Universidade Veiga de Almeida

Aluno: Raphael Ricardo Gonçalves

Matrícula: 20182101572

Disciplina: Cálculo Numérico

**Introdução**

Encontrar o valor das variáveis x1, x2 e x3 através dos métodos de Gauss-Jacobi e Gauss-Seidel para a solução de um dado sistema de equações lineares (ver a seguir).

3x1 – 0.1x2 – 0.2x3 = 7.85

0.1x1 + 7x2 – 0.3x3 = -19.3

0.3x1 – 0.2x2 +10x3 = 71.4

**Desenvolvimento**

Método de Gauss-Jacobi:

Para solucionar o problema foi escrito um programa em python conforme abaixo:

import numpy as np

import csv

def jacobi\_method(A, x, b, err=0.001, prec=3, iteration=0):

    #Calcula o tamanho do sistema

    n = len(A.shape) + 1

    #Mostra 3 casas decimais

    np.set\_printoptions(precision=prec)

    #Monta a matriz diagonal

    lst = []

    for i in range(0, n):

        lst.append(A[i][i])

    D = np.diag(lst)

    #Isola as variaveis

    R = A - D

    Dinv = np.linalg.inv(D)

    Rx = R.dot(x)

    #Resultado = D^-1\*(b - Rx)

    xf = Dinv.dot(b-Rx)

    #CSV para análise dos resultados

    with open('jacobi.csv', 'a', newline='\n') as csvfile:

        writter = csv.writer(csvfile)

        cols = [iteration, xf[0][0], xf[1][0], xf[2][0]]

        writter.writerow(cols)

    print("Valor de Xf")

    print(xf)

    #Análise de convergência

    conv = True

    for i in range(0, n):

        if round(xf[i][0], prec) != round(x[i][0], prec):

            conv = False

    if conv:

        return xf

    else:

        #Caso não convirja, é feita a análise de erro

        Ax = A.dot(x)

        ok = True

        for i in range(0, n):

            if Ax[i] > b[i]+b[i]\*err or Ax[i] < b[i]-b[i]\*err:

                ok = False

        if not ok:

        #Realiza outra iteração

            return jacobi\_method(A, xf, b, n, iteration=iteration+1)

        else:

        #Erro aceitável

            return xf

#Matriz de coeficientes

A = np.array([[3, -0.1, -0.2],

               [0.1, 7, -0.3],

               [0.3, -0.2, 10]])

#Matriz de variaveis

x = np.array([[0],  #x1

              [0],  #x2

              [0]]) #x3

#Matriz de constantes

b = np.array([[7.85],

              [-19.3],

              [71.4]])

resposta = jacobi\_method(A, x, b)

Método de Gauss-Seidel:

Para solução por este método, foi escrito outro programa em python que realiza a implementação. Segue o código:

import numpy as np

import csv

def seidel\_method(A, x, b, err=0.001, prec=3, iteration=0):

    #Calcula o tamanho do sistema

    n = len(A.shape) + 1

    #Mostra 3 casas decimais

    np.set\_printoptions(precision=prec)

    #Monta a matriz trangular superior

    U = np.zeros(shape=(n,n))

    for i in range(0, n):

        for j in range(0, n):

            if j > i:

                U[i][j] = A[i][j]

    #Monta a matriz triangular inferior

    L = A - U

    #Isola as variaveis

    Linv = np.linalg.inv(L)

    Ux = U.dot(x)

    #Resultado = L^-1\*(b - Ux)

    xf = Linv.dot(b - Ux)

    #CSV para análise dos resultados

    with open('seidel.csv', 'a', newline='\n') as csvfile:

        writter = csv.writer(csvfile)

        cols = [iteration, xf[0][0], xf[1][0], xf[2][0]]

        writter.writerow(cols)

    print("Valor de Xf")

    print(xf)

    #Análise de convergência

    conv = True

    for i in range(0, n):

        if round(xf[i][0], prec) != round(x[i][0], prec):

            conv = False

    if conv:

        return xf

    else:

        #Caso não convirja, é feita a análise de erro

        Ax = A.dot(x)

        ok = True

        for i in range(0, n):

            if Ax[i] > b[i]+b[i]\*err or Ax[i] < b[i]-b[i]\*err:

                ok = False

        if not ok:

        #Realiza outra iteração

            return seidel\_method(A, xf, b, iteration=iteration+1)

        else:

        #Erro aceitável

            return xf

#Matriz de coeficientes

A = np.array([[3, -0.1, -0.2],

               [0.1, 7, -0.3],

               [0.3, -0.2, 10]])

#Matriz de variaveis

x = np.array([[0],  #x1

              [0],  #x2

              [0]]) #x3

#Matriz de constantes

b = np.array([[7.85],

              [-19.3],

              [71.4]])

resposta = seidel\_method(A, x, b)

**Resultados e Análise**

Cada programa ao rodar, gerou um arquivo CSV que possui os dados de x2, x2 e x3 em cada iteração. Ambos programas encontraram os mesmos resultados finais: x1 = 3, x2 = -2,5 e x3 = 7. Seguem os dados gerados ao longo de cara iteração pelo método de Gauss-Jacobi e Gauss-Seidel respectivamente:

0,2.6166666666666663,-2.757142857142857,7.140000000000001

1,3.0007619047619043,-2.4885238095238096,7.006357142857144

2,3.000806349206349,-2.4997384353741494,7.000206666666668

3,3.0000224965986395,-2.5000026621315192,6.999981040816327

4,2.999998647316704,-2.500001133916424,6.999999271859412

0,2.6166666666666663,-2.7945238095238096,7.005609523809524

1,2.9905565079365077,-2.499624684807256,7.00029081106576

2,3.0000318979108087,-2.499987992353051,6.999999283215615

3,3.0000003524692724,-2.500000035754606,6.99999998871083

Os valores de x1, x2 e x3 são representados respectivamente pelas cores azul amarelo e verde.

Pode se observar que o método de Gauss-Seidel se mostrou mais eficiente, ao convergir para o resultado com uma iteração a menos. Porém a diferença não é tão perceptível ao analisar os gráficos de convergência de cada um, que mostra que os dois se comportaram de uma maneira aceitável. Seguem os gráficos dos métodos de Gauss-Jacobi e Gauss-Seidel respectivamente:

