Álgebra Linear Numérica
Aula Prática 1

Pedro Henrique Coterli
Ciência de Dados e Inteligência Artificial

Março de 2024

Questão 1

Para esse exercício, foram feitos três testes. Abaixo estão as matrizes A, os vetores b e as saídas x da função, com a comparação entre o produto Ax e o vetor objetivo b.

Teste 1

$$A\_teste\_1 = \begin{bmatrix} 1 & 4 \\ 8 & 5 \end{bmatrix} \qquad b\_teste\_1 = \begin{bmatrix} 26 \\ 46 \end{bmatrix}$$

```
"x:"

 2.
 6.

"Ax:"

 26.
 46.
```

Resultado do teste 1

Teste 2

$$A\_teste\_2 = \begin{bmatrix} 7 & 5 & 1 \\ 0 & 6 & 8 \\ 9 & 3 & 4 \end{bmatrix} \qquad b\_teste\_2 = \begin{bmatrix} 45 \\ 44 \\ 40 \end{bmatrix}$$

```
"x:"

 2.
 6.
 1.0000000

"Ax:"

 45.
 44.
 40.
```

Resultado do teste 2

Teste 3

$$A\_teste\_3 = \begin{bmatrix} 2 & 8 & 9 & 3 \\ 6 & 3 & 1 & 1 \\ 5 & 1 & 6 & 4 \\ 7 & 5 & 7 & 3 \end{bmatrix} \qquad b\_teste\_3 = \begin{bmatrix} 95 \\ 46 \\ 54 \\ 85 \end{bmatrix}$$

```
"x:"

 3.
 7.0000000
 2.0000000
 5.0000000

"Ax:"

 95.
 46.000000
 54.000000
 85.000000
```

Resultado do teste 3

Portanto, aparentemente, a função está funcionando.

Questão 2

Testando com a matriz e o vetor indicados, obtemos o seguinte resultado para x:

```
"x1:"

 Nan
 Nan
 Nan
 Nan
```

Resultado do experimento da questão 2

Isso se deve ao fato de um dos pivôs dessa matriz ser 0, como mostrado abaixo.

$$\begin{bmatrix} 1 & -2 & 5 & 0 & 1 \\ 2 & -4 & 1 & 3 & 0 \\ -1 & 1 & 0 & 2 & 0 \\ 0 & 3 & 3 & 1 & 0 \end{bmatrix} \xrightarrow[]{\substack{L2 - 2 \cdot L1 \\ L3 + L1}} \begin{bmatrix} 1 & -2 & 5 & 0 & 1 \\ 0 & \mathbf{0} & -9 & 3 & -2 \\ 0 & -1 & 5 & 2 & 1 \\ 0 & 3 & 3 & 1 & 0 \end{bmatrix}$$

Devido a isso, o algoritmo acaba realizando divisões por 0 com as linhas da matriz, o que causa a impossibilidade de interpretação numérica para o valor de x.

Questão 3

Com a nova função `Gaussian_Elimination_2`, agora, permutamos as linhas da matriz quando um pivô é igual a 0, trocando essa linha com a primeira abaixo dela que possua um valor diferente de 0 na coluna do pivô analisado.

```
function [x, C]=Gaussian_Elimination_2(A, b)
    C=[A,b];
    [n]=size(C,1);
    for j=1:(n-1)
        // Se o pivô for 0, permuta as linhas
        if C(j,j) == 0 then
            // Encontrando a primeira linha sem zero abaixo do pivô
            row_without_zero = find(C(j+1:n,j),1)
            // Trocando essas linhas
            C([j j+row_without_zero],:) = C([j+row_without_zero j],:)
        end
```

Alterações para a função `Gaussian_Elimination_2`

Usando essa função, obtemos um resultado mais adequado:

```
"A1*x1_2:"

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

"b1:"

 1.
 0.
 0.
 0.
```

Novo resultado para A1 com a função `Gaussian_Elimination_2`

Apesar das aproximações, esses 3 últimos valores são muito próximos de 0. Dessa forma, agora temos uma saída aceitável para a função nesse caso.

