EP1 - Decomposição LU para matrizes tridiagonais

Bruno Matutani Santos - 11804682
Ligia Corunha Palma - 11352268

30 de Abril de 2022

1 Introdução

Este relatório possui como objetivo descrever as atividades e análises realizadas para decomposição LU de matrizes tridiagonais. Dito isso, nesse exercício computacional, serão desenvolvidas e estudados três algoritmos, sendo eles: decomposição LU (geral), decomposição de matrizes tridiagonais e decomposição de matrizes tridiagonais cíclicas.

2 Resolução e testes

2.1 Exercício 1 - decomposição LU (geral)

2.1.1 Resolução

Este trecho do exercício computacional possui como objetivo encontrar as matrizes triangulares superior (U) e inferior (L) que contém os multiplicadores necessários para decompor uma matriz, de forma que:

$$A = LU$$

Para implementar o algoritmo que realiza tal tarefa, foram criadas duas matrizes (U e L) n x n, sendo n a dimensão da matriz A a ser decomposta. Essas matrizes foram inicializadas preenchidas com zeros e com a matriz identidade, respectivamente.

```
def decLU(A,n):
    U = np.zeros((n,n))
    L = np.eye(n)
```

Figure 1: Criação das matrizes L e U

Em seguida, as matrizes L e U foram preenchidas com os coeficientes calculados por meio das seguintes fórmulas:

Primeira iteração(i=0):

$$U[i, i:n] = A[i, i:n]$$

$$L[(i+1):n, i] = (1/(U[i, i])).(A[(i+1):n, i])$$

Demais iterações(i=1,2,3...n-1):

$$U[i, i:n] = A[i, i:n] - (L[i, 0:(i)].U[0:(i), i:n])$$

Demais iterações(i=1,2,3...n-1) se $i+1 \neq n$:

$$L[(i+1):n, i] = (1/(U[i, i])).(A[(i+1):n, i] - (L[(i+1):n, 0:(i)].U[0:(i), i]))$$

Após o preenchimento das matrizes, são retornados os valores de L e U resultantes da decomposição.

2.1.2 Testes Exercício 1

Dito isso, para testar o algoritmo foram utilizados 2 casos de teste:

$$A_1 = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & -1 \\ 2 & -1 & 1 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 1 & -0.9 & 0 & 4.1 \\ 0.8 & 100 & -1.18 & -2.9 \\ 22 & -11 & 110 & 1.1 \\ 19 & 0 & 18 & 9 \end{bmatrix}$$

Para o caso de A_1:

O programa criado gerou as seguintes matrizes:

$$L_1 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix} \quad U_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -3 \\ 0 & 0 & 8 \end{bmatrix}$$

Multiplicando as matrizes, obtêm-se:

$$L_1.U_1 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -3 \\ 0 & 0 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & -1 \\ 2 & -1 & 1 \end{bmatrix} = A_1$$

Logo,a decomposição da matriz A_1 resultou nas matrizes L_1 e L_2 corretas.

Para o caso de A_2:

O programa criado gerou as seguintes matrizes:

$$L_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.8 & 1 & 0 & 0 \\ 22 & 0.08737093 & 1 & 0 \\ 19 & 0.1697776 & 0.16530268 & 1 \end{bmatrix} \quad U_2 = \begin{bmatrix} 1 & -0.9 & 0 & 4.1 \\ 0 & 100.72 & -1.18 & -6.18 \\ 0 & 0 & 110.1030977 & -88.56004766 \\ 0 & 0 & 0 & -53.21156085 \end{bmatrix}$$

Multiplicando as matrizes, obtêm-se:

$$L_2.U_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.8 & 1 & 0 & 0 \\ 22 & 0.08737093 & 1 & 0 \\ 19 & 0.1697776 & 0.16530268 & 1 \end{bmatrix} . \begin{bmatrix} 1 & -0.9 & 0 & 4.1 \\ 0 & 100.72 & -1.18 & -6.18 \\ 0 & 0 & 110.1030977 & -88.56004766 \\ 0 & 0 & 0 & -53.21156085 \end{bmatrix}$$

$$L_2.U_2 = \begin{bmatrix} 1 & -0.9 & 0 & 4.1 \\ 0.8 & 100 & -1.18 & -2.9 \\ 22 & -11 & 110 & 1.1 \\ 19 & 0 & 18 & 9 \end{bmatrix} = A_2$$

A multiplicação entre as matrizes L_1 e L_2 resultaram em A_2, o que demonstra de que a decomposição da matriz analisada está correta.

