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
```