Universidade Federal da Bahia
Programa de Pós-Graduação em Ciência de Dados e Big Data
Carlos Magno Santos Ribeiro de Brito

---

Prof Dra. Majela Penton Machado

**Respostas da segunda lista de exercício**

Questão 01) Determine as condições que as constantes $b_1$, $b_2$, $b_3$ devem satisfazer para garantir a consistência do sistema linear.

$$\begin{cases} x - 2y + 5z = b_1 \\ 4x - 5y + 8z = b_2 \\ -3x + 3y - 3z = b_3 \end{cases}$$

Duas perguntas têm que ser feitas para essa verificação:

(a) É um sistema possível de ser resolvido?

(b) É um sistema possível determinada/única ou indeterminadas soluções?

Para responder essas questões, primeiramente devemos encontrar a **matriz escalonada**, por meio de operações elementares na matriz dos coeficientes.

Usando eliminação de gauss, tem-se:

$$M = \begin{bmatrix} 1 & -2 & 5 \\ 4 & -5 & 8 \\ -3 & 3 & -3 \end{bmatrix} R_2 - 4R_1 \to R_2 \begin{bmatrix} 1 & -2 & 5 \\ 0 & 3 & -12 \\ -3 & 3 & -3 \end{bmatrix} R_3 - (-3)R_1 \to R_3 \begin{bmatrix} 1 & -2 & 5 \\ 0 & 3 & -12 \\ 0 & -3 & 12 \end{bmatrix}$$

$$R_3 - (-1)R_2 \to R_3 \begin{bmatrix} 1 & -2 & 5 \\ 0 & 3 & -12 \\ 0 & 0 & 0 \end{bmatrix}$$

A partir disso, obtemos o seu posto, isto é, o número de linhas não-nulas quando a matriz está escalonada $p(M) = 2$. Ainda, para a matriz aumentada teremos que a única condição de consistência é caso o seu posto seja também 2. Nessa situação, o sistema será possível e com infinitas soluções, já que o número de variáveis é igual a 3 e $p(M) = p(M|B) < n$. Caso tenhamos $p(M|B) = 3$ o sistema será inconsistente. Para a consistência, temos:

$$(M|B) = \left[\begin{array}{ccc|c} 1 & -2 & 5 & b_1 \\ 0 & 3 & -12 & -4b_1 + b_2 \\ 0 & 0 & 0 & -b_1 + b_2 + b_3 \end{array}\right]$$

Onde devemos atender na terceira linha $-b_1 + b_2 + b_3 = 0$ para que o sistema seja possível e indeterminado.

Questão 02) Usando o método de Gauss-Jordan resolva o sistema de equações lineares.

$$\begin{cases} x_1 + 2x_2 - 3x_4 + x_5 = 2 \\ x_1 + 2x_2 + x_3 - 3x_4 + x_5 + 2x_6 = 3 \\ x_1 + 2x_2 - 3x_4 + 2x_5 + x_6 = 4 \\ 3x_1 + 6x_2 + x_3 - 9x_4 + 4x_5 + 3x_6 = 9 \end{cases}$$

No processo de eliminação gaussiana, encontra-se a matriz escalonada a partir da matriz aumentada:

$$\left[\begin{array}{cccccc|c} 1 & 2 & 0 & -3 & 1 & 0 & 2 \\ 1 & 2 & 1 & -3 & 1 & 2 & 3 \\ 1 & 2 & 0 & -3 & 2 & 1 & 4 \\ 3 & 6 & 1 & -9 & 4 & 3 & 9 \end{array}\right] R_2 - R_1 \to R_2 \left[\begin{array}{cccccc|c} 1 & 2 & 0 & -3 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 1 & 2 & 0 & -3 & 2 & 1 & 4 \\ 3 & 6 & 1 & -9 & 4 & 3 & 9 \end{array}\right] R_3 - R_1 \to R_3$$

$$\left[\begin{array}{cccccc|c} 1 & 2 & 0 & -3 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 3 & 6 & 1 & -9 & 4 & 3 & 9 \end{array}\right] R_4 - 3R_1 \to R_4 \left[\begin{array}{cccccc|c} 1 & 2 & 0 & -3 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 & 1 & 3 & 3 \end{array}\right] R_4 - R_2 \to R_4 \left[\begin{array}{cccccc|c} 1 & 2 & 0 & -3 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \end{array}\right]$$

