

# Algebra Linear Computacional COC473 - Lista 1

Bruno Dantas de Paiva  
DRE: 118048097

September 24, 2020

## 1 Questão 1

$$\text{Step I: } \left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ -4 & 6 & -4 & 1 & 2 \\ 1 & -4 & 6 & -4 & 1 \\ 0 & 1 & -4 & 5 & 3 \end{array} \right] \xrightarrow{\substack{L_2 \leftarrow L_2 + \frac{4}{5}L_1 \\ L_3 \leftarrow L_3 - \frac{1}{5}L_1}} \left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ 0 & -14/5 & -16/5 & 1 & 6/5 \\ 0 & -16/5 & 29/5 & -4 & 6/5 \\ 0 & 1 & -4 & 5 & 3 \end{array} \right]$$

$$\text{Step II: } \left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ 0 & -14/5 & -16/5 & 1 & 6/5 \\ 0 & -16/5 & 29/5 & -4 & 6/5 \\ 0 & 1 & -4 & 5 & 3 \end{array} \right] \xrightarrow{\substack{L_3 \leftarrow L_3 + 8/7L_2 \\ L_4 \leftarrow L_4 - 5/14L_2}} \left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ 0 & 14/5 & -16/5 & 1 & 6/5 \\ 0 & 0 & 15/7 & -20/7 & 18/7 \\ 0 & 0 & -20/7 & 65/14 & 18/7 \end{array} \right]$$

$$\text{Step III: } \left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ 0 & 14/5 & -16/5 & 1 & 6/5 \\ 0 & 0 & 15/7 & -20/7 & 18/7 \\ 0 & 0 & -20/7 & 65/14 & 18/7 \end{array} \right] \xrightarrow{L_4 \leftarrow L_4 + 4/3L_3} \left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ 0 & 14/5 & -16/5 & 1 & 6/5 \\ 0 & 0 & 15/7 & -20/7 & 18/7 \\ 0 & 0 & 0 & 5/6 & 6 \end{array} \right]$$

$$\text{Step IV: } \left\{ \begin{array}{l} 5x_1 - 4x_2 + x_3 = -1 \Rightarrow x_1 = 29/5, \\ 14/5x_2 - 16/5x_3 + x_4 = 6/5 \Rightarrow x_2 = 51/5, \\ 15/7x_3 - 20/7x_4 = 18/7 \Rightarrow x_3 = 54/5, \\ 0 + 5/6x_4 = 6 \Rightarrow x_4 = 36/5. \end{array} \right.$$

$$\frac{1}{5} [29, 51, 54, 36].$$

Figure 1: Imagem contendo os cálculos da resolução da matriz A

## 2 Questão 2

### 2.1 Decomposição LU

```
def lu_decomposition(matrix):
    if(Matrix_Utils.matrix_determinant(matrix) == 0):
        return "Error"

    number_of_rows = len(matrix)
    number_of_columns = len(matrix[0])

    if(number_of_rows != number_of_columns):
        return "Error"

    result = copy.deepcopy(matrix)

    for k in range(number_of_rows):
        for i in range(k+1, number_of_rows):
            result[i][k] = float(result[i][k]/result[k][k])

        for j in range(k+1, number_of_columns):
            for i in range(k+1, number_of_columns):
                result[i][j] = float(result[i][j]-result[i][k]*result[k][j])

    return result
```

Para verificar se é possível realizar a decomposição, foi utilizada a rotina de cálculo de determinante onde pode ser observada na alternativa mais abaixo.

## 2.2 Decomposição Cholesky

```

def cholesky_decomposition(matrix):
    number_of_rows      = len(matrix)
    number_of_columns   = len(matrix[0])

    if(number_of_rows != number_of_columns):
        return "Error"

    if(not Matrix_Utils.positive_definite(matrix)):
        return "Error"

    l = [[0.0] * len(matrix) for _ in range(len(matrix))]

    for i in range(len(matrix)):
        for j in range(i + 1):

            if(i==j):
                summation = sum(l[i][k]**2 for k in range(i))
                l[i][i]   = (matrix[i][i]-summation)**0.5
                continue;

            summation = sum(l[i][k]*l[j][k] for k in range(i))
            l[i][j]   = (1.0/l[j][j])*(matrix[i][j]-summation)

    return l