2.2 Exercício 2 - decomposição LU de matrizes tridiagonais

2.2.1 Resolução

Neste exercício, foi programado um algoritmo capaz de realizar a decomposição LU de matrizes tridiagonais, de maneira mais eficiente que a fórmula geral. Para implementar este algoritmo mais eficiente, utilizam-se as propriedades das matrizes tridiagonais para reduzir o número de operações realizadas para o cálculo de L e U, uma vez que não é necessário calcular os resultados nulos das matrizes.

Além disso, para facilitar a armazenagem da matriz A e o cálculo de LU, criam-se os vetores a, b e c (unidimensionais), que armazenarão os elementos pertencentes a diagonal principal, a diagonal inferior e a diagonal superior. Para a implementação do algoritmo em Python, primeiro inicializam-se os vetores a,b,c (sendo o primeiro elemento de a o número 0).

```
def MatrizesTridiagonais(A, d, n_tam):
    #---Criação dos vetores a,b e c (separação de A em 3 vetores)
    a = np.array([]) #Arrays Unidimensionais
    b = np.array([])
    c = np.array([])
    a = np.append(a,[0]) #Adicionando o primeiro elemento de a
```

Figure 2: Criação e inicialização dos vetores a,b,c

Após esse processo, preenchem-se os vetores criados com os elementos de A correspondentes a: diagonal principal (b), não-nulos acima da diagonal principal(c) e não-nulos abaixo da diagonal principal (a). Também adiciona-se a c o seu último elemento (o número 0).

```
for i in range(n_tam):
    for j in range (n_tam):
        if (i==j): #Elemento da diagonal principal
            b = np.append(b,A[i,j]) #Adiciona elemento no fim da lista
        if (i == j-1):
            c = np.append(c,A[i,j])
        if (i == j+1):
            a = np.append(a,A[i,j])

c = np.append(c,0)
```

Figure 3: Preenchimento dos vetores a,b,c

Depois da criação de a, b e c, os vetores u e l são inicializados com o primeiro elemento de b e o número 0, respectivamente. Em seguida, u e l são preenchidos utilizando-se as fórmulas para os elementos de l e u demonstradas na figura a seguir:

```
#---Decomposição LU da matriz tridiagonal
u = np.array([b[0]])
l = np.array([0])

for v in range(1,n_tam):
    l = np.append(l,a[v]/u[v-1])
    u = np.append(u,b[v] -(l[v]*c[v-1]) )
```

Figure 4: Decomposição LU do exercício 2

Tendo-se L e U armazenados, resolve-se a equação

$$Ly = d$$

Sendo d um vetor dos termos independentes fornecido pelo usuário. Para calcular y, inicia-se y[0] com o valor do primeiro elemento de d (d[0]). Os outros elementos de y são calculados conforme a fórmula presente dentro do loop da figura a seguir:

```
#---Solucao do sistema de equações
y = np.array([d[0]])

#Cálculo de y
for k in range(1,n_tam):
    y = np.append(y, d[k]- (l[k]*y[k-1]))
```

Figure 5: Cálculo de y

Por último, para encontrar a solução final do sistema linear tridiagonal, calcula-se o vetor x (resultados), de forma que:

$$Ux = y$$

Tal vetor é preenchido começando pela última posição $(n-1)$ e finalizando na primeira. Os valores que serão utilizados para o preenchimento de x são calculados por meio da fórmula presente no loop mostrado na figura a seguir:

Assim, o vetor contendo o resultado do sistema (x) é retornado.

```python
#Cálculo de x (resultado)
x = np.zeros(n_tam)
x[n_tam-1] = y[n_tam-1]/u[n_tam-1]

for h in reversed(range(0,n_tam-1)):
    x[h]= (y[h]-c[h]*x[h+1])/(u[h])

return x
```

Figure 6: Cálculo de x (vetor contendo os resultados)

