Giovanna Girotto - 11803416
Vittor Braide Costa - 11806920

Relatório EP1 MAP3121 - Decomposição LU para Matrizes Tridiagonais

São Paulo, SP

Abril, 2022

Giovanna Girotto - 11803416
Vittor Braide Costa - 11806920

Relatório EP1 MAP3121 - Decomposição LU para Matrizes Tridiagonais

Relatório da Tarefa 1 - *Decomposição LU para Matrizes Tridiagonais* proposto pela disciplina MAP3121 - Métodos Numéricos e Aplicações

Escola Politécnica da Universidade de São Paulo

São Paulo, SP

Abril, 2022

Lista de ilustrações

Sumário

1 Introdução

O presente relatório tem como objetivo estudar a decomposição LU para matrizes tridiagonais. Para isso, primeiro apresenta-se o método de eliminação de Gauss, que possui ampla aplicação na resolução de sistemas de equações. A partir disso, chega-se no caso de matrizes que não necessitam trocas de linhas e nem demandam condensação pivotal para a estabilidade numérica quando triagularizadas - as quais podem sofrer decomposição LU.

Dessa forma, parte-se para o estudo e demonstração da decomposição LU, aplicando lógica de programação e atribuição de variáveis. É a partir disso que cria-se a base para prosseguir com o estudo de matrizes tridiagonais e a resolução de sistemas lineares a partir delas.

Assim, apresenta-se algumas das diversas aplicações que este método têm na realidade, de forma a entender seu contexto e sua importância computacional. Ao fim, mostra-se o código em linguagem Python para a decomposição LU de uma matriz A, bem como os vetores de sua matriz tridiagonal e o algoritmo para resolução de sistemas lineares.

2 Método de Eliminação de Gauss

O método de eliminação de Gauss consiste em uma das diversas maneiras de resolver um sistema de equações lineares. Para tanto, operações sucessivas são realizadas de modo a alterar o sistema, conservando a solução original. Nesse sentido, é possível visualizar a aplicação do método para o sistema de 3 equações abaixo:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{2.1}$$

Tais equações podem ser representadas na forma matricial:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2.2}$$

Ante o exposto, o método atuará na matriz A de coeficientes a_{ij}, tornando-a uma matriz escalonada. Assim sendo, o primeiro passo do método de Gauss consiste em manter a primeira equação e a aplicar algumas operações na segunda e terceira. Para a segunda equação, sendo $a_{11} \neq 0$ utiliza-se um coeficiente m_{21}, dado pela divisão de a_{21} por a_{11}, e subtrai-se a segunda equação pela primeira equação multiplicada pelo coeficiente m_{21}. Analogamente, o processo repete-se para a terceira equação, sendo o coeficiente, agora, m_{31}.

$$\begin{cases} i \ \rightarrow \ i \\ ii \ \rightarrow \ ii - m_{21}i \\ iii \ \rightarrow \ iii - m_{31}i \end{cases} \tag{2.3}$$

Como resultado, tem-se:

$$\begin{cases} i \ \rightarrow \ a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ ii \ \rightarrow \ \cancelto{0}{(a_{21} - \frac{a_{21}}{a_{11}}a_{11})}x_1 + (a_{22} - m_{21}a_{12})x_2 + (a_{23} - m_{21}a_{13})x_3 = b_2 - m_{21}b_1 \\ iii \ \rightarrow \ \cancelto{0}{(a_{31} - \frac{a_{31}}{a_{11}}a_{11})}x_1 + (a_{32} - m_{31}a_{12})x_2 + (a_{33} - m_{31}a_{13})x_3 = b_3 - m_{31}b_1 \end{cases} \tag{2.4}$$

É possível perceber que as operações realizadas transformaram a matriz A. Nesse sentido, podemos representar todas as alterações pela multiplicação da matriz original por uma outra matriz dada por:

$$\tilde{L} = \begin{bmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & 0 & 1 \end{bmatrix} \tag{2.5}$$

Em sequência, realiza-se uma operação semelhante com a matriz resultante da multiplicação $\tilde{L}A$. Substituindo os resultados obtidos em 2.4 pelas equações abaixo e realizando a operação também disponível a seguir, é possível encontrar a matriz escalonada final.

$$\begin{cases} i \;\rightarrow\; a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ ii - m_{21}i \;\rightarrow\; a'_{22}x_2 + a'_{23}x_3 = b'_2 \\ iii - m_{31}i \;\rightarrow\; a'_{32}x_2 + a'_{33}x_3 = b'_3 \end{cases} \tag{2.6}$$

$$\begin{cases} i \;\rightarrow\; a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ ii' \;\rightarrow\; a'_{22}x_2 + a'_{23}x_3 = b'_2 \\ iii' - m_{32}ii' \;\rightarrow\; \cancelto{0}{(a'_{32} - \tfrac{a'_{32}}{a'_{22}})}x_2 + (a'_{33} - m_{32}a'_{23})x_3 = (b'_3 - m_{32}b'_2) \end{cases} \tag{2.7}$$

Ao final, as operações realizadas podem ser escritas na forma matricial:

$$Ax = b \rightarrow \tilde{L}_2(\tilde{L}_1 A)x = \tilde{L}_2(\tilde{L}_1 b) \tag{2.8}$$

$$\tilde{L}_2 = \begin{bmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & 0 & 1 \end{bmatrix} \tag{2.9}$$

Por último, a solução do sistema é:

$$\begin{cases} x_3 = \frac{b''_3}{a''_{33}} \\ x_2 = \frac{1}{a'_{22}}(b'_2 - a'_{23}x_3) \\ x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{33}x_3) \end{cases} \tag{2.10}$$

Em suma, é fácil de identificar o potencial do método para resolução de sistemas, já que, ao final, com baixo esforço computacional, chega-se rapidamente à solução de um sistema de n equações.

