Álgebra Linear - Aula Prática 1

Iara Cristina Mescua Castro

7 de novembro de 2022

1. Teste a função dada usando algumas matrizes quadradas A e respectivos vetores b. Use exemplos dos quais você saiba a resposta para verificar se a função realmente está funcionando corretamente.

   Resolução:

$$A = \begin{bmatrix} 3 & 1 & -1 \\ 1 & -1 & 1 \\ 2 & 1 & 1 \end{bmatrix} b = \begin{bmatrix} 1 \\ -3 \\ 0 \end{bmatrix}$$

Resolvendo por eliminação de Gauss:

$$\begin{bmatrix} 3 & 1 & -1 & | & 1 \\ 1 & -1 & 1 & | & -3 \\ 2 & 1 & 1 & | & 0 \end{bmatrix}$$

$R_1 \longleftrightarrow R_2$

$$\begin{bmatrix} 1 & -1 & 1 & | & -3 \\ 3 & 1 & -1 & | & 1 \\ 2 & 1 & 1 & | & 0 \end{bmatrix}$$

$-3R_1 + R_2 \longleftrightarrow R_2$

$$\begin{bmatrix} 1 & -1 & 1 & | & -3 \\ 0 & 4 & -4 & | & 10 \\ 2 & 1 & 1 & | & 0 \end{bmatrix}$$

$-2R_1 + R_3 \longleftrightarrow R_3$

$$\begin{bmatrix} 1 & -1 & 1 & | & -3 \\ 0 & 4 & -4 & | & 10 \\ 0 & 3 & -1 & | & 6 \end{bmatrix}$$

$R_2/4 \longleftrightarrow R_2$

$$\begin{bmatrix} 1 & -1 & 1 & | & -3 \\ 0 & 1 & -1 & | & \frac{10}{4} \\ 0 & 3 & -1 & | & 6 \end{bmatrix}$$

$R_2 + R_1 \longleftrightarrow R_1$

$$\begin{bmatrix} 1 & 0 & 0 & | & -\frac{1}{2} \\ 0 & 1 & -1 & | & \frac{5}{2} \\ 0 & 3 & -1 & | & 6 \end{bmatrix}$$

$-3R_2 + R_3 \longleftrightarrow R_3$

$$\begin{bmatrix} 1 & 0 & 0 & | & -\frac{1}{2} \\ 0 & 1 & -1 & | & \frac{5}{2} \\ 0 & 0 & 2 & | & -\frac{3}{2} \end{bmatrix}$$

$R_3/2 \longleftrightarrow R_3$

$$\begin{bmatrix} 1 & 0 & 0 & | & -\frac{1}{2} \\ 0 & 1 & -1 & | & \frac{5}{2} \\ 0 & 0 & 1 & | & -\frac{3}{4} \end{bmatrix}$$

$R_3 + R_2 \longleftrightarrow R_2$

$$\begin{bmatrix} 1 & 0 & 0 & | & -\frac{1}{2} \\ 0 & 1 & 0 & | & \frac{7}{4} \\ 0 & 0 & 1 & | & -\frac{3}{4} \end{bmatrix}$$

Temos que:

$$x = \begin{bmatrix} -\frac{1}{2} \\ \frac{7}{4} \\ -\frac{3}{4} \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1.75 \\ -0.75 \end{bmatrix}$$

Decomposição LU de A:

$$\begin{bmatrix} 3 & 1 & -1 \\ 1 & -1 & 1 \\ 2 & 1 & 1 \end{bmatrix}$$

$R_2 \longleftrightarrow R_2 - \frac{1}{3}R_1$

$$\begin{bmatrix} 3 & 1 & -1 \\ 0 & -\frac{4}{3} & \frac{4}{3} \\ 2 & 1 & 1 \end{bmatrix}$$

$R_3 \longleftrightarrow R_3 - \frac{2}{3}R_1$

$$\begin{bmatrix} 3 & 1 & -1 \\ 0 & -\frac{4}{3} & \frac{4}{3} \\ 0 & \frac{1}{3} & \frac{5}{6} \end{bmatrix}$$