$$R_4 - R_3 \to R_4 \left[\begin{array}{cccccc|c} 1 & 2 & 0 & -3 & 1 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] R_1 - R_3 \to R_1 \left[\begin{array}{cccccc|c} 1 & 2 & 0 & -3 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Essa matriz escalonada resulta em um sistema de equações lineares:

$$\begin{cases} x_1 + 2x_2 - 3x_4 - x_6 = 0 \\ x_3 + 2x_6 = 1 \\ x_5 + x_6 = 2 \end{cases}$$

Pela matriz aumentada escalonada é possível notar que seu posto é igual a $p(M|B) = 3$. Sabe-se também que o posto da matriz dos coeficientes é igual a $p(M) = 3$. Com base nisso e tendo como número de variáveis $n = 6$, podemos afirmar que o sistema é possível, mas indeterminado. Ou seja, apresenta infinitas soluções.

$$\begin{cases} x_1 = -2x_2 + 3x_4 + x_6 \\ x_2 = x_2 \\ x_3 = 1 - 2x_6 \\ x_4 = x_4 \\ x_5 = 2 - x_6 \\ x_6 = x_6 \end{cases} \tag{1}$$

Para a resolução desse sistema de equações lineares com python, é retornada uma exceção, pois a matriz dos coeficientes não é quadrada e temos o seguinte código:

```
import numpy as np

    # Seta precisao de 3 decimais
    np.set_printoptions(precision=3, suppress=True)

    #Matriz dos coeficientes
    A = np.array([
```

```
    [1, 2, 0, -3, 1, 0],
    [1, 2, 1, -3, 1, 2],
    [1, 2, 0, -3, 2, 1],
    [3, 6, 1, -9, 4, 3]
    ])

    #Vetor com valores atribuidos
    B = np.array([2,3,4,9])

    #Resolucao do sistema
    x = np.linalg.solve(A, B)

    print(x)

# raise LinAlgError('Last 2 dimensions of the array must be square')
# numpy.linalg.LinAlgError: Last 2 dimensions of the array must be
# square
```

Questão 03) A Regra de Cramer é um método para determinar a solução de um sistema linear $Ax = b$ de n equações e n incógnitas quando a matriz dos coeficientes A é invertível.

**Regra de Cramer**: Se $Ax = b$ é um sistema de n equações lineares em n incógnitas tal que $det(A) \neq 0$, então a única solução do sistema é dada por:

$$x_1 = \frac{det(A_1)}{det(A)}, \ x_2 = \frac{det(A_2)}{det(A)}, \ \ldots, \ x_n = \frac{det(A_n)}{det(A)}$$

onde $A_j$ é a matriz obtida substituindo as entradas da $j-$ésima coluna de $A$ pelas entradas da matriz $b$.

a) Mostre a Regra de Cramer no caso em que $A$ é uma matriz quadrada de ordem 2.

Tem-se então:

$$\begin{cases} a_1x_1 + a_2x_2 = b_1 \\ a_3x_1 + a_4x_2 = b_2 \end{cases}$$

Onde podemos obter a matriz dos coeficientes:

$$A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

o determinante de $A$ é igual a:

$$\det(A) = \begin{vmatrix} a_1 & a_2 \\ a_3 & a_4 \end{vmatrix} = -a_2 \cdot a_3 + a_1 \cdot a_4$$

Os determinantes de $A_1$ e $A_2$ são iguais a:

$$\det(A_1) = \begin{vmatrix} b_1 & a_2 \\ b_2 & a_4 \end{vmatrix} = a_4 \cdot b_1 - a_2 \cdot b_2$$

$$\det(A_2) = \begin{vmatrix} a_1 & b_1 \\ a_3 & b_2 \end{vmatrix} = -a_3 \cdot b_1 + a_1 \cdot b_2$$