Entretanto, ao testá-la com a matriz $A2$ e com o vetor $b2$ determinados, obtemos o seguinte:

```
"A2*x2:"

 1.
 0.
-1.000D+20

"b2:"

 1.
 0.
 0.
```

Resultado para $A2$ com a função `Gaussian_Elimination_2`

Definitivamente, isso não está correto. Depois de algumas análises das iterações do programa, descobre-se que esse problema ocorre devido a um arredondamento realizado pelo código no momento de fazer as operações entre linhas, e isso se dá pelo fato de termos um pivô muito pequeno (10^{-20}). Vamos ver como isso acontece:

$$\begin{bmatrix} \mathbf{0} & 10^{-20} & 1 & 1 \\ 10^{-20} & 1 & 1 & 0 \\ 1 & 2 & 1 & 0 \end{bmatrix} \xrightarrow{L1 \longleftrightarrow L2}$$

$$\begin{bmatrix} 10^{-20} & 1 & 1 & 0 \\ 0 & 10^{-20} & 1 & 1 \\ 1 & 2 & 1 & 0 \end{bmatrix} \xrightarrow{L3 - 10^{-20} \cdot L1}$$

$$\begin{bmatrix} 10^{-20} & 1 & 1 & 0 \\ 0 & 10^{-20} & 1 & 1 \\ 0 & 2 - 10^{20} & 1 - 10^{20} & 0 \end{bmatrix} \xrightarrow{L3 - \frac{2 - 10^{20}}{10^{-20}} \cdot L2}$$

$$\begin{bmatrix} 10^{-20} & 1 & 1 & 0 \\ 0 & 10^{-20} & 1 & 1 \\ 0 & 0 & \mathbf{1 - 3 \cdot 10^{20} + 10^{40}} & \mathbf{10^{40} - 2 \cdot 10^{20}} \end{bmatrix}$$

Nesse momento, o programa acaba aproximando os dois termos destacados para 10^{40}, desconsiderando os outros valores.

```
--> 1-3*10^20+10^40
 ans  =

   1.000D+40

--> 10^40-2*10^20
 ans  =

   1.000D+40
```

Aproximações da função `Gaussian_Elimination_2`

Assim, a matriz se torna a mostrada abaixo.

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

Matriz U da $A2$ obtida com a função `Gaussian_Elimination_2`

E o vetor x obtido do processo de substituição com essa matriz é o seguinte:

```
"x2:"

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

Vetor $x2$ do sistema com $A2$ e $b2$ obtido com a função `Gaussian_Elimination_2`

que, multiplicado pela matriz $A2$, gera algo diferente de $b2$. Portanto, esse é o erro cometido por essa função com esse sistema.

Questão 4

Agora, quando temos um pivô zero, a função `Gaussian_Elimination_3` não apenas troca sua linha pela primeira logo abaixo cujo valor na coluna correspondente não é nulo, mas sim pela linha tal que esse valor é o maior em módulo dentre os disponíveis.

```
function [x, C]=Gaussian_Elimination_3(A, b)
    C=[A,b];
    [n]=size(C,1);
    for j=1:(n-1)
        // Se o pivô for 0, permuta com a linha com o maior pivô em módulo
        if C(j,j) == 0 then
            // Encontrando a linha com o maior pivô em módulo
            moduled_vector = abs(C(j+1:n,j))
            max_pivot_index = find(moduled_vector == max(moduled_vector))
            // Trocando essas linhas
            C([j j+max_pivot_index],:) = C([j+max_pivot_index j],:)
        end
```

Alterações para a função `Gaussian_Elimination_3`

Dessa forma, nossa intenção é evitar pivôs pequenos. Vamos ver se isso funciona:

$$\begin{bmatrix} \mathbf{0} & 10^{-20} & 1 & 1 \\ 10^{-20} & 1 & 1 & 0 \\ 1 & 2 & 1 & 0 \end{bmatrix} \xrightarrow{L1 \longleftrightarrow L3}$$

$$\begin{bmatrix} 1 & 2 & 1 & 0 \\ 10^{-20} & 1 & 1 & 0 \\ 0 & 10^{-20} & 1 & 1 \end{bmatrix} \xrightarrow{L2 - 10^{-20} \cdot L1}$$

$$\begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 - 2 \cdot 10^{-20} & 1 - 10^{-20} & 0 \\ 0 & 10^{-20} & 1 & 1 \end{bmatrix} \xrightarrow{L3 - \frac{10^{-20}}{1 - 2 \cdot 10^{-20}} \cdot L2}$$

$$\begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 - \mathbf{2 \cdot 10^{-20}} & 1 - \mathbf{10^{-20}} & 0 \\ 0 & 0 & 1 - \dfrac{\mathbf{1 - 10^{-20}}}{\mathbf{10^{20} - 2}} & 1 \end{bmatrix}$$

Os termos destacados são muito próximos de 0. Assim, o programa os desconsidera.