2.2.2 Testes Exercício 2

A fim de testar essa função, multiplicou-se a matriz A pelo vetor x (calculado nessa equação) e verificou-se se o resultado era igual ao vetor d, visto que Ax = d. Se o resultado fosse diferente, mostraria que x estava errado, visto que A e d são fixos durante a execução. Foram feitos 2 testes diferentes, os quais podem ser vistos na figura 7 e na figura 8 . Nas imagens, pode-se ver que o valor de A*x é, de fato, igual ao valor de d.

```
X:
[ 1.00000000e+00  2.00000000e+00 -1.00000000e+00 -3.55271368e-16
  1.00000000e+00]
A =
[[2. 1. 0. 0. 0.]
 [1. 2. 1. 0. 0.]
 [0. 1. 2. 1. 0.]
 [0. 0. 1. 2. 1.]
 [0. 0. 0. 1. 2.]]
A*x =
[4.00000000e+00 4.00000000e+00 2.22044605e-16 0.00000000e+00
 2.00000000e+00]
d =
[4. 4. 0. 0. 2.]
```

Figure 7: Primeiro teste para o exercćio 2

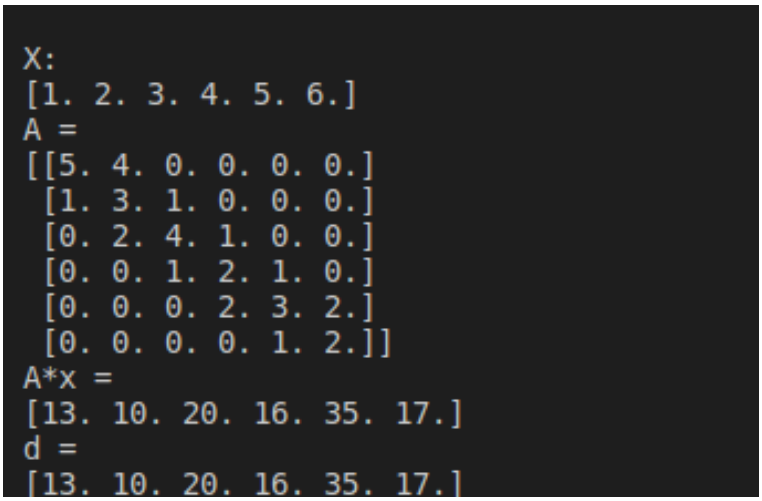


Figure 8: Segundo teste para o exercício 2

2.3 Exercício 3 - sistemas tridiagonais cíclicos

2.3.1 Resolução

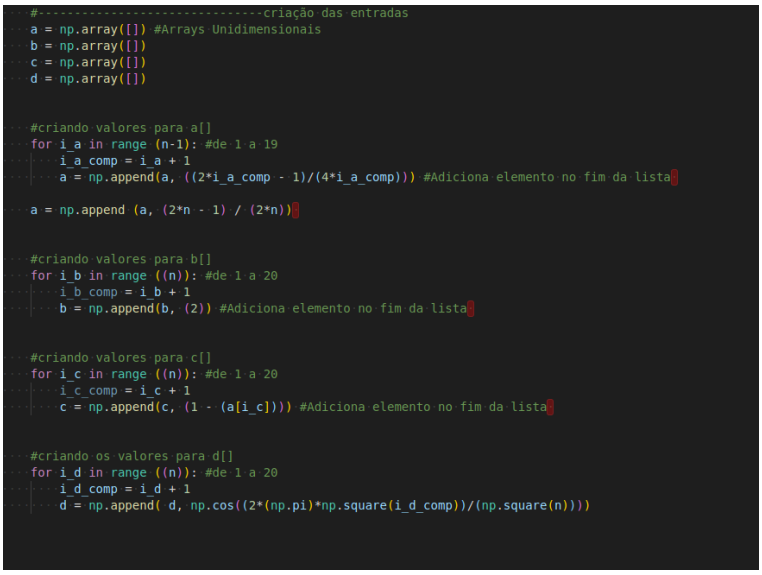
Neste último exercício, o objetivo é a resolver um sistema tridiagonal cíclico. Para criar a solução, primeiro foi criado o código para o caso teste, que foi base do código para o caso genérico.

Para essa resolução, foi criada uma função do código criado na seção 2.2 (decomposição LU de matrizes tridiagonais), a fim de otimizar o processo de solução das matrizes tridiagonais criadas a partir de A.

