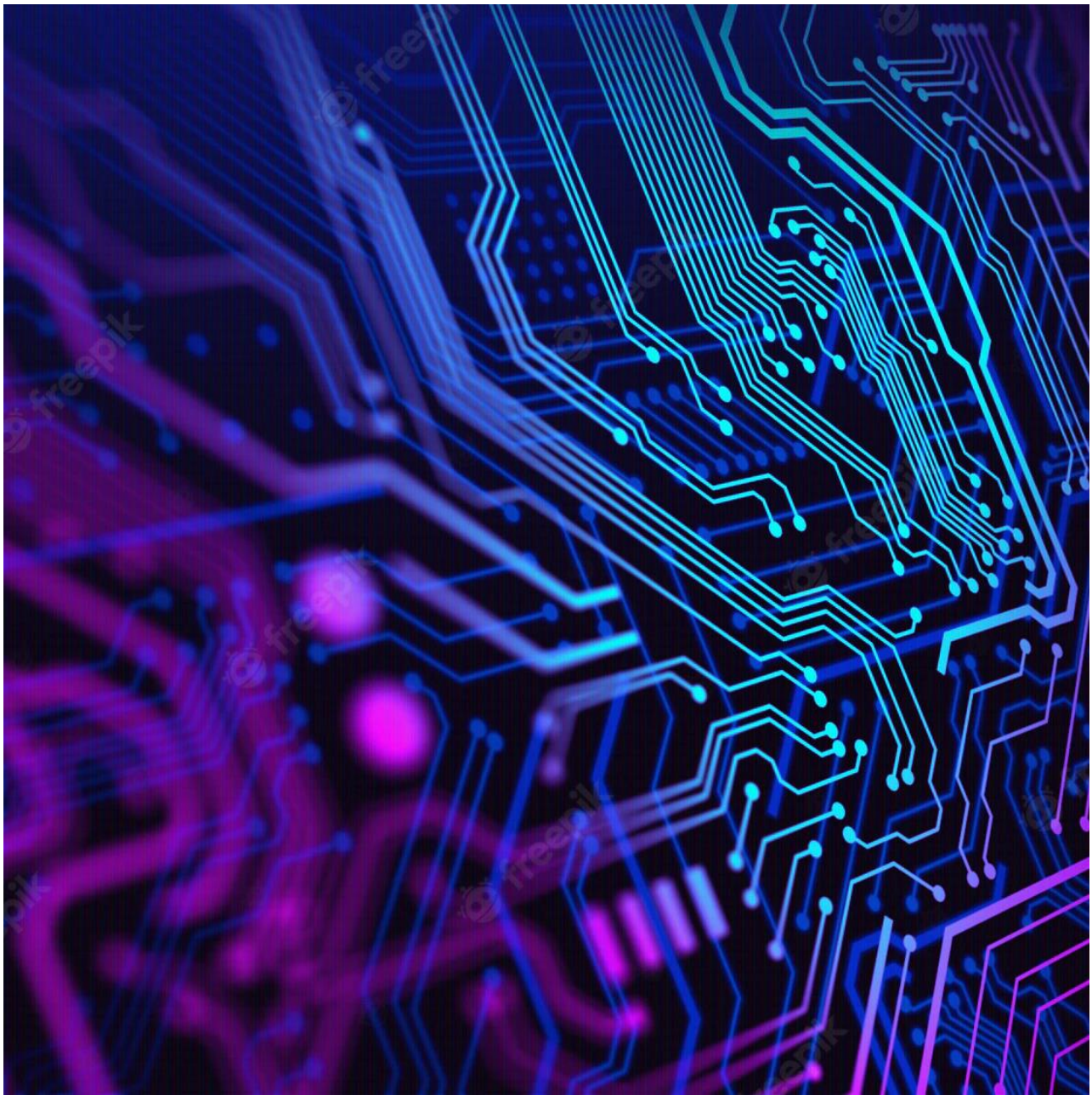
Matrizes tridiagonais, tridiagonais cíclicas e sistemas.

Decomposição LU para Matrizes Tridiagonais

Marco Antônio Rudas Napoli, n°USP: 11857970



Decomposição LU

Certas matrizes podem ser triangularizadas pelo Método de Eliminação de Gauss sem que haja troca de linhas. Assim, conseguimos montar duas matrizes, L e U, a partir de uma matriz A.

A matriz L é triangular inferior com os itens da diagonal principal igual a 1 e com os demais iguais aos multiplicadores resultantes do Método de Eliminação de Gauss.

Enquanto isso, a Matriz U é simplesmente a matriz A escalonada.

Assim, conseguimos montar a relação A = LU.

