

Relatório da Lista 1

COC473 - Semestre 2020/PLE

Gabriel G. Milan¹

¹Departamento de Engenharia Eletrônica e de Computação – Universidade Federal do Rio de Janeiro (UFRJ)
21.941-909 – Rio de Janeiro – RJ – Brasil

1. Introdução

Para as implementações em código solicitadas nessa lista (e nas seguintes) foi criado um pacote Python chamado "alc" (álgebra linear computacional). Esse pacote foi completamente escrito em Python e não está disponível no *Python Package Index* (ou PyPI). Em outras palavras, não pode ser instalado via *pip install alc*. No entanto, de forma a tornar possível replicar os resultados demonstrados nesse relatório, o código está disponibilizado abertamente em <https://github.com/gabriel-milan/linear-algebra>.

Nesse pacote, são implementadas todas as funcionalidades solicitadas (no caso da lista 1: decomposição LU, decomposição de Cholesky, método de Jacobi, método de Gauss-Seidel, eliminação de Gauss, eliminação de Gauss-Jordan, cálculo de determinante, inversas e resolução de sistemas $Ax = B$ usando os métodos previamente citados).

A fim de conseguir adquirir tais funcionalidades, também foram implementadas operações básicas, como adição, subtração e multiplicação de matrizes, algumas utilidades como o cálculo do traço, da transposta, de normas de ordem p , verificações de simetria, se é estritamente diagonal dominante e geradores de matrizes preenchidas com zeros, preenchidas com números aleatórios e matrizes identidade.

Devido a todas essas funcionalidades serem implementadas do zero, o código tornou-se grande, fazendo-o inviável de ser explicitado por completo nesse relatório. De qualquer forma, as partes mais relevantes serão explicitadas, de forma a tornar compreensível a implementação.

Os nomes de variáveis e comentários espalhados ao longo do código foram escritos em inglês, de forma a, ao final dessa matéria, possibilitar a exposição desse trabalho como portfólio.

Ao final desse relatório, serão acrescentadas páginas escaneadas das resoluções manuais da lista e, portanto, os exercícios cuja resolução for estritamente manual (1 e 4) não serão citados em seções desse relatório.

Para os exercícios restantes, cada seção desse relatório fará referência a um deles.

2. Exercício 2

2.1. Implementação da decomposição LU

A implementação da decomposição LU foi realizada na função *lu_decomposition*, disponível em *alc.decomposition*. Ao iniciar, realiza-se o método da eliminação de Gauss (implementado na função *gauss_elimination*, disponível em *alc.gauss*), tornando a matriz inicial A uma triangular superior U .

Como a função *gauss_elimination* pode ser configurada, através dos parâmetros, para retornar as matrizes intermediárias $M_i, i \in [1, n]$, indicando quais são pivô, pode-se utilizá-las para computar a matriz L , de forma que $L = L_n \times L_{n-1} \times \dots \times L_1$, sendo $L_i, i \in [1, n]$ as matrizes M_i com os elementos externos à diagonal multiplicados por -1 . Como as matrizes M_i possuem valores apenas abaixo da diagonal, a matriz L será, necessariamente, triangular inferior.

O código utilizado para o método da eliminação de Gauss é o seguinte:

```

from alc.utils import eye # Identity matrices

def gauss_elimination (
    arr, return_intermediates=False,
    show_steps=True,
    return_pivots=False
):
    m_n = []
    pivots = []
    # One column at a time
    for col_idx in range(arr.shape[1] - 1):
        if show_steps:
            print ("A = {}".format(arr))
        # Check for zeros on the diagonal
        if arr[col_idx][col_idx] == 0:
            # Switch row with one that has no zero values on diag
            for row_idx in range(col_idx + 1, arr.shape[0]):
                if arr[col_idx][row_idx] != 0:
                    # Get identity as base for pivoting
                    p = eye(arr.shape[0])
                    # Switch rows
                    tmp = p[row_idx]
                    p[row_idx] = p[col_idx]
                    p[col_idx] = tmp
                    if show_steps:
                        print ("P = {}".format(p))
                    m_n.append(p)
                    pivots.append(1)
                    arr = p * arr
                    if show_steps:
                        print ("P * A = {}".format(arr))
                    break
            # Get identity as base for combination
            m = eye(arr.shape[0])
            for row_idx in range(col_idx + 1, arr.shape[0]):
                # Fill coefficients of combination
                m[row_idx][col_idx] = -arr[row_idx][col_idx] / arr[col_idx][col_idx]
            m_n.append(m)
            pivots.append(0)
            if show_steps:
                print ("M = {}".format(m))
            # Multiply
            arr = m * arr
            if show_steps:
                print ("A' = {}".format(arr))
                print ("-----")

```

```

if return_intermediates:
    if return_pivots:
        return arr, m_n, pivots
    return arr, m_n
else:
    return arr

```

O código utilizado para a decomposição LU é o seguinte:

```

from alc.utils import eye # Identity matrices
from alc.gauss import gauss_elimination

def lu_decomposition (arr, return_det=False):
    # First of all, gauss elimination (arr will be U)
    arr, intermediates, pivots = gauss_elimination(
        arr,
        return_intermediates=True,
        show_steps=False,
        return_pivots=True
    )
    # Then, Li=Mi with elements other than diagonal multiplied by -1
    l_n = []
    for i, m in enumerate(intermediates):
        if not pivots[i]:
            l_n.append(mi_to_li(intermediates[i]))
        else:
            l_n.append(intermediates[i])
    # L = Ln * ... * L2 * L1
    L = eye(arr.shape[0])
    for li in l_n:
        L = L * li
    # U=arr, L=L
    det = 1
    for i in range(arr.shape[0]):
        for j in range(arr.shape[1]):
            if (i == j):
                det *= arr[i][j]
    if (det == 0):
        print ("WARNING: Essa é uma matriz singular e, portanto, essa decomposição pode não existir")
    if return_det:
        return L, arr, det
    return L, arr