```
--> 1-2*10^(-20)
 ans  =

   1.

--> 1-10^(-20)
 ans  =

   1.

--> 1-(1-10^(-20))/(10^20-2)
 ans  =

   1.
```

Aproximações da função `Gaussian_Elimination_3`

Isso transforma a matriz em

```
1.   2.   1.   0.
0.   1.   1.   0.
0.   0.   1.   1.
```

Matriz U da $A2$ obtida com a função `Gaussian_Elimination_3`

Dessa forma, fazendo o processo de substituição para obter o vetor x, chegamos a

```
"x2_2:"

 1.
-1.
 1.
```

Vetor $x2\_2$ do sistema com $A2$ e $b2$ obtido com a função `Gaussian_Elimination_3`

Por fim, multiplicando esse vetor pela matriz $A2$, o resultado obtido é exatamente o vetor $b2$, como mostrado no código abaixo.

```
"x2_2:"

 1.
-1.
 1.

"A2*x2_2:"

 1.
 0.
 0.

"b2:"

 1.
 0.
 0.
```

Resultado para $A2$ com a função `Gaussian_Elimination_3`

Portanto, mais um problema resolvido!

Entretanto, mais uma vez, os resultados agora com a matriz $A3$ e o vetor $b3$ fornecidos não são muito animadores:

```
"x3:"

 0.
-1.
 1.

"A3*x3:"

 1.
 0.
-1.

"b3:"

 1.
 0.
 0.
```

Resultado para $A3$ com a função `Gaussian_Elimination_3`

Vamos ver o que essa 3° função faz com essa matriz:

$$
\begin{bmatrix} 10^{-20} & 10^{-20} & 1 & 1 \\ 10^{-20} & 1 & 1 & 0 \\ 1 & 2 & 1 & 0 \end{bmatrix}
\xrightarrow[]{\substack{L2 - L1 \\ L3 - 10^{-20} \cdot L1}}
$$

$$
\begin{bmatrix} 10^{-20} & 10^{-20} & 1 & 1 \\ 0 & 1 - 10^{-20} & 0 & -1 \\ 0 & 1 & 1 - 10^{20} & -10^{20} \end{bmatrix}
\xrightarrow{L3 - \frac{1}{1 - 10^{-20}} \cdot L2}
$$

$$
\begin{bmatrix} 10^{-20} & 10^{-20} & 1 & 1 \\ 0 & 1 - \mathbf{10^{-20}} & 0 & -1 \\ 0 & 0 & 1 - 10^{20} & \dfrac{2 - 10^{20}}{1 - 10^{-20}} \end{bmatrix}
$$

Feito isso, constata-se que o programa arredonda o termo destacado para 0. Além disso, ele aproxima os dois últimos termos para -10^{20}.

```
--> 1-10^(-20)
 ans  =

   1.

--> 1-10^20
 ans  =

  -1.000D+20

--> (2-10^20)/(1-10^-20)
 ans  =

  -1.000D+20
```

Aproximações da função `Gaussian_Elimination_3`

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

Matriz U da $A3$ obtida com a função `Gaussian_Elimination_3`

Com isso, no processo de calcular o vetor x por substituição, no momento de calcular o primeiro termo do vetor, cuja fórmula seria $\dfrac{1 - (1 - 10^{-20})}{10^{-20}} = 1$, o programa acaba calculando primeiro a subtração dentro dos parênteses e retornando 1 como aproximação, tornando o numerador e, consequentemente, toda a fórmula 0. Esse é o erro cometido por essa função.

```
--> 1-10^(-20)
 ans  =


   1.
```

Aproximação da função 3

Questão 5

Nossa última versão da função de eliminação Gaussiana sempre confere se o pivô atual é o maior valor do vetor correspondente a ele e aos elementos abaixo dele. Caso não seja, ela troca as linhas do pivô atual e desse maior valor para que tenhamos o maior valor possível como pivô. E isso nos ajuda pois é uma melhora do processo de tentar evitar pivôs muito pequenos, que a versão anterior já tinha tentado fazer, mas sem sucesso total.