O primeiro passo foi a criação dos elementos dos vetores a, b, c e d a partir das equações dadas no enunciado (figura 9). A partir deles, foram criadas as matrizes A e T (matriz A sem a última linha e a última coluna) (figura 10). Os vetores v, w, e d' (d' é o d til) foram criados a partir das expecificações do enunciado (figura 11). Após a realização de cada passo, todos os novos valores eram checados por meio de prints.

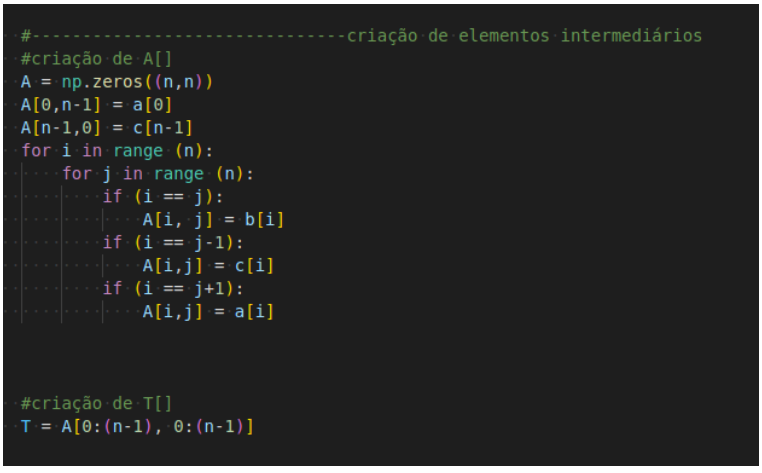
Em seguida, foi criada a parte do código que calcula z til e y til (chamados de z' e y') . Na primeira tentativa, foi copiada a parte final da segunda parte duas vezes, uma vez modificada para resolver a equação Ty' = d' e outra para resolver Tz' = v. Logo em seguida, foram criadas as equações para os valores do vetor de saída x (figura 13). Quando o código foi executado, os elementos de x tinham valores com ordem de grandeza muito distante do esperado. Então, fizemos a multiplicação A*x (usando a linguagem Python), a resposta foi diferente de d e assim tivemos a certeza de que o código estava errado. Por conta da aferição realizada antes da criação de y' e z', soube-se que o erro se encontrava nessa segunda parte do código.

Como já era sabido da existência do erro e havia uma noção de onde estava,



```
#------------------------------criação das entradas
a = np.array([]) #Arrays Unidimensionais
b = np.array([])
c = np.array([])
d = np.array([])


#criando valores para a[]
for i_a in range (n-1): #de 1 a 19
    i_a_comp = i_a + 1
    a = np.append(a, ((2*i_a_comp - 1)/(4*i_a_comp))) #Adiciona elemento no fim da lista

a = np.append (a, (2*n - 1) / (2*n))


#criando valores para b[]
for i_b in range ((n)): #de 1 a 20
    i_b_comp = i_b + 1
    b = np.append(b, (2)) #Adiciona elemento no fim da lista


#criando valores para c[]
for i_c in range ((n)): #de 1 a 20
    i_c_comp = i_c + 1
    c = np.append(c, (1 - (a[i_c]))) #Adiciona elemento no fim da lista


#criando os valores para d[]
for i_d in range ((n)): #de 1 a 20
    i_d_comp = i_d + 1
    d = np.append( d, np.cos((2*(np.pi)*np.square(i_d_comp))/(np.square(n))))
```

Figure 9: Criação das entradas do exercício 3.



```
#--------------------------------criação de elementos intermediários
#criação de A[]
A = np.zeros((n,n))
A[0,n-1] = a[0]
A[n-1,0] = c[n-1]
for i in range (n):
    for j in range (n):
        if (i == j):
            A[i, j] = b[i]
        if (i == j-1):
            A[i,j] = c[i]
        if (i == j+1):
            A[i,j] = a[i]



#criação de T[]
T = A[0:(n-1), 0:(n-1)]
```

Figure 10: Criação das matrizes A e T do exercício 3.

```
#criação de v[]
v = np.array([])
v = np.append(v, a[0])
for i_v in range (1, n - 2): #de 1 a 19
    v = np.append(v, (0))
v = np.append (v, c[n-2])



#criação de w[]
w = np.array([])
w = np.append(w, c[n-1])
for i_w in range (1, n - 2): #de 1 a 19
    w = np.append(w, (0))
w = np.append (w, a[n-1])


#criação de d_linha[]
d_linha = np.array([])
for i_d_linha in range(n-1):
    d_linha = np.append (d_linha , d[i_d_linha])
```