Decomposição LU de uma matriz tridiagonal:

Sabemos que uma matriz tridiagonal possui o seguinte formato:

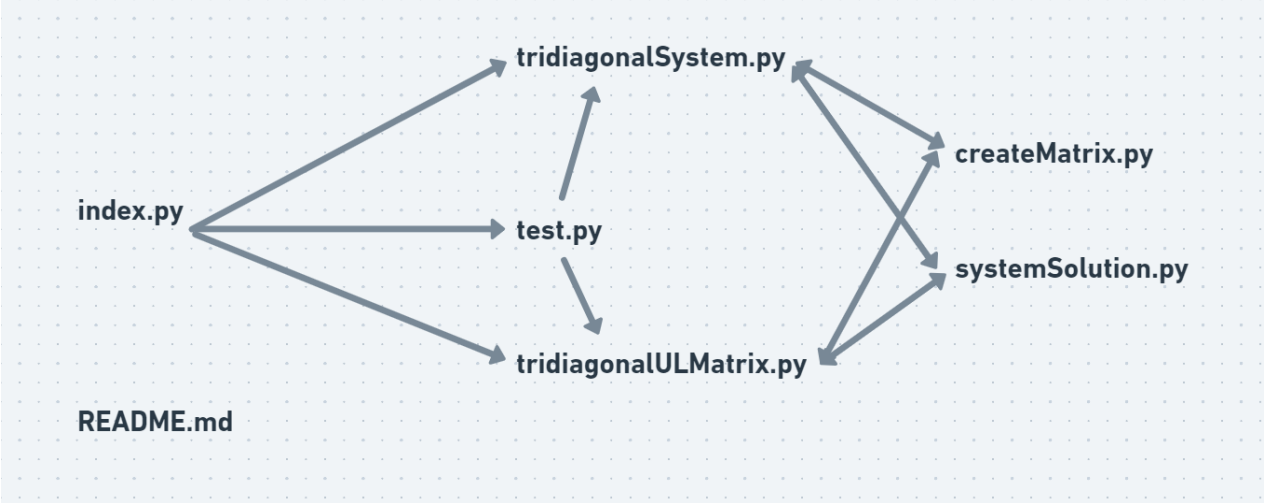
$$A = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{bmatrix}$$

Ao fazermos a decomposição, encontramos que a matriz L possui itens apenas na

diagonal secundária inferior. A matriz U possui a diagonal principal seguindo o Método de Eliminação de Gauss e a diagonal secundária superior como os mesmos itens da Matriz A, enquanto todos os outros tem valor igual a “0”.

Dessa maneira podemos economizar processamento e armazenar os nossos dados de maneira mais inteligente.

Estrutura de Funções do Projeto:



Solução 1 - Decomposição LU de Matriz Tridiagonal

Após introduzirmos os valores dos vetores a, b e c a partir do seguinte código podemos iniciar o algorítimo para achar o as matrizes L e U.

```
#Mapeamento da diagonal a
i = 0
while i < ordemMatrix-1 :
    numberCell = float(input(f'Insira o valor de a{i+2}:'))
    vectora.append(numberCell)

    i = i + 1

#Mapeamento da diagonal b
i = 0
while i < ordemMatrix :
    numberCell = float(input(f'Insira o valor de b{i+1}:'))
    vectorb.append(numberCell)

    i = i + 1

#Mapeamento da diagonal c
i = 0
while i < ordemMatrix-1 :
    numberCell = float(input(f'Insira o valor de c{i+2}:'))
    vectorc.append(numberCell)

    i = i + 1
```

A partir disso, introduzimos os valores mapeados numa matriz, criada pela da função “newSquareMatrix(ordemMatrix)”, utilizando a função “addVectorOnMatrix(order, vector, matrix)”. (obs: O algorítmo desta última foi desenvolvido de maneira a atender o segundo exercício proposto).

```
def addVectorOnMatrix(order, vector, matrix):
    matrix = matrix
    if order == "a":
        if len(vector) == len(matrix):
            matrix[0][len(matrix)-1] = vector[0]
            for i in range(len(vector)-1):
                matrix[i+1][i] = vector[i+1]
        else:
            for i in range(len(vector)):
                matrix[i+1][i] = vector[i]
    elif order == "b":
        for i in range(len(vector)):
            matrix[i][i] = vector[i]
    elif order == "c":
        if len(vector) == len(matrix):
            matrix[len(matrix)-1][0] = vector[len(matrix)-1]
            for i in range(len(vector)-1):
                matrix[i][i+1] = vector[i]
        else:
            for i in range(len(vector)):
                matrix[i][i+1] = vector[i]
    else:
        print("Ocorreu algum ERRO, verifique os seus dados")
```

Possuindo a matriz A em mão já podemos começar a trabalhar para acharmos as matrizes U e L.