3 Decomposição LU

Matrizes pertencentes a certas classes, como matrizes diagonais dominantes e matrizes simétricas definidas positivas, podem ser triangularizadas pelo Método de Eliminação de Gauss sem trocas de linhas e sem a necessidade de condensação pivotal para a estabilidade numérica.

Dessa forma, chamando de U a matriz triangular superior obtida da triangularização e L a matriz inferior com os multiplicadores $L_{ij} = \begin{cases} L_{ij}, i > j \\ 1, i = j \end{cases}$, a decomposição LU de uma matriz A triangularizável pelo Método de Eliminação de Gauss sem trocas de linhas é definida pela equação:

$$A = LU \tag{3.1}$$

Além disso, pode-se obter os coeficientes de L e de U sem a necessidade de fazer as contas na ordem da eliminação de Gauss. Se U é uma matriz triangular superior, então: $U_{ij} = 0$ para todo $i > j$.

Assim, a multiplicação LU fica:

$$A = L \cdot U \Rightarrow A = \begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} & \cdots & U_{1,j} \\ U_{2,1} & U_{2,2} & U_{2,3} & \cdots & U_{2,j} \\ U_{3,1} & U_{3,2} & U_{3,3} & \cdots & U_{3,j} \\ \vdots & \vdots & \vdots & \ddots & \\ U_{i,1} & U_{i,2} & U_{i,3} & \cdots & U_{i,j} \end{bmatrix} \cdot \begin{bmatrix} L_{1,1} & L_{1,2} & L_{1,3} & \cdots & L_{1,j} \\ L_{2,1} & L_{2,2} & L_{2,3} & \cdots & L_{2,j} \\ L_{3,1} & L_{3,2} & L_{3,3} & \cdots & L_{3,j} \\ \vdots & \vdots & \vdots & \ddots & \\ L_{i,1} & L_{i,2} & L_{i,3} & & L_{i,j} \end{bmatrix} \tag{3.2}$$

Aplicando os conceitos de multiplicação de matrizes, obtém-se:

$$A = \begin{bmatrix} L_{1,1}U_{1,1} & L_{1,1}U_{1,2} & L_{1,1}U_{1,3} & \cdots & L_{1,1}U_{1,j} \\ L_{2,1}U_{1,1} & L_{2,1}U_{1,2} + L_{2,2}U_{2,2} & L_{2,1}U_{1,3} + L_{2,2}U_{2,3} & \cdots & \sum_{k=1}^{j} L_{2,k}U_{k,j} \\ L_{3,1}U_{1,1} & L_{3,1}U_{1,2} + L_{3,2}U_{2,2} & L_{3,1}U_{1,3} + L_{3,2}U_{2,3} + L_{3,3}U_{3,3} & \cdots & \sum_{k=1}^{j} L_{3,k}U_{k,j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ L_{i,1}U_{1,1} & \sum_{k=1}^{i} L_{i,k}U_{k,2} & \sum_{k=1}^{i} L_{i,k}U_{k,3} & \cdots & \sum_{k=1}^{\min\{i,j\}} L_{i,k}U_{k,j} \end{bmatrix}$$

Generalizando, tem-se que:

$$\boxed{A = \sum_{k=1}^{\min\{i,j\}} L_{i,k}U_{k,j}} \tag{3.3}$$

Mas, usando o fato que $L_{ii} = 1$, conclui-se que o último termo da soma $\sum_{k=1}^{\min\{i,j\}} L_{i,k}U_{k,j}$ fica somente U_{ij}. Assim, retira-se tal termo da soma, resultando em:

$$A_{ij} = \sum_{k=1}^{i-1} L_{ik}U_{kj} + U_{ij} \Rightarrow \boxed{U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik}U_{kj}} \text{ onde } j = 1, \ldots, n \tag{3.4}$$

Mantendo a mesma nomenclatura de i e j, para calcular L é necessário inverter no somatório ij para ji, dado que são "matrizes simétricas" em relação à diagonal de termos não nulos. Para tal inversão, basta trocar i por j e j por i:

$$A_{ij} = \sum_{k=1}^{i} L_{ik}U_{kj} \rightarrow A_{ji} = \sum_{k=1}^{j} L_{jk}U_{ki} \tag{3.5}$$