$R_3 \longleftrightarrow R_3 + \frac{1}{4}R_2$

$$U = \begin{bmatrix} 3 & 1 & -1 \\ 0 & -\frac{4}{3} & \frac{4}{3} \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 1 & -1 \\ 0 & -1.333333 & 1.3333333 \\ 0 & 0 & 2 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{2}{3} & -\frac{1}{4} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.33 & 1 & 0 \\ 0.66 & -0.25 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 3 & 1 & -1 \\ 0.33 & -1.33 & 1.33 \\ 0.66 & -0.25 & 2 \end{bmatrix}$$

Conferindo no Scilab, verificamos que a função está operando corretamente para este exemplo.

```
--> A
 A  =

   3.   1.  -1.
   1.  -1.   1.
   2.   1.   1.

--> b
 b  =

   1.
  -3.
   0.

--> [x, C] = Gaussian_Elimination_1(A,b)
 x  =

  -0.5000000
   1.7500000
  -0.7500000
 C  =

   3.          1.          -1.
   0.3333333  -1.3333333   1.3333333
   0.6666667  -0.25        2.
```

2. Agora teste com a matriz: $A1 = [1 \quad -2 \quad 5 \quad 0; 2 \quad -4 \quad 1 \quad 3; -1 \quad 1 \quad 0 \quad 2; 0 \quad 3 \quad 3 \quad 1]$ e com o vetor $b1 = [1; 0; 0; 0]$.

   Resolução:

```
A1  =

   1.  -2.   5.   0.
   2.  -4.   1.   3.
  -1.   1.   0.   2.
   0.   3.   3.   1.

--> b1=[1;0;0;0]
 b1  =

   1.
   0.
   0.
   0.

--> [x, C] = Gaussian_Elimination_1(A1, b1)
 x  =

   Nan
   Nan
   Nan
   Nan
 C  =

   1.  -2.    5.    0.
   2.   0.   -9.    3.
  -1.  -Inf  -Inf   Inf
   0.   Inf   Nan   Nan
```

   Houve um erro, pois na quarta linha teremos um elemento nulo, que conflita com a divisão pelo pivô $A(j,j)$ na função.

3. Modifique a função dada trocando linhas quando no início da iteração j o elemento na posição (j,j) é nulo. Chame esta nova função de *Gaussian_Elimination_2* e teste-a com a matriz A1 e o vetor b1 dados. Agora teste-a com a matriz $A2 = [0 \quad 10^{-20} \quad 1; 10^{-20} \quad 1 \quad 1; 1 \quad 2 \quad 1]$ e o vetor $b2 = [1; 0; 0]$.

   Resolução:

```
function [x, C]=Gaussian_Elimination_2(A, b)
C=[A,b];
[n]= length(b);

for j=1:(n-1)
        if C(j,j) == 0
                diff_zero = find(C(:,j));
                x = diff_zero(diff_zero>j);
                C([j x(1)],:) = C([x(1) j],:);
        end
        for i=(j+1):n
                C(i,j)=C(i,j)/C(j,j);
                C(i,j+1:n+1)=C(i,j+1:n+1)-C(i,j)*C(j,j+1:n+1);
        end
end

x=zeros(n,1);

// Calcula x, sendo Ux=C(1:n,n+1)

x(n)=C(n,n+1)/C(n,n);
for i=n-1:-1:1
        sum = b(i);
```

```
            for k=i+1:n
                sum = sum-A(i,j)*x(k);
            end
            x(i) = sum/A(i,i);
end

C=C(1:size(C,1),1:size(C,1));

endfunction
```

Com a nova função, dentro do loop há uma nova condição que quando encontra $C(j,j)$ igual a 0, verifica as linhas da matriz que tem um número diferente de 0 naquela coluna j com a função $find()$ e depois encontra qual dessas linhas é a próxima depois da linha j, para então, realizar a troca de linhas toda vez que durante o loop encontrarmos um 0 no pivô na diagonal.