Sabemos que a matriz U possui a seguinte propriedade: $u_{i,\,i+1} = a_{i,\,i+1}$. Assim temos que, a matriz U, inicialmente formada por zeros, terá sua matriz secundária superior identica à matriz A, economizando processamento para sua construção. Além disso, sabemos que a primeira linha é identica a da matriz A.

```
#A Matriz U possui a seguinte consição u(i,i+1) = a(i,i+1)
uMatrix = newSquareMatrix(ordemMatrix)
addVectorOnMatrix("c", vectorc, uMatrix)
uMatrix[0] = matrix[0]
```

Seguindo para a construção da matriz L, temos que os únicos itens que não são obrigatoriamente zeros são pertencentes à matriz principal e à secundária inferior. Tendo essa propriedade em mão, conseguimos dar o formato da matriz da seguinte forma:

```
"""A Matriz L é composta por uma diagonal principal formada por "1" e os multiplicadores
Neste caso, os unicos multiplicadores não nulos são l(i+1,i). Portanto, podemos escrever os dados em um vetor"""
lMatrix = newSquareMatrix(ordemMatrix)
addVectorOnMatrix("b", [1]*ordemMatrix, lMatrix)
lVector = []
```

Agora que já possuímos tanto o formato da matriz U e L, quanto a matriz A, podemos começar a rodar o algorítmo apresentado no enunciado deste EP, que é:

$u_1 = b_1$

para $i = 2, \cdots, n$ **faça**

$l_i = a_i / u_{i-1}$ (multiplicador)

$u_i = b_i - l_i c_{i-1}$

fim

Por ser uma matriz tridiagonal, não há a necessidade de passar por todos os itens da matriz para calcularmos os multiplicadores, apenas aqueles que não possuem 0 no item acima. Além disso, conseguimos utilizar os vetores a, b e c para melhorar a intermpretação dos cálculos. Logo, o código para acharmos a matriz L e U possue a seguinte estrutura:

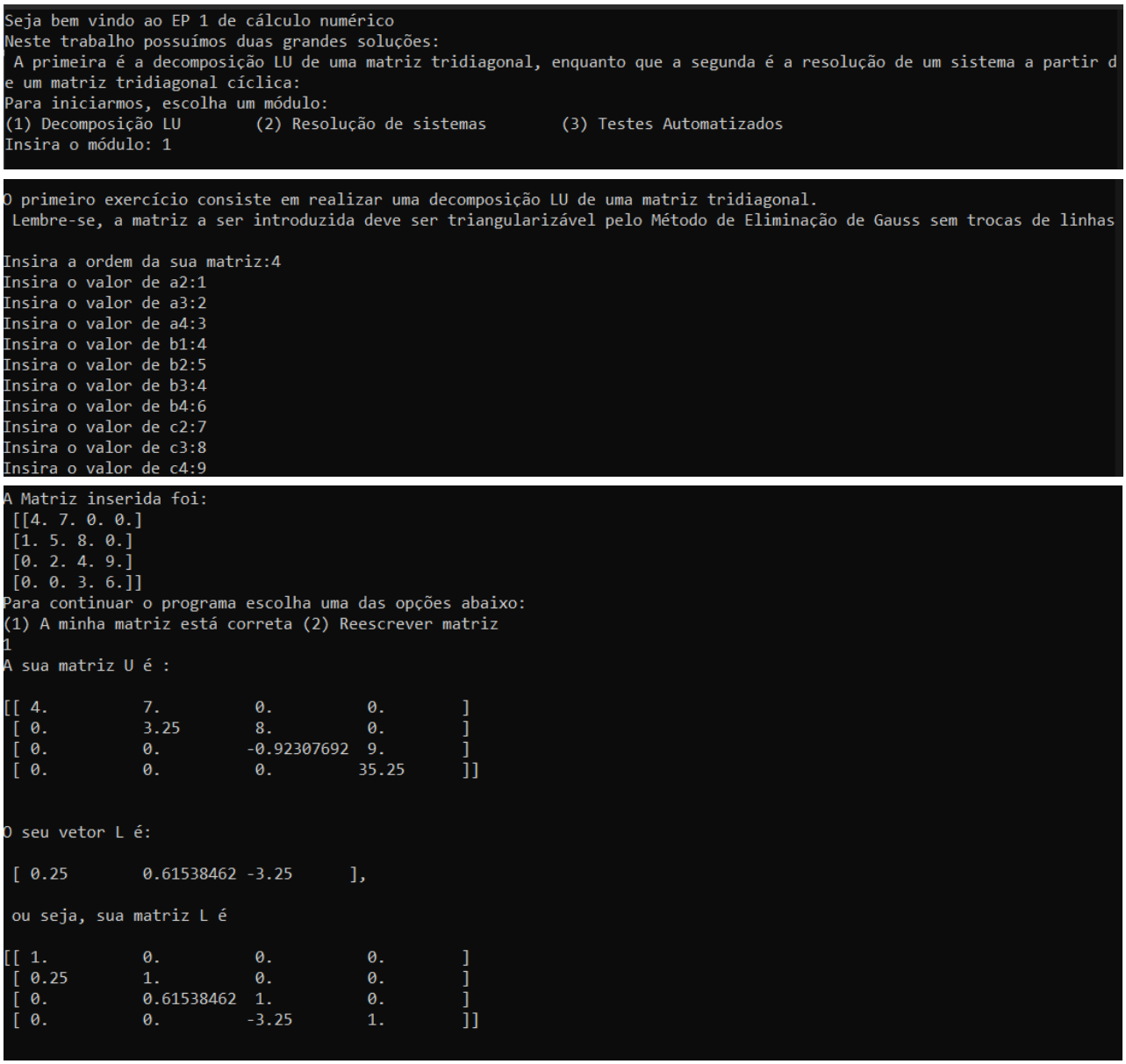
```
line = 1
column = 0

while line < ordemMatrix:
    multi = matrix[line][column]/matrix[line-1][column]
    lMatrix[line][column] = multi
    lVector.append(multi)
    for i in range(2):
        matrix[line][column] = matrix[line][column] - multi*matrix[line-1][column]
        column = column + 1
        if line == column:
            uMatrix[line][column] = vectorb[line] - multi*vectorc[line-1]
    line = line + 1
    column = line - 1
```