No processo de reversão, $U_{ij} \rightarrow U_{ji} = 0$, fato que anula o último termo da equação acima:

$$A_{ji} = \sum_{k=1}^{j-1} L_{jk}U_{ki} + L_{jj}\cancelto{0}{U_{ji}} \Rightarrow A_{ji} = \sum_{k=1}^{j-2} L_{jk}U_{ki} + L_{j,(j-1)}U_{j-1,i} \tag{3.6}$$

Sabendo que os únicos termos de L não nulos são da forma $L_{i+1,i}$, substitui-se $j = 1 + 1$:

$$\sum_{k=1}^{j-2} L_{jk}U_{ki} + L_{j,(j-1)U_{j-1,i}} \rightarrow \sum_{k=1}^{i-1} L_{i+i,k}U_{ki} + L_{i+1,i}U_{i,i} \tag{3.7}$$

Assim, isolando $L_{i+1,i}$ e retomando $j = i + 1$:

$$\boxed{L_{ji} = (A_{ji} - \sum_{k=1}^{i-1} L_{jk}U_{ki})/U_{ii}} \text{ onde } j = i + 1, ..., n \tag{3.8}$$

Contudo, a decomposição *LU* pode ser feita de maneira eficiente e com um número de operações aritméticas reduzidas quando feita para uma **matriz tridiagonal**. A matriz A é tridiagonal se possui elementos diferentes de zero somente na diagonal principal e nas diagonais secundárias acima e abaixo da diagonal principal, ou seja, $a_{ij} = 0$ se $|i - j| > 1$. Define-se:

$$A = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{bmatrix} \tag{3.9}$$

Em que seu armazenamento pode ser feito em três vetores:

$$a = (0, a_2, \ldots, a_{n-1}, a_n) \tag{3.10}$$

$$b = (b_1, b_2, \ldots, b_{n-1}, b_n) \tag{3.11}$$

$$c = (c_1, c_2, \ldots, c_{n-1}, 0) \tag{3.12}$$

Consequentemente, os únicos elementos de U que podem ser não nulos são U_{ii} e $U_{i,i+1}$, e os únicos multiplicadores diferentes de zero são os do tipo $L_{i+1,i}$, uma vez que é válido que:

$$\text{Para } j > i+1 \Rightarrow L_{ij} = 0:$$

$$U_{ij} = \cancelto{0}{A_{ij}} - \sum_{k=1}^{i-1} \cancelto{0}{L_{ik}} U_{kj} \Rightarrow \boxed{U_{ij} = 0 \text{ para } j > i+1}$$

De maneira análoga, se $U_ij = 0$ para $i > j+1$:

$$L_{ij} = (\cancelto{0}{A_{ij}} - \sum_{k=1}^{i-1} L_{ik} \cancelto{0}{U_{kj}})/U_{ii} \Rightarrow \boxed{L_{ij} = 0 \text{ para } i > j+1}$$

Tem-se, ainda, que $U_{i,i+1} = c_i - \sum_{k=1}^{i} \cancelto{0}{L_{ik}} U_{kj} \Rightarrow \boxed{U_{i,i+1} = c_i}$

Assim, definindo $U_{ii} = u_i$, $L_{i+1,i} = l_{i+1}$ e sabendo que $U_{i,i+1} = c_i$, a decomposição LU fica:

$$A = LU \Rightarrow \tag{3.13}$$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_2 & 1 & 0 & \cdots & 0 \\ 0 & l_3 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & l_i & 1 \end{bmatrix} \cdot \begin{bmatrix} u_1 & c_1 & 0 & \cdots & 0 \\ 0 & u_2 & c_2 & \cdots & 0 \\ 0 & 0 & u_3 & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & c_{i-1} \\ 0 & 0 & 0 & \cdots & u_i \end{bmatrix} = \begin{bmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{bmatrix} \tag{3.14}$$

O que resulta em:

$$\begin{cases} b_1 = u_1 \\ b_2 = l_2 c_1 + u_2 \\ b_3 = l_3 c_2 + u_3 \\ \vdots \\ b_i = l_i c_{i-1} + u_i \end{cases} \Rightarrow \boxed{u_i = b_i - l_i c_{i-1}} \begin{cases} a_2 = l_2 u_1 \\ a_3 = l_3 u_2 \\ \vdots \\ a_{n-1} = l_{n-1} u_{n-2} \\ a_n = l_n u_{n-1} \end{cases} \Rightarrow \boxed{l_i = a_i / u_{i-1}} \tag{3.15}$$

3.1 Implementação da decomposição LU em Python

Portanto, é possível obter os coeficientes de L e de U por uma ordem diferente a partir das últimas equações obtidas.