Testando com a matriz A_1 e b_1:

```
A1 =

     1    -2     5     0
     2    -4     1     3
    -1     1     0     2
     0     3     3     1

>> b1

b1 =

     1
     0
     0
     0

>> [x1, C1] = Gaussian_Elimination_2(A1, b1)

x1 =

   -0.3248
   -0.1709
    0.1966
   -0.0769


C1 =

     1    -2     5     0
    -1    -1     5     2
     2     0    -9     3
     0    -3    -2    13
```

Podemos verificar a diferença através do comando de divisão de matrizes que está funcionando corretamente:

```
>> A1\b1

ans =

   -0.3248
   -0.1709
    0.1966
   -0.0769
```

Testando com a matriz $A2 = [0 \quad 10^{-20} \quad 1; 10^{-20} \quad 1 \quad 1; 1 \quad 2 \quad 1]$ e o vetor $b2 = [1; 0; 0]$, temos:

```
--> A2
 A2  =

   0.         1.000D-20   1.
   1.000D-20  1.          1.
   1.         2.          1.

--> b2
 b2  =

   1.
   0.
   0.

--> [x2,C2] = Gaussian_Elimination_2(A2,b2)
 x2  =

  -1.000D+20
   0.
   1.
 C2  =

   1.000D-20   1.          1.
   0.          1.000D-20   1.
   1.000D+20  -1.000D+40   1.000D+40
```

Mesmo sem erros na execução, a solução está incorreta. Podemos verificar a diferença através do comando de divisão de matrizes:

```
--> A2\b2
 ans  =

   1.
  -1.
   1.
```

4. Modifique a função do item 3 para escolher o maior pivô em módulo quando no início da iteração j o elemento na posição (j, j) é nulo. Chame esta nova função de *Gaussian_Elimination_3* e teste-a com a matriz $A2$ e o vetor $b2$ dados. Agora com a matriz $A3 = [10^{-20} \quad 10^{-20} \quad 1; 10^{-20} \quad 1 \quad 1; 1 \quad 2 \quad 1]$ e o vetor $b3 = b2$. **Resolução:**

```
function [x, C]=Gaussian_Elimination_3(A, b)
C=[A,b];
[n]= length(b);

for j=1:(n-1)
        if C(j,j) == 0
                x = L(:,j);
                x = (max(abs(x)));
                index = find(L(:,j) == x);
                L([j index(1)],:) = L([index(1) j],:);
                C([j index(1)],:) = C([index(1) j],:);
        end
        for i=(j+1):n
```

```
                    C(i,j)=C(i,j)/C(j,j);
                    C(i,j+1:n+1)=C(i,j+1:n+1)-C(i,j)*C(j,j+1:n+1);
          end
end

x=zeros(n,1);

// Calcula x, sendo Ux=C(1:n,n+1)

x(n)=C(n,n+1)/C(n,n);
for i=n-1:-1:1
          sum = b(i);
          for k=i+1:n
                    sum = sum-A(i,j)*x(k);
          end
          x(i) = sum/A(i,i);
end

C=C(1:size(C,1),1:size(C,1));

endfunction
```

Agora não precisamos apenas do índice, mas do seu elemento correspondente que será o maior pivô para a troca de linhas. Para isso, indexamos eles na matriz para visualizar os valores e devolvemos o $max()$ (função que pega o valor máximo de uma matriz) do módulo desse vetor $abs()$ x. Agora temos o maior valor, então criamos uma condição com $find(C(:,j) == x)$ para ver qual a linha desse valor e por fim, fazer a troca.

Esta simples mudança oferece mais estabilidade para a função.

Testando com a matriz $A2$ e o vetor $b2$, agora teremos a solução correta.

```
--> A2
 A2  =

   0.           1.000D-20   1.
   1.000D-20    1.          1.
   1.           2.          1.

--> b2
 b2  =

   1.
   0.
   0.

--> [x2,C2] = Gaussian_Elimination_3(A2,b2)
 x2  =

   1.
  -1.
   1.
 C2  =

   1.           2.          1.
   1.000D-20    1.          1.
   0.           1.000D-20   1.
```

Testando com a matriz $A3 = [10^{-20} \quad 10^{-20} \quad 1; 10^{-20} \quad 1 \quad 1; 1 \quad 2 \quad 1]$ e o vetor $b3 = b2$:

```
>> A3

A3 =

    0.0000    0.0000    1.0000
    0.0000    1.0000    1.0000
    1.0000    2.0000    1.0000

>> b3

b3 =

     1
     0
     0

>> [x3, C3] = Gaussian_Elimination_3(A3, b3)

x3 =

     0
    -1
     1


C3 =

   1.0e+20 *

    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000
    1.0000    0.0000   -1.0000
```