Observe que sempre estamos caminhando uma coluna a menos quando comparado à linha, isso porque não faz sentido passar pelos itens que já foram zerados.

Por fim, aplicando esse algorítmo chegamos no resultado final, com a função retornando tanto a matrix U quando a L.

Segue abaixo um exemplo de funcionamento da primeira solução:



```
Seja bem vindo ao EP 1 de cálculo numérico
Neste trabalho possuímos duas grandes soluções:
 A primeira é a decomposição LU de uma matriz tridiagonal, enquanto que a segunda é a resolução de um sistema a partir d
e um matriz tridiagonal cíclica:
Para iniciarmos, escolha um módulo:
(1) Decomposição LU        (2) Resolução de sistemas        (3) Testes Automatizados
Insira o módulo: 1

O primeiro exercício consiste em realizar uma decomposição LU de uma matriz tridiagonal.
 Lembre-se, a matriz a ser introduzida deve ser triangularizável pelo Método de Eliminação de Gauss sem trocas de linhas

Insira a ordem da sua matriz:4
Insira o valor de a2:1
Insira o valor de a3:2
Insira o valor de a4:3
Insira o valor de b1:4
Insira o valor de b2:5
Insira o valor de b3:4
Insira o valor de b4:6
Insira o valor de c2:7
Insira o valor de c3:8
Insira o valor de c4:9

A Matriz inserida foi:
 [[4. 7. 0. 0.]
 [1. 5. 8. 0.]
 [0. 2. 4. 9.]
 [0. 0. 3. 6.]]
Para continuar o programa escolha uma das opções abaixo:
(1) A minha matriz está correta (2) Reescrever matriz
1
A sua matriz U é :

[[ 4.          7.          0.          0.        ]
 [ 0.          3.25        8.          0.        ]
 [ 0.          0.         -0.92307692  9.        ]
 [ 0.          0.          0.         35.25      ]]


O seu vetor L é:

 [ 0.25        0.61538462 -3.25      ],

 ou seja, sua matriz L é

[[ 1.          0.          0.          0.        ]
 [ 0.25        1.          0.          0.        ]
 [ 0.          0.61538462  1.          0.        ]
 [ 0.          0.         -3.25        1.        ]]
```

Solução 2 - Resolução de sistemas:

A solução dois, por sua vez, é dependente do entendimento da solução 1, já que é necessário encontrarmos a matriz U e L para resolvermos um sistema tridiagonal cícliclo.

Temos o mapeamento dos itens da matriz A e do vetor d a partir da função “mapVectorSecond()” mostrada a seguir:

```python
def mapVectorSecond():
    correctMatrix = False

    ordemMatrix = int(input("Insira a ordem da sua matriz:"))

    while correctMatrix == False:

        vectora = []
        vectorb = []
        vectorc = []
        matrix = newSquareMatrix(ordemMatrix)
        vectord = []

        #Mapeamento da diagonal a
        i = 0
        while i < ordemMatrix :
            numberCell = float(input(f'Insira o valor de a{i+1}:'))
            vectora.append(numberCell)

            i = i + 1
```

```python
        #Mapeamento da diagonal b
        i = 0
        while i < ordemMatrix :
            numberCell = float(input(f'Insira o valor de b{i+1}:'))
            vectorb.append(numberCell)

            i = i + 1

        #Mapeamento da diagonal c
        i = 0
        while i < ordemMatrix :
            numberCell = float(input(f'Insira o valor de c{i+1}:'))
            vectorc.append(numberCell)

            i = i + 1

        #Mapeamento do vetor d
        print("Muito bem, agora que você já inseriu a sua matriz A, siga os próximos passos para o mapeamento do veto
        for i in range(len(matrix) ):
            itemd = float(input(f'Insira o item d{i+1} do seu vetor d: '))
            vectord.append(itemd)
```

```python
        addVectorOnMatrix("a", vectora, matrix)
        addVectorOnMatrix("b", vectorb, matrix)
        addVectorOnMatrix("c", vectorc, matrix)

        print(f'A Matriz inserida foi: \n {numpy.array(matrix)} \n \n e o vetor d foi: {numpy.array(vectord)}')
        print('Para continuar o programa escolha uma das opções abaixo:')
        validate = int(input('(1) Os meus dados estão corretos (2) Reescrever dados\n'))
        if validate != 2:
            correctMatrix = True
```

Após mapearmos os vetores e adicionarmos na Matriz A é necessário acahrmos a subMatriz T de tamanho (n-1)x(n-1) para que possamos solucionar o sistema. Para isso, basta excluirmos as ultimas linha e coluna da matriz A.

