Rotinas Desenvolvidas – P1T2 ALC

Matheus Lomba de Rezende Conde – DRE: 117085216

As rotinas desenvolvidas para o trabalho estão divididas em 4 agrupamentos: Main, Inputs, Métodos e Suporte.

Main: agrupamento principal por onde as funções são chamadas e o programa é executado.

from inputs import cria\_matriz, ordemN, icod, idet, tolM  
from metodos import met\_potencia, met\_jacobi  
from sup import finaliza  
  
#Inputs do programa  
var\_ordemN = ordemN()  
var\_icod = icod()  
var\_idet = idet()  
var\_tolM = tolM()  
A = cria\_matriz(var\_ordemN)  
  
if var\_icod == 1:  
 met\_potencia(A, var\_ordemN, var\_idet, var\_tolM)  
elif var\_icod == 2:  
 met\_jacobi(A, var\_ordemN, var\_idet, var\_tolM)  
finaliza()

Inputs: agrupamento onde ficam as rotinas que requisitam a entrada de dados fornecidos pelo usuário.

from sup import finaliza  
  
def cria\_matriz(ordemN):  
 matriz = [[]]  
 print('=' \* 30)  
  
 for i in range(0, ordemN):  
 for j in range(0, ordemN):  
 try:  
 matriz[i].append(float(input(f'Insira um valor para A({i+1},{j+1}): ')))  
 except ValueError:  
 print("Favor rodar o programa novamente e inserir um número válido (float).")  
 finaliza()  
 exit(0)  
 if i < ordemN-1:  
 matriz.append(list())  
  
 #Printar Matriz  
 print('='\*30)  
 print('Matriz A:')  
 for i in range(len(matriz)):  
 print(matriz[i])  
  
 return matriz  
  
#-------------------------------------  
  
def ordemN():  
 print('=' \* 30)  
 while True:  
 try:  
 var\_ordemN = int(input('Qual é a ordem N do sistema de equações? '))  
 if var\_ordemN < 1:  
 print('Favor inserir um número inteiro positivo e não nulo.')  
 else:  
 break  
 except ValueError:  
 print('Favor inserir um número inteiro.')  
 return var\_ordemN  
  
  
def icod():  
 print('=' \* 30)  
 print('1 = Método da Potência\n2 = Método de Jacobi')  
 while True:  
 try:  
 var\_icod = int(input('Qual método será utilizado? '))  
 if var\_icod != 1 and var\_icod != 2:  
 print('Favor inserir um valor válido para o método.')  
 else:  
 break  
 except ValueError:  
 print('Favor inserir um valor válido para o método (1 ou 2)')  
 return var\_icod  
  
  
def idet():  
 print('=' \* 30)  
  
 while True:  
 var\_idet = str(input('Deseja calcular a determinante da Matriz A? (s/n) '))  
 if var\_idet.lower() != 's' and var\_idet.lower() != 'n':  
 print('Favor inserir uma resposta válida (s/n).')  
 else:  
 if var\_idet.lower() == 's':  
 return True  
 else:  
 return False  
  
  
def tolM():  
 print('=' \* 30)  
 while True:  
 try:  
 var\_tolM = float(input('Qual será a tolerância máxima para a solução iterativa? '))  
 break  
 except ValueError:  
 print('Favor inserir um número float.')  
 return var\_tolM

Métodos: agrupamento onde estão armazenados os métodos requisitados no trabalho para a solução de sistemas lineares.

import numpy as np  
import math  
from sup import simetrica, maiorTolM, maiorForaDiag  
  
#------------------------------------------------------------  
  
def met\_potencia(matrizA, ordemN, idet, tolM):  
  
 l0 = 1 #lambda0  
 x = []  
 print('=' \* 30)  
 for i in range(0, ordemN):  
 x.append(float(input('Insira um valor para o vetor solução X: ')))  
  
 r = 100  
 n\_it = 0  
 while r > tolM:  
  
 y = np.dot(matrizA, x)  
  
 l = y[0]  
 for i in range(0, len(x)):  
 x[i] = y[i]/l  
  
 r = abs(l-l0)/abs(l)  
 l0 = l  
 n\_it += 1  
 print('=' \* 30)  
 print(f'R da iteração {n\_it}: {r}')  
  
 print('=' \* 30)  
 autovalores = np.linalg.eigvals(matrizA)  
 av = []  
 for a in autovalores:  
 av.append(a)  
 print(f'Autovalores:\n{av}')  
 print('=' \* 30)  
 print(f'Autovetor:\n{x}')  
 print('=' \* 30)  
 if idet:  
 print(f'Determinante de A = {np.linalg.det(matrizA)}')  
  
 return 0  
  
#------------------------------------------------------------  
  
def met\_jacobi(matrizA, ordemN, idet, tolM):  
  