```

Para verificar se é possível realizar a decomposição, foram utilizadas duas rotinas: uma para verificar a simetria da matriz e outra para determinar se a matriz é definida positiva, onde foi utilizada o critério de silvester, como podemos observar abaixo:

```

def positive_definite(matrix):
    if(not check_simetry(matrix)):
        return False
    if(not sylvesters_criterion(matrix)):
        return False
    return True

def check_simetry(matrix):
    number_of_rows      = len(matrix)
    number_of_columns   = len(matrix[0])

    if(number_of_rows != number_of_columns):
        return -1

    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if(matrix[i][j]!= matrix[j][i]):
                return False
    return True

```

```
def sylvesters_criterion(matrix):
    for i in range(len(matrix)):
        secondary = get_main_minor(matrix, i)
        if(matrix_determinant(secondary) <= 0):
            return False

    return True

def get_main_minor(matrix, index):
    result = [[matrix[row][column] for row in range(index + 1)] for column in range(index + 1)]
    return result
```

## 2.3 Resolver o sistema AX=B

Para resolver o sistema AX=B foram criadas duas rotinas, onde é possível utilizar tanto a decomposição LU quanto a decomposição Cholesky. Além disso, foi necessário implementar as substituições pra trás e pra frente, a fim de auxiliar a obtenção do X, como podemos observar abaixo:

```
def solve_matrix(matrix_a, matrix_b, use_cholesky):
    matrix_lu = copy.deepcopy(matrix_a)
    if(not use_cholesky):
        matrix_lu = lu_decomposition(matrix_lu)
        if(matrix_lu == "Error"):
            return "Error"

        matrix_y = Matrix_Utils.forward_substitution(matrix_lu, matrix_b)
        return Matrix_Utils.backward_substitution(matrix_lu, matrix_y)

    matrix_lu = cholesky_decomposition(matrix_lu)
    if(matrix_lu == "Error"):
        return "Error"

    matrix_y = Matrix_Utils.forward_substitution(matrix_lu, matrix_b, True)
    return Matrix_Utils.backward_substitution(Matrix_Utils.get_transposed_matrix(matrix_lu), matrix_b)

def backward_substitution(matrix_u, matrix_y):
    number_of_rows = len(matrix_u)
    matrix_x = [0 for i in range(number_of_rows)]

    matrix_x[number_of_rows-1] = matrix_y[number_of_rows-1]/matrix_u[number_of_rows-1][number_of_rows-1]

    for i in range(number_of_rows-2, -1, -1):
        summation = matrix_y[i]
        for j in range(i+1, number_of_rows):
            summation -= matrix_u[i][j]*matrix_x[j]

        matrix_x[i] = summation/float(matrix_u[i][i])

    return matrix_x
```

```

def forward_substitution(matrix_l, matrix_b, control=False):
    number_of_rows = len(matrix_l)
    matrix_y = [0 for i in range(number_of_rows)]

    if(not control):
        matrix_y[0] = matrix_b[0]
    else:
        matrix_y[0] = matrix_b[0]/matrix_l[0][0]

    for i in range(1, number_of_rows):
        summation = matrix_b[i]
        for j in range(i):
            summation -= matrix_l[i][j]*matrix_y[j]

        if(not control):
            matrix_y[i] = summation
        else:
            matrix_y[i] = summation/matrix_l[i][i]

    return matrix_y

```

Nota-se que a função que executa a substituição para frente possui um parâmetro booleano, que permite que, quando esse parâmetro é true, execute a substituição para frente na decomposição cholesky, caso contrário executa para a decomposição LU.