Mesmo sem erros na execução, a solução está incorreta. Podemos verificar a diferença através do comando de divisão de matrizes:

```
>> A3\b3

ans =

     1
    -1
     1
```

5. Modifique a função do item 4 para escolher sempre o maior pivô em módulo no início da iteração j independente do elemento na posição (j,j) ser nulo ou não. Nessa função, retorne também a matriz de permutação P utilizada. Chame esta nova função de *Gaussian_Elimination_4* e teste-a com a matriz A3 e o vetor b3 dados.

Resolução:
A mudança é simples, basta remover as linhas da condição de $C(j, j)$ ser nulo ou não. O erro será solucionado, devemos fazer a troca de linhas com valores muito próximos a zero, o que não acontecia originalmente, mas com a nova função a troca de linhas será feita de qualquer forma para o maior pivô em módulo, evitando erros de aproximação.

```
function [x, C]=Gaussian_Elimination_4(A, b)
C=[A, b];
[n]=size(C, 1);
P = eye(n, n);
for j = 1:(n - 1)
// O pivo esta na posicao (j,j)
p_col = abs(C(j:n, j));
// Extrai os valores absolutos da coluna pivo atual
// em linhas maiores que a atual
[m, i] = max(p_col);
```

```
// Encontra onde ela tem seu valor maximo
i = i + j - 1
if i > j
P([j i], :) = P([i j], :);
C([j i], :) = C([i j], :);
// Reordena as linhas se a atual nao tem o maior valor
end
for i = (j + 1):n
// O elemento C(i,j) eh o elemento na posicao (i, j) de L
// na decomposicao LU de A
C(i, j) = C(i, j) / C(j, j);
// Linha i <- Linha i - C(i, j) * Linha j
// Somente os elementos da diagonal ou acima sao computados
// (aqueles que compoem a matrix U)
C(i, j+1:n+1)=C(i, j+1:n+1) - C(i, j) * C(j, j+1:n+1);
end
end
x = zeros(n, 1);
// Calcula x, sendo Ux=C(1:n,n+1)
x(n) = C(n, n+1) / C(n, n);
for i = n-1:-1:1
x(i) = (C(i, n+1) - C(i, i:n) * x(i:n)) / C(i, i);
end
C = C(1:n, 1:n);
endfunction
```

Testando com a matriz $A3$ e o vetor $b3$, teremos:

```
--> A3
 A3  =

   1.000D-20   1.000D-20   1.
   1.000D-20   1.          1.
   1.          2.          1.

--> b3
 b3  =

   1.
   0.
   0.

--> [x3,C3,P3] = Gaussian_Elimination_4(A3,b3)
 x3  =

   1.
  -1.
   1.
 C3  =

   1.          2.          1.
   1.000D-20   1.          1.
   1.000D-20  -1.000D-20   1.
 P3  =

   0.   0.   1.
   0.   1.   0.
   1.   0.   0.
```

Agora a solução está correta.

```
--> A3
 A3  =

   1.000D-20   1.000D-20   1.
   1.000D-20   1.          1.
   1.          2.          1.

--> b3
 b3  =

   1.
   0.
   0.

--> A3\b3
 ans  =

   1.
  -1.
   1.
```

6. Uma vez que você tem a decomposição LU de uma matriz quadrada A de ordem n (ou de PA, sendo P uma matriz permutação) a resolução de um sistema linear Ax=b pode ser obtida mais rapidamente usando a decomposição LU já feita, em vez de fazer todo o escalonamento de novo. Escreva uma função Scilab de nome *Resolve_com_LU*, que receba como variáveis de entrada uma matriz C com a decomposição LU de A (ou de PA, conforme matriz retornada pelas funções anteriores) e uma matriz B de ordem n x m e retorne uma matriz X, com a mesma ordem de B, cujas colunas sejam as soluções dos sistemas lineares $A_x i = bi$. Observação: talvez você ache necessário passar outra(s) variável(is) de entrada para essa função. Teste a sua função com a matriz A1 dada anteriormente e com a matriz

$B1 = [24-15; 0103; 22-11; 0115]$. Teste também com a matriz A2 dada anteriormente e com a matriz $B2 = [112; 1-10; 101]$. Finalmente, teste com a matriz A3 dada anteriormente e com a matriz B3=B2.