```

Um exemplo de código para execução da decomposição em uma matriz quadrada qualquer (por exemplo a do exercício 4) é o seguinte:

```

import alc

A = alc.Array([
    [5, -4, 1, 0],
    [-4, 6, -4, 1],
    [1, -4, 6, -4],
    [0, 1, -4, 5],
])

```

```

])
L, U = alc.decomposition.lu_decomposition(A)

print ("A = {}".format(A))
print ("L = {}".format(L))
print ("U = {}".format(U))

```

O resultado será o seguinte:

```

A = alc.Array(
[5, -4, 1, 0],
[-4, 6, -4, 1],
[1, -4, 6, -4],
[0, 1, -4, 5],
)
L = alc.Array(
[1.0, 0.0, 0.0, 0.0],
[-0.8, 1.0, 0.0, 0.0],
[0.2, -1.142857142857143, 1.0, 0.0],
[0.0, 0.35714285714285715, -1.33333333333337, 1.0],
)
U = alc.Array(
[5.0, -4.0, 1.0, 0.0],
[0.0, 2.8, -3.2, 1.0],
[0.0, 0.0, 2.142857142857142, -2.8571428571428568],
[0.0, 0.0, 0.0, 0.83333333333333],
)

```

2.2. Implementação da decomposição de Cholesky

A implementação da decomposição de Cholesky foi realizada na função *cholesky_decomposition*, disponível em *alc.decomposition* utilizando o algoritmo genérico provido nos slides. O código para essa foi o seguinte:

```

from alc.utils import zeros # Zero-filled matrices

def cholesky_decomposition (arr):
    from alc.utils import is_definite_positive
    if (not is_definite_positive(arr)):
        raise ValueError(
            """Não é possível realizar a decomposição de Cholesky. A matriz fornecida não é simétrica positiva definida."""
        )
    L = zeros(arr.shape)
    for i in range(arr.shape[0]):
        L[i][i] = arr[i][i]
        for k in range(0, i):
            L[i][i] -= (L[i][k] ** 2)
        L[i][i] = (L[i][i] ** (1/2))
    for j in range(i + 1, arr.shape[1]):
        L[j][i] = arr[i][j]
        for k in range(0, i):

```

```
L[j][i] -= (L[i][k]*L[j][k])
L[j][i] /= L[i][i]
return L, L.t
```

É possível observar que foi reservado um espaço para verificação antes da decomposição em si. A verificação se a matriz é positiva definida ainda não foi implementada no momento de escrita desse relatório, porém tem seu espaço reservado para futura implementação.

Possivelmente, no momento de entrega desse relatório, a implementação dessa verificação já tenha sido realizada.

Um código para realizar a decomposição da matriz A do exercício 4 usando esse método é o seguinte:

```
import alc

A = alc.Array([
    [5, -4, 1, 0],
    [-4, 6, -4, 1],
    [1, -4, 6, -4],
    [0, 1, -4, 5],
])
L, LT = alc.decomposition.cholesky_decomposition(A)

print ("A = {}".format(A))
print ("L = {}".format(L))
print ("LT = {}".format(LT))
```

O resultado obtido seria o seguinte:

```
A = alc.Array(
[5, -4, 1, 0],
[-4, 6, -4, 1],
[1, -4, 6, -4],
[0, 1, -4, 5],
)
L = alc.Array(
[2.23606797749979, 0, 0, 0],
[-1.7888543819998317, 1.6733200530681511, 0, 0],
[0.4472135954999579, -1.9123657749350298, 1.4638501094227998, 0],
[0.0, 0.5976143046671968, -1.9518001458970664, 0.9128709291752769],
)
LT = alc.Array(
[2.23606797749979, -1.7888543819998317, 0.4472135954999579, 0.0],
[0, 1.6733200530681511, -1.9123657749350298, 0.5976143046671968],
[0, 0, 1.4638501094227998, -1.9518001458970664],
[0, 0, 0, 0.9128709291752769],
```

2.3. Resolvendo um sistema $Ax = B$ usando a decomposição LU

Em *alc.systems* foi implementada a função *solve*, que resolve um sistema $Ax = B$, dados A e B como parâmetros e, opcionalmente, o método de solução selecionado.

Para a solução utilizando a decomposição LU, a implementação faz o seguinte:

$$Ax = B$$

$$A = LU$$

$$LUx = B$$

$$y = Ux$$

$$Ly = B$$

A partir disso, é possível resolver y usando o método de *forward substitution*, conforme demonstrado em sala. Uma vez resolvido, pode-se resolver $Ux = y$ com retro-substituição, encontrando x .