## 2.4 Calcular o determinante de A

```

def matrix_determinant(matrix):
    number_of_rows = len(matrix)
    number_of_columns = len(matrix[0])

    result = 0

    if(number_of_rows != number_of_columns):
        return "Error"

    if(number_of_rows == 1):
        return matrix[0][0]

    for k in range(number_of_rows):
        result += matrix[k][0]*((-1)**k)*matrix_determinant(get_auxiliar_matrix(matrix, k))

    return result

```

### 3 Questão 3

#### 3.1 Jacobi

```
def iterative_jacobi(matrix_a, matrix_b):
    if (not Matrix_Utils.converge(matrix_a)):
        return "Error"

    n = len(matrix_a)

    solution_zero = [0.0 for i in range(n)]
    next_solution = [0.0 for i in range(n)]

    tol      = 10**(-5)
    residue = 1
    step     = 0

    while (residue > tol):

        numerator   = 0
        denominator = 0

        for j in range(n):
            next_solution[j] = matrix_b[j]

            for k in range(n):
                if (j!=k):
                    next_solution[j] += (-1)*(matrix_a[j][k] * solution_zero[k])

            next_solution[j] /= matrix_a[j][j]

        for z in range(n):
            numerator   += (next_solution[z]-solution_zero[z])**2
            denominator += next_solution[z]**2

        residue = float(numerator**0.5)/(denominator**0.5)

        for i in range(len(next_solution)):
            solution_zero[i] = next_solution[i]
        step+=1

    print("x1: ", next_solution)
    print("residue: ", residue)
    print("iteration_number: ", step)
```

### 3.2 Gauss-Seidel

```
def gauss_seidel(matrix_a, matrix_b):
    if (not Matrix_Utils.converge(matrix_a)):
        if (not Matrix_Utils.positive_definite(matrix_a)):
            return "Error"

    n = len(matrix_a)

    solution_zero = [0.0 for i in range(n)]
    next_solution = [0.0 for i in range(n)]

    tol      = 10**(-5)
    residue = 1
    step    = 0

    while (residue > tol):
        numerator      = 0
        denominator    = 0
        second_summation = 0
        second_summation = 0

        for j in range(n):
            first_summation = Matrix_Utils.sum_of_vector_multiplication(matrix_a[j][:j])
            second_summation = Matrix_Utils.sum_of_vector_multiplication(matrix_a[j][j+1:])
            next_solution[j] = (matrix_b[j] - first_summation - second_summation)/matrix_a[j][j]

        for z in range(n):
            numerator += (next_solution[z] - solution_zero[z])**2
            denominator += next_solution[z]**2

        residue = float(numerator**0.5)/(denominator**0.5)

        for i in range(len(next_solution)):
            solution_zero[i] = next_solution[i]

        step += 1

    print("x1: ", next_solution)
    print("residue: ", residue)
    print("iteration_number: ", step)
```

Nota-se que ambas as funções utilizam as rotinas converge, que definem se a matriz converge ou não. E, o método de Gauss-Seidel também utiliza uma função auxiliar que calcula separadamente os somatórios. A imagem destas funções se encontram abaixo:

```

def converge(matrix):
    for i in range(len(matrix)):
        lines_summation = 0
        columns_summation = 0;
        for j in range(len(matrix)):
            if (i != j):
                lines_summation += math.fabs(matrix[i][j])
                columns_summation += math.fabs(matrix[j][i])

    if (matrix[i][i] < lines_summation or matrix[i][i] < columns_summation):
        return False

    return True

def sum_of_vector_multiplication(matrix_a, matrix_b):
    summation = 0

    for i in range(len(matrix_a)):
        for j in range(len(matrix_b)):
            if (i==j):
                summation += matrix_a[i]*matrix_b[i]

    return summation

```

## 4 Questão 4

### 4.1 Letra A

Resolver o sistema de equações por:

#### 4.1.1 Método de Eliminação de Gauss

The image shows handwritten steps for solving a 4x4 system of linear equations using the method of elimination. It includes the augmented matrix, row operations, and the resulting row echelon form, followed by the system of equations and its solution.