Figure 11: Criação dos vetores v, w e d' do exercício 3.

essa parte do código foi revisada. Revisar as equações para os valores dos elementos de x era mais simples que revisar a criação dos elementos de y' e z'. Por isso foi decidido começar com x. Após aferir as equações algumas vezes, assumiu-se que estavam certas. Assim, o próximo passo foi checar y' e z', durante a realização do debug, percebeu-se que era mais simples criar uma função do exercício 2 para solucionar as equações Tz' = v e Ty' = d' (figura 12) que debugar o código escrito, pois estava com muitas variáveis e índices.

```
y_linha = np.array([])
z_linha = np.array([])
y_linha = MatrizesTridiagonais(T, d_linha, n-1)
z_linha = MatrizesTridiagonais(T, v, n-1)
```

Figure 12: Criação dos vetores y' e z' do exercício 3.

```
x = np.array([])

x_n = (d[n-1] - (c[n-1]*y_linha[0]) - (a[n-1] * y_linha[n-2] )) / (b[n-1] - (c[n-1]*z_linha[0]) - (a[n-1]*z_linha[n-2] ))

for var in range (n-1):
    x = np.append(x, y_linha[var] - x_n*z_linha[var])

x = np.append(x, x_n)
```

Figure 13: Criação dos vetor x do exercício 3.

Ao fazer isso e rodar o código, verificamos que a o vetor x estava correto, pois ao multiplicar Ax, a resposta dada era igual ao vetor d.

2.3.2 Testes Exercício 3

Para testar essa função, fizemos 2 testes diferentes: primeiro foi realizado o caso teste (n = 20) e depois para n = 30 (igual o caso teste, a única diferença é o tamamnho n da matriz). Esses dois valores foram escolhidos para mostrar que o caso teste funciona (n = 20) e que a função pode rodar para qualquer valor de n.

Para checar os valores, a matriz de entrada A foi multiplicada pelo vetor resposta x (figura 14) e verificou-se se o resultado era igual ao vetor d (Ax = d). Para n = 20, pode-se ver que Ax = d, como esperado (figura 15). O mesmo ocorre quando n = 30 (figura 16).

3 Main.py e interface

Após a criação separada de cada código (um para cada um dos três exercícios), foram criadas funções com cada um dos códigos, a fim de serem utilizados em

```
    resp = A@np.transpose(x)
    print("Ax = ")
    print(resp)
```

Figure 14: Multiplicação de A por x no teste do exercício 3.

```
Ax =
[ 9.99876632e-01  9.98026728e-01  9.90023658e-01  9.68583161e-01
  9.23879533e-01  8.44327926e-01  7.18126298e-01  5.35826795e-01
  2.94040325e-01  6.93889390e-17 -3.23917418e-01 -6.37423990e-01
 -8.83765630e-01 -9.98026728e-01 -9.23879533e-01 -6.37423990e-01
 -1.71929100e-01  3.68124553e-01  8.18149717e-01  1.00000000e+00]


d =
[ 9.99876632e-01  9.98026728e-01  9.90023658e-01  9.68583161e-01
  9.23879533e-01  8.44327926e-01  7.18126298e-01  5.35826795e-01
  2.94040325e-01  6.12323400e-17 -3.23917418e-01 -6.37423990e-01
 -8.83765630e-01 -9.98026728e-01 -9.23879533e-01 -6.37423990e-01
 -1.71929100e-01  3.68124553e-01  8.18149717e-01  1.00000000e+00]
```