O código utilizado para essa implementação é o seguinte:

```
from alc.utils import zeros
from alc.decomposition import lu_decomposition

# Backward substitution algorithm
def retrosub (A, B):
    # Initialize x
    x = zeros((A.shape[0], B.shape[1]))
    n = A.shape[0]
    # Retro-substitution
    x[n-1] = [B[n-1][0]/A[n-1][n-1]]
    for i in range(n-2, -1, -1):
        x[i][0] = B[i][0]
        for j in range(i+1, n):
            x[i][0] -= A[i][j]*x[j][0]
        x[i][0] /= A[i][i]
    return x

# Forward substitution algorithm
def forwardsub (L, B):
    # Initialize y
    y = zeros((L.shape[0], B.shape[1]))
    n = L.shape[0]
    # Forward sub
    y[0][0] = B[0][0] / L[0][0]
    for i in range(1, n):
        y[i][0] = B[i][0]
        for j in range(0, i):
            y[i][0] -= L[i][j]*y[j][0]
        y[i][0] /= L[i][i]
    return y

def solve (A, B, method='gauss', threshold=1e-3):
    ...
```

```

elif method == "lu":
    L, U = lu_decomposition (A, return_det=False)
    y = forwardsub(L, B)
    x = retrosub(U, y)
    return x

```

O código para resolver o exercício 4 através da decomposição LU é o seguinte:

```

from alc.systems import solve

A = Array([
    [5, -4, 1, 0],
    [-4, 6, -4, 1],
    [1, -4, 6, -4],
    [0, 1, -4, 5],
])
B = Array([
    [-1],
    [2],
    [1],
    [3]
])
print ("A = {}".format(A))
print ("B = {}".format(B))
x = solve(A, B, method='lu')
print ("x = {}".format(x))

```

O resultado adquirido seria o seguinte:

```

A = alc.Array(
[5, -4, 1, 0],
[-4, 6, -4, 1],
[1, -4, 6, -4],
[0, 1, -4, 5],
)
B = alc.Array(
[-1],
[2],
[1],
[3],
)
x = alc.Array(
[5.800000000000009],
[10.200000000000014],
[10.800000000000011],
[7.200000000000005],
)

```

2.4. Calculando o determinante através da decomposição LU

Um resultado interessante que pode ser obtido facilmente através da decomposição LU é o determinante. Fazendo uso do teorema $\det(AB) = \det(A) \cdot \det(B)$, fica fácil computar através da decomposição LU.

Sejam $L_{n \times n}$ uma matriz triangular inferior e $U_{n \times n}$ uma matriz triangular superior, ambas adquiridas através da decomposição LU de uma matriz $A_{n \times n}$ qualquer. Como L e U são triangulares, o valor de seu determinante corresponde ao produtório dos elementos na sua diagonal. Dessa forma,

$$\det(A) = \det(L) \cdot \det(U)$$

$$\det(A) = \prod_{i=1}^n l_{i,i} \cdot \prod_{i=1}^n u_{i,i}$$

Como os elementos na diagonal de L são sempre iguais a 1, $\prod_{i=1}^n l_{i,i} = 1$ e, portanto, o cálculo do determinante de A se resume a

$$\det(A) = \prod_{i=1}^n u_{i,i}$$

O cálculo do determinante usando esse método foi implementado dentro da função de decomposição LU (ver seção 2.1), porém um *wrapper* foi implementado em *alc.utils* de forma a facilitar o uso:

```
def det (arr):
    from alc.decomposition import lu_decomposition
    _, _, det = lu_decomposition(arr, return_det=True)
    return det
```

Assim, pode-se utilizá-lo da seguinte maneira:

```
import alc

A = alc.Array([
    [5, -4, 1, 0],
    [-4, 6, -4, 1],
    [1, -4, 6, -4],
    [0, 1, -4, 5],
])
print ("A = {}".format(A))
det = alc.utils.det(A)
print ("det(A) = {}".format(det))
```

E, finalmente, o resultado obtido seria 24.99999999.

3. Exercício 3

3.1. Implementação do método de Jacobi

O método de Jacobi é um método iterativo de solução para sistemas $Ax = B$, assim como Gauss-Seidel, também implementado para esse exercício. Ambos os métodos possuem uma condição para a convergência da solução: a matriz A deve ser estritamente diagonal dominante, ou seja: $|a_{i,i}| > \sum_{i \neq j} |a_{i,j}|$ para todo $i \in [1, n]$.

Para Gauss-Seidel há uma outra garantia de convergência, descrita na seção 3.2.

De qualquer forma, o algoritmo usado para a implementação foi o fornecido em sala de aula, usando como vetor inicial um vetor coluna preenchido com valores aleatórios. O código, a seguir, encontra-se em *alc.jacobi*:

```
from copy import deepcopy
from alc.utils import random_array, zeros, vector_norm, is_diagonally_dominant

def jacobi (A, B, threshold=1e-3):
    if (not is_diagonally_dominant(A)):
        raise ValueError("""
            A matriz \"A\" não é estritamente diagonal dominante e, portanto,
            o método de Jacobi não irá convergir!
        """)
    prev_x = random_array(B.shape)
    r = 1000
    n = B.shape[0]
    while (r > threshold):
        x = zeros(B.shape)
        for i in range(x.shape[0]):
            x[i][0] = B[i][0]
            for j in range(0, n):
                if (i != j):
                    x[i][0] -= A[i][j]*prev_x[j][0]
            x[i][0] /= A[i][i]
        r = vector_norm(x - prev_x, 2) / vector_norm(x, 2)
        prev_x = deepcopy(x)
    return x
```

Para a verificação de estritamente diagonal dominante, implementado em *alc.utils*, o código é o seguinte:

```
from alc.utils import zeros

def is_diagonally_dominant (arr):
    dom = zeros((arr.shape[0], 1))
    for i in range(arr.shape[0]):
        for j in range(arr.shape[1]):
            if (i != j):
                dom[i][0] += abs(arr[i][j])
    for i in range(arr.shape[0]):
        for j in range(arr.shape[1]):
            if (i == j):
                if (abs(arr[i][j]) < dom[i][0]):
                    return False
    return True
```

O código do método de Jacobi pode ser utilizado através da função *solve*, de *alc.systems*, com o argumento *method="jacobi"*.