Obtem-se então a solução do sistema:

$$\begin{aligned} x_1 &= \frac{-a_4 \cdot b_1 + a_2 \cdot b_2}{a_2 \cdot a_3 - a_1 \cdot a_4} \\ x_2 &= \frac{a_3 \cdot b_1 - a_1 \cdot b_2}{a_2 \cdot a_3 - a_1 \cdot a_4} \end{aligned} \tag{2}$$

b) Use a Regra de Cramer para resolver $x'$ e $y'$ em termos de $x$ e $y$

$$\begin{cases} x = \frac{3}{5}x' - \frac{4}{5}y' \\ y = \frac{4}{5}x' + \frac{3}{5}y' \end{cases}$$

A matriz dos coeficientes:

$$A = \begin{bmatrix} \frac{3}{5} & \frac{-4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix}$$

$$det(A) = 1$$

$$\det(A_1) = \begin{vmatrix} x & -4/5 \\ y & 3/5 \end{vmatrix} = \frac{3x + 4y}{5}$$

$$\det(A_2) = \begin{vmatrix} 3/5 & x \\ 4/5 & y \end{vmatrix} = \frac{-4x + 3y}{5}$$

Assim sendo, a solução do sistema é:

$$\begin{aligned} x' &= \frac{3x + 4y}{5}/1 = \frac{3x + 4y}{5} \\ y' &= \frac{-4x + 3y}{5}/1 = \frac{-4x + 3y}{5} \end{aligned} \tag{3}$$

Questão 04) Considere o seguinte sistema de equações lineares

$$\begin{bmatrix} 2 & -4 & 0 & 0 \\ 1 & 2 & 12 & 0 \\ 0 & -1 & -4 & -5 \\ 0 & 0 & 2 & 11 \end{bmatrix} \cdot \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 8 \\ 0 \\ 1 \\ 0 \end{Bmatrix}$$

a) Use o Python para determinar uma decomposição LU da matriz dos coeficientes deste sistema.

```
import numpy as np

# Algoritmo para decomposicao LU (scipy apenas resolve com
# pivotacao)
def fatoraLU(A):
```

```
    U = np.copy(A)
    n = len(U)
    L = np.eye(n)
    for j in np.arange(n-1):
        for i in np.arange(j+1,n):
            L[i,j] = U[i,j]/U[j,j]
            for k in np.arange(j+1,n):
                U[i,k] = U[i,k] - L[i,j]*U[j,k]
            U[i,j] = 0
    return L, U

A = np.array([[2,-4,0,0],[1,2,12,0],[0,-1,-4,-5],[0,0,2,11]])

# realiza decomposicao
l, u = fatoraLU(A)
# imprime valores
print(l)
# [[ 1.    0.    0.    0.  ]
#  [ 0.5   1.    0.    0.  ]
#  [ 0.   -0.25  1.    0.  ]
#  [ 0.    0.   -2.    1.  ]]
print(u)
# [[ 2 -4  0  0]
#  [ 0  4 12  0]
#  [ 0  0 -1 -5]
#  [ 0  0  0  1]]
```

b) Use a decomposição LU para resolver o sistema.

Usando os resultados anteriores obtidos pelo Python para L e U, podemos resolver o sistem como:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 \\ 0 & -1/4 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & -4 & 0 & 0 \\ 0 & 4 & 12 & 0 \\ 0 & 0 & -1 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Completada a fatoração LU, resolvemos, primeiramente, o sistema $Ly = b$:

$$
\begin{aligned}
y_1 &= 8 \\
\frac{1}{2}y_1 + y_2 &= 0 \\
-\frac{1}{4}y_2 + y_3 &= 1 \\
-2y_3 + y_4 &= 0
\end{aligned}
$$