Resolução:

```
function X=Resolve_com_LU(C, B, P)
B = P*B;
m = size(B,2); //colunas de B
n = size(C,1); //linhas
L = tril(C); //Matriz L (triangular inferior)
L(boolean(eye(n)))=1;
U = triu(C); //Matriz U (triagular superior)
X = zeros(n,m);

for k = 1:m
b_i = B(1:n, k); //vetores b_i
y_i = resolveL(L,b_i); //vetores y_i
x_i = resolveU(U,y_i); //x_i = U\ y_i
X(1:n, k) = x_i; //x_i em X
end
endfunction

//Ly = b
function y = resolveL(L,b)
n = size(L,1);
m = size(b,1);
y = zeros(m,1);
y(1) = b(1)/L(1,1);
for i=2:n
y(i) = (b(i) - L(i,1:i-1)*y(1:i-1))/L(i,i);
end
endfunction

//Ux = y
function x = resolveU(U,y)
n = size(U,1);
x = zeros(n,1);
x(n) = y(n)/U(n,n);
for i=n-1:-1:1
x(i) = (y(i) - U(i,1:i-1)*x(1:i-1))/U(i,i);
end
endfunction
```

Nessa função, recebemos a matriz C, BeP. Primeiro usamos a matriz de permutação P para $B = P*B$ e separamos a matriz C ela nas duas matrizes L e U com o comando $tril()$ e $triu()$. Depois, criamos duas funções a parte para resolver o seguinte sistema:

$$Ax = b$$

Sabendo que $A = LU$

$$LUx = b$$

$$L(Ux) = b$$

Chamamos $Ux = y$ Substituindo:

$$Ly = b$$

Então ficamos com essas duas equações: $Ux = y$ e $Ly = b$. Resolvemos ela com as funções $resolveL(L, b)$ e $resolveU(U, y)$ e por último, chamamos essas funções em um loop na função inicial para encontrar as soluções dos sistemas lineares $Ax_i = b_i$ e armazenamos elas na matriz X.

Testando com todas as matrizes:

```
C1 =

    2.0000   -4.0000    1.0000    3.0000
         0    3.0000    3.0000    1.0000
    0.5000         0    4.5000   -1.5000
   -0.5000   -0.3333    0.3333    4.3333

>> B1

B1 =

     2     4    -1     5
     0     1     0     3
     2     2    -1     1
     0     1     1     5

>> P1

P1 =

     0     1     0     0
     0     0     0     1
     1     0     0     0
     0     0     1     0

>> [X] = Resolve_com_LU(C1,B1,P1)

X =

         0    0.5000         0    1.5000
         0    0.3333    0.3333    1.6667
    0.4444    0.7778   -0.2222    0.7778
    0.3077    0.3846   -0.0769    0.6923
```

Figura 1: A1

```
>> C2

C2 =

    1.0000    2.0000    1.0000
    0.0000    1.0000    1.0000
         0    0.0000    1.0000

>> B2

B2 =

     1     1     2
     1    -1     0
     1     0     1

>> P2

P2 =

     0     0     1
     0     1     0
     1     0     0

>> [X2] = Resolve_com_LU(C2,B2,P2)

X2 =

    1.0000         0    1.0000
    1.0000   -1.0000   -0.0000
    1.0000    1.0000    2.0000
```

Figura 2: A2

```
>> C3

C3 =

    1.0000    2.0000    1.0000
    0.0000    1.0000    1.0000
    0.0000   -0.0000    1.0000

>> B3

B3 =

     1     1     2
     1    -1     0
     1     0     1

>> P3

P3 =

     0     0     1
     0     1     0
     1     0     0

>> [X3] = Resolve_com_LU(C3,B3,P3)

X3 =

    1.0000         0    1.0000
    1.0000   -1.0000   -0.0000
    1.0000    1.0000    2.0000
```

Figura 3: A3