 #Condição: só matriz simétrica  
 if simetrica(matrizA, ordemN):  
  
 detA = np.linalg.det(matrizA)  
  
 x = np.eye(ordemN)  
  
 n\_it = 0  
 while maiorTolM(matrizA, ordemN, tolM):  
  
 maior, maior\_i, maior\_j = maiorForaDiag(matrizA, ordemN)  
  
 if matrizA[maior\_i][maior\_i] == matrizA[maior\_j][maior\_j]:  
 phi = math.pi / 4  
 else:  
 phi = 0.5 \* ( math.atan( (2 \* matrizA[maior\_i][maior\_j]) / (matrizA[maior\_i][maior\_i] - matrizA[maior\_j][maior\_j]) ) )  
  
 p = np.eye(ordemN)  
 p[maior\_i][maior\_i] = math.cos(phi)  
 p[maior\_i][maior\_j] = -1 \* math.sin(phi)  
 p[maior\_j][maior\_i] = math.sin(phi)  
 p[maior\_j][maior\_j] = math.cos(phi)  
  
 pt = np.transpose(p)  
 matrizA = np.dot(np.dot(pt, matrizA), p)  
  
 x = np.dot(x, p)  
  
 n\_it += 1  
  
 av = []  
 for i in range(0, ordemN):  
 av.append(matrizA[i][i])  
  
 print('=' \* 30)  
 print(f'Matriz P:\n{p}')  
 print('=' \* 30)  
 print(f'Matriz A:\n{matrizA}')  
 print('=' \* 30)  
 print(f'Autovalores: {av}')  
 print('=' \* 30)  
 print(f'Matriz X:\n{x}')  
 print('=' \* 30)  
 print(f'{n\_it} iterações para convergência.')  
 print('=' \* 30)  
 if idet:  
 print(f'Determinante de A = {detA}')  
 print('=' \* 30)  
  
 return 0

Suporte: agrupamento onde estão armazenadas funções de suporte, que auxiliam as funções principais a realizarem seus processos corretamente.

import numpy as np  
  
def finaliza():  
 # Impede que o terminal feche automaticamente assim que o programa finaliza.  
 input('Pressione qualquer tecla para finalizar...')  
  
def sim\_def\_pos(matriz, ordemN):  
 return def\_pos(matriz) and simetrica(matriz, ordemN)  
  
def def\_pos(matriz):  
 return np.all(np.linalg.eigvals(matriz) > 0)  
  
def simetrica(matriz, ordemN):  
 matrizt = np.transpose(matriz)  
 for i in range(0, ordemN-1):  
 for j in range(0, ordemN - 1):  
 if matrizt[i][j] == matriz[i][j]:  
 return True  
 else:  
 return False  
  
def not\_diag\_dom(matriz, ordemN):  
 for i in range(0, ordemN):  
 a = 0  
 b = 0  
 for j in range(0, ordemN):  
 if i == j:  
 a = matriz[i][j]  
 else:  
 b += matriz[i][j]  
 if a < b:  
 print("A matriz A inputada não é diagonal dominante e, portanto, a condição para convergência da solução não é atendida.\nFavor inputar novamente uma matriz válida.")  
 return False  
 for j in range(0, ordemN):  
 a = 0  
 b = 0  
 for i in range(0, ordemN):  
 if i == j:  
 a = matriz[i][j]  
 else:  
 b += matriz[i][j]  
 if a < b:  
 print("A matriz A inputada não é diagonal dominante e, portanto, a condição para convergência da solução não é atendida.\nFavor inputar novamente uma matriz válida.")  
 return False  
  
  
def maiorTolM(matrizA, ordemN, tolM):  
  
 for i in range(0, ordemN):  
 for j in range(0, ordemN):  
 if i != j:  
 if abs(matrizA[i][j]) > tolM and abs(matrizA[i][j]) != 0:  
 return True  
 return False  
  
  
def maiorForaDiag(matrizA, ordemN):  
  
 maior = maior\_i = maior\_j = 0  
 for i in range(0, ordemN):  
 for j in range(0, ordemN):  
 if i != j:  
 if abs(matrizA[i][j]) > abs(maior):  
 maior = matrizA[i][j]  
 maior\_i = i  
 maior\_j = j  
 return maior, maior\_i, maior\_j