Como, no exercício 4, a matriz A fornecida foi verificada como não estritamente diagonal dominante, o método de Jacobi não convergeria. De qualquer forma, um exemplo de utilização com a matriz A citada poderia ser feito da seguinte maneira:

```

import alc

A = alc.Array([
    [5, -4, 1, 0],
    [-4, 6, -4, 1],
    [1, -4, 6, -4],
    [0, 1, -4, 5],
])
B = alc.Array([
    [-1],
    [2],
    [1],
    [3]
])
print ("A = {}".format(A))
print ("B = {}".format(B))
x = alc.systems.solve(A, B, method='jacobi')
print ("x = {}".format(x))

```

Quando executar esse código, uma mensagem dizendo que a matriz A não é estritamente diagonal dominante será exibida e, portanto, o método de Jacobi não chegará à convergência.

3.2. Implementação do método de Gauss-Seidel

Para o método de Gauss-Seidel, valem as mesmas condições de convergência do método de Jacobi (seção 3.1). No entanto, a convergência é garantida para o caso de a matriz A ser simétrica e positiva definida.

Novamente, o algoritmo utilizado para a implementação foi o fornecido em sala de aula e ele está implementado em *alc.gauss_seidel*. Vale notar que também está disponível na função *alc.systems.solve* através do argumento *method="gauss_seidel"*.

```

from copy import deepcopy
from alc.utils import random_array, zeros, vector_norm
from alc.utils import is_diagonally_dominant, is_definite_positive

def gauss_seidel (A, B, threshold=1e-3):
    if (not is_diagonally_dominant(A)):
        if (not is_definite_positive(A)):
            raise ValueError("""
                A matriz \"A\" não é estritamente diagonal dominante nem positiva
                definida e, portanto, o método de Gauss-Seidel não irá convergir!
            """)
    prev_x = random_array(B.shape)
    r = 1000
    n = B.shape[0]
    while (r > threshold):
        x = zeros(B.shape)
        for i in range(x.shape[0]):
            x[i][0] = B[i][0]
            for j in range(0, i):
                x[i][0] -= A[i][j]*x[j][0]
            for j in range(i+1, n):
                x[i][0] -= A[i][j]*x[j][0]
        r = vector_norm(x - prev_x)
        prev_x = x

```

```

for j in range(i+1, n):
    x[i][0] -= A[i][j]*prev_x[j][0]
    x[i][0] /= A[i][i]
    r = vector_norm(x - prev_x, 2) / vector_norm(x, 2)
    prev_x = deepcopy(x)
return x

```

Se o único critério avaliado para determinar a capacidade de convergência do algoritmo fosse o descrito para o método de Jacobi (seção 3.1), também não seria possível resolver o sistema do exercício 4 através do método de Gauss-Seidel. No entanto, a convergência é garantida devido à matriz A fornecida ser simétrica e positiva definida.

Dessa forma, para resolver o exercício 4, poderia se utilizar o seguinte código:

```

import alc

A = alc.Array([
    [5, -4, 1, 0],
    [-4, 6, -4, 1],
    [1, -4, 6, -4],
    [0, 1, -4, 5],
])
B = alc.Array([
    [-1],
    [2],
    [1],
    [3]
])
print ("A = {}".format(A))
print ("B = {}".format(B))
x = alc.systems.solve(A, B, method='gauss_seidel')
print ("x = {}".format(x))

```

Com isso, o resultado obtido é bem próximo dos resultados obtidos com outros métodos de solução, previamente mostrados nas seções 2.1 e 2.2.

```

A = alc.Array(
    [5, -4, 1, 0],
    [-4, 6, -4, 1],
    [1, -4, 6, -4],
    [0, 1, -4, 5],
)
B = alc.Array(
    [-1],
    [2],
    [1],
    [3],
)
x = alc.Array(
    [5.673377707783734],
    [10.002467141572232],
    [10.610972334046272],
    [7.088284438922571],
)

```

4. Exercício 5

4.1. Resolvendo um sistema $Ax = B$

Para a solução de sistemas $Ax = B$, foi desenvolvida uma função genérica e configurável dentro de `alc.systems` chamada `solve`. Essa função recebe como argumento as matrizes A e B , além do opcional `method`, que permite ao usuário selecionar com qual método deseja resolver o sistema.

