida
        n2 = len(N[0])
        termos = n*n2 #numero de termos da Matriz (Linhas X Colunas)
        res = 0
        for i in range(n):
            for j in range(n2):
                res+=N[i][j] #Soma todos os valores da matriz

        media = res/termos #Divide o Resultado da Soma pelo numero
            de Termos para obter a Media

        res2 = 0

        for k in range(n):
            for l in range(n2):
                res2 += (media - N[i][j])**2 #Calculo dos termos do
                    numerador da formula do Desvio Padrao
        dp = np.sqrt(res2/termos) #Divisao pelo numero de termos e
            raiz quadrada para se obter o Desvio Padrao (dp)

        return media, dp
    else:
        return('Impossivel de se tomar a media e o desvio padrao de
            uma matriz vazia')
```

```
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
print('A:')
print(A)
print(mediaedesvio(A))
```

C Código Exercício 3

```
import numpy as np
import random

def mediaedesvio(N):
    n = len(N)
    if n > 0:
        n2 = len(N[0])
        termos = n*n2
        res = 0
        for i in range(n):
            for j in range(n2):
                res+=N[i][j]

        media = res/termos

        res2 = 0

        for k in range(n):
            for l in range(n2):
                res2 += (media - N[k][l])**2
        dp = np.sqrt(res2/termos)

        return media, dp


N = np.random.randint(2,1001)
A = np.random.uniform(-2,2,(N,N)) #Cria uma Matriz NXN aleatoria
print('A:')
```

```
print(A)
print(mediaedesvio(A))
```

D Código Exercício 4

D.1 Código da Parte 1

```
nome, idade = input('Digite seu nome e idade separados por
    espao:').split() #Requisita Nome e Idade
print(f'Seu nome  {nome} e sua idade  {idade} ano(s)') #Imprime
    Nome e Idade

arquivo = open("amigos.txt", "a") #Abre ou Cria um Arquivo para
    armazenar o Nome e Idade
arquivo.write('Seu nome  ' + nome + ' e sua idade  ' + idade + '
    ano(s)\n') #Sentena Armazenada
```

D.2 Código da Parte 2

```
arquivo = open("amigos.txt", "r") #Abre o Arquivo
for i in arquivo.readlines(): #Le as linhas do Arquivo
    print(i) #Imprime o Nome e a Idade
```

E Código Exercício 5

```
from secrets import choice
import numpy as np
import random

print("Vamos jogar pedra, papel e tesoura!!!")

#Requisita o modo de jogo vlido
while True:
```

```
        print("Primeiro, selecione o modo de jogo entre facil e
            dificil sem acento")
        dificuldade = input()
        dificuldade = dificuldade.lower()
        if(dificuldade != "dificil" and dificuldade != "facil"):
            print("Selecione entre facil e dificil")
        else:
            break

print("Selecione entre pedra, papel ou tesoura")
while True: #O Jogo nao possui empate, e jogado ate possuir um
    vencedor ou perdedor

    #Selecao entre pedra, papel e tesoura vlidos
    while True:
        jogador = input()
        jogador = jogador.lower()
        if(jogador != "pedra" and jogador != "papel" and jogador !=
            "tesoura"):
            print("Selecione entre pedra, papel ou tesoura, por
                favor")
        else:
            break
    #Definicao das probabilidades de acordo com a dificuldade do jogo
    lista_aux = [0,1,2]
    if dificuldade == 'dificil':
        if jogador == 'pedra':
            pesos = [20,20,60] #Peso caso o jogador escolha pedra
        if jogador == 'tesoura':
            pesos = [60,20,20] #Peso caso o jogador escolha tesoura
        if jogador == 'papel':
            pesos = [20,60,20] #Peso caso o jogador escolha papel

        maquina = random.choices(lista_aux, pesos) #escolhe um valor
            aleatorio entre 0,1 e 2 para os pesos dados
        maquina = maquina[0]
    else:
        maquina = np.random.randint(0,3) #Simplesmente, escolhe um
            valor inteiro aleatrio entre 0, 1 e 2
    #Converte os numeros da maquina em 'pedra, papel e tesoura'
```

```
    if maquina == 0:
        maquina = 'pedra'
    elif maquina == 1:
        maquina = 'tesoura'
    else:
        maquina = 'papel'
    #Jogo em si
    print(maquina)
    if jogador == maquina:
        print('empate. De novo...') #Empate, repete o jogo
    else:
        if jogador == 'pedra':
            if maquina == 'tesoura':
                print('Voce venceu!!!')
            if maquina == 'papel':
                print('Voce perdeu. Que pena...')

        if jogador == 'papel':
            if maquina == 'pedra':
                print('Voce venceu!!!')
            if maquina == 'tesoura':
                print('Voce perdeu. Que pena...')