Implementando esse código na linguagem Python, obtém-se:

```python
import matplotlib
import numpy as np

def decomposicaoLU(a,b,c): #recebe os vetores a,b,c
    U,L=np.zeros(len(a)),np.zeros(len(a)) #cria U e L vazios
    #valores iniciais
    U[0] = b[0]
    L[0] = 1
    #atribui os valores de u e l
    for i in range(1,len(A[0])):
        L[i]=a[i]/U[i-1]
        U[i]=b[i]-L[i]*c[i-1]
    return L,U
```

Caso queira fazer a decomposição de uma matriz tridiagonal que não está em formato de três vetores (a,b,c), criou-se uma função para fazer esta transformação:

```python
    def lista_de_listas_para_vetores(A):
    a,b,c=np.zeros(len(A[0])),np.zeros(len(A[0])),np.zeros(len(A[0])) #cria os vetores a, b e c vazios
    for i in range(0,len(A[0])): #atribui valores aos vetores
        if i==0:
            a[i]=0
        else:
            a[i]=A[i][i-1]
        b[i]=A[i][i]
        if i==len(A[0])-1:
            c[i]=0
        else:
            c[i] = A[i][i + 1]
    return a,b,c
```

Para testar o código, inputou-se a matriz tridiagonal $A = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 5 & 6 \\ 0 & 8 & 9 \end{bmatrix}$, e o resultado obtido foi:

$$u = [1, -3, 25]; l = [1, 4, -2.67] \tag{3.16}$$

O que representa:

$$U = \begin{bmatrix} 1 & 2 & 0 \\ 0 & -3 & 6 \\ 0 & 0 & 25 \end{bmatrix}; L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 0 & -8/3 & 1 \end{bmatrix} \tag{3.17}$$

4 Resolução de Sistemas

Uma das aplicações da decomposição LU consiste em facilitar a resolução de sistemas de equações. Para tanto, utiliza-se tal processo para simplificar o problema descrito na forma matricial, aplicando a transformação obtida pela decomposição. Assim, seja uma matriz A de tamanho $i \times j$ é possível representar um sistema $Ax = d$ da seguinte forma:

$$Ax = d \rightarrow LUx = d \tag{4.1}$$

Já encaminhando as operações para um estrutura mais aplicável em algoritmos, é interessante substituir o termo Ux por uma nova variável y de tamanho $i, 1$. Dessa forma, o sistema de equações fica descrito como:

$$Ax = d \rightarrow \begin{cases} Ly = d \\ Ux = y \end{cases} \tag{4.2}$$

Nessa estrutura, é necessário primeiramente encontrar as matriz L e U obtidas pela decomposição. Para tal, na resolução do problema proposto utiliza-se a função desenvolvida e já apresentada no presente relatório. Em sequência, deve-se calcular a variável y introduzida, a qual por definição corresponde ao resultado de Ux. Entretanto, x é a incógnita do sistema e, desse modo, utiliza-se a relação $Ly = d$ para encontrar y, assim como representado abaixo:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_2 & 1 & 0 & \cdots & 0 \\ 0 & l_3 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & l_i & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \cdots \\ d_n \end{bmatrix} \tag{4.3}$$

Para o problema em questão, d é conhecido e L é fornecido pela decomposição de A. Diante disso. é possível interpretar os resultados obtidos para $y = [y_1, \cdots, y_n]$ como o laço representado abaixo e aplicá-lo no algoritmo em Python.

$$\begin{cases} y_1 = d_1 \\ y_2 = d_2 - l_2 y_1 \\ y_3 = d_3 - l_3 y_2 \\ \vdots \\ y_i = d_i - l_i y_{i-1} \end{cases} \tag{4.4}$$

```
1     y=np.zeros(len(d))
2     y[0]=d[0]
3     for i in range(1,len(d)):
4         y[i]=d[i]-L[i]*y[i-1]
```

Tal parte representada consiste em um trecho da função completa desenvolvida. Para o cálculo de y, cria-se primeiro um vetor com todos os elementos nulos, passo que pode ser visualizado na primeira linha. Em seguida, os valores são atualizados por um laço utilizando a função *for* do Python. Desse modo, é possível otimizar o código, facilitando a execução para matrizes de dimensão maior.

Posteriormente, utiliza-se a segunda equação obtida pela separação de $Ax = d$, dada por $Ux = y$. Em forma matricial, o problema fica representado da seguinte forma:

$$\begin{bmatrix} u_1 & c_1 & 0 & \cdots & 0 \\ 0 & u_2 & c_2 & \cdots & 0 \\ 0 & 0 & u_3 & \ddots & 0 \\ \vdots & \vdots & \vdots & \ddots & c_{i-1} \\ 0 & 0 & 0 & \cdots & u_i \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \tag{4.5}$$

$$\begin{cases} y_1 = u_1 x_1 + c_1 x_2 \\ y_2 = u_2 x_2 + c_2 x_3 \\ \vdots \\ y_{n-1} = u_{n-1} x_{n-1} + c_{n-1} x_n \\ y_n = u_n x_n \end{cases} \tag{4.6}$$

De maneira análoga ao processo realizado com a equação $Ly = d$, deseja-se descobrir x, conhecendo y e U. Dessa forma, colocando x em função das variáveis conhecidas, tem-se novamente um laço que também foi implementado no algoritmo. Entretanto, diferentemente do laço desenvolvido para y, para encontrar x, é necessário iniciar pelo último termo e aplicar o laço até o primeiro termo.

$$\begin{cases} x_n = \frac{y_n}{u_n} \\ x_{n-1} = \frac{y_{n-1}-c_{n-1}x_n}{u_n} \\ \vdots \\ x_2 = \frac{y_2-c_2x_3}{u_2} \\ x_1 = \frac{y_1-c_1x_2}{u_1} \end{cases} \tag{4.7}$$

```
x=np.zeros(len(d))
x[len(d)-1]=y[len(d)-1]/U[len(d)-1]
for i in range(len(d)-2,-1,-1):
    x[i]=(y[i]-A[2][i]*x[i+1])/U[i]
```

Assim como o trecho do código demonstrado anteriormente, para construir o vetor x, criou-se uma lista somente com valores nulos para, posteriormente, atualizá-los com um laço *for*.