Figure 15: Resultado de A*x quando n = 20.

```
Ax =
[ 9.99975631e-01  9.99610115e-01  9.98026728e-01  9.93767919e-01
  9.84807753e-01  9.68583161e-01  9.42057453e-01  9.01832526e-01
  8.44327926e-01  7.66044443e-01  6.63926213e-01  5.35826795e-01
  3.81070376e-01  2.01077921e-01  6.24500451e-17 -2.14735327e-01
 -4.32085749e-01 -6.37423990e-01 -8.13100761e-01 -9.39692621e-01
 -9.98026728e-01 -9.71961001e-01 -8.51726934e-01 -6.37423990e-01
 -3.42020143e-01  6.98126030e-03  3.68124553e-01  6.89619544e-01
  9.16362730e-01  1.00000000e+00]


d =
[ 9.99975631e-01  9.99610115e-01  9.98026728e-01  9.93767919e-01
  9.84807753e-01  9.68583161e-01  9.42057453e-01  9.01832526e-01
  8.44327926e-01  7.66044443e-01  6.63926213e-01  5.35826795e-01
  3.81070376e-01  2.01077921e-01  6.12323400e-17 -2.14735327e-01
 -4.32085749e-01 -6.37423990e-01 -8.13100761e-01 -9.39692621e-01
 -9.98026728e-01 -9.71961001e-01 -8.51726934e-01 -6.37423990e-01
 -3.42020143e-01  6.98126030e-03  3.68124553e-01  6.89619544e-01
  9.16362730e-01  1.00000000e+00]
```

Figure 16: Resultado de A*x quando n = 30.

um arquivo chamado main.py (figura 17).

Nesse arquivo, criou-se o código em que o usuário escolhe qual dos 3 exercício ele pretende resolver, o tamanho da matriz quadrada e os valores dos elementos. Ele também pode escolher resolver o caso teste pedido no enunciado do terceiro exercício (figura 18).

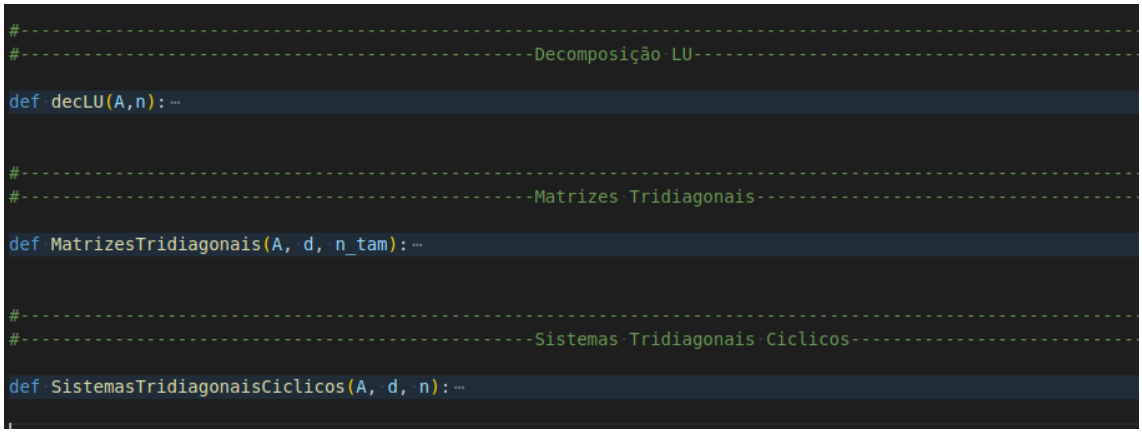


Figure 17: Funções moduláveis no arquivo main.py.

3.1 Opção 1

Caso o usuário escolha a opção 1, ele terá a opção de escolher o n (tamanho da matriz quadrada) e quais os valores dos elementos da matriz. Para passar esses valores, ele precisa preencher os valores de uma linha inteira antes de escolher os valores da próxima, de cima pra baixo e da esquerda pra direita; é necessário teclar 'enter' entre os valores (figura 19).

Após a criação da matriz A, a função decLU(A,n) é chamada (referente ao exercício 1). Os parâmetros passados são: a matriz A e o tamanho n (figura 20).

Por fim, as matrizes L e U são apresentadas para o usuário (figura 21).

3.2 Opção 2

Se a opção 2 for a escolhida, o usuário também deve escolher o tamanho n da matriz quadrada e o valor de seus elementos, igual feito na seção anterior (figura 22).

Além disso, também deve passar o valor dos elementos do vetor d, o qual tem tamanho n (figura 23). O programa, nesse momento chama a função MatrizesTridiagonais(A, d, n), a qual retorna o vetor x (solução do sistema Ax = d)(figura 24). O vetor x será apresentado para o usuário por meio da interface (figura 25).