No momento, os métodos disponíveis são `gauss`, `gauss_jordan`, `jacobi`, `gauss_seidel`, `lu` e `cholesky`. A implementação da função `solve` é a seguinte:

```
from alc.gauss import gauss_elimination, gauss_jordan_elimination
from alc.utils import zeros
from alc.jacobi import jacobi
from alc.gauss_seidel import gauss_seidel
from alc.decomposition import lu_decomposition, cholesky_decomposition
from alc.systems import retrosub, forwardsub

def solve (A, B, method='gauss', threshold=1e-3):
    if method == "gauss":
        # Turn A into an upper triangular matrix
        A, intermediates = gauss_elimination (
            A, return_intermediates=True, show_steps=False, return_pivots=False
        )
        # To keep it an equation, apply transformations to B as well
        for m in intermediates:
            B = m * B
        x = retrosub(A, B)
        return x
    elif method == "gauss_jordan":
        # Turn A into an upper triangular matrix
        A, intermediates = gauss_elimination (
            A, return_intermediates=True, show_steps=False, return_pivots=False
        )
        # To keep it an equation, apply transformations to B as well
        for m in intermediates:
            B = m * B
        # Turn A into diagonal matrix
        A, intermediates = gauss_jordan_elimination (
            A, return_intermediates=True, show_steps=False
        )
        # Apply transformations to B as well
        for m in intermediates:
            B = m * B
        # Initialize x
        x = zeros((A.shape[0], B.shape[1]))
        # Computing x
        for i in range(A.shape[0]):
            x[i][0] = B[i][0]/A[i][i]
        return x
    elif method == "jacobi":
        x = jacobi(A, B, threshold=threshold)
        return x
    elif method == "gauss_seidel":
```

```

x = gauss_seidel(A, B, threshold=threshold)
return x

elif method == "lu":
    L, U = lu_decomposition (A, return_det=False)
    y = forwardsub(L, B)
    x = retrosub(U, y)
    return x

elif method == "cholesky":
    try:
        L, Lt = cholesky_decomposition (A)
        y = forwardsub(L, B)
        print (y)
        x = retrosub(Lt, y)
        return x
    except ValueError as e:
        raise e
else:
    raise NameError("Solving method not allowed!")

```

Como não é o propósito dessa seção, não será explicitado aqui o código dos métodos de eliminação de Gauss-Jordan. Todos os outros já foram descritos previamente. De qualquer forma, o código está disponível por completo na página do GitHub, citada na seção introdutória desse documento.

No código a seguir, será demonstrado como resolver o sistema fornecido no enunciado usando LU e Cholesky. Suas implementações já foram descritas nas seções 2.1 e 2.2, respectivamente.

```

import alc

A = alc.Array([
    [16, 9, 8, 7, 6, 5, 4, 3, 2, 1],
    [9, 17, 9, 8, 7, 6, 5, 4, 3, 2],
    [8, 9, 18, 9, 8, 7, 6, 5, 4, 3],
    [7, 8, 9, 19, 9, 8, 7, 6, 5, 4],
    [6, 7, 8, 9, 18, 9, 8, 7, 6, 5],
    [5, 6, 7, 8, 9, 17, 9, 8, 7, 6],
    [4, 5, 6, 7, 8, 9, 16, 9, 8, 7],
    [3, 4, 5, 6, 7, 8, 9, 15, 9, 8],
    [2, 3, 4, 5, 6, 7, 8, 9, 14, 9],
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 13],
])
B = alc.Array([
    [4],
    [0],
    [8],
    [0],
    [12],
    [0],
    [8],
    [0],
    [4],
    [0],
])

```

```

print ("A = {}".format(A))
print ("B = {}".format(B))
x_lu = alc.systems.solve(A, B, method='lu')
x_cholesky = alc.systems.solve(A, B, method='cholesky')
print ("x (LU) = {}".format(x_lu))
print ("x (Cholesky) = {}".format(x_cholesky))

```

Os resultados obtidos podem ser vistos a seguir:

```

A = alc.Array(
[16, 9, 8, 7, 6, 5, 4, 3, 2, 1],
[9, 17, 9, 8, 7, 6, 5, 4, 3, 2],
[8, 9, 18, 9, 8, 7, 6, 5, 4, 3],
[7, 8, 9, 19, 9, 8, 7, 6, 5, 4],
[6, 7, 8, 9, 18, 9, 8, 7, 6, 5],
[5, 6, 7, 8, 9, 17, 9, 8, 7, 6],
[4, 5, 6, 7, 8, 9, 16, 9, 8, 7],
[3, 4, 5, 6, 7, 8, 9, 15, 9, 8],
[2, 3, 4, 5, 6, 7, 8, 9, 14, 9],
[1, 2, 3, 4, 5, 6, 7, 8, 9, 13],
)
B = alc.Array(
[4],
[0],
[8],
[0],
[12],
[0],
[8],
[0],
[4],
[0],
)
x (LU) = alc.Array(
[0.1624042654217077],
[-0.4399144089052846],
[0.49836798312315045],
[-0.4388861601607006],
[0.9044166164750982],
[-0.5386464945717035],
[0.6910538591753264],
[-0.5109441019087019],
[0.43058690551090606],
[-0.3798034784491446],
)
x (Cholesky) = alc.Array(
[0.1624042654217077],
[-0.4399144089052845],
[0.4983679831231503],
[-0.43888616016070064],
[0.9044166164750985],
[-0.5386464945717038],
[0.6910538591753262],
[-0.5109441019087019],
[0.430586905510906],