(obs: armazenamos a ultima coluna sem o ultimo item um vetor chamado de v e, análogamente, armazenamos a ultima linha sem o último item num vettor denominado de w)

A partir do momento que possuimos a matriz T, é ncessário encontrarmos suas respectivas matrizes U e L, ou seja, conseguimos reutilizar o algorítmo feito no primeiro exercício, obviamente tratando os novos dado

```
#Criação da estrutura da Matriz L      You, 4 days ago • add: functionSolveSystems …
lMatrix = newSquareMatrix(ordemMatrix - 1)
addVectorOnMatrix("b", [1]*(ordemMatrix-1), lMatrix)
lVector = []

#Criação da estrutura da Matriz U
uMatrix = newSquareMatrix(ordemMatrix - 1)
vectorcCopy = vectorc.copy()
del(vectorcCopy[-1])
del(vectorcCopy[-2])
addVectorOnMatrix("c", vectorcCopy, uMatrix)
uMatrix[0] = matrix[0].copy()
uMatrix[0].pop()
TMatrixCopy = numpy.copy(TMatrix)
```

```
#Achando as Matrizes U e L da Matriz T

line = 1
column = 0

while line < ordemMatrix - 1:
    multi = TMatrixCopy[line][column]/TMatrixCopy[line-1][column]
    lMatrix[line][column] = multi
    lVector.append(multi)
    for i in range(2):
        TMatrixCopy[line][column] = TMatrixCopy[line][column] - multi*TMatrixCopy[line-1][column]
        column = column + 1
        if line == column:
            uMatrix[line][column] = vectorb[line] - multi*vectorc[line-1]
    line = line + 1
    column = line - 1
```

Obtendo as respectivas matrizes L e U partimos para o algorítmo passado no EP para resolução de sistemas. Esta tarefa tem como objetivo encontrar os ^y e ^z, pois com eles será possível encontrar a solução do nosso sistema inicial.

$$T\tilde{y} = \tilde{d}$$

$$T\tilde{z} = v$$

Para encontrarmos os vetores citados anteriormente foi utilizado o algorítmo proposto pelo EP, que foi:

$Ly = d$:

$y_1 = d_1$
para $i = 2, \cdots, n$ **faça**
$y_i = d_i - l_i y_{i-1}$
fim

$Ux = y$:

$x_n = y_n / u_n$
para $i = n-1, \cdots, 1$ **faça**
$x_i = (y_i - c_i\, x_{i+1}) / u_i$
fim

Assim, traduzindo para a linguagem python temos:

```python
def SystemSolutionUL(lMatrix, uMatrix, vectorD):      You, 3 days ago
    vectorY = []
    vectorY.append(vectorD[0])

    for i in range(len(lMatrix) - 1):
        yi = vectorD[i+1] - lMatrix[i+1][i]*vectorY[i]
        vectorY.append(yi)

    vectorX = []
    vectorX.append(vectorY[-1]/uMatrix[-1][-1])

    for i in range(len(uMatrix) - 1):
        j = len(uMatrix) - 2 - i
        xi = (vectorY[j] - uMatrix[j][j+1]*vectorX[0])/uMatrix[j][j]
        vectorX.insert(0, xi)

    return vectorX
```

Observe que o algorítmo é achamado a partir de uma função, podendo ser utilizado para qualquer sistema tridiagonal.

```python
#Encontrando o vetor y~ e o vetor z~

vectordCopy = vectord.copy()
vectordCopy.pop()
vectory = SystemSolutionUL(lMatrix, uMatrix, vectordCopy)
vectorz = SystemSolutionUL(lMatrix, uMatrix, VVector)
```

Após encontrarmos ^y e ^z, basta seguirmos as recomendações de cálculos mostrados no enunciado:

$$x_n = \frac{d_n - c_n\tilde{y}_1 - a_n\tilde{y}_{n-1}}{b_n - c_n\tilde{z}_1 - a_n\tilde{z}_{n-1}} \quad e \quad \tilde{x} = \tilde{y} - x_n\tilde{z}$$

Que, transformando para a linguagem python, obtemos:

```
#Encontrando xn
xn = (vectord[-1] - (vectorc[-1]*vectory[0]) - (vectora[-1]*vectory[-2]))/
(vectorb[-1] - (vectorc[-1]*vectorz[0]) - (vectora[-1]*vectorz[-2]))

#Encontrando o x~
for i in range(len(vectorz)):
    vectorz[i] = vectorz[i]*xn
vectorx = subtractMatrixOneLine(vectory, vectorz)
vectorx.append(xn)


print(f'O seu vetor solução é: \n x = {numpy.array(vectorx)}')
```

Assim, o usuário obterá o vetor solução do sistema inicial.