4.1 Implementação da resolução de sistemas em Python

Ao final, a função construída para resolução de sistemas deve integrar os dois trechos apresentados anteriormente. Para isso, usa-se uma estrutura padrão de funções no Python, sendo as entradas dadas pela matriz A e o vetor d. A matriz A deve ser fornecida assim como no caso de decomposição, sendo uma estrutura de *array* com 3 linhas compostas por a, b e c. Caso o usuário queira utilizar o código com um *input* de A como uma matriz completa, basta utilizar a função complementar *lista_de_listas_para_vetores* criada. Com relação a matriz d, ela deve ser fornecida também em formato de lista com uma única linha.

```
import numpy as np
def solucao_sistema(A,d):
    LU=decomposicaoLU(A)
    L,U=LU[0],LU[1]
    x,y=np.zeros(len(d)),np.zeros(len(d))
    y[0]=d[0]
    for i in range(1,len(d)):
        y[i]=d[i]-L[i]*y[i-1]
    x[len(d)-1]=y[len(d)-1]/U[len(d)-1]
    for i in range(len(d)-2,-1,-1):
        x[i]=(y[i]-A[2][i]*x[i+1])/U[i]
    return x
```

As principais informações que auxiliam no entendimento do código já foram destacadas anteriormente. Entretanto, vale comentar sobre a chamada da função 'decomposiçãoLU' para obter as matrizes L e U.

5 Aplicações

5.1 Tarefa

Para validar os resultados obtidos no código idealizado, foi repassado no enunciado da tarefa que propõe esse relatório, uma matriz A tridiagonal representada por três vetores a, b e c obtidos pelas seguintes expressões:

$$a_i = \frac{2i-1}{4i},\ 1 \leq i \leq n-1,\ a_n = \frac{2n-1}{2n} \tag{5.1}$$

$$c_i = 1 - a_i,\ 1 \leq i \leq n \tag{5.2}$$

$$b_i = 2, 1\ \leq i \leq n \tag{5.3}$$

Já o vetor d que representa o resultado da multiplicação Ax é dado por:

$$d_i = \cos\left(\frac{2\pi i^2}{n^2}\right),\ 1 \leq i \leq n \tag{5.4}$$

Dessa forma, construiu-se um código que permita uma fácil utilização e que resolva o sistema para os valores propostos. O código por completo pode ser encontrado no Apêndice A e consiste na combinação das funções já apresentadas em decomposição LU e resolução de sistemas. O principal detalhe é que o código final trata-se de uma função *main*, além de conter algumas chamadas para que o usuário comente o que deseja fazer.

Nesse sentido, no início do código a matriz A é construída no modelo de agrupamento dos três vetores a, b e c para os valores desejados. Nesse processo, utilizou-se um laço do tipo *for* com a construção do vetor de valores nulos como processo antecessor. Tal etapa pode ser melhor visualizada no trecho recortado da função *main* e disponibilizado abaixo.

```
    a=np.zeros(n)
    a[n-1]=(2*n-1)/(2*n)
    d=np.zeros(n)
    for i in range(0,n-1):
        a[i]=(2*(i+1)-1)/(4*(i+1))
        d[i]=np.cos((2*np.pi*(i+1)**2)/(n**2))
    c=1-a
    b=np.full(n,2)
    A=[a,b,c]
```

Em sequência, o código principal solicita algumas informações para que o usuário descreva o que deseja fazer. A priori, questiona-se sobre qual parte deseja-se rodar, sendo duas opções ofertadas, resolver somente o problema de decomposição LU da matriz A ou resolver o sistema $Ax = d$. Para escolher entre as opções, o usuário deve digitar 1 ou 2, respectivamente, para cada opção.

```
print("Algoritmo para decomposição LU de uma matriz A e resolução de um sistema Ax=d.")
print("Para demonstração dos resultados do código, utiliza-se a matriz 'A' e o vetor 'd' do teste disponibilizado no final do enunciado.")
funcao = int(input("Digite 1 para resolver somente o problema de decomposição LU ou digite 2 para resolver um sistema tridiagonal"))
```

Por conseguinte, o código dá sequência dependendo do interesse do usuário. Caso seja escolhido somente a opção de resolver a decomposição LU, o código entrega como *Output* o vetor l e o vetor u. Logo após, questiona-se o interesse em visualizar a matriz L e a matriz U, em formato completo, não mais representadas no agrupamento de 3 vetores. Assim sendo, o usuário digita 1 para visualizar ou 0 para dar sequência no código.

```
imprimir=int(input(print("Caso queira visualizar as matrizes L e U em forma de lista de listas digite 1, caso contrário digite 0 para dar sequência.")))
#Condicional para imprimir as matrizes em formato array.
if imprimir==1:
    print("Matriz U:", print_matrizU(U,c))
    print("Matriz L:", print_matrizL(L))
```

As funções *print_matrizU* e *print_matrizL* também são funções complementares para ampliar a utilização do código. O desenvolvimento de tais funções pode ser encontrado no Apêndice A.