        if jogador == 'tesoura':
            if maquina == 'papel':
                print('Voce venceu!!!')
            if maquina == 'pedra':
                print('Voce perdeu. Que pena...')
        break #Quando ha um ganhador, o jogo acaba
```

F Código Exercício 6

```
import numpy as np
import matplotlib.pyplot as plt

#Valores do dolar nos ultimos 20 dias
dolar = np.array([4.752, 4.804, 4.789, 4.779, 4.796, 4.874, 4.890,
    4.916, 4.987, 5.115, 5.133, 5.027, 5.027, 5.143, 5.187, 5.155,
```

```
    5.177, 5.229, 5.253, 5.233])

#Valores dos dias
dias = np.array ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
    15, 16, 17, 18, 19])

#Definicao de parametros do grafico
graf = plt.figure()
ax = graf.add_subplot()

#Legendas e nomes dos Eixos do Grafico
graf.suptitle('Dolar Comercial ao longo de 20 dias', fontsize = 12,
    fontweight = 'bold', usetex = False)
ax.set_ylabel(r'Dolar', fontsize = 12, usetex = False)
ax.set_xlabel(r'Dias', fontsize = 12, usetex = False)

#Tracando o Grafico
plt.plot(dias, dolar, '-r')
plt.plot(dias, dolar, '--b')

plt.show()

#Salvando o Grafico como uma figura
plt.savefig('dolar.pdf', format = 'pdf', dpi = 1200, bbox_inches =
    'tight')
```

G Código Exercício 7

```
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    return 2*x - (2.0*np.cosh(x/4.0))
x = 6.0 #hipotese inicial
k = 0

atol = 1.e-8
```

```
dif = 2.0*atol

while dif >= atol:
    novo_valor = f(x)

    dif = np.abs(novo_valor - x) #nova diferenca
    x = novo_valor
    k+=1
    print(f'Na iteracao {k}, x e igual a {x}')
```

H Código Exercício 8

```
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    return 2.0*np.cosh(x/4.0) - x #funcao a ser verificada
#Hipoteses Iniciais
x_antigo = 7.0
x = 9.0

#Parametros
atol = 1.e-8
k = 0
dif = 2.0*atol

#Algoritmo do Metodo da Secante
while dif >= atol:
    x_novo = x - (f(x)*(x-x_antigo))/(f(x)-f(x_antigo))
    dif = np.abs(x_novo-x)
    x_antigo = x
    x = x_novo
    k+=1
    print(f'Na iteracao {k}, x e igual a {x}')
```

I Código Exercício 9

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline
from scipy.interpolate import BSpline



def f(x):
    return 2.0*np.cosh(x/4.0) - x #funcao a ser verificada
def derivada(x):
    return 0.5*np.sinh(x/4.0) - 1.0 #derivada de f(x)


# Mtodo  da Bisseccao

#Variaveis de Amostragem

k = np.arange(0,50, 1)
erro_1 = np.zeros(50, float)
erro_2 = np.zeros(50, float)
erro_3 = np.zeros(50, float)
erro_4 = np.zeros(50, float)

a,b = 1.0, 20.0
for i in k:
    t = (a+b)/2.0
    if f(a)*f(t) < 0.0:
        b = t
    else:
        a = t
    t_novo = (a+b)/2.0
    erro_1[i] = np.abs(t_novo - t)
    if erro_1[i] == 0:
        break

#Metodo do Ponto Fixo
x = 1.0 #hipotese inicial
```

```
for i in k:
    novo_valor = f(x)
    erro_2[i] = np.abs(novo_valor - x) #nova diferenca
    x = novo_valor
    if erro_2[i] == 0:
        break

#Metodo de Newton
x = 5.0 #hipotese inicial

for i in k:
    novo_valor = x - f(x)/derivada(x)
    erro_3[i] = np.abs(novo_valor - x) #nova diferenca
    x = novo_valor
    if erro_3[i] == 0:
        k_3 = np.arange(0,i, 1)
        break

#Metodo da Secante

#Hipoteses Iniciais
x_antigo = 6.0
x = 9.0

for i in k:
    x_novo = x - (f(x)*(x-x_antigo))/(f(x)-f(x_antigo))
    erro_4[i] = np.abs(x_novo-x)
    x_antigo = x
    x = x_novo
    if erro_4[i] == 0:
        break

#Inicializando o Grafico

graf = plt.figure()
ax = graf.add_subplot()

#Parametros do Grafico
```

```
graf.suptitle('Comparacao entre os Metodos Numericos', fontsize =
    12, fontweight = 'bold', usetex = False)
ax.set_ylabel(r'Erro', fontsize = 12, usetex = False)
ax.set_xlabel(r'K', fontsize = 12, usetex = False)

#Tracando o Grafico

plt.plot(k,erro_1, label = 'Bisseccao')
plt.plot(k,erro_2, label = 'Ponto Fixo')
plt.plot(k,erro_3, label = 'Newton')
plt.plot(k,erro_4, label = 'Secante')
plt.yscale('log')
plt.show()

#Salvando uma figura

plt.savefig('comp_metodos.pdf', format = 'pdf', dpi = 1200,
    bbox_inches = 'tight')
```

