Aula prática 1
Algoritmos de resolução de sistemas lineares

Will Sena*

Contents

*wllsena@protonmail.com

Algoritmo de eliminação Gaussiana

A função Scilab a seguir, implementa o algoritmo de eliminação Gaussiana para resolver um sistema quadrado $A\boldsymbol{x} = \boldsymbol{b}$ com A invertível, obtendo também a decomposição LU da matriz A. Nesta versão, supomos que os elementos diagonais (pivôs) da matriz dos coeficientes ao longo do processo são sempre não nulos.

```
///////////////////////////////////////////////////////////////
//Variáveis de saída:
///////////////////////////////////////////////////////////////
//x: solução do sistema Ax=b (assumimos que tal solução existe)
//C: Seja A=LU a decomposição LU de A.
//Então C(i,j)=L(i,j) para i>j e C(i,j)=U(i,j) para j>=i.
///////////////////////////////////////////////////////////////

function [x,C]=Gaussian_Elimination_1(A,b)

  C=[A,b];
  [n]=size(C,1);

  for j=1:(n-1)
    //O pivô está na posição (j,j)
    for i=(j+1):n
      //O elemento C(i,j) é o elemento na posição (i,j) of L
      //na decomposição 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 da diagonal
      // são computados (aqueles que compõem 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
```

1)

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

```
--> A = [1 2; 3 -5;];

--> b = [5; 4];

--> x = Gaussian_Elimination_1(A, b)
x =

3.
1.

--> inv(A) * b
ans =

3.
1.

--> A = [2 1; 1 3;];

--> b = [2; -4];

--> x = Gaussian_Elimination_1(A, b)
x =

2.
-2.
```

```
--> inv(A) * b
ans =

2.
-2.
```

Dado que $\boldsymbol{x} = inv(A) * \boldsymbol{b}$, os resultados da função estão corretos.

2)

Agora teste com a matriz $A\mathbf{1} = [1\ \ -2\ 5\ 0; 2\ \ -4\ 1\ 3; -1\ 1\ 0\ 2; 0\ 3\ 3\ 1]$ e com o vetor $\boldsymbol{b1} = [1; 0; 0; 0]$:

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

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

--> x1 = Gaussian_Elimination_1(A1, b1)
x1 =

Nan
Nan
Nan
Nan
```

Como pivô3 $= A1_{3,3} = 0$, a função retorna um vetor de **Nan (Not a number)** resultante das divisões por 0.

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 $A\mathbf{1}$ e o vetor $\boldsymbol{b1}$ dados. Agora teste-a com a matriz $A\mathbf{2} = [0\ 10^{-20}\ 1; 10^{-20}\ 1\ 1; 1\ 2\ 1]$ e o vetor $\boldsymbol{b2} = [1; 0; 0]$:

Definindo *Gaussian_ Elimination_ 2* como *Gaussian_ Elimination_ 1* acrescentando o código abaixo na linha 17 (abaixo do iterador de j):

```
17 if C(j,j) == 0
18   p = find(C([j:n], j), 1);
19   k = j + p - 1
20   C([j, k], :) = C([k, j], :);
21 end
```

Testando com $A1$ e $\boldsymbol{b}1$:

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

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

--> x1 = Gaussian_Elimination_2(A1, b1)
x1 =

-0.3247863
-0.1709402
0.1965812
-0.0769231

--> inv(A1) * b1
ans =

-0.3247863
-0.1709402
0.1965812
-0.0769231
```

Testando com $A2$ e $\boldsymbol{b}2$:

```
--> A2 = [0 10^(-20) 1; 10^(-20) 1 1; 1 2 1];

--> b2 = [1; 0; 0];
```

```
--> x2 = Gaussian_Elimination_2(A2, b2)
x2 =

-1.000D+20
0.
1.

--> inv(A2) * b2
ans =

1.
-1.
1.
```

$\boldsymbol{x}3 \neq inv(A2) * \boldsymbol{b}2$. Esta diferença é causada por pivô1 $= A2_{2,1} = 10^{-20}$ ser extremamente pequeno, assim o computador o arredonda para um número um pouco maior, especificamente para 10^{-16} (16 dígitos de precisão). Quando valores são divididos pelo pivô1' = pivô1 $* 10^4$ há grandes discrepâncias.

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 $A\mathbf{2}$ e o vetor $\boldsymbol{b}\mathbf{2}$ dados. Agora com a matriz $A\mathbf{3} = [10^{-20}\ 10^{-20}\ 1; 10^{-20}\ 1\ 1; 1\ 2\ 1]$ e o vetor $\boldsymbol{b}\mathbf{3} = \boldsymbol{b}\mathbf{2}$

Definindo *Gaussian_ Elimination_ 3* como *Gaussian_ Elimination_ 2* alterando a linha 18 (onde **p** é definido no código do exercício anterior) para:

```
18 [_, p] = max(abs(C([j:n], j)));
```

Testando com $A2$ e $b2$:

```
--> A2 = [0 10^(-20) 1; 10^(-20) 1 1; 1 2 1];
```

```
--> b2 = [1; 0; 0];

--> x2 = Gaussian_Elimination_3(A2, b2)
x2 =

1.
-1.
1.

--> inv(A2) * b2
ans =

1.
-1.
1.
```

Testando com $A3$ e $b3$:

```
--> A3 = [10^(-20) 10^(-20) 1; 10^(-20) 1 1; 1 2 1];

--> b3 = [1; 0; 0];

--> x3 = Gaussian_Elimination_3(A3, b3)
x3 =

0.
-1.
1.

--> inv(A3) * b3
ans =

1.
-1.
1.
```

Estes resultados foram explicados no exercício anterior.

