Universidade de São Paulo – USP

Instituto de Física

Cálculo Numérico com Aplicações em Física - EP2

Aluno: Raphael Rolim
Professor: Arnaldo Gammal

Setembro/2024

Conteúdo

Preâmbulo

```python
import numpy as np
import pandas as pd
```

Funções

```python
def root_by_GE(matrix):
    n = len(matrix)
    matrix = matrix.astype(float)

    for i in range(n):
        max_row = i + np.argmax(abs(matrix[i:, i]))
        if abs(matrix[max_row][i]) < 1e-12:
            raise ValueError('Matriz singular, não há solução única.')
        if i != max_row:
            matrix[[i, max_row]] = matrix[[max_row, i]]
            print(f'Pivotamento: Trocando linha {i+1} com linha {max_row+1}')
            print(matrix, '\n')
        pivot = matrix[i, i]
        matrix[i] /= pivot
        print(f'Normalizando linha {i+1} por {pivot}')
        print(matrix, '\n')
        for k in range(i + 1, n):
            factor = matrix[k, i]
            matrix[k] -= factor * matrix[i]
            print(f'Eliminando elemento abaixo do pivô na linha {k+1}')
            print(matrix, '\n')

    for i in range(n-1, -1, -1):
        for k in range(i - 1, -1, -1):
            factor = matrix[k, i]
            matrix[k] -= factor * matrix[i]
            print(f'Eliminando elemento acima do pivô na linha {k+1}')
            print(matrix, '\n')

    solution = matrix[:, -1]
    return matrix, solution

def root_by_jacobi(matrix, b, x0, error_acceptance, iterations=10, save_data=False, data_name='jacobi_data'):
    x = np.zeros(len(matrix))
    data = {'k': [], 'x': [], 'error': []}
    for it in range(iterations):
        x0 = np.copy(x)
        for i in range(len(x)):
            prod = 0
            for j in range(len(x)):
                if j != i:
                    prod += matrix[i][j] * x0[j]
            x[i] = (b[i] - prod) / matrix[i][i]

        epsilon = np.max(np.abs(x-x0))

        data['k'].append(it+1)
        data['x'].append(np.round(np.copy(x), 4))
```

```python
        data['error'].append(round(epsilon, 4))

        if epsilon < error_acceptance:
            break

    df = pd.DataFrame(data=data)
    if save_data:
        df.to_csv(f'{data_name}.csv', index=False)

    return x, df

def root_by_GS(matrix, b, x, error_acceptance, iterations=10, save_data=False, data_name='gauss_seidel_data'):
    size = len(matrix)
    beta = np.ones(size)
    new_matrix = np.zeros((size, size))
    data = {'k': [], 'x': [], 'error': []}

    for i in range(size):
        max_location = np.argmax(np.abs(matrix[i]))
        new_matrix[i] = matrix[max_location]

    for i in range(size):
        for j in range(size):
            if j != i:
                beta[i] += beta[j] * np.abs(new_matrix[i][j]) / np.abs(new_matrix[i][i])
        beta[i] -= 1

    if np.max(beta) >= 1:
        raise ValueError('Matriz não satisfaz o critério de Sassenfeld.')

    for it in range(iterations):
        x_old = np.copy(x)
        for i in range(size):
            prod = 0
            for j in range(size):
                if j != i:
                    prod += matrix[i][j] * x[j]
            x[i] = (b[i] - prod) / matrix[i][i]

        epsilon = np.max(np.abs(x - x_old))

        data['k'].append(it+1)
        data['x'].append(np.round(np.copy(x), 4))
        data['error'].append(round(epsilon, 4))

        if epsilon < error_acceptance:
            break

    df = pd.DataFrame(data=data)
    if save_data:
        df.to_csv(f'{data_name}.csv', index=False)

    return x, df
```

Item a)

Para a identificação dos nós, podemos perceber que:

1. I_1 flui através de R_1 da esquerda para a direita;
2. I_2 flui através de R_3 da direita para a esquerda;
3. I_3 flui através de R_4 para cima;

sendo assim, no nó entre R_1, R_3 e R_4, temos que

$$I_1 = I_2 + I_3 \rightarrow I_1 - I_2 - I_3 = 0. \tag{1}$$

A partir das Leis das Tensões de Kirchhoff, obtemos

$$-I_2R_3 - U_3 + U_2 + I_3R_4 = 0 \rightarrow 0 \cdot I_1 + R_3I_2 - R_4I_3 = U_2 - U_3, \tag{2}$$

e também

$$-I_1R_1 - I_3R_4 - U_2 - I_1R_2 + U_1 = 0 \rightarrow (R_1 + R_2)I_1 + 0 \cdot I_2 + R_4I_3 = U_1 - U_2. \tag{3}$$

