Relatório - Aula Prática 1

Gustavo Ramalho*

20 de Março de 2022

1 Questão 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.

```
function [x, C]= Gaussian_Elimination_1(A, b)
C=[A,b];
[n]=size(C,1);
for j=1:(n-1)
 //O pivo esta na posicao (j,j)
 for i=(j+1):n
//O elemento C(i,j) eh o elemento na posicao (i,j) of 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 dela 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


```

*gustavoramalho384@gmail.com

Testarei com a matriz a seguir, para conferir se o código funcionou corretamente.

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 1 \\ 4 & -1 & 3 & -4 \\ -2 & 5 & 5 & 9 \end{array}\right] \qquad \mathrm{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & -1 \end{bmatrix} \; \mathrm{U} = \begin{bmatrix} 2 & 3 & -1 \\ 0 & -2 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -5 \\ 3 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 & -1 \\ 0 & -2 & 1 \\ 0 & 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -5 \\ 3 \end{bmatrix} \qquad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -3 \\ 3 \\ 1 \end{bmatrix} \qquad \mathrm{C} = \begin{bmatrix} 2 & 3 & -1 \\ 2 & -2 & 1 \\ -1 & 0 & 3 \end{bmatrix}$$

O Programa foi testado com esses mesmos valores e o resultado da variável x e C foi semelhante ao encontrado. Também testei com outras matrizes que podiam ser decompostas e o resultado foi igual. Assim, concluimos que o programa funciona.

Figura 1: console 1

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

--> x
 x  =

  -3.
   3.
   1.

--> C
 C  =

   2.   3.  -1.
   2.  -2.   1.
  -1.   0.   3.
```

2 Questão 2

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

Ao testarmos essa matriz, obtemos valores NaN para o x (Not a Number), o que implica que tem algo que impede que essa decomposição aconteça. No caso dessa matriz A1, o motivo disso é o pivô da terceira linha ser igual a 0, o que não pode acontecer porque o programa não suportaria essa particularidade. Assim, alterações nó codigo precisam ser feitas para ter o funcionamento que queremos. O C também tem partes em que o resultado é NaN, infinito e menos infinito também.

Figura 2: console 2

```
--> A1=[1 -2 5 0; 2 -4 1 3; -1 1 0 2; 0 3 3 1] ;

--> 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
```

3 Questão 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 10^{-20} 1; 10^{-20} 1 1; 1 2 1] e o vetor b2=[1; 0; 0].

```
1
2  function [x, C]=Gaussian_Elimination_2(A, b)
3  C=[A,b];
4  [n]=size(C,1);
5
6  for j=1:(n-1)
7      if C(j,j) == 0
8          C([j j+1], :) = C([j+1 j], :)
9       end
10
11   //O pivo esta na posicao (j,j)
12       for i=(j+1):n
13      //O elemento C(i,j) eh o elemento na posicao (i,j) of L na decomposicao LU de A
14          C(i,j)=C(i,j)/C(j,j);
15       //Linha i < Linha i - C(i,j)*Linha j
16      //Somente os elementos da diagonal ou acima dela sao computados
17      //(aqueles que compoem a matrix U)
18          C(i,j+1:n+1)=C(i,j+1:n+1)-C(i,j)*C(j,j+1:n+1);
19      end
20 end
21
22 x=zeros(n,1);
23 // Calcula x, sendo Ux=C(1:n,n+1)
24 x(n)=C(n,n+1)/C(n,n);
25 for i=n-1:-1:1
26       x(i)=(C(i,n+1)-C(i,i:n)*x(i:n))/C(i,i);
27 end
28 C=C(1:n,1:n);
29 endfunction
30
31
```

O "for"é criado para selecionar da primeira coluna até a penúltima, com exceção de b apenas. Na linha seguinte, o "if"é a condição do pivô ser 0 para podermos efetuar a troca de linhas. E, por fim, a troca de linhas é exercida caso a condição seja satisfeita, através da troca feita de j por j+1. Agora, devido ao problema resolvido do pivô da terceira linha ser igual a 0 que encontramos anteriormente, foi possível encontrar x e C de A1 e B1. Assim, a terceira linha foi trocada com a quarta e o pivô não é mais 0. Foi encontrado resultados que serão mostrados na imagem a seguir.

Figura 3: console 3.1

```
--> [x, C]=Gaussian_Elimination_2(A1, b1)
 x  =

  -0.3247863
  -0.1709402
   0.1965812
  -0.0769231
C  =

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

Na 2ª matriz dada, o scilab nos retorna os valores. Entretanto, quando tentamos fazer a mão a

multiplicação Ax = b, a resposta não é igual a encontrada pelo programa. Isso se deve ao "erro de arredondamento"citado por David Poole em "Álgebra Linear: Uma Introdução Moderna": esses erros de arredondamento podem causar efeitos dramáticos nos cálculos, o que aconteceu nessa situação. É um tipo de problema que chamamos de "malcondicionados".

