Fundação Getúlio Vargas
Escola de Matemática Aplicada
Álgebra Linear Numérica

Algoritmos de Resolução de Sistemas Lineares

Aluno:
- Jeann da Rocha Silva
Professor:
- Antonio Carlos Saraiva Branco

Março
2024

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.

Vamos efetuar testes simples com a função *Gaussian_Elimination_1* (arquivo em anexo) para matrizes de dimensões $1, 2, 3$ e 4, que mostrarão a eficácia da função para matrizes inversíveis, exceto por alguns detalhes a serem devidamente justificados nos próximos tópicos.

Teste 1 - $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}$ $\left(\textbf{solução esperada: } x = \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right)$

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

   1.   0.   0.
   0.   1.   0.
   0.   0.   1.

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

   1.
   2.
  -3.

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

   1.
   2.
  -3.
 C  =

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

Teste 2 - $A = [5], b = [11]$ $\left(\textbf{solução esperada: } x = \left[\frac{11}{5}\right] = [2.2], C = [5]\right)$

```
--> A = [5]
 A  =

   5.

--> b = [11]
 b  =

   11.

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

   2.2
 C  =

   5.
```

Teste 3 - $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$ $\left(\textbf{solução esperada: } x = \begin{bmatrix} -4 \\ \frac{9}{2} \end{bmatrix}, C = \begin{bmatrix} 1 & 2 \\ 3 & -2 \end{bmatrix}\right)$

```
--> A = [1 2; 3 4]
 A  =

   1.   2.
   3.   4.

--> b = [5; 6]
 b  =

   5.
   6.

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

  -4.
   4.5
 C  =

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

Teste 4 - $A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ $\left(\textbf{sol. esp.: } x = \begin{bmatrix} -\frac{5}{6} \\ 3 \\ -\frac{3}{2} \\ \frac{1}{3} \end{bmatrix}, C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 3 \\ 1 & 3 & 2 & 6 \\ 1 & 7 & 6 & 6 \end{bmatrix} \right)$

```
--> A = [1 1 1 1; 1 2 3 4; 1 4 9 16; 1 8 27 64]
 A  =

   1.   1.   1.    1.
   1.   2.   3.    4.
   1.   4.   9.    16.
   1.   8.   27.   64.

--> b = [1; 2; 3; 4]
 b  =

   1.
   2.
   3.
   4.

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

  -0.8333333
   3.
  -1.5
   0.3333333
 C  =

   1.   1.   1.   1.
   1.   1.   2.   3.
   1.   3.   2.   6.
   1.   7.   6.   6.
```

Como se vê, não houve problemas para descrever a decomposição LU (compactada na matriz C) e achar a solução x do sistema. Entretanto, veremos que a depender da matriz, podem haver problemas no processo de escalonamento, que serão corrigidos no decorrer dos itens.

2) Agora teste com a matriz $A_1 = \begin{bmatrix} 1 & -2 & 5 & 0 \\ 2 & -4 & 1 & 3 \\ -1 & 1 & 0 & 2 \\ 0 & 3 & 3 & 1 \end{bmatrix}$ **e com o vetor** $b_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$**.**

Aplicando a matriz A_1 e o vetor b_1 à função *Gaussian_Elimination_1*, encontramos o seguinte problema na solução:

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

que decorre do fato de não estarmos tratando o possível surgimento de zeros nas posições pivôs durante o processo de eliminação, gerando respostas inexistentes para a solução (Nan) e a ocorrência de valores inexistentes (resultantes de indeterminações de 0 com infinito) ou de valor tão grande quanto se queira, isto é, infinito (Inf ou -Inf), devido à divisão por 0 para a matriz C. Isto exige que permutemos linhas, portanto, modificando a função (como será feito na parte 3), a fim de que não ocorram eventuais divisões por zero nas linhas subsequentes. Vale ressaltar que ao permutar linhas, não estamos mais num caso de decomposição $A = LU$, mas sim de decomposição $PA = LU$, uma vez que as permutações P_i utilizadas também são levadas em conta.

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_1 e o vetor b_1 dados. Agora teste-a com a matriz $A_2 = \begin{bmatrix} 0 & 10^{-20} & 1 \\ 10^{-20} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$ e o vetor $b_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.

A nova função *Gaussian_Elimination_2* terá a estrutura semelhante à *Gaussian_Elimination_1*, porém tendo no início do primeiro **for** uma estrutura de verificação e procura do primeiro elemento não-nulo da coluna para preencher a posição de pivô a partir de uma permutação de linhas, se necessário (detalhes do código no arquivo anexado).
Aplicando esta função à matriz A_1 e ao vetor b_1, conseguimos encontrar a solução ("aproximada"), como segue abaixo:

```
--> [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.
```

A fim, de analisar o quão eficaz é a solução, isto é, o quão pequeno é o erro de aproximação, analisamos o produto A_1x (que deve ser $\approx b$) e, mais precisamente, a norma $||A_1x - b_1||$ (que deve ser ≈ 0), como segue abaixo:

```
--> A1*x
 ans  =

   1.0000000
  -1.388D-16
   1.110D-16
  -1.943D-16

--> norm(A1*x-b1)
 ans  =

   2.858D-16
```

Então, de fato, temos um resultado muito bem aproximado, sendo o erro de cerca de 10^{-16}.

Agora, aplicando a matriz A_2 e o vetor b_2 à função *Gaussian_Elimination_2*, encontramos os seguintes resultados:

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

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

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

   1.
   0.
   0.

--> [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
```

Entretanto, fazendo A_2x e $||A_2x - b_2||$, obtemos:

```
--> A2*x
 ans  =

   1.
   0.
  -1.000D+20

--> norm(A2*x-b2)
 ans  =

   1.000D+20
```

que certamente demonstra um problema no algoritmo implementado, já que o erro é de 10^{20}. Isto ocorre pois no escalonamento, após a permutação da primeira linha com a segunda, ao zerar o primeiro elemento da terceira linha, estamos efetuando uma divisão por -10^{-20} na primeira linha, isto é, um produto por -10^{20} que, ao ser somado à terceira linha não é entendido pelo computador como $2-10^{20}$, mas sim como -10^{20} devido a ordem de grandeza enorme de 10^{20}, ou seja, $2-10^{20} \approx -10^{20}$ (e, analogamente, $1-10^{2}0 \approx -10^{20}$), como se observa no teste abaixo:

```
--> 2 - 10^20
 ans  =

  -1.000D+20

--> -10^20
 ans  =

  -1.000D+20
```

A priori, isto não deve causar problemas, mas conforme vamos efetuando mais e mais operações envolvendo números com essa escala de grandeza (como ocorre na terceira linha pela segunda), obtemos aproximações cada vez piores gerando um resultado errado ao final. Logo, é necessário corrigir este problema tomando valores para o pivô que não tenham valor absoluto pequeno para que na operação de divisão para as linhas subsequentes não hajam erros de aproximação devido à baixa escala de grandeza na divisão (ou alta escala na multiplicação), como será feito no próximo item.

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_2 e o vetor b_2 dados. Agora com a matriz $A_3 = \begin{bmatrix} 10^{-20} & 10^{-20} & 1 \\ 10^{-20} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}$ **e o vetor** $b_3 = b_2$**.**

Novamente, A nova função *Gaussian_Elimination_3* terá a estrutura semelhante à função *Gaussian_Elimination_2*, porém, em vez de procurar o primeiro elemento não-nulo da coluna correspondente abaixo da posição pivô, ela transforma todos em suas versões de valor absoluto (com a função **abs**) para tomar a posição do maior deles (com a função **max**) para preencher a posição de pivô e permuta as linhas em seguida tal como era na função anterior (detalhes do código no arquivo anexado).
Aplicando esta função à matriz A_2 e ao vetor b_2, conseguimos encontrar a solução, como segue abaixo:

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

   1.
  -1.
   1.
 C  =

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

Para verificar que, de fato, a solução está correta, efetuamos novamente o cálculo de A_2x e $||A_2x - b_2||$ abaixo:

```
--> A2*x
 ans  =

   1.
   0.
   0.

--> norm(A2*x - b2)
 ans  =

   0.
```

Agora, aplicando a matriz A_3 e o vetor b_3 à função *Gaussian_Elimination_2*, encontramos o seguinte resultado:

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

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

--> b3=b2
 b3  =

   1.
   0.
   0.