```

```
[-0.37980347844914447],  
)
```

5. Exercício 6

5.1. Calculando o determinante de uma matriz

Nesse exercício, o objetivo é calcular o determinante da matriz A do exercício 5. Para isso, será feito uso do *wrapper* descrito na seção 2.4:

```
import alc  
  
A = alc.Array([  
    [16, 9, 8, 7, 6, 5, 4, 3, 2, 1],  
    [9, 17, 9, 8, 7, 6, 5, 4, 3, 2],  
    [8, 9, 18, 9, 8, 7, 6, 5, 4, 3],  
    [7, 8, 9, 19, 9, 8, 7, 6, 5, 4],  
    [6, 7, 8, 9, 18, 9, 8, 7, 6, 5],  
    [5, 6, 7, 8, 9, 17, 9, 8, 7, 6],  
    [4, 5, 6, 7, 8, 9, 16, 9, 8, 7],  
    [3, 4, 5, 6, 7, 8, 9, 15, 9, 8],  
    [2, 3, 4, 5, 6, 7, 8, 9, 14, 9],  
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 13],  
])  
print ("A = {}".format(A))  
det = alc.utils.det(A)  
print ("det (A) = {}".format(det))
```

O resultado obtido é o seguinte:

```
A = alc.Array(  
[16, 9, 8, 7, 6, 5, 4, 3, 2, 1],  
[9, 17, 9, 8, 7, 6, 5, 4, 3, 2],  
[8, 9, 18, 9, 8, 7, 6, 5, 4, 3],  
[7, 8, 9, 19, 9, 8, 7, 6, 5, 4],  
[6, 7, 8, 9, 18, 9, 8, 7, 6, 5],  
[5, 6, 7, 8, 9, 17, 9, 8, 7, 6],  
[4, 5, 6, 7, 8, 9, 16, 9, 8, 7],  
[3, 4, 5, 6, 7, 8, 9, 15, 9, 8],  
[2, 3, 4, 5, 6, 7, 8, 9, 14, 9],  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 13],  
)  
det (A) = 20385044096.0
```

6. Soluções manuais

A seguir, será anexado a esse relatório as soluções manuais escaneadas.

ALGEBRA LINEAR COMPUTACIONAL COC473

GABRIEL GAZÓN MILAN · DRE 116034397

① RESOLVER MANUALMENTE O SISTEMA $Ax = B$

$$A = \begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} \quad B = \begin{bmatrix} -1 \\ 2 \\ 1 \\ 3 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 4/5 & 1 & 0 & 0 \\ -1/5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad M_1 A = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & -10/5 & 20/5 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix}; \quad M_1 B = \begin{bmatrix} -1 \\ 6/5 \\ 6/5 \\ 3 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3/7 & 1 & 0 \\ 0 & -5/14 & 0 & 1 \end{bmatrix}; \quad M_2 M_1 A = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & 0 & 15/7 & -20/7 \\ 0 & 0 & -20/7 & 65/14 \end{bmatrix}; \quad M_2 M_1 B = \begin{bmatrix} -1 \\ 6/5 \\ 18/7 \\ 12/7 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 4/3 & 1 \end{bmatrix}; \quad M_3 M_2 M_1 A = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & 0 & 15/7 & -20/7 \\ 0 & 0 & 0 & 5/6 \end{bmatrix}; \quad M_3 M_2 M_1 B = \begin{bmatrix} -1 \\ 6/5 \\ 18/7 \\ 6 \end{bmatrix}$$

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & 0 & 15/7 & -20/7 \\ 0 & 0 & 0 & 5/6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 6/5 \\ 18/7 \\ 6 \end{bmatrix} \Rightarrow X = \begin{bmatrix} 29/5 \\ 51/5 \\ 54/5 \\ 36/5 \end{bmatrix}$$

4) a) • MÉTODO DA ELIMINAÇÃO DE GAUSS:

* FEITO NO EX. ①

* MÉTODO DE GAUSS-JORDAN

* APÓS A ELIMINAÇÃO DE GAUSS, TEMOS:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 4/5 & 1 & 0 & 0 \\ -1/5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2/7 & 1 & 0 \\ 0 & -5/14 & 0 & 1 \end{bmatrix}; \quad M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 4/3 & 1 \end{bmatrix}$$

$$U = M_3 M_2 M_1 A = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & 0 & 15/7 & -29/7 \\ 0 & 0 & 0 & 5/6 \end{bmatrix}; \quad B' = M_3 M_2 M_1 B = \begin{bmatrix} -1 \\ 5/5 \\ 12/7 \\ 6 \end{bmatrix}$$

* A PARTIR DAÍ, FAZEMOS:

$$M_1^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -8/5 \\ 0 & 0 & 1 & 24/7 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad M_1^* U = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 0 \\ 0 & 0 & 15/7 & 0 \\ 0 & 0 & 0 & 5/6 \end{bmatrix}; \quad M_1^* B' = \begin{bmatrix} -1 \\ -6 \\ 162/7 \\ 6 \end{bmatrix}$$

$$M_2^* = \begin{bmatrix} 1 & 0 & -7/15 & 0 \\ 0 & 1 & 112/75 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad M_2^* M_1^* U = \begin{bmatrix} 5 & -4 & 0 & 0 \\ 0 & 14/5 & 0 & 0 \\ 0 & 0 & 15/7 & 0 \\ 0 & 0 & 0 & 5/6 \end{bmatrix}; \quad M_2^* M_1^* B' = \begin{bmatrix} -59/5 \\ 214/25 \\ 162/7 \\ 6 \end{bmatrix}$$

$$M_3^* = \begin{bmatrix} 1 & 19/7 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad M_3^* M_2^* M_1^* U = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 14/5 & 0 & 0 \\ 0 & 0 & 15/7 & 0 \\ 0 & 0 & 0 & 5/6 \end{bmatrix}; \quad M_3^* M_2^* M_1^* B' = \begin{bmatrix} 29 \\ 714/25 \\ 162/7 \\ 6 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 14/5 & 0 & 0 \\ 0 & 0 & 15/7 & 0 \\ 0 & 0 & 0 & 5/6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 29 \\ 714/25 \\ 162/7 \\ 6 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 29/5 \\ 51/5 \\ 54/5 \\ 36/5 \end{bmatrix}$$