Figura 4: console 3.2

```
--> [x, C]=Gaussian_Elimination_2(A2, b2)
 x  =

  -1.000D+20
   0.
   1.
 C  =

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

4 Questão 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=$[10^{-20}\ 10^{-20}$ **1;** 10^{-20} **1 1; 1 2 1]** **e o vetor b3=b2.**

```
function [x, C]=Gaussian_Elimination_3(A, b)
C=[A,b];
[n]=size(C,1);


for j=1:(n-1)
    if C(j,j) == 0
        m = j:n
        [p,l] = max(abs(C(m,j)))
        C([j, l], :)=C([l, j], :)
     end



 //O pivo esta na posicao (j,j)
     for i=(j+1):n
    //O elemento C(i,j) eh o elemento na posicao (i,j) of 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 dela 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

```

O código acima mostra a parte que foi alterada, que foi o "for"que troca as linhas. Eu declaro a variável m que recebe de j até o último elemento n da coluna, e logo depois na linha seguinte encontro qual desses elementos é o maior em módulo através do uso de max e abs. Após isso, efetuo a troca de linhas.

Figura 5: console 4.1

```
--> [x, C]=Gaussian_Elimination_3(A2,b2);

--> x
 x  =

   1.
  -1.
   1.

--> C
 C  =

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

Agora, a tabela teria a linha do número 1 como pivô A_{jj} (1 2 1), que é o maior número possível para esse pivô. Sendo assim, os cálculos para a fatoração LU serão feitos de modo mais simples e um resultado "mais correto"para x será encontrado, porque um número com maior valor absoluto no pivô em relação aos abaixos nessa mesma coluna favorecem isso. O mais ideal é que pivôs sejam os elementos com maior valor absoluto da coluna, o que faremos pelas trocas de linhas. Entretanto, as questões relacionadas ao arredondamento permanecem em C.

Agora, é exposto o resultado com a nova matriz A3 e b3:

Figura 6: console 4.2

```
--> [x, C]=Gaussian_Elimination_3(A3,b3);

--> x
 x  =

   0.
  -1.
   1.

--> C
 C  =

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

Nessa nova situação, o x ainda tem valores inteiros e corretos, mas o C permanece com o mesmo problema que anteriormente.

5 Questão 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.

```
function [x, C, P]=Gaussian_Elimination_4(A, b)
C=[A,b];
[n]=size(C,1);

P = eye(n,n);

for j=1:(n-1)
    m = j:n
    [p,l] = max(abs(C(m,j)))
    C([j, l], :)=C([l, j], :)
    P([j, l],:)=P([l, j],:)
    break


 //O pivo esta na posicao (j,j)
     for i=(j+1):n
    //O elemento C(i,j) eh o elemento na posicao (i,j) of 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 dela 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


```

Agora, tive que adicionar uma nova matriz identidade P, para assim poder retornar a matriz de permutação utilizada. Seguindo a mesma lógica do item 4, apenas tirei o "if"nesse laço for da linha 7 e coloquei as trocas de linha na matriz de permutação. Assim, retornou os valores que desejamos. A seguir, mostro a saída encontrada.

Figura 7: console 5

```
--> [x, C, P]=Gaussian_Elimination_4(A3,b3);

--> x
 x  =

   1.
  -1.
   1.

--> C
 C  =

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

--> P
 P  =

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

Parece que o C nos dá mais informações, porém, curiosamente, os valores de x são diferentes do encontrado no item anterior. Esses problemas de arredondamento se mostram mais evidentes quando claramento alteram o resultado final x.

6 Questão 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 Axi=bi, 1 i m . 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=[2 4 -1 5 ;0 1 0 3 ; 2 2 -1 1 ; 0 1 1 5]. 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.

```
function[X] = Resolve_Com_LU(P, C, B)

    [t] = size(C,1);
    ident = eye(t,t)

    L = tril(C, -1) + ident;
    U = triu(C);
    X =[];

    [SC] = size(B,2);

    Z = (P')*L*U;


    for j=1:SC
        b = B(:,j);
        x = inv(Z)*b;
        X = [X x];
    end

endfunction


```

No código, usamos P, C e B como argumentos de entrada. P e C já foram definidos em questões anteriores, enquanto B é a nova matriz que vamos declarar (B1,B2,B3). t é o tamanho de C, para fazermos em seguida uma matriz identidade desse tamanho. Depois, através de funções do próprio scilab, as matrizes L e U são criadas. Em seguida, pego o tamanho da coluna de C para ser o tamanho de j no laço for criado a seguir, e nesse laço as operações são feitas.