J Código Exercício 10

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return (1.0-x**2)/2.0


x = 0.5 #hipotese inicial
k = 0

atol = 1.e-10
dif = 2.0*atol

while dif >= atol:

    novo_valor = f(x)
    dif = np.abs(novo_valor - x) #nova diferenca
```

```
    x = novo_valor
    k+=1
    print(f'Na iteracao {k}, x e igual a {x}')

x+=1
print(f'Valor Final: {x}')
```

K Código Exercício 11

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return 2.0*np.cosh(x/4.0) - x #funcao a ser verificada

#Metodo da Bisseccao
def bisseccao(a,b,atol):
    if a > b:
        a,b = b,a
    kmax = np.log2((b-a)/(2.0*atol))
    i = 0

    if f(a)*f(b) < 0:
        while True:
            t = (a+b)/2.0
            if f(a)*f(t) < 0.0:
                b = t
            else:
                a = t

            t_novo = (a+b)/2.0
            dif = np.abs(t_novo-t)

            if i >= kmax + 1:
                if dif < atol:
                    return t_novo
                else:
```

```
                return('Numero maximo de iteracoes realizado,
                    porem, sem sucesso em se adequar a tolerancia
                    requisitada')
            i+=1
    else:
        return('Nao e possivel garantir que o metodo va convergir')

x = bisseccao(15.0,-15.0,1.e-8)
print(x)
```

L Código Exercício 13

```
import numpy as np
import math


def termos_cos(x,atol):

    delta_x = np.abs(x - np.radians(45))
    ntermos = 0

    while True:
        aux = (delta_x**ntermos)/math.factorial(ntermos)
        if aux <= atol:
            break
        ntermos+=1
    return ntermos

x = termos_cos(np.radians(41),1.e-6)
print('Ordem da Serie de Taylor para a Tolerancia desejada', x)
```

M Código Exercício 14

```
import numpy as np
import matplotlib.pyplot as plt
```

```
#Aproximando f pelo Metodo da Secante
def f(x,ed,ref):
    return 1/np.sqrt(x) + 1/((-2*np.log10( ed/(3.7) +
        2.51/(ref*np.sqrt(x) )) )**2) #f desejado


# Mtodo  da Secante

#Hipoteses Iniciais dadas pelo Exerccio
ed = 0.001
ref = np.arange(4.0,7.1,0.207)
res = np.zeros(len(ref))

x_antigo = 0.05
x = 0.16*(10**4.0)*np.exp(-0.16)

for i in range(len(ref)):
    x_novo = x -
        (f(x,ed,10**ref[i])*(x-x_antigo))/(f(x,ed,10**ref[i])-f(x_antigo,ed,10**ref[i]))
    dif = np.abs(x_novo-x)
    if dif == 0:
        break
    x_antigo = x
    x = x_novo
    res[i] = f(x,ed,10**ref[i])


#Inicializando o Grafico

graf = plt.figure()
ax = graf.add_subplot()

#Parametros do Grfico

graf.suptitle('Fator de Atrito de Darcy de acordo com o Numero de
    Reynolds', fontsize = 12, fontweight = 'bold', usetex = False)
ax.set_ylabel(r'f', fontsize = 12, usetex = False)
ax.set_xlabel(r'Re', fontsize = 12, usetex = False)
```

```
#Tracando o Grafico

plt.plot(ref,res)
plt.xscale('log')
plt.yscale('log')
plt.show()

#Salvando uma figura

plt.savefig('Diagrama_Moody.pdf', format = 'pdf', dpi = 1200,
    bbox_inches = 'tight')
```

Referências

[UFR20] UFRGS. *Método de Newton-Raphson.* https://www.ufrgs.br/reamat/CalculoNumerico/livro-oct/sdeduv-metodo$_d e_n ewton - raphson.html : : text = Do\%20teorema\%20do\%20ponto\%20fixo, x\%20)\%20f\%20(\%20x\%20)\%20..$ Agosto de 2020.

[Ros22] Adriano Possebon Rosa. *Equações não Lineares.* https://aprender3.unb.br/pluginfile.php/2182428/mod$_r esource/content/5/Equacoes_n a$ Jan. de 2022.