Encontrando os seguintes valores:

$$
\begin{cases}
y_1 = 8 \\
y_2 = -4 \\
y_3 = 0 \\
y_4 = 0
\end{cases}
$$

Esses valores serão usados para resolver o sistema $Ux = y$, encontrando assim um *array* com os valores de $x$:

$$
\begin{aligned}
2x_1 - 4x_2 &= 8 \\
4x_2 + 12x_3 &= -4 \\
-x_3 - 5x_4 &= 0 \\
x_4 &= 0
\end{aligned}
$$

$$
x = \begin{bmatrix} 2 & -1 & 0 & 0 \end{bmatrix} \tag{4}
$$

Questão 05) Dado $n$ um número natural maior ou igual a 2, considere a matriz $A$ de tamanho $n \times n$ cuja entrada na posição $i, j$ é $maxi, j$. Na seguinte função foi implementado o método de Gauss-Jordan para resolver sistemas lineares cuja matriz dos coeficientes é uma matriz $A$ da forma anterior e os termos independentes são matrizes colunas com entradas inteiras quaisquer.

```
import numpy as np


def GaussJordan(A, b):
```

```
    b = np.array(b, dtype=float)
    A = np.array(A, dtype=float)
    Ab = np.column_stack((A, b))
    n = np.shape(Ab)[0]
    m = np.shape(Ab)[1]
    for i in range(0, n - 1, 1):
        if Ab[i, i] == 0:
            print("Divisao por zero")
            break
        for k in range(i + 1, n, 1):
            fator = Ab[k, i] / Ab[i, i]
            Ab[k, :] = Ab[k, :] - Ab[i, :] * fator
    for i in range(n - 1, -1, -1):
        for k in range(i - 1, -1, -1):
            fator = Ab[k, i] / Ab[i, i]
            Ab[k, :] = Ab[k, :] - Ab[i, :] * fator
        Ab[i, :] = Ab[i, :] / Ab[i, i]
    x = np.copy(Ab[:, m - 1])
    return x
```

Usando Python defina matrizes da forma acima e matrizes colunas $b$ com entradas inteiras aleatórias de tamanhos 5, 10, 15, 50, 100, 500, 1000 (especifique os comandos utilizados). Use as seguintes linhas para comparar o tempo de execução da função Gauss-Jordan (acima) e da função do numpy (linalg.solve) ao resolver cada sistema linear $AX = b$ definido anteriormente

```
inicio = time.time()
xgauss = GaussJordan(A,b)
fim = time.time()
print("Tempo de Gauss-Jordan:",fim - inicio)
inicio = time.time()
xnump = np.linalg.solve(A,b)
fim = time.time()
print("Tempo de numpy-solve:",fim - inicio)
```

Qual a conclusão que você pode tirar deste experimento?

Inicialmente, comparando os tempos de execução para a questão anterior, percebeu-se que o tempo de execução do numpy é menor que o de Gauss-Jordan, porém, há uma oscilação: vezes a diferença é bastante superior, por outras não muito. No geral, o método do numpy apresenta uma tendencia de sempre permanecer com tempos menores na execução. Isso possivelmente se deve a algoritmos otimizados para resolução interna do sistema, que apresentam um big O mais rápido.

Para testes em matrizes de tamanho $2 \times 2$ até $1000 \times 1000$ elaborou-se um gráfico *tempo* $\times$ *passoiteração* com o seguinte algoritmo e percebeu-se que o tempo de execução

do numpy continua sendo menor que o de Gauss-Jordan.

```
import numpy as np
import time as time
import matplotlib.pyplot as plt

initial = 2
final = 1000
times_gauss = []
times_numpy = []
for i in range(initial, final):
    A = np.random.randint(-5, 5, size=(i, i))
    b = np.random.randint(-5, 5, size=(i))
    if np.linalg.det(A) != 0:
        inicio = time.time()
        xgauss = GaussJordan(A, b)
        fim = time.time()
        times_gauss.append((fim-inicio)*1000) #tempo em s
        inicio = time.time()
        xnump = np.linalg.solve(A, b)
        fim = time.time()
        times_numpy.append((fim-inicio)*1000) #tempo em s

plt.title("Tempo de execucao")
plt.xlabel("Passo")
plt.ylabel("Tempo (s)")

df={'x_values': np.arange(initial, final), 'y1_values': times_gauss,
'y2_values': times_numpy }

plt.plot( 'x_values', 'y1_values', data=df, marker='', color='olive',
linewidth=2, label="gauss")
plt.plot( 'x_values', 'y2_values', data=df, marker='', color='blue',
linewidth=2, linestyle='dashed', label="numpy")
# show legend
plt.legend()

plt.show()h
```