Segue abaixo o funcionamento da segunda solução

```
Insira a ordem da sua matriz:5
Insira o valor de a1:1
Insira o valor de a2:1
Insira o valor de a3:2
Insira o valor de a4:3
Insira o valor de a5:4
Insira o valor de b1:4
Insira o valor de b2:5
Insira o valor de b3:4
Insira o valor de b4:6
Insira o valor de b5:7
Insira o valor de c1:7
Insira o valor de c2:8
Insira o valor de c3:9
Insira o valor de c4:10
Insira o valor de c5:11
Muito bem, agora que você já inseriu a sua matriz A, siga os próximos passos para o mapeamento do vetor d

Insira o item d1 do seu vetor d: 1
Insira o item d2 do seu vetor d: 2
Insira o item d3 do seu vetor d: 3
Insira o item d4 do seu vetor d: 4
Insira o item d5 do seu vetor d: 5
```

```
A Matriz inserida foi:
 [[ 4.  7.  0.  0.  1.]
 [ 1.  5.  8.  0.  0.]
 [ 0.  2.  4.  9.  0.]
 [ 0.  0.  3.  6. 10.]
 [11.  0.  0.  4.  7.]]

 e o vetor d foi: [1. 2. 3. 4. 5.]
Para continuar o programa escolha uma das opções abaixo:
(1) Os meus dados estão corretos (2) Reescrever dados
1
O seu vetor solução é:
 x = [ 0.19678328 -0.00861568  0.23078689  0.20940838  0.27317663]
```

Solução 3 – Testes automatizados

Basicamente na solução 3 o usuário possui a opção de realizar os testes 1 e 2 a partir de vetores a, b, c e d definidos no enunciado do EP, ou seja:

$$a_i = \frac{2i-1}{4i},\ 1 \leq i \leq n-1,\ a_n = \frac{2n-1}{2n},$$
$$c_i = 1 - a_i,\ 1 \leq i \leq n,$$
$$b_i = 2,\ 1 \leq i \leq n,$$

$$d_i = \cos\left(\frac{2\pi i^2}{n^2}\right),\ 1 \leq i \leq n.$$

Que, ao passarmos para o algorítmo, o

```python
def createVectors(number, module):
    vectora = []
    vectorb=[]
    vectorc=[]
    vectord = []
    for j in range(number):
        i = j + 1
        if i == number:
            ai = (2*number - 1)/(2*number)
        else:
            ai = (2*i - 1)/(4*i)
        ci = 1-ai
        vectora.append(ai)
        vectorc.append(ci)
        vectorb.append(2)
        if module == 2:
            di = math.cos((2*math.pi*(i**2))/(number**2))
            vectord.append(di)
```

btemos:

```
    ordemMatrix = len(vectorb)

    if module == 1:
        matrix = newSquareMatrix(len(vectora))
        vectora.pop()
        vectorc.pop()
        addVectorOnMatrix("a", vectora, matrix)
        addVectorOnMatrix("b", vectorb, matrix)
        addVectorOnMatrix("c", vectorc, matrix)
        return  matrix, vectora, vectorb, vectorc, ordemMatrix
    if module == 2:
        matrix = newSquareMatrix(len(vectora))
        addVectorOnMatrix("a", vectora, matrix)
        addVectorOnMatrix("b", vectorb, matrix)
        addVectorOnMatrix("c", vectorc, matrix)

        return  matrix, vectora, vectorb, vectorc, vectord, ordemMatrix
```

Exercícios:

1.

Sabendo-se da existência da decomposição LU, podemos obter os coeficientes de L e de U usando-se somente as propriedades dessas matrizes, sem termos de necessariamente fazer as contas na ordem da eliminação de Gauss. Note que, se $A = LU$, com L triangular inferior e U triangular superior, então

$$A_{ij} = \sum_{k=1}^{\min\{i,j\}} L_{ik}U_{kj}. \qquad \text{(exercício)}$$

Ao pensarmos numa matriz A, tridiagonal, começamos aplicar o método de Gauss da seguinte maneira.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & & & & \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix} \xrightarrow[/\ \frac{a_{21}}{a_{11}} = \mu_{21}]{L_2 - \frac{a_{21}}{a_{11}} \cdot L_1} \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22}-\mu_{21}\cdot a_{12} & a_{23}-\mu_{21}\cdot a_{12} & \dots & a_{2n}-\mu_{21}\cdot a_{1n} \\ \vdots & & & & \\ a_{1n} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix}$$

p/ a linha 3, fazemos $L_3 - \frac{a_{31}}{a_{11}} \cdot L_1$. p/ a linha 4: $L_4 - \frac{a_{41}}{a_{11}} \cdot L_4$, e assim por diante.