**Augmented Matrix:**

$$\left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ -4 & 6 & -4 & 1 & 2 \\ 1 & -4 & 6 & -4 & 1 \\ 0 & 1 & -4 & 5 & 3 \end{array} \right]$$

**Row Operations:**

- Step 1:  $L_2 \leftarrow 5L_2 + 4L_1$ ;  $L_3 \leftarrow 5L_3 - L_1$ ;  $L_4 \leftarrow 5L_4 + L_1$
- Step 2:  $L_3 \leftarrow 7L_3 + 8L_2$ ;  $L_4 \leftarrow 14L_4 - L_2$
- Step 3:  $L_4 \leftarrow 15L_4 + 8L_3$

**Row Echelon Form:**

$$\left[ \begin{array}{cccc|c} 5 & -4 & 1 & 0 & -1 \\ 0 & 14 & -16 & 5 & 6 \\ 0 & -16 & 29 & -20 & 6 \\ 0 & 0 & -40 & 65 & 36 \end{array} \right]$$

**System of Equations:**

$$\begin{cases} 5x_1 - 4x_2 + x_3 + 0x_4 = -1 \\ 0x_1 + 14x_2 - 16x_3 + 5x_4 = 6 \\ 0x_1 + 0x_2 + 7x_3 + (-100)x_4 = 90 \\ 0x_1 + 0x_2 + 0x_3 + 175x_4 = 1260 \end{cases}$$

**Solution:**

$$\begin{aligned} x_4 &= 1260/175 \Rightarrow 7,2 \\ x_3 &= (90 + 100 \cdot 7,2)/75 \Rightarrow 10,8 \Rightarrow x_2 = 10,2 \\ x_2 &= (6 - 5 \cdot 7,2 + 16 \cdot 10,8)/14 \Rightarrow 10,2 \\ x_1 &= (-10,8 + 4 \cdot 10,2)/5 \Rightarrow 5,8 \end{aligned}$$

Figure 2: Imagem contendo a resolução de  $AX= B$  utilizando o método de eliminação de gauss

#### 4.1.2 Método de Eliminação Gauss-Jordan

The image shows handwritten steps for solving a system of linear equations using the Gauss-Jordan elimination method. It includes several augmented matrices labeled (I) through (VII), showing row operations like L1 ↔ L1/5, L2 ↔ L2 + 4L1, L3 ↔ L3 - L1, L2 ↔ L2/2.8, L3 ↔ L3/2.14, and L4 ↔ L4 + 2.8L3. The final solution is given as  $x \Rightarrow [5, 8, 10, 2, 10, 8, 7, 2]$ .

Figure 3: Imagem contendo a resolução de  $AX = B$  utilizando o método de eliminação de gauss-jordan

#### 4.1.3 Decomposição $A = LU$

The image shows handwritten steps for decomposing matrix  $A$  into  $LU$  components and solving for  $X$ . It includes augmented matrices (I) through (VII) for row operations and a note about pivoting. The final solution is given as  $Y = B$ , Partindo; Resolvendo por Gauss-Jordan.

Figure 4: Imagem contendo a resolução de  $AX = B$  utilizando a decomposição LU - Parte 1

Nota-se que a primeira decomposição, nós obtemos somente a matriz  $L$  e  $U$ , onde estas são utilizadas para obter  $Y$  (um vetor auxiliar necessário para obter a solução particular  $X$ ).

$\text{U } X = Y$ , Portanto, Resolvendo pela eliminação de gauss:

$$\left[ \begin{array}{ccc|c} 5 & -4 & 1 & -1 \\ 0 & 2,8 & -3,2 & 1 \\ 0 & 0 & 2,14 & -2,868 \\ 0 & 0 & 0 & 0,833 \end{array} \right] \quad \begin{aligned} x_4 &= 7,19 & x_4 &= 7,19 \\ x_3 &= (2,57 + 2,868)/2,14 & = 10,8 & \\ x_2 &= (1,2 + 7,19 + 3,2 \times 3) / 2,8 & = 10,2 & \\ x_1 &= (-1 - 10,8 + 4 \times 10,8) / 5 & & \\ x_1 &= 5,83 & & \end{aligned}$$

$$X = [5,83, 10,2, 10,8, 7,19]$$

Figure 5: Imagem contendo a resolução de  $AX = B$  utilizando a decomposição LU - Parte 2

Comparando então os resultados, podemos observar (imagem abaixo) que ambos são iguais, conforme era esperado.

```
bdantas@Oracle:~/Área de Trabalho/ALC_Lists/List_1$ python3 main.py
[5.800000000000009, 10.20000000000014, 10.80000000000011, 7.20000000000005]
```