Instantes depois, pergunta-se se é desejável visualizar, também, a resolução do sistema $Ax = d$. Na hipótese do usuário apresentar tal interesse, é necessário digitar 1, caso contrário, digita-se 0 para encerrar o código.

```
print("Algoritmo para decomposição LU de uma matriz A e resolução de um sistema Ax=d.")
print("Para demonstração dos resultados do código, utiliza-se a matriz 'A' e o vetor 'd' do teste disponibilizado no final do enunciado.")
funcao = int(input("Digite 1 para resolver somente o problema de decomposição LU ou digite 2 para resolver um sistema tridiagonal"))
```

Se porventura, o usuário optar logo no primeiro questionamento pela resolução do sistema $Ax = d$, o vetor x é dado como *Output* e o código é encerrado. Tal esquema de questionário pode ser visualizado mais facilmente no trecho do código abaixo.

```
if funcao==1:
    L,U=decomposicaoLU(A)
```

```
        print("Vetor u:", U)
        print("Vetor l:", L)
        imprimir=int(input(print("Caso queira visualizar as matrizes L e U em forma de lista de listas digite 1, caso contrário digite 0 para dar sequência.")))
        #Condicional para imprimir as matrizes em formato array.
        if imprimir==1:
            print("Matriz U:", print_matrizU(U,c))
            print("Matriz L:", print_matrizL(L))
        #Condicional para continuar da decomposição para resolução de sistemas
        continua = int(input("Caso queira visualizar a solução do sistema Ax=d do teste, digite 1. Caso contrário digite 0 para encerrar o código"))
        if continua==1:
            print("As raízes do sistema são:")
            print("x:",solucao_sistema(A,d))
        else:
            exit()
    elif funcao==2:
        print("As raízes do sistema são:")
        print("x:",solucao_sistema(A,d))
```

Nesse sentido, rodando o código para os valores do enunciado, obteve-se os seguintes resultados utilizando ponto como separador decimal:

Tabela 1 – Resultados obtidos no código

i	**a**	**b**	**c**	**u**	**l**	**y**	**x**	**d**
1	0.2500	2.0000	0.7500	2.0000	1.0000	0.9999	0.3784	0.9999
2	0.3750	2.0000	0.6250	1.8594	0.1875	0.8105	0.3240	0.9980
3	0.4167	2.0000	0.5833	1.8599	0.2241	0.8084	0.3330	0.9900
4	0.4375	2.0000	0.5625	1.8628	0.2352	0.7784	0.3241	0.9686
5	0.4500	2.0000	0.5500	1.8641	0.2416	0.7358	0.3107	0.9239
6	0.4583	2.0000	0.5417	1.8648	0.2459	0.6634	0.2850	0.8443
7	0.4643	2.0000	0.5357	1.8651	0.2490	0.5530	0.2438	0.7181
8	0.4688	2.0000	0.5313	1.8654	0.2513	0.3969	0.1835	0.5358
9	0.4722	2.0000	0.5278	1.8655	0.2532	0.1936	0.1027	0.2940
10	0.4750	2.0000	0.5250	1.8656	0.2546	-0.0493	0.0036	0.0000
11	0.4773	2.0000	0.5227	1.8657	0.2558	-0.3113	-0.1067	-0.3239
12	0.4792	2.0000	0.5208	1.8657	0.2568	-0.5575	-0.2147	-0.6374
13	0.4808	2.0000	0.5192	1.8658	0.2577	-0.7401	-0.3011	-0.8838
14	0.4821	2.0000	0.5179	1.8658	0.2584	-0.8068	-0.3436	-0.9980
15	0.4833	2.0000	0.5167	1.8659	0.2590	-0.7149	-0.3201	-0.9239
16	0.4844	2.0000	0.5156	1.8659	0.2596	-0.4518	-0.2278	-0.6374
17	0.4853	2.0000	0.5147	1.8659	0.2601	-0.0544	-0.0519	-0.1719
18	0.4861	2.0000	0.5139	1.8659	0.2605	0.3823	0.0824	0.3681
19	0.4868	2.0000	0.5132	1.8659	0.2609	0.7184	0.4446	0.8182
20	0.9750	2.0000	0.0250	1.7319	0.5225	-0.3754	-0.2168	0.0000

5.2 Matriz inversa

Para melhor apresentar e descrever as aplicações da decomposição LU de uma matriz A no universo matemático, mostra-se a possibilidade de calcular a inversa de uma matriz por este método. Para isso, a resolução é semelhante a de sistemas lineares do tipo $Ax = b$, conforme mostra-se a seguir.