Sabemos que $R_1 = 4.7\ \Omega$, $R_2 = 7.2\ \Omega$, $R_3 = 5.3\ \Omega$, $R_4 = 1.8\ \Omega$ $U_1 = 24.0$ V, $U_2 = 9.0$ V e $U_3 = 5.9$ V, logo, substituindo os valores em 2 e 3, temos

$$\begin{cases} 0 \cdot I_1 + 5.3I_2 - 1.8I_3 = 3.1 \\ 11.9I_1 + 0 \cdot I_2 + 1.8I_3 = 15 \\ I_1 - I_2 - I_3 = 0 \end{cases} \tag{4}$$

Podemos também transformar as desigulades lineares de 4 em um produto de matrizes

$$\begin{bmatrix} 0.0 & 5.3 & -1.8 \\ 11.9 & 0.0 & 1.8 \\ 1.0 & -1.0 & -1.0 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 3.1 \\ 15.0 \\ 0.0 \end{bmatrix}. \tag{5}$$

Item b)

```
matrix = np.array([[0, 5.3, -1.8, 3.1], [11.9, 0, 1.8, 15], [1, -1, -1, 0]])
final_matrix, solution = root_by_GE(matrix)
```

A partir do método de Eliminação de Gauss, é possível encontrar a solução do sistema 5 utilizando a forma matricial

$$\left[\begin{array}{ccc|c} 0.0 & 5.3 & -1.8 & 3.1 \\ 11.9 & 0.0 & 1.8 & 15.0 \\ 1 & -1 & -1 & 0.0 \end{array}\right]. \tag{6}$$

A saída do código rodado é

```
Pivotamento: Trocando linha 1 com linha 2
[[11.9  0.   1.8 15. ]
 [ 0.   5.3 -1.8  3.1]
 [ 1.  -1.  -1.   0. ]]

Normalizando linha 1 por 11.9
[[ 1.          0.          0.1512605   1.2605042]
 [ 0.          5.3        -1.8         3.1      ]
 [ 1.         -1.         -1.          0.       ]]

Eliminando elemento abaixo do pivô na linha 2
[[ 1.          0.          0.1512605   1.2605042]
 [ 0.          5.3        -1.8         3.1      ]
 [ 1.         -1.         -1.          0.       ]]

Eliminando elemento abaixo do pivô na linha 3
[[ 1.          0.          0.1512605   1.2605042]
 [ 0.          5.3        -1.8         3.1      ]
 [ 0.         -1.         -1.1512605  -1.2605042]]

Normalizando linha 2 por 5.3
[[ 1.           0.           0.1512605    1.2605042 ]
 [ 0.           1.          -0.33962264   0.58490566]
 [ 0.          -1.          -1.1512605   -1.2605042 ]]

Eliminando elemento abaixo do pivô na linha 3
[[ 1.           0.           0.1512605    1.2605042 ]
 [ 0.           1.          -0.33962264   0.58490566]
 [ 0.           0.          -1.49088315  -0.67559854]]

Normalizando linha 3 por -1.4908831457111145
[[ 1.           0.           0.1512605    1.2605042 ]
 [ 0.           1.          -0.33962264   0.58490566]
 [-0.          -0.           1.           0.45315325]]

Eliminando elemento acima do pivô na linha 2
[[ 1.           0.           0.1512605    1.2605042 ]
 [ 0.           1.           0.           0.73880676]
 [-0.          -0.           1.           0.45315325]]

Eliminando elemento acima do pivô na linha 1
[[ 1.           0.           0.           1.19196001]
 [ 0.           1.           0.           0.73880676]
```

```
 [-0.          -0.          1.          0.45315325]]

Eliminando elemento acima do pivô na linha 1
[[ 1.          0.          0.          1.19196001]
 [ 0.          1.          0.          0.73880676]
 [-0.          -0.          1.          0.45315325]]
```

Onde a solução final para o problema é

$$\begin{cases} I_1 = 1.19196001 \text{ A} \\ I_2 = 0.73880676 \text{ A} \\ I_3 = 0.45315325 \text{ A} \end{cases} \tag{7}$$

Item c)

```
jacobi_matrix = np.array([[11.9, 0.0, 1.8], [0.0, 5.3, -1.8], [1, -1, -1]])
jacobi_b = np.array([15.0, 3.1, 0.0])
jacobi_x0 = np.array([1, 1, 1])

jacobi_solution, jacobi_data = root_by_jacobi(jacobi_matrix, jacobi_b, jacobi_x0, 1e-3, iterations
    =50, save_data=True)
```

Permutando as linhas 1 e 2 da relação 5, obtemos

$$\begin{bmatrix} 11.9 & 0.0 & 1.8 \\ 0.0 & 5.3 & -1.8 \\ 1.0 & -1.0 & -1.0 \end{bmatrix} \begin{bmatrix} I_2 \\ I_1 \\ I_3 \end{bmatrix} = \begin{bmatrix} 15.0 \\ 3.1 \\ 0.0 \end{bmatrix}. \tag{8}$$

Utilizando o método de Jacobi para encontrar os valores de I_1, I_2 e I_3 para um $\epsilon = 10^{-3}$ e chute inicial de $x_0 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$, obtemos os valores da Tabela 1.