• Decomposição LU:

* PASSO A ELIMINAÇÃO DE GAUSS, TERMOS:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 4/5 & 1 & 0 & 0 \\ -1/5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2/3 & 1 & 0 \\ 0 & -6/11 & 0 & 1 \end{bmatrix}; \quad M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 4/3 & 1 \end{bmatrix}$$

$$U = M_3 M_2 M_1 A = \begin{bmatrix} 5 & -11 & 1 & 0 \\ 0 & 14/5 & -19/5 & 1 \\ 0 & 0 & 15/7 & -29/7 \\ 0 & 0 & 0 & 5/6 \end{bmatrix}; \quad B' = M_3 M_2 M_1 B = \begin{bmatrix} -1 \\ 6/5 \\ 17/7 \\ 6 \end{bmatrix}$$

* $L = L_3 L_2 L_1 \Rightarrow L_i = M_i$ c/ ELEMENTOS FORA DA DIAGONAL INVERTIDOS

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -4/3 & 1 \end{bmatrix}; \quad L^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -4/5 & 1 & 0 & 0 \\ 1/5 & -8/7 & 1 & 0 \\ 0 & 9/4 & -4/3 & 1 \end{bmatrix}$$

* MANIPULANDO

$$Ax = B \Rightarrow LUx = B \Rightarrow Ly = B \quad (y = Ux)$$

* RESOLVENDO Y

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -4/5 & 1 & 0 & 0 \\ 1/5 & -8/7 & 1 & 0 \\ 0 & 9/4 & -4/3 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 1 \\ 3 \end{bmatrix} \Rightarrow y_i = b_i - \sum_{j=1}^3 L_{ij} y_j, \quad i = 2, 3, \dots, n$$

$$\begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -16/5 & 1 \\ 0 & 0 & 15/7 & -29/7 \\ 0 & 0 & 0 & 5/6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 6/5 \\ 18/7 \\ 6 \end{bmatrix} \Rightarrow x_n = y_n / u_{nn}$$

$$x = \begin{bmatrix} 29/5 \\ 54/5 \\ 54/5 \\ 26/5 \end{bmatrix}$$

• DECOMPOSIÇÃO DE CHOLESKY

* ALGORITMO: $A = LL^T$

- $i = 1, 2, \dots, n$

$$-- l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} (l_{i,k})^2}$$

--- $j = i+1, i+2, \dots, n$

$$--- l_{ji} = \frac{1}{l_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk} \right)$$

* APÓS APLICAÇÃO DO ALGORITMO, TERÍAMOS:

$$L = \begin{bmatrix} +2,2361 & 0 & 0 & 0 \\ -1,7889 & +1,6733 & 0 & 0 \\ +0,4472 & -1,9124 & +1,4639 & 0 \\ 0 & +0,5976 & -1,9518 & +0,9129 \end{bmatrix}$$

+ FAZEMOS $LL^T x = B$, TERÍAMOS, SEMELHANTE À SOLUÇÃO com LU

$$LL^T x = B \Rightarrow Ly = B \quad (L^T x = y)$$

$$y = \begin{bmatrix} -0,4472 \\ +0,7171 \\ +1,7566 \\ +6,5727 \end{bmatrix}$$

\Rightarrow RESOLVENDO \Rightarrow

$$L^T x = y$$

$$x = \begin{bmatrix} +5,8 \\ +10,2 \\ +10,8 \\ +7,2 \end{bmatrix}$$

• MÉTODO DE JACOBI:

$$\Rightarrow \text{A MATRIZ } A = \begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} \text{ NÃO É ESTRITAMENTE}$$

DIAGONAL DOMINANTE E, PORTANTO, O MÉTODO DE JACOBI
NÃO IRÁ CONVERGIR.

• MÉTODO DE GAUSS-SEIDEL:

* APESAR DE NÃO SER ESTRITAMENTE DIAGONAL DOMINANTE, POR SER POSITIVA DEFINIDA, CONVERGE.

INICIANDO C/ $\bar{x}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$, CALCULA-SE $x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij}x_j^{(k+1)} - \sum_{j < i} a_{ij}x_j^{(k)} \right)$

E $R = \frac{\|x^{(k+1)} - x^{(k)}\|_2}{\|x^{(k+1)}\|_2}$ ATÉ QUE $R \leq \theta$,

SENDO θ UM LIMIAZ. P/ ESSE EXERCÍCIO, $\theta = 10^{-3}$

$$\bar{x}_1 = \begin{bmatrix} +0,4 \\ +1,1 \\ +1,5 \\ +1,58 \end{bmatrix}, R_1 \approx 0,4 \quad \bar{x}_2 = \begin{bmatrix} +0,38 \\ +1,32 \\ +2,04 \\ +1,97 \end{bmatrix}, R_2 \approx 0,22 \quad \bar{x}_3 = \begin{bmatrix} +0,45 \\ +1,67 \\ +2,51 \\ +2,28 \end{bmatrix}, R_3 \approx 0,17$$