Observe que obtemos: $\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ \vdots & & & & \\ 0 & a'_{n2} & a'_{n3} & \dots & a'_{nn} \end{pmatrix} = A'$

Logo podemos escrever: $A \cdot \tilde{L_1} = A'$ / $\tilde{L_1} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -\mu_{21} & 1 & 0 & \dots & 0 \\ \vdots & & & & \\ -\mu_{n1} & 0 & 0 & \dots & 1 \end{pmatrix}$

/ $\mu_{n1} = \frac{a_{n1}}{a_{11}}$

Agora, podemos realizar as mesmas operações, entretanto comparando todas as linhas abaixo da 2 com a 2, da seguinte forma:

$$A' \xrightarrow[/\ \frac{a'_{32}}{a'_{22}} = \mu_{32}]{L_3 - \mu_{32} L_2} \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} - \mu_{32}\cdot a'_{23} & \dots & a'_{3n} - \mu_{32}\cdot a_{2n} \\ \vdots & & & & \\ 0 & a'_{n2} & a'_{n3} & \dots & a'_{nn} \end{pmatrix}$$

Seguindo a mesma lógica p/ todas as linhas, obtemos:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & a''_{33} & \dots & a''_{3n} \\ \vdots & & & & \\ 0 & 0 & a''_{n3} & \dots & a''_{nn} \end{pmatrix} = A'' \quad \text{tal que: } A'' = A' \cdot \tilde{L_2}$$

com $\tilde{L_2} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & -\mu_{32} & 1 & \dots & 0 \\ \vdots & & & & \\ 0 & -\mu_{n2} & 0 & \dots & 1 \end{pmatrix}$

ou seja $A \cdot \tilde{L_1} \cdot \tilde{L_2} \cdot \tilde{L_3} \cdot \dots \cdot \tilde{L_n} = U$ se continuarmos o processo p/ todas as linhas, tal que U é a matriz escalonada. Definimos $\prod_{i=1}^{n} \tilde{L_i} = (L)^{-1}$

$$\Rightarrow A \cdot L^{-1} = U \Rightarrow A \underbrace{L^{-1} \cdot L}_{I} = U \cdot L \Rightarrow A \cdot I = L \cdot U \Rightarrow A = L \cdot U,$$

ou seja: $A_{ij} = \sum_{k=1}^{\min\{i,j\}} L_{ik} \cdot U_{kj}$

Usando-se também o fato de que $L_{ii} = 1$, os coeficientes podem ser calculados em uma ordem diferente do Método de Eliminação de Gauss da seguinte maneira (exercício):

para $i = 1, \cdots, n$ **faça**

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj}, \quad j = i, \cdots, n \tag{1}$$

$$L_{ji} = \left(A_{ji} - \sum_{k=1}^{i-1} L_{jk} U_{ki} \right) / U_{ii}, \quad j = i+1, \cdots, n \tag{2}$$

fim

2.

Suponha uma matriz $A_{n\times n}$. Vamos acompanhar seu escalonamento:

(tridiagonal)

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ \vdots & & & & & \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{pmatrix} \rightarrow \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ L_{21} & a_{22}-L_{21}a_{12} & a_{23}-L_{21}\cdot a_{13} & \cdots & a_{2n}-L_{21}\cdot a_{1n} \\ L_{31} & a_{32}-L_{31}a_{12} & a_{33}-L_{31}\cdot a_{13} & \cdots & a_{32}-L_{2v}a_{13} \\ \vdots & & & & \\ L_{n1} & a_{n2}-L_{n1}\cdot a_{12} & a_{n3}-L_{n1}\cdot a_{13} & \cdots & a_{nn}-L_{n1}\cdot a_{1n} \end{pmatrix}$$

Podemos escrever a última matriz como:

$$\begin{pmatrix} v_{11} & v_{12} & v_{13} & \cdots & v_{1n} \\ L_{21} & v_{22} & v_{23} & \cdots & v_{2n} \\ L_{31} & a_{32}-L_{31}v_{12} & a_{33}-L_{31}v_{13} & \cdots & a_{3n}-L_{31}\cdot v_{1n} \\ L_{n1} & a_{n2}-L_{n1}v_{12} & a_{n3}-L_{n1}\cdot v_{13} & \cdots & a_{nn}-L_{n1}\cdot v_{1n} \end{pmatrix}$$

e $L_{n1} = \dfrac{a_{n1}}{v_{11}}$

e $L_{32} = \dfrac{a_{32}-L_{31}v_{12}}{v_{22}}$

Vamos realizar mais uma etapa:

$$\begin{pmatrix} v_{11} & v_{12} & v_{13} & \cdots & v_{1n} \\ L_{21} & v_{22} & v_{23} & \cdots & v_{2n} \\ L_{31} & L_{32} & a_{33}-L_{31}\cdot v_{13}-L_{32}\cdot v_{23} & \cdots & a_{3n}-L_{31}v_{1n}-L_{32}\cdot v_{2n} \\ \vdots & & & & \\ L_{n1} & L_{n2} & a_{n3}-L_{31}v_{13}-L_{n2}\cdot v_{23} & \cdots & a_{nn}-L_{n1}\cdot v_{1n}-L_{n2}\cdot v_{2n} \end{pmatrix}$$