Tabela 1: Tabela com os valores do item c). k é o número de iterações e o erro é a relação $\max |I_i^{(k+1)} - I_i^{(k)}|$.

k	$I^k = \begin{bmatrix} I_1^k & I_2^k & I_3^k \end{bmatrix}$	Erro
1	[1.2605 0.5849 0.0000]	1.2605
2	[1.2605 0.5849 0.6756]	0.6756
3	[1.1583 0.8144 0.6756]	0.2294
4	[1.1583 0.8144 0.3440]	0.3316
5	[1.2085 0.7017 0.3440]	0.1126
6	[1.2085 0.7017 0.5068]	0.1628
7	[1.1839 0.7570 0.5068]	0.0553
8	[1.1839 0.7570 0.4268]	0.0799
9	[1.1959 0.7299 0.4268]	0.0271
10	[1.1959 0.7299 0.4661]	0.0392
11	[1.1900 0.7432 0.4661]	0.0133
12	[1.1900 0.7432 0.4468]	0.0193
13	[1.1929 0.7367 0.4468]	0.0065
14	[1.1929 0.7367 0.4563]	0.0095
15	[1.1915 0.7399 0.4563]	0.0032
16	[1.1915 0.7399 0.4516]	0.0046
17	[1.1922 0.7383 0.4516]	0.0016
18	[1.1922 0.7383 0.4539]	0.0023
19	[1.1918 0.7391 0.4539]	0.0008

Onde a solução para o problema final é

$$\begin{cases} I_1 = 1.19184657 \text{ A} \\ I_2 = 0.73906147 \text{ A} \\ I_3 = 0.45390323 \text{ A} \end{cases} \tag{9}$$

Item d)

Temos que, para a relação 8 A matriz $\mathbb{J}$ se dá por

$$\mathbb{J} = -\begin{bmatrix} 0 & \frac{0}{11.9} & \frac{1.8}{11.9} \\ \frac{0}{5.3} & 0 & -\frac{1.8}{5.3} \\ -\frac{1}{1} & \frac{1}{1} & 0 \end{bmatrix} \approx -\begin{bmatrix} 0.0 & 0.0 & 0.15 \\ 0.0 & 0.0 & -0.34 \\ -1.0 & 1.0 & 0.0 \end{bmatrix} \tag{10}$$

Os autovalores da matriz $\mathbb{J}$ são

$$\begin{cases} \lambda_1 = 0 \\ \lambda_2 = 0.7i \\ \lambda_3 = -0.7i \end{cases} \tag{11}$$

Como o ρ_s é o módulo do maior autovalor, então $\rho_s = 0.7$, já que $|i| = 1$. Para $p = 10^{-3}$, temos que

$$k \approx \frac{p \ln 10}{\ln \rho_s} = \frac{10^{-3} \ln 10}{\ln 0.7} \approx -0.0065. \tag{12}$$

Para verificar a convergência, precisamos que

$$\rho_s^k \approx 10^{-p} \to 0.7^{-0.0065} \approx 10^{-10^{-3}} \to 0.99 \approx 1.00 \blacksquare \tag{13}$$

Item e)

```
gs_matrix = np.array([[11.9, 0.0, 1.8], [0.0, 5.3, -1.8], [1, -1, -1]])
gs_b = np.array([15.0, 3.1, 0.0])
gs_x0 = np.array([1, 1, 0.5])

gs_solution, gs_data = root_by_GS(gs_matrix, gs_b, gs_x0, 1e-3, iterations=50, save_data=True)
```

Agora, utilizando o método de Gauss-Seidel para encontrar os valores de I_1, I_2 e I_3 para um $\epsilon = 10^{-3}$ e chute inicial de $x_0 = \begin{bmatrix} 1 & 1 & 0.5 \end{bmatrix}$, obtemos os valores da Tabela 2.

Tabela 2: Tabela com os valores do item c). k é o número de iterações e o erro é a relação $\max |I_i^{(k+1)} - I_i^{(k)}|$.

k	$I^k = \begin{bmatrix} I_1^k & I_2^k & I_3^k \end{bmatrix}$	Erro
1	[1.1849 0.7547 0.4302]	0.2453
2	[1.1954 0.7310 0.4644]	0.0343
3	[1.1903 0.7426 0.4476]	0.0168
4	[1.1928 0.7369 0.4559]	0.0083
5	[1.1915 0.7397 0.4518]	0.0041
6	[1.1922 0.7384 0.4538]	0.0020
7	[1.1919 0.7390 0.4528]	0.0010

Onde a solução para o problema final é

$$\begin{cases} I_1 = 1.19186087 \text{ A} \\ I_2 = 0.73902937 \text{ A} \\ I_3 = 0.45283149 \text{ A} \end{cases} \tag{14}$$