Rotinas Desenvolvidas – P1T1 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, cria\_vetor, ordemN, icod, idet, tolM  
from metodos import dec\_LU, dec\_Cholesky, proc\_Jacobi, proc\_Gauss\_Seidel  
from sup import finaliza  
  
#Inputs do programa  
var\_ordemN = ordemN()  
var\_icod = icod()  
var\_idet = idet()  
if var\_icod == 3 or var\_icod == 4:  
 var\_tolM = tolM()  
  
A = cria\_matriz(var\_ordemN)  
B = cria\_vetor(var\_ordemN)  
  
if var\_icod == 1:  
 dec\_LU(A, B, var\_ordemN, var\_idet)  
elif var\_icod == 2:  
 dec\_Cholesky(A, B, var\_ordemN, var\_idet)  
elif var\_icod == 3:   
 proc\_Jacobi(A, B, var\_ordemN, var\_idet, var\_tolM)  
elif var\_icod == 4:  
 proc\_Gauss\_Seidel(A, B, 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 cria\_vetor(ordemN):  
 vetor = []  
 print('=' \* 30)  
  
 for i in range(0, ordemN):  
 try:  
 vetor.append(float(input('Insira um valor para o vetor B: ')))  
 except ValueError:  
 print("Favor rodar o programa novamente e inserir um número válido (float).")  
 finaliza()  
 exit(0)  
  
 #Printar Vetor  
 print('=' \* 30)  
 print(f'Vetor B: {vetor}')  
  
 return vetor  
  
#-------------------------------------

def ordemN():  
 print('=' \* 30)  
 while True:  
 try:  
 var\_ordemN = int(input('Qual é a ordem N do sistema de equações? '))  
 if var\_ordemN < 2:  
 print('Favor inserir um número maior ou igual a 2.')  
 else:  
 break  
 except ValueError:  
 print('Favor inserir um número maior ou igual a 2.')  
 return var\_ordemN  
  
  
def icod():  
 print('=' \* 30)  
 print('1 = Decomposição LU\n2 = Decomposição de Cholesky\n3 = Procedimento Iterativo Jacobi\n4 = Procedimento Iterativo Gauss-Seidel')  
 while True:  
 try:  
 var\_icod = int(input('Qual método será utilizado? '))  
 if var\_icod < 1 or var\_icod > 4:  
 print('Favor inserir um valor válido para o método.')  
 else:  
 break  
 except ValueError:  
 print('Favor inserir um valor válido para o método (1, 2, 3 ou 4)')  
 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  
from sup import sim\_def\_pos, not\_diag\_dom

def dec\_LU(matrizA, vetorB, ordemN, idet):  
  
 if ordemN == 2:  
 m = matrizA[1][0]/matrizA[0][0]  
 l = [[1, 0], [m, 1]]  
 u = np.linalg.solve(l, matrizA)  
  
 y = np.linalg.solve(l, vetorB)  
 x = np.linalg.solve(u, y)  
  
 elif ordemN == 3:  
 m1 = [[1, 0, 0], [-1 \* matrizA[2-1][1-1]/matrizA[1-1][1-1], 1, 0], [-1 \* matrizA[3-1][1-1]/matrizA[1-1][1-1], 0, 1]]  
 m2 = [[1, 0, 0], [0, 1, 0], [0, -1 \* np.dot(m1, matrizA)[3-1][2-1]/np.dot(m1, matrizA)[2-1][2-1], 1]]  
  
 u = np.dot(np.dot(m2, m1), matrizA)  
 l = np.linalg.inv(np.dot(m2, m1))  
  
 y = np.linalg.solve(l, vetorB)  
 x = np.linalg.solve(u, y)  
  
 print('=' \* 30)  
 print(f'M1: {m1}')  
 print('=' \* 30)  
 print(f'M2: {m2}')  
 print('=' \* 30)  
 print(f'U:\n{u}')  
 print('=' \* 30)  
 print(f'L:\n{l}')  
 print('=' \* 30)  
 print(f'Y: {y}')  
 print('=' \* 30)  
 print(f'X: {x}')  
 print('=' \* 30)  
 if idet:  
 print(f'Determinante de A = {np.linalg.det(matrizA)}')  
 print('=' \* 30)  
 return 0

def dec\_Cholesky(matrizA, vetorB, ordemN, idet):  
  
 if sim\_def\_pos(matrizA):  
 l = [[]]  
 for i in range(0, ordemN):  
 for j in range(0, ordemN):  
 if i == j:  
 sub = 0  
 for n in range(0, i):  
 sub += (l[i][n])\*(l[i][n])  
 l[i].append(np.sqrt(matrizA[i][j] - sub))  
 elif i > j:  
 sub = 0  
 for n in range(0, j):  
 sub += l[j][0] \* l[i][0]  
 l[i].append((matrizA[i][j] - sub)/l[j][j])  
 else:  
 l[i].append(0)  
 if i < ordemN - 1:  
 l.append(list())  
  
 u = np.transpose(l)  
  
 y = np.linalg.solve(l, vetorB)  
 x = np.linalg.solve(u, y)  
  
 print('=' \* 30)  
 print(f'L:\n{l}')  
 print('=' \* 30)  
 print(f'U:\n{u}')  
 print('=' \* 30)  
 print(f'Y: {y}')  
 print('=' \* 30)  
 print(f'X: {x}')  
 print('=' \* 30)  
 if idet:  
 print(f'Determinante de A = {np.linalg.det(matrizA)}')  
 print('=' \* 30)  
 return 0  
  
 else:  
 print(f'A matriz {matrizA} não é simétrica definida positiva.')  
 return 0  
  
  
def proc\_Jacobi(matrizA, vetorB, ordemN, idet, tolM):  
  
 if not not\_diag\_dom(matrizA, ordemN): #Condição para convergência: matrizA diagonal dominante  
 x = []  
 xi = []  
 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:  
 for i in range(0, ordemN):  
 sub = 0  
 for j in range(0, ordemN):  
 if i != j:  
 sub += matrizA[i][j] \* x[j]  
 xi.append( (vetorB[i] - sub)/matrizA[i][i] )  
  
 v\_aux = np.subtract(xi, x)  
 r = np.linalg.norm(v\_aux)/np.linalg.norm(xi)  
  
 x = xi[:]  
 xi = []  
 n\_it += 1  
 print(f'R da iteração {n\_it}: {r}')  
 print('=' \* 30)  
  
 print('=' \* 30)  
 print(f'X:\n{x}')  
 print('=' \* 30)  
 if idet:  
 print(f'Determinante de A = {np.linalg.det(matrizA)}')  
 print('=' \* 30)  
 return 0  
  
  
def proc\_Gauss\_Seidel(matrizA, vetorB, ordemN, idet, tolM):  
  
 if not not\_diag\_dom(matrizA, ordemN) or sim\_def\_pos(matrizA, ordemN): #Condição para convergência  
 x = []  
 print('=' \* 30)  
 for i in range(0, ordemN):  
 x.append(float(input('Insira um valor para o vetor solução X: ')))  
 xi = x[:]  
  
 r = 100  
 n\_it = 0  
 while r > tolM:  
 for i in range(0, ordemN):  
 sub = 0  
 for j in range(0, ordemN):  
 if i != j:  
 sub += matrizA[i][j] \* xi[j]  
 xi[i] = (vetorB[i] - sub) / matrizA[i][i]  
  
 v\_aux = np.subtract(xi, x)  
 r = np.linalg.norm(v\_aux) / np.linalg.norm(xi)  
  
 x = xi[:]  
 n\_it += 1  
 print('=' \* 30)  
 print(f'R da iteração {n\_it}: {r}')  
  
 print('=' \* 30)  
 print(f'X:\n{x}')  
 print('=' \* 30)  
 if idet:  
 print(f'Determinante de A = {np.linalg.det(matrizA)}')  
 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