Figure 6: Imagem contendo os cálculos da determinante da matriz A

#### 4.1.4 Decomposição de Cholesky $A = LL^T$

$$\left[ \begin{array}{ccc|c} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{array} \right] \quad \begin{aligned} L_{1,1}^2 &= 5 & L_{1,1}L_{2,1} &= L_{1,1}L_{3,1} \\ L_{1,1}L_{2,1} &= -4 & L_{1,1}L_{2,1}^2 &= L_{1,1}L_{2,1}L_{3,1} \\ L_{1,1}L_{3,1} &= -4 & L_{1,1}L_{2,1}L_{3,1} &= L_{1,1}L_{2,1}L_{3,1}L_{4,1} \\ L_{1,1}L_{4,1} &= 0 & L_{1,1}L_{2,1}L_{3,1}L_{4,1} &= 0 \\ L_{2,1}^2 &= 6 & L_{2,1}^2 &= L_{2,1}L_{3,1} \\ L_{2,1}L_{3,1} &= -4 & L_{2,1}L_{3,1} &= L_{2,1}L_{3,1}L_{4,1} \\ L_{2,1}L_{4,1} &= 1 & L_{2,1}L_{3,1}L_{4,1} &= L_{2,1}L_{3,1}L_{4,1}L_{4,1} \\ L_{3,1}^2 &= 6 & L_{3,1}^2 &= L_{3,1}L_{4,1} \\ L_{3,1}L_{4,1} &= -4 & L_{3,1}L_{4,1} &= L_{3,1}L_{4,1}L_{4,1} \\ L_{4,1}^2 &= 5 & L_{4,1}^2 &= L_{4,1}L_{4,1} \\ L_{4,1}L_{4,1} &= 5 & L_{4,1}L_{4,1} &= L_{4,1}L_{4,1}L_{4,1} \\ L_{1,1} &= 2,24 & L_{1,1} &= 2,24 \\ L_{2,1} &= 1,79 & L_{2,1} &= 1,79 \\ L_{3,1} &= 1,46 & L_{3,1} &= 1,46 \\ L_{4,1} &= 0,913 & L_{4,1} &= 0,913 \end{aligned}$$

$$L = \begin{bmatrix} 2,24 & 0 & 0 & 0 \\ 1,79 & 1,79 & 0 & 0 \\ 1,46 & 1,46 & 1,46 & 0 \\ 0,913 & 0,913 & 0,913 & 0,913 \end{bmatrix} \quad L \cdot Y = B.$$

Resolvendo por eliminação de Gauss:

$$\left[ \begin{array}{ccc|c} 2,24 & 0 & 0 & -1 \\ 1,79 & 1,79 & 0 & 0 \\ 1,46 & 1,46 & 1,46 & 0 \\ 0,913 & 0,913 & 0,913 & 0,913 \end{array} \right] \quad \begin{aligned} L_2 &\leftarrow L_2 + 0,799L_1 \\ L_3 &\leftarrow L_3 + 0,624L_1 \\ L_4 &\leftarrow L_4 + 0,350L_1 \\ 0,913 &\leftarrow 0,913 - 0,913 \cdot 0,799 \\ 0,913 &\leftarrow 0,913 - 0,913 \cdot 0,624 \\ 0,913 &\leftarrow 0,913 - 0,913 \cdot 0,350 \end{aligned}$$

$$\left[ \begin{array}{ccc|c} 2,24 & 0 & 0 & -1 \\ 0 & 1,67 & 0 & 0 \\ 0 & 0 & 1,46 & 0 \\ 0 & 0 & 0,913 & 0,913 \end{array} \right] \quad \begin{aligned} 2,24x_1 &= -1 & x_1 &= 0,447 \\ 1,67x_2 &= 0 & x_2 &= 0,717 \\ 1,46x_3 &= 0 & x_3 &= 1,76 \\ 0,913x_4 &= 0,913 & x_4 &= 1,013 \end{aligned}$$

Figure 7: Imagem contendo a resolução de  $AX = B$  utilizando a decomposição Cholesky - Parte 1

Nota-se que assim como a decomposição LU, nós obtemos duas matrizes, onde a primeira é utilizada para obter Y (um vetor auxiliar necessário para obter a solução particular X).