Considerando a definição de matriz inversa $AA^{-1} = I$, sendo A_{ij} os elementos da matriz A e a_{ij} os de A^{-1}, ambas de dimensões $n \times n$, tem-se:

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1,n} \\ A_{21} & A_{22} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,n} \\ a_{21} & a_{22} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{5.5}$$

Para resolver essa equação, interpreta-se como n sistemas lineares do tipo $Ax = b$, de forma que x é a a enésima coluna de A^{-1} e b a matriz transposta da enésima linha da matriz identidade (B).

Assim, obtém-se para uma coluna k arbitrária:

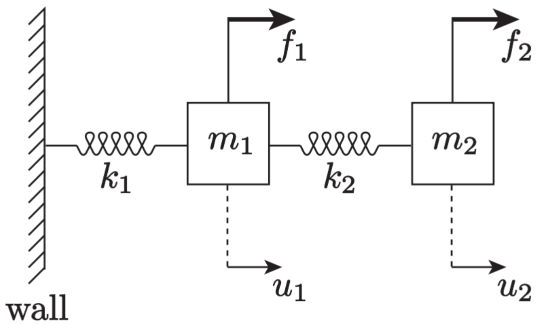
$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1,n} \\ A_{21} & A_{22} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{bmatrix} \cdot \begin{bmatrix} a_{1,k} \\ a_{2,k} \\ \vdots \\ a_{n,k} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 1 & \cdots & 0 \end{bmatrix}^T \tag{5.6}$$

Em que o fator "1"ocupa a k-ésima coluna na matriz. Dessa forma é possível calcular os coeficientes da matriz inversa através da resolução de sistemas lineares por decomposição LU conforme apresentado anteriormente.

5.3 Exemplo práticos

Além dos casos já apresentados, é possível introduzir, ainda, um exemplo físico. Nesse sentido, vale comentar sobre um sistema massa-mola e suas condições de equilíbrio. Inicialmente, vale tratar um caso simples que contém duas massas, assim como ilustrado na figura abaixo.

Figura 1 – Sistema massa-mola para $n = 2$



Fonte:(PATERA; YANO, 2014)

Em tal exemplo, f_1 representa forças externas aplicadas e u_1 o deslocamento do bloco de massa m_1. De maneira análoga, u_2 e f_2 representar o deslocamento e a força do bloco 2, respectivamente. Desse modo, utilizando o equilíbrio de forças para resolver o problema, chega-se nas seguintes equações:

$$f_1 - k_1 u_1 + k_2(u_2 - u_1) = 0 \tag{5.7}$$

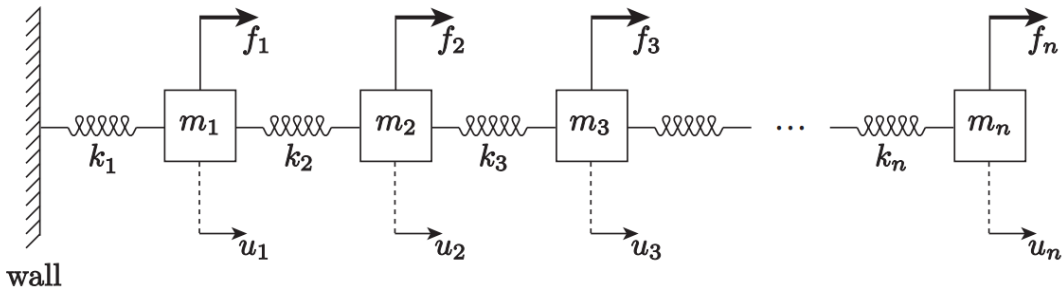
$$f_2 - k_2(u_2 - u_1) = 0 \tag{5.8}$$

Para tal problema, deseja-se conhecer os deslocamentos de cada bloco. Dessa forma, reescrevendo as equações e passando para forma matricial, tem-se:

$$\begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{5.9}$$

Assim sendo, generalizando o problema para n molas e n massas, percebe-se que o sistema de equações final é dado por uma matriz tridiagonal, além de encontrar-se na forma $Ax = d$. Dessa forma, é possível aplicar a decomposição LU, além de resolver o sistema, assim como já apresentado no presente relatório.

Figura 2 – Sistema de n molas e n massas



Fonte:(PATERA; YANO, 2014)

$$\begin{bmatrix} k_1 + k_2 & -k_2 & 0 & & \cdots & 0 \\ -k_2 & k_2 + k_3 & -k_3 & & \cdots & 0 \\ 0 & -k_3 & k_3 + k_4 & & & \vdots \\ & & & & \ddots & \\ \vdots & \vdots & & \ddots & \ddots & -k_n \\ & & & & & \\ 0 & 0 & \cdots & & -k_n & k_n \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix} \tag{5.10}$$

Por conclusão, ressalta-se a importância dos métodos apresentados para os problemas de engenharia. Com tais ferramentas é possível resolver os sistemas apresentados, além de diversas outras aplicações.

Referências

PATERA, A.; YANO, M. *Linear Systems of Equations. . . in a Nutshell*. 2014. Disponível em: <https://ocw.mit.edu/ans7870/2/2.086/F14/MIT2_086F14_Linear_Sys.pdf>. Acesso em: 30 abr 2022. Citado 2 vezes nas páginas 18 e 19.