$$\bar{x}_4 = \begin{bmatrix} +0,03 \\ +2,05 \\ +2,95 \\ +2,55 \end{bmatrix}, R_4 \approx 0,15 \quad \bar{x}_5 = \begin{bmatrix} -0,85 \\ 2,44 \\ 3,35 \\ 2,79 \end{bmatrix}, R_5 \approx 0,13 \quad \bar{x}_6 = \begin{bmatrix} 1,08 \\ 2,82 \\ 3,73 \\ 3,02 \end{bmatrix}, R_6 \approx 0,11$$

$$\bar{x}_7 = \begin{bmatrix} 1,31 \\ 3,19 \\ 4,09 \\ 3,23 \end{bmatrix}, R_7 \approx 0,10, \quad \bar{x}_8 = \begin{bmatrix} 1,53 \\ 3,54 \\ 4,43 \\ 3,43 \end{bmatrix}, R_8 \approx 0,08, \quad \bar{x}_9 = \begin{bmatrix} 1,75 \\ 3,88 \\ 4,75 \\ 3,62 \end{bmatrix}, R_9 \approx 0,07$$

$$\bar{x}_{10} = \begin{bmatrix} 1,95 \\ 4,20 \\ 5,05 \\ 3,81 \end{bmatrix}, R_{10} \approx 0,07, \quad \bar{x}_{11} = \begin{bmatrix} 2,15 \\ 4,50 \\ 5,35 \\ 3,98 \end{bmatrix}, R_{11} \approx 0,06, \quad \bar{x}_{12} = \begin{bmatrix} 2,33 \\ 4,79 \\ 5,62 \\ 4,14 \end{bmatrix}, R_{12} \approx 0,05$$

Como o valor inicial é distante do resultado esperado, seriam necessárias muitas iterações para convergir. Verificando pelo computador, com $\theta = 10^{-3}$, seriam necessárias 76 iterações e o resultado seria $\bar{x}_{76} = \begin{bmatrix} 5,67 \\ 10,00 \\ 10,61 \\ 7,09 \end{bmatrix}$.

b) OBTER A INVERSA DE A USANDO O MÉTODO DE GAUSS-JORDAN.

A PARTIR DA MATRIZ DIAGONAL OBTIDA PELO MÉTODO DE GAUSS-JORDAN,

$\begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & \frac{4}{5} & 0 & 0 \\ 0 & 0 & \frac{15}{7} & 0 \\ 0 & 0 & 0 & \frac{5}{6} \end{bmatrix}$, É POSSÍVEL GERAR UMA MATRIZ M_4^* QUE A NORMALIZE. DESSA FORMA, $M_4^* M_3^* M_2^* M_1^* M_3 M_2 M_1 A = I$, SENDO M_3^* , M_2^* , M_1^* , M_3 , M_2 , M_1 AS MATRIZES INTERMEDIÁRIAS ENCONTRADAS PELOS MÉTODOS DE ELIMINAÇÃO DE GAUSS E GAUSS-JORDAN.

A PARTIR DA EXPRESSÃO $M_4^* M_3^* M_2^* M_1^* M_3 M_2 M_1 A = I$, É POSSÍVEL VERIFICAR QUE $C = A^{-1}$.

• ENCONTRANDO M_4^* :

$$M_4^* \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & \frac{4}{5} & 0 & 0 \\ 0 & 0 & \frac{15}{7} & 0 \\ 0 & 0 & 0 & \frac{5}{6} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow M_4^* = \begin{bmatrix} \frac{1}{5} & 0 & 0 & 0 \\ 0 & \frac{5}{4} & 0 & 0 \\ 0 & 0 & \frac{7}{15} & 0 \\ 0 & 0 & 0 & \frac{6}{5} \end{bmatrix}$$

• ENCONTRANDO A INVERSA:

$$M_4^* M_3^* M_2^* M_1^* M_3 M_2 M_1 = A^{-1} = \begin{bmatrix} \frac{6}{5} & \frac{8}{5} & \frac{7}{5} & \frac{4}{5} \\ \frac{8}{5} & \frac{13}{5} & \frac{12}{5} & \frac{7}{5} \\ \frac{7}{5} & \frac{12}{5} & \frac{13}{5} & \frac{8}{5} \\ \frac{4}{5} & \frac{7}{5} & \frac{8}{5} & \frac{6}{5} \end{bmatrix}$$

c) Cálculo do $\det(A)$:

• A PARTIR DA DECOMPOSIÇÃO LU É POSSÍVEL CALCULAR O DETERMINANTE USANDO O SEGUINTE TEOREMA: $\det(AB) = \det(A) * \det(B)$.

Como $A = LU$, $\det(A) = \det(L) \cdot \det(U)$.

SENDO L UMA MATRIZ TRIANGULAR INFERIOR CUOS ELEMENTOS NA DIAGONAL SÃO, IGUAIS A 1, $\det(L) = 1$.

DESSA FORMA, É POSSÍVEL REDUZIR A EXPRESSÃO PARA $\det(A) = 1 \cdot \det(U) \Rightarrow \det(A) = \det(U)$.

SENDO U UMA MATRIZ TRIANGULAR SUPERIOR, SEU DETERMINANTE É IGUAL AO PRODUTO RÍO DE SEUS ELEMENTOS NA DIAGONAL.

$$U = \begin{bmatrix} 5 & -4 & 1 & 0 \\ 0 & 14/5 & -13/5 & 1 \\ 0 & 0 & 15/7 & -20/7 \\ 0 & 0 & 0 & 5/6 \end{bmatrix} \Rightarrow \det(U) = \prod_{i=1}^n u_{ii} = 25$$