$\text{J} X = Y$ , onde  $U = L^T$ .

$$\left[ \begin{array}{cccc|c} 2,24 & -1,79 & 0,447 & 0 & -0,447 \\ 0 & 1,67 & -1,91 & 0,598 & 0,717 \\ 0 & 0 & 1,46 & -1,95 & 1,76 \\ 0 & 0 & 0 & 0,913 & 6,57 \end{array} \right] \xrightarrow{\text{Step 1}} \left[ \begin{array}{cccc|c} & & & & x_4 = 7,20 \\ & & & & x_3 = (1,76 + 1,95 \cdot 7,2) / 1,46 = 10,8 \\ & & & & x_2 = (0,717 - 0,598 \cdot x_4 + 1,91 \cdot x_3) / 1,67 \\ & & & & x_1 = (-0,447 + 0,447 \cdot x_3 + 1,79 \cdot x_2) / 2,24 \end{array} \right]$$

$$X = [5,8, 10,2, 10,8, 7,20]$$

Figure 8: Imagem contendo a resolução de  $AX = B$  utilizando a decomposição Cholesky - Parte 2

Comparando então os resultados, podemos observar (imagem abaixo) que ambos são iguais, conforme era esperado.

```
bdantas@Oracle:~/Área de Trabalho/ALC_Lists/List_1$ python3 main.py
[5.799999999999995, 10.199999999999992, 10.799999999999992, 7.199999999999994]
```

Figure 9: Imagem contendo os cálculos da determinante da matriz A

#### 4.1.5 Método Iterativo Jacobi

Antes de mostrar a resolução é necessário ter em mente que a matriz dada não converge, pois, a linha 3, por exemplo, possui o elemento  $A_{3,3}$  menor que o somatório dos outros elementos dessa linha. Contudo, ainda assim, foi realizado passo a passo seus cálculos, porém não será realizada a comparação com o código desenvolvido devido à problemática apresentada.

A	x	b	Tentativas	x <sub>1</sub>	x <sub>2</sub>	x <sub>3</sub>	x <sub>4</sub>
$\begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix}$	$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$	$\begin{bmatrix} -1 \\ 2 \\ 1 \\ 3 \end{bmatrix}$	1	1	1	1	1
			2	0,4	1,5	1,33	1,2
			3	0,73	1,51	1,85	1,77
			4	0,64	1,7	2,4	2,2
			5	0,70	2,0	2,84	2,5
			6	0,84	2,4	3,3	2,7
			7	1,05	2,75	3,64	3,0
			8	1,30	3,11	4,01	3,2
			9	1,50	3,50	4,35	3,4
			10	1,70	3,81	4,7	3,6
$x_1^{(1)} = b_1 - (A_{1,2} \cdot x_2 + A_{1,3} \cdot x_3 + A_{1,4} \cdot x_4)$							
$x_2^{(1)} = b_2 - (A_{2,1} \cdot x_1 + A_{2,3} \cdot x_3 + A_{2,4} \cdot x_4)$							
$x_3^{(1)} = b_3 - (A_{3,1} \cdot x_1 + A_{3,2} \cdot x_2 + A_{3,4} \cdot x_4)$							
$x_4^{(1)} = b_4 - (A_{4,1} \cdot x_1 + A_{4,2} \cdot x_2 + A_{4,3} \cdot x_3)$							
Por 10 tentativas: $X = [1,70, 3,81, 4,7, 3,6]$							

Figure 10: Imagem contendo a resolução de  $AX = B$  utilizando o método iterativo Jacobi

#### 4.1.6 Método Iterativo Gauss-Seidel

Antes de mostrar a resolução é necessário ter em mente que a matriz dada converge, diferente do algoritmo anterior. Contudo, pelo número de passos para convergir é um número muito alto, foi considerado um número baixo de tentativas (10) para realizar seus cálculos, deste modo, o valor obtido nos cálculos manuais, divergem dos resultados obtidos pelo programa.