Apêndices

APÊNDICE A – Código completo

Código desenvolvido no *Python*

```
import numpy as np
#Função principal para obter os resultados do problema proposto no enunciado
def main():
    np.set_printoptions(precision=4,suppress=True)
    print("Algoritmo para decomposição LU de uma matriz A e resolução de um sistema Ax=d.")
    print("Para demonstração dos resultados do código, utiliza-se a matriz 'A' e o vetor 'd' do teste disponibilizado no final do enunciado.")
    funcao = int(input("Digite 1 para resolver somente o problema de decomposição LU ou digite 2 para resolver um sistema tridiagonal"))
    n=20
    #Cria as variáveis com os dados fornecidos pelo enunciado
    a=np.zeros(n)
    a[n-1]=(2*n-1)/(2*n)
    d=np.zeros(n)
    for i in range(0,n-1):
        a[i]=(2*(i+1)-1)/(4*(i+1))
        d[i]=np.cos((2*np.pi*(i+1)**2)/(n**2))
    c=1-a
    b=np.full(n,2)
    A=[a,b,c]
    #Condicional para mostrar os resultados conforme deseja o usuário
    if funcao==1:
        L,U=decomposicaoLU(A)
        print("Vetor u:", U)
        print("Vetor l:", L)
        imprimir=int(input(print("Caso queira visualizar as matrizes L e U em forma de lista de listas digite 1, caso contrário digite 0 para dar sequência.")))
        #Condicional para imprimir as matrizes em formato array.
        if imprimir==1:
            print("Matriz U:", print_matrizU(U,c))
            print("Matriz L:", print_matrizL(L))
        #Condicional para continuar da decomposição para resolução de sistemas
        continua = int(input("Caso queira visualizar a solução do sistema Ax=d do teste, digite 1. Caso contrário digite 0 para encerrar o código"))
        if continua==1:
            print("As raízes do sistema são:")
```

```
            print("x:",solucao_sistema(A,d))
        else:
            exit()
    elif funcao==2:
        print("As raízes do sistema são:")
        print("x:",solucao_sistema(A,d))
#'A' deve ser uma matriz com n colunas e 3 linhas, sendo a primeira linha 'a', a segunda 'b' e a terceira 'c'. Caso a matriz A não esteja no formato comentado, pode-se utilizar a função lista_de_listas_para_vetores adiciona em funções complementares

#Primeira função, utilizada para realizar a decomposição LU de uma matriz A
def decomposicaoLU(A):
    a,b,c=A[0],A[1],A[2]
    U,L=np.zeros(len(A[0])),np.zeros(len(A[0])) #cria os vetores U e L vazios
    #Valores iniciais
    U[0]=b[0]
    L[0]=1
    #Laço para encontrar L,U, assim como disponibilizado no enunciado
    for i in range(1,len(A[0])):
        L[i]=a[i]/U[i-1]
        U[i]=b[i]-L[i]*c[i-1]
    return L,U
#Segunda função, utilizada para resolver um sistema Ax=d.
def solucao_sistema(A,d):
    LU=decomposicaoLU(A)
    L,U=LU[0],LU[1]
    x,y=np.zeros(len(d)),np.zeros(len(d)) #cria os vetores x e y vazios
    y[0]=d[0]
    for i in range(1,len(d)):
        y[i]=d[i]-L[i]*y[i-1]
    x[len(d)-1]=y[len(d)-1]/U[len(d)-1]
    for i in range(len(d)-2,-1,-1):
        x[i]=(y[i]-A[2][i]*x[i+1])/U[i]
    return x

#Códigos complementares
#Função criada para transformar uma matriz A tridiagonal da forma de lista de listas (array) para vetores, conforme descrito no enunciado.
def lista_de_listas_para_vetores(A):
    a,b,c=np.zeros(len(A[0])),np.zeros(len(A[0])),np.zeros(len(A[0])) #cria os vetores a, b e c vazios
    for i in range(0,len(A[0])): #atribui valores aos vetores
        if i==0:
            a[i]=0
        else:
```

```
                a[i]=A[i][i-1]
            b[i]=A[i][i]
            if i==len(A[0])-1:
                c[i]=0
            else:
                c[i] = A[i][i + 1]
    B=a,b,c
    return(B)
#Função para imprimir a Matriz U em forma de lista de listas (array)
def print_matrizU(u,c):
    U=np.zeros((len(u),len(u))) #cria a matriz U vazia
    for i in range(len(u)): #atribui valores para matriz U
        U[i][i]=u[i]
        if i != len(u)-1:
            U[i][i+1]=c[i]
    U_matrix=np.matrix(U)
    return U_matrix
#Função para imprimir a Matriz L em forma de lista de listas (array)
def print_matrizL(l):
    L=np.zeros((len(l),len(l))) #cria a matriz U vazia
    for i in range(len(l)): #atribui valores para matriz U
        L[i][i]=1
        if i != len(l)-1:
            L[i+1][i]=l[i+1]
    L_matrix=np.matrix(L)
    return L_matrix

main()
```