Após isso, fiz testes com a matriz A1 e B1:

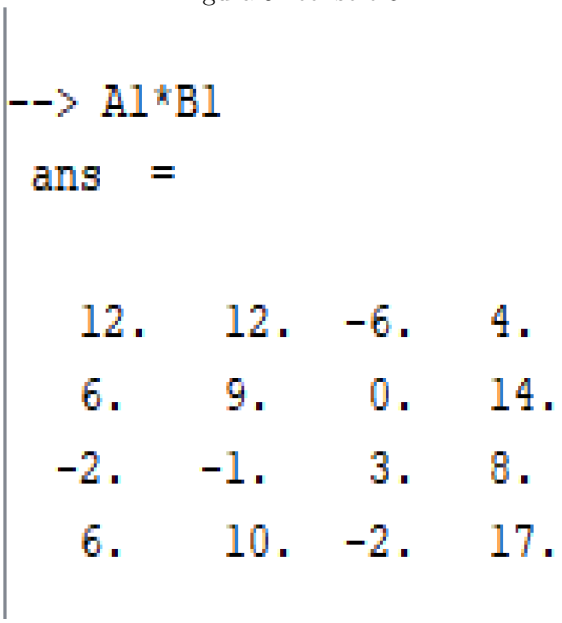
Figura 8: console 6.1

```
--> [X] = Resolve_Com_LU(P, C, B1)
 X  =

   3.243D+16   4.323D+16  -2.162D+16   4.323D+16
   1.801D+16   2.402D+16  -1.201D+16   2.402D+16
   7.206D+15   9.608D+15  -4.804D+15   9.608D+15
   0.          0.          0.          1.
```

Parece ter sido um resultado com várias casas decimais, com exceção da linha debaixo. Vamos conferir se o resultado está correto com a multiplicação A1*X. Se for igual a B1, é que a matriz encontrada é a certa.

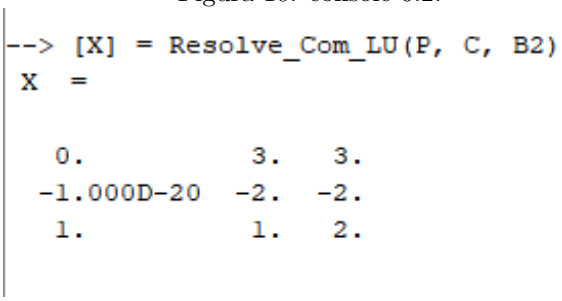
Figura 9: console 6.1.1



A matriz encontrada não foi a esperada, e isso infere que algo não deu certo. Quando executo a Eliminação Gaussiana 4 com essa matriz, os resultados encontrados são NaN, o que nos diz que possivelmente o problema está no código quando ele trata de matrizes 4x4, tendo em vista que em questões anteriores, com códigos que fiz também, essa matriz nos deu números reais.

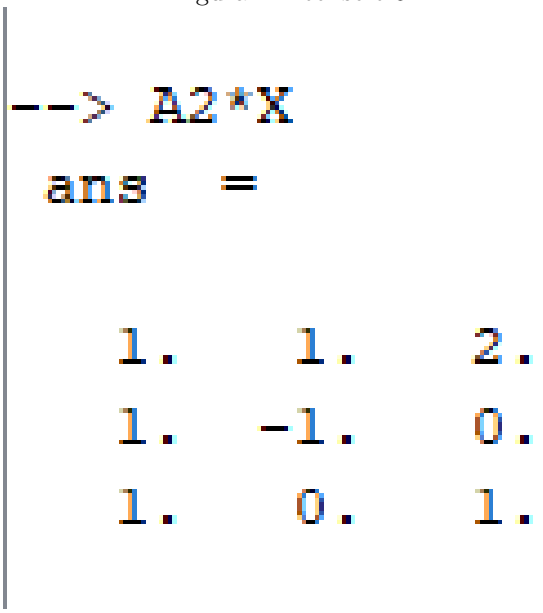
Agora, vamos pra A2, b2 e B2:

Figura 10: console 6.2.



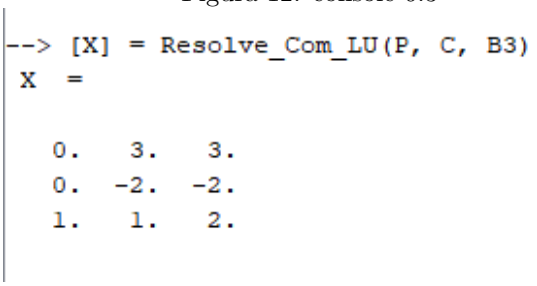
Testando a multiplicação A2*X para vermos se o resultado bate com B2, concluimos que funcionou, pois o resultado é a matriz B2!

Figura 11: console 6.2.1



Agora, a matriz tem 8 de seus 9 valores inteiros, e apenas um número com várias casas decimais. Testaremos com A3, b3 E B3:

Figura 12: console 6.3



Finalmente, temos um item com todos os valores inteiros, sem nenhum problema de arredondamento! Testamos, com os mesmos métodos anteriores, se o resultado é correto. Chegamos a conclusão que sim, e, no final, conseguimos encontrar resultados satisfatórios.