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

   0.
  -1.
   1.
 C  =

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

Analisando A_3x e $||A_3x - b_3||$, vemos novamente outro erro:

```
--> A3*x
 ans  =

   1.
   0.
  -1.

--> norm(A3*x-b3)
 ans  =

   1.
```

Tal erro (de valor 1) ocorre pois, neste caso, o pivô não é 0, porém é um valor muito pequeno e a mesma ideia do item anterior se aplica, sendo, portanto, necessário, efetuar permutações para obter os maiores pivôs possíveis (em módulo), independente do pivô inicial não ser 0, a fim de não obter erros de aproximação evidentes devido a números pequenos, como será feito no item a seguir.

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 A_3 e o vetor b_3 dados.

Para esta nova função *Gaussian_Elimination_4*, basta repetir toda a estrutura da função *Gaussian_Elimination_3* e remover a condicional que efetuava a permutação para a linha com o pivô máximo caso o pivô inicial fosse 0, permanecendo, assim, obrigatória a permutação de linhas (se houver um valor maior em módulo). O arquivo da função está anexado.
Aplicando esta função à matriz A_3 e ao vetor $\boldsymbol{b}_3$, conseguimos encontrar a solução, como segue abaixo:

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

   1.
  -1.
   1.
 C  =

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

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

Para verificar que, de fato, a solução está correta, efetuamos novamente o cálculo de $A_3\boldsymbol{x}$ e $||A_3\boldsymbol{x} - \boldsymbol{b}_3||$ abaixo:

```
--> A3*x
 ans  =

   1.
   0.
   0.

--> norm(A3*x-b3)
 ans  =

   0.
```

Observe que a matriz de permutação também foi obtida e é a matriz que tem apenas a diagonal secundária composta por 1's.

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 $n \times m$ e retorne uma matriz X, com a mesma ordem de B, cujas colunas sejam as soluções dos sistemas lineares $Ax_i = b_i, 1 \leq i \leq m$. Observação: talvez você ache necessário passar outra(s) variável(is) de entrada para essa função.

Teste a sua função com a matriz A_1 dada anteriormente e com a matriz $B_1 = \begin{bmatrix} 2 & 4 & -1 & 5 \\ 0 & 1 & 0 & 3 \\ 2 & 2 & -1 & 1 \\ 0 & 1 & 1 & 5 \end{bmatrix}$.

Teste também com a matriz A_2 dada anteriormente e com a matriz $B_2 = \begin{bmatrix} 1 & 1 & 2 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$.

Finalmente, teste com a matriz A_3 dada anteriormente e com a matriz $B_3 = B_2$.

O algoritmo em questão resolverá o sistema $LUx_i = Ax_i = b_i$ em dois passos. Primeiro resolver $Ly_i = b_i$ e, para o vetor y encontrado, resolver $Ux_i = y_i$. Supondo possíveis permutações de linha para chegar na forma LU, teremos ainda que passar a matriz P como parâmetro para a função *Resolve_com_LU* e, o sistema em questão será $LUX = PAX = PB$ (isto é, se $AX = B$, então $PAX = PB$).
Aplicando esta função à matriz A_1 e à matriz B_1, conseguimos encontrar a solução:

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

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

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

Como sempre, calculando A_1X_1 e $||A_1X_1 - B_1||$, esperamos obter um resultado com menor erro de aproximação, que de fato ocorre, como segue abaixo:

```
--> A1*X1
 ans  =

   2.          4.  -1.          5.
   4.441D-16   1.   5.551D-17   3.
   2.          2.  -1.          1.
   0.          1.   1.          5.

--> norm(A1*X1 - B1)
 ans  =

   1.164D-15
```

Analogamente, para a matriz A_2 e a matriz B_2, tem-se:

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

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

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

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

--> A2*X2
 ans  =

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

--> norm(A2*X2-B2)
 ans  =

   0.
```

Por fim, para a matriz A_3 e a matriz B_3, temos:

```
--> B3=B2
 B3  =

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

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

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

--> A3*X3
 ans  =

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

--> norm(A3*X3-B3)
 ans  =

   0.
```

Como se pode ver, a função executa de forma eficiente para os exemplos em questão.