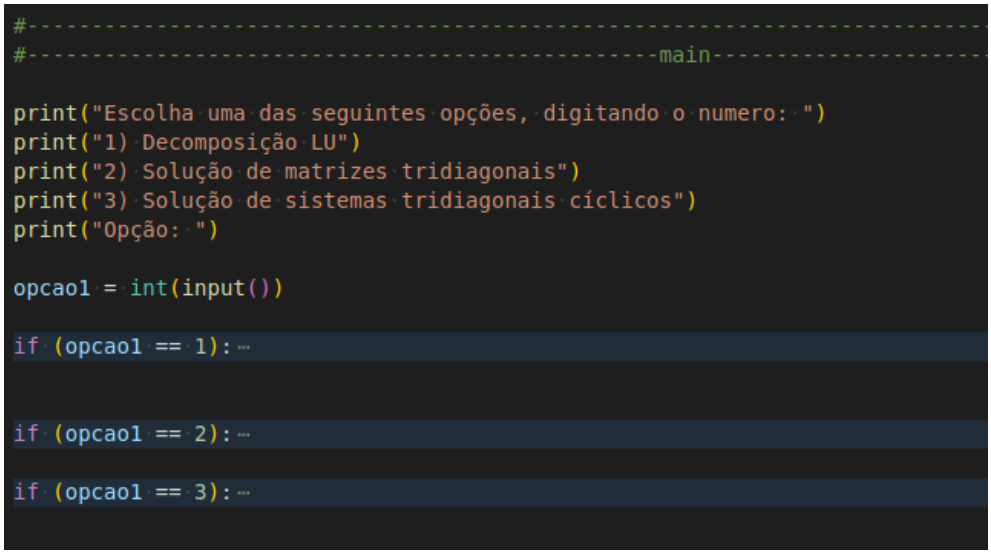


Figure 18: Primeiras opções do usuário no arquivo main.py.

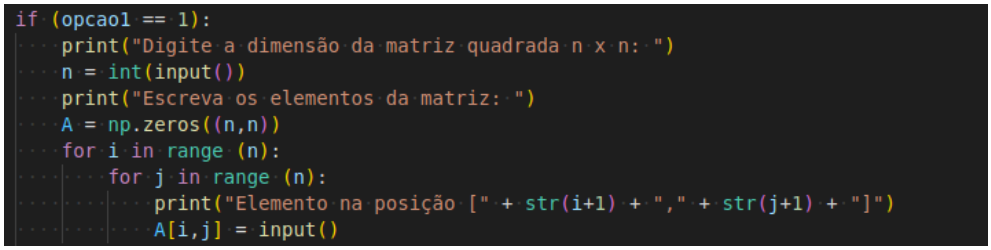


Figure 19: Criação da matriz A na opção 1

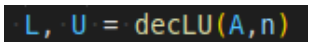


Figure 20: Chamada da função que resolve A = LU na opção 1

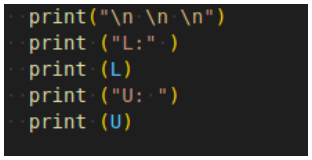


Figure 21: Print das matrizes L e U na opção 1

```
if (opcao1 == 2):
    print("Digite a dimensão da matriz quadrada n x n: ")
    n = int(input())
    print("Escreva os elementos da matriz: ")
    A = np.zeros((n,n))
    for i in range (n):
        for j in range (n):
            print("Elemento na posição [" + str(i+1) + "," + str(j+1) + "]")
            A[i,j] = input()
```

Figure 22: Criação da matriz A na opção 2.

```
    print("Digite o vetor d a ser utilizado: ")
    d = np.zeros(n)
    for i in range (n):
        print("Elemento na posição [" + str(i+1) + "]")
        d[i] = int(input())
```

Figure 23: Criação do vetor d na opção 2.

```
    x = MatrizesTridiagonais(A, d, n)
```

Figure 24: Chamada da função que calcula a solução do sistema Ax = d na opção 2.

```
    print ("\n \n \n")
    print ("X: ")
    print (x)
```

Figure 25: Print do vetor x na opção 2.