Assim, observe que conforme vai vamos escalonando a matriz, encontramos um padrão, tal que: $v_{33} = a_{33}-L_{31}v_{13}-L_{32}\cdot v_{23} = a_{33}-\sum_{i=1}^{2} L_{3i}v_{i3}$

Ou seja, de maneira geral, para qualquer célula da matriz V, podemos escrever: $v_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik}v_{kj}$, $j = i, \cdots, n$

Ao observarmos o multiplicador, temos que: $L_{32} = \dfrac{a_{32}-L_{31}v_{12}}{v_{22}}$, ou seja, podemos escrever: $L_{ji} = \dfrac{\left(A_{ji} - \sum_{k=1}^{i-1} L_{jk}v_{ki}\right)}{v_{ii}}$, $j = i+1, \cdots, n$

O armazenamento também pode ser feito de forma eficiente, sendo desnecessário guardar os valores que sabemos que são nulos. A matriz tridiagonal

$$A = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{bmatrix}$$

pode ser armazenada em três vetores

$$a = (0, a_2, \ldots, a_{n-1}, a_n), \quad b = (b_1, b_2, \ldots, b_{n-1}, b_n) \quad c = (c_1, c_2, \ldots, c_{n-1}, 0)$$

e os coeficientes $u_i = U_{ii}$ e $l_{i+1} = L_{i+1,i}$ da decomposição LU podem ser calculados pelo algoritmo (exercício)

$u_1 = b_1$

para $i = 2, \cdots, n$ **faça**

$l_i = a_i / u_{i-1}$ (multiplicador)

$u_i = b_i - l_i c_{i-1}$

fim

3. sendo possível armazená-los também em vetores. Lembre-se que $U_{i,i+1} = c_i$.

Suponha que temos uma matriz tridiagonal, ou seja: $\begin{pmatrix} b_1 & c_1 & 0 & \dots & 0 \\ a_1 & b_2 & c_2 & \dots & 0 \\ 0 & a_2 & b_3 & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & \dots & a_n & b_n \end{pmatrix}$

Ao iniciarmos o processo de escalonamento observamos os seguintes fatos:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ a_1 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & \dots & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & a_n & b_n \end{pmatrix} \xrightarrow{L_2 - \mu_{21} L_1} \begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ 0 & b_2 - \mu_{21} c_1 & c_2 - \mu_{21} \cdot 0 & 0 & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & \dots & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & a_n & b_n \end{pmatrix}$$

1. b_1 se mantém constante, ou seja, $u_1 = b_1$ / $v_i = v_{i,i}$.
2. Toda célula que possui 0 acima se mantém constante, ou seja, toda célula da diagonal secundária superior ($v_{i,i+1} = c_i$)

Assim, já podemos uma previsão do formato de U, ou seja, $\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ 0 & \mu_{22} & c_2 & 0 & \dots & 0 \\ 0 & 0 & \mu_{33} & c_3 & \dots & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & & \mu_{n,n-1,\mu_n} \end{pmatrix}$

Vamos realizar mais uma etapa do escalonamento:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ 0 & b_2 - \mu_{21} c_1 & c_2 & 0 & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & \dots & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & a_n & b_n \end{pmatrix} \xrightarrow[\mu_{32} = \frac{a_2}{b_2 - \mu_{21} c_1}]{L_3 - \mu_{32} \cdot L_2} \begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 \\ 0 & b_2 & c_2 & 0 & \dots & 0 \\ 0 & 0 & b_3 - \mu_{32} \cdot c_2 & c_3 & \dots & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & a_n & b_n \end{pmatrix}$$

Observe que o próximo multiplicador, ou seja, da troca $L_4 - \mu_{43} L_3$ será: $\mu_{43} = \dfrac{a_3}{b_3 - \mu_{32} \cdot c_2}$, ou seja, podemos escrever $\mu_{i,i-1} = \dfrac{a_i}{b_{i-1} - \mu_{i-1,i-2} \cdot c_{i-2}}$

Adicionais, conseguimos definir o denominador como $v_{i,i}$, ou melhor, neste caso, $v_{i-1,i-1} = v_{i-1}$

Logo, para encontrar o multiplicador podemos aplicar o seguinte algoritmo:

$$\mu_{i,i-1} = l_i = \frac{a_i}{v_{i-1}}$$

Da mesma maneira, vamos observar v_i. Este segue o seguinte formato:

$u_i = b_i - \mu_{i,i-1} \cdot c_{i-1}$, ou seja $u_i = b_i - l_i \cdot c_{i-1}$