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 $\boldsymbol{b3}$ dados.

Definindo *Gaussian_Elimination_4* como *Gaussian_Elimination_3* com a seguinte alterações:

1. Trocando a linha 10 (onde a função é definida) por:

```
10 function [x, C, P] = Gaussian_Elimination_4(A, b)
```

2. Adicionado o seguinte código na linha 14 (abaixo de onde **n** é definido):

```
14 P = eye(n, n);
```

3. Trocando todo o corpo do **if** nas linhas 18, 19, 20, 21 e 22 (onde o pivô foi verificado como nulo) por:

```
18 [_, p] = max(abs(C([j:n], j)));
19 k = j + p - 1
20 C([j, k], :) = C([k, j], :);
21 P([j, k], :) = P([k, j], :);
```

Testando com $A3$ e $b3$:

```
--> A3 = [10^(-20) 10^(-20) 1; 10^(-20) 1 1; 1 2 1];

--> 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.

--> inv(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 $A\boldsymbol{x} = \boldsymbol{b}$ 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 $n \times m$ e retorne uma matriz X, com a mesma ordem de B, cujas colunas sejam as soluções dos sistemas lineares $A\boldsymbol{x}_i = \boldsymbol{b}_i$, $1 \leq i \leq m$.
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 $B2 = [1\ \ 1\ \ 2; 1\ \ -1\ \ 0; 1\ \ 0\ \ 1]$. Finalmente, teste com a matriz $A3$ dada anteriormente e com a matriz $B3 = B2$.

```
function [X] = Resolve_com_LU(C, B, P)
  n = size(C, 1);

  if n ~= size(B, 1)
    error("Inconsistent row/column dimensions.");
  end

  if nargin < 3
    P = eye(n, n);
  end

  Y = P * B;
  for i = 2:n
    Y(i, :) = Y(i, :) - C(i, 1:i-1) * Y(1:i-1, :);
  end

  X = Y;
  X(n, :) = X(n, :)/C(n, n);
  for i = n-1:-1:1
    X(i, :) = (X(i, :) - C(i, i+1:n) * X(i+1:n, :))/C(i, i);
  end
endfunction
```

Testando para $A1$ e $B1$

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

--> B1 = [2 4 -1 5; 0 1 0 3; 2 2 -1 1; 0 1 1 5];

--> [_ C1 P1] = Gaussian_Elimination_4(A1, B1(:, 1));

--> X1 = Resolve_com_LU(C1, B1, P1)
X1  =
```

```
-2.034188   -1.9316239   1.4529915   0.8119658
-0.6495726  -0.7008547   0.6068376   0.4273504
0.5470085   0.9059829  -0.2478632   1.008547
0.3076923   0.3846154  -0.0769231   0.6923077

--> inv(A1) * B1
ans  =

-2.034188   -1.9316239   1.4529915   0.8119658
-0.6495726  -0.7008547   0.6068376   0.4273504
0.5470085   0.9059829  -0.2478632   1.008547
0.3076923   0.3846154  -0.0769231   0.6923077
```

Testando para $A2$ e $B2$

```
--> A2 = [0 10^(-20) 1; 10^(-20) 1 1; 1 2 1];

--> B2 = [1 1 2; 1 -1 0; 1 0 1];

--> [_ C2 P2] = Gaussian_Elimination_4(A2, B2(:, 1));

--> X2 = Resolve_com_LU(C2, B2, P2)
X2  =

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

--> inv(A2) * B2
ans  =

0.          3.   3.
-1.000D-20  -2.  -2.
1.          1.   2.
```

Testando para $A3$ e $B3$

```
--> A3 = [10^(-20) 10^(-20) 1; 10^(-20) 1 1; 1 2 1];
```

```
--> B3 = [1 1 2; 1 -1 0; 1 0 1];

--> [_ C3 P3] = Gaussian_Elimination_4(A3, B3(:, 1));

--> X3 = Resolve_com_LU(C3, B3, P3)
X3  =

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

--> inv(A3) * B3
ans  =

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

Todos os resultados estão corretos.

Extra)

Definindo *Gaussian_Elimination* como *Gaussian_Elimination_4*:

```
/////////////////////////////////////////////////////////////////
//Variáveis de saída:
/////////////////////////////////////////////////////////////////
//x: solução do sistema Ax=b (assumimos que tal solução existe)
//C: Seja A=LU a decomposição LU de A.
//Então C(i,j)=L(i,j) para i>j e C(i,j)=U(i,j) para j>=i.
/////////////////////////////////////////////////////////////////

function [x, C, P] = Gaussian_Elimination(A, b)

  C=[A,b];
  [n]=size(C,1);
  P = eye(n, n);

```

```
  for j=1:(n-1)
    //O pivô está na posição (j,j)
    [_, p] = max(abs(C([j:n], j)));
    k = j + p - 1
    C([j, k], :) = C([k, j], :);
    P([j, k], :) = P([k, j], :);
    for i=(j+1):n
      //O elemento C(i,j) é o elemento na posição (i,j) of L
      //na decomposição 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 da diagonal
      // são computados (aqueles que compõem 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
```