A	X	b	Tentativa	X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>
$\begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix}$	$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$	$\begin{bmatrix} -1 \\ 2 \\ 1 \\ 3 \end{bmatrix}$	1	1	1	1	1
			2	0,4	1,1	1,5	1,58
			3	0,38	1,32	2,04	1,97
			4	0,45	1,67	2,51	2,28
			5	0,63	2,05	2,95	2,55
			6	0,85	2,44	3,35	2,79
			7	1,08	2,82	3,73	3,02
			8	1,31	3,19	4,09	3,23
			9	1,53	3,54	4,43	3,43
			10	1,75	3,88	4,75	3,62
			:				
			Para 10 tentativas:	x = [1,75, 3,88, 4,75, 3,62]			

Figure 11: Imagem contendo a resolução de  $AX = B$  utilizando o método iterativo Gauss-Seidel

Comparação:

```
bdantas@Oracle:~/Área de Trabalho/ALC_Lista$ python3 main.py
x1: [5.798749912228412, 10.198049842515994, 10.798133810646098, 7.19889708001368]
residue: 9.65638161129393e-06
```

Figure 12: Imagem contendo a resolução de  $AX = B$  utilizando o método iterativo Gauss-Seidel

## 4.2 Letra B

### 4.2.1 Obtenha também a inversa de A usando o método de eliminação Gauss-Jordan

Tomando como base a matriz obtida pela eliminação de Gauss e, seguindo os passos em um matriz identidade:

Sendo:

(I)	(II)	(III)
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0,2 & 0 & 0 & 0 \\ 0,8 & 1 & 0 & 0 \\ -0,2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0,2 & 0 & 0 & 0 \\ 0,286 & 0,357 & 0 & 0 \\ 0,714 & 1,14 & 1 & 0 \\ -0,286 & -0,357 & 0 & 1 \end{bmatrix}$

  

(IV)	(V)
$\begin{bmatrix} 0,2 & 0 & 0 & 0 \\ 0,286 & 0,357 & 0 & 0 \\ 0,333 & 0,533 & 0,467 & 0 \\ 0,667 & 1,17 & 1,33 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

  

(VI)
$\begin{bmatrix} 0,2 & 0 & 0 & 0 \\ 0 & 1,6 & 2,6 & 2,4 \\ 0,4 & 2,4 & 2,6 & 1,6 \\ 1,8 & 1,4 & 1,6 & 1,2 \end{bmatrix}$

  

$$A^{-1} = \begin{bmatrix} 1,2 & 1,6 & 1,4 & 0,8 \\ 1,6 & 2,6 & 2,4 & 1,4 \\ 1,4 & 2,4 & 2,6 & 1,6 \\ 0,8 & 1,4 & 1,6 & 1,2 \end{bmatrix}$$

Figure 13: Imagem contendo os cálculos para gerar  $A^{-1}$  por meio do método de eliminação Gauss-Jordan

## 4.3 Letra C

### 4.3.1 Calcular o determinante de A

Para calcular o determinante, foi utilizada a matriz U da decomposição LU (Página 8).

$$U = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 2,8 & -3,2 & 1 \\ 0 & 0 & 2,14 & -2,86 \\ 0 & 0 & 0 & 0,833 \end{bmatrix}$$

, como U é uma matriz triangular superior:

$$\text{Det}(A) = 5 \cdot 2,8 \cdot 2,14 \cdot 0,833 \approx 25$$

Figure 14: Imagem contendo os cálculos da determinante da matriz A

## 5 Questão 5

Para ambos os métodos utilizados a solução foi idêntica, portanto, colocarei somente uma imagem referente à execução dos métodos de resolução da matrix  $AX=B$ .

```
bdantas@Oracle:~/Área de Trabalho/ALC_Lists/List_1$ python3 main.py
[0.1624042654217077, -0.4399144089652845, 0.4983679831231562, -0.4388861601607006, 0.9044166164750983, -0.5386464945717034, 0.6910538591753261
, -0.5109441019087015, 0.4305869055109058, -0.37980347844914464]
```

Figure 15: Imagem contendo a solução da matriz  $AX=B$  da questão 5

## 6 Questão 6

Para a resolução da determinante foi utilizado o método que está presente na questão 1. Tal como a questão anterior, colocarei somente uma imagem referente à execução do método.

```
bdantas@Oracle:~/Área de Trabalho/ALC_Lists/List_1$ python3 main.py
20385044096
```

Figure 16: Imagem contendo a solução da determinante da matriz da questão 5