3.3 Opção 3

Quando essa opção é escolhida, o usuário deve fazer outra escolha logo em seguida: se ele quer resolver o caso teste, se ele quer resolver o caso teste mas com uma matriz com tamanho n arbitrário (escolhe mais adiante) ou se ele quer uma matriz arbitrária com tamanho arbitrário (figura 26).

```
if (opcao1 == 3):
    print("Escolha: ")
    print("1) Usar o caso teste do enunciado")
    print("2) Usar o caso teste, porém com outra dimensão")
    print("3) Escolher os valores e o tamanho da matriz")
    print("\n \n")
    print("Opção: ")
    opcao2 = int(input())
```

Figure 26: Escoha da opção na opção 3.

Se a opção 1 é escolhida, as os vetores a, b, c e d e a matriz A são criados (caso teste e tamnanho n = 20) (figura 27). Logo depois, a função SistemasTridiagonaisCiclicos(A, d, n) é chamada (figura 28). Essa função retorna x, o vetor solução do sistema Ax= D, que é apresentado ao usuário por meio da interface (figura 29).

Se a opção 2 é escolhida, tudo acontece exatamente igual ao o que acontece na opção 1, a única diferença é que no começo o usuário deve escolher o tamamho da matriz quadrada (n) (figura 30). As entradas serão criadas de acordo com as mesmas equações do caso teste (n=20).

Se a opção 3 é escolhida, o usuário deve passar para o programa o tamanho da matriz quadrada A (n), os valores dos elementos da matriz A (esquerda para direita, de cima a baixo) e também os valores do vetor d (figura 31). Ao passar os números, o usuário deve teclar 'enter' entre eles.

Após a criação da matriz e do vetor, a função MatrizesTridiagonais(A, d, n) é chamada (figura 28) e o vetor x recebe o vetor que a função retorna como solução de Ax = d. O programa mostra ao usuário o vetor solução 'x' por meio de prints na interface (figura 29).

```python
if (opcao2 == 1):
    #------------------------------criação das entradas
    a = np.array([]) #Arrays Unidimensionais
    b = np.array([])
    c = np.array([])
    d = np.array([])
    n = 20
    #criando valores para a[]
    for i_a in range (n-1): #de 1 a 19
        i_a_comp = i_a + 1
        a = np.append(a, ((2*i_a_comp - 1)/(4*i_a_comp))) #Adiciona elemento no fim da lista
    a = np.append (a, (2*n - 1) / (2*n))
    #criando valores para b[]
    for i_b in range ((n)): #de 1 a 20
        i_b_comp = i_b + 1
        b = np.append(b, (2)) #Adiciona elemento no fim da lista
    #criando valores para c[]
    for i_c in range ((n)): #de 1 a 20
        i_c_comp = i_c + 1
        c = np.append(c, (1 - (a[i_c]))) #Adiciona elemento no fim da lista
    #criando os valores para d[]
    for i_d in range ((n)): #de 1 a 20
        i_d_comp = i_d + 1
        d = np.append( d, np.cos((2*(np.pi)*np.square(i_d_comp))/(np.square(n))))
    #criação de A[]
    A = np.zeros((n,n))
    A[0,n-1] = a[0]
    A[n-1,0] = c[19]
    for i in range (n):
        for j in range (n):
            if (i == j):
                A[i, j] = b[i]
            if (i == j-1):
                A[i,j] = c[i]
            if (i == j+1):
                A[i,j] = a[i]
```

Figure 27: Criação das entradas na opção 1 da opção 3.

```python
x = SistemasTridiagonaisCiclicos(A, d, n)
```

Figure 28: Chamada da função na opção 1 da opção 3.

```python
print()
print()
print()
print ("X: " )
for i in range (n):
    print("X[" + str(i+1) + "] = " + str(x[i]))
```

Figure 29: Print do vetor solução x na opção 3.

```
    print()
    print("Escreva o tamanho da matriz quadrada")
    n = int(input())
```

Figure 30: Escolha do tamanho da matriz A na opção 2 da opção 3.

```
if (opcao2 == 3):
    print("Digite o número de linhas da matriz quadrada")
    n = int(input())
    print("Escreva os elementos da matriz:")
    A = np.zeros((n,n))
    for i in range (n):
        print("Linha número " + str(i+1))
        for j in range (n):
            A[i,j] = input()
    print("Escreva o vetor d: ")
    d = np.zeros(n)
    for i in range (n):
        d[i] = int(input())
```

Figure 31: Criação de A e d pelo usuário na opção 3 da opção 3.