```
function [x, C, P]=Gaussian_Elimination_4(A, b)
    C=[A,b];
    [n]=size(C,1);
    // Inicializando a matriz P
    P=eye(n,n)
    for j=1:(n-1)
        // Se o pivô não for o maior valor de sua coluna, troca as linhas
        if max(abs(C(j:n,j))) ~= abs(C(j,j)) then
            // Encontrando a linha com o maior pivô em módulo
            moduled_vector = abs(C(j:n,j))
            max_pivot_index = find(moduled_vector == max(moduled_vector))
            // Trocando essas linhas na C e na P
            C([j j+max_pivot_index-1],:) = C([j+max_pivot_index-1 j],:)
            P([j j+max_pivot_index-1],:) = P([j+max_pivot_index-1 j],:)
        end
```

Alterações para a função `Gaussian_Elimination_4`

Assim, vamos ver como essa nova função se aplica à matriz $C3 = [A3 \quad b3]$:

$$
\begin{bmatrix}
10^{-20} & 10^{-20} & 1 & 1 \\
10^{-20} & 1 & 1 & 0 \\
1 & 2 & 1 & 0
\end{bmatrix}
\xrightarrow{L1 \longleftrightarrow L3}
$$

$$
\begin{bmatrix}
1 & 2 & 1 & 0 \\
10^{-20} & 1 & 1 & 0 \\
10^{-20} & 10^{-20} & 1 & 1
\end{bmatrix}
\xrightarrow[]{\substack{L2 - 10^{-20} \cdot L1 \\ L3 - 10^{-20} \cdot L1}}
$$

$$
\begin{bmatrix}
1 & 2 & 1 & 0 \\
0 & 1 - 2 \cdot 10^{-20} & 1 - 10^{-20} & 0 \\
0 & -10^{-20} & 1 - 10^{-20} & 1
\end{bmatrix}
\xrightarrow{L3 - \frac{10^{-20}}{1 - 2 \cdot 10^{-20}} \cdot L2}
$$

$$
\begin{bmatrix}
1 & 2 & 1 & 0 \\
0 & 1 - \mathbf{2 \cdot 10^{-20}} & 1 - \mathbf{10^{-20}} & 0 \\
0 & 0 & \dfrac{1 - \mathbf{4 \cdot 10^{-20}} + \mathbf{3 \cdot 10^{-40}}}{1 - \mathbf{2 \cdot 10^{-20}}} & 1
\end{bmatrix}
$$

O programa considerará todos os valores destacados como 0.

```
--> 1-2*10^(-20)
 ans  =

   1.

--> 1-10^(-20)
 ans  =

   1.

--> (1-4*10^(-20)+3*10^(-40))/(1-2*10^(-20))
 ans  =

   1.
```

Aproximações da função `Gaussian_Elimination_4`

Desse modo, a matriz U final de $A3$ será a seguinte:

```
1.   2.   1.   0.
0.   1.   1.   0.
0.   0.   1.   1.
```

Matriz U da $A3$ obtida com a função `Gaussian_Elimination_4`

Com isso, realizando a substituição, obtém-se o seguinte vetor x:

```
"x3_2:"

 1.
-1.
 1.
```

Resultado para $A3$ com a função `Gaussian_Elimination_4`

Multiplicando-o por $A3$, o resultado é exatamente $b3$, como desejado.

```
"A3*x3_2:"

 1.
 0.
 0.

"b3:"

 1.
 0.
 0.
```

Funcionamento correto da função `Gaussian_Elimination_4`

Portanto, com esse novo mecanismo, essa função consegue evitar melhor problemas de aproximação que levam a resultados equivocados.

Questão 6

Por fim, esta é a função `Resolve_com_LU` exigida na última questão:

```
function [X]=Resolve_com_LU(C, B, P)
    n = size(B,1)
    m = size(C,2)
    // Obtendo a L e a U da C
    L = eye(n,n) + tril(C,-1)
    U = triu(C)
    // Multiplicando dos dois lados por P para poder utilizar a decomposição LU de PA
    D = P*B
    // Resolvendo LY = D, onde Y = UX
    Y = zeros(n,m)
    Y(1,:) = D(1,:)
    for row = 2:n
        sub_factor = L(row,1:(row-1))*Y(1:(row-1),:)
        Y(row,:) = D(row,:) - sub_factor
    end
    // Resolvendo UX = Y
    X = zeros(n,m)
    X(n,:) = Y(n,:)/U(n,n)
    for row = (n-1):-1:1
        sub_factor = U(row,(row+1):m)*X((row+1):n,:)
        X(row,:) = (Y(row,:) - sub_factor)/U(row,row)
    end
endfunction
```

Função `Resolve_com_LU`

Inicialmente, essa função recebe como argumentos a matriz C com a decomposição LU de A, a matriz B de resultados e a matriz P presente na decomposição LU de PA (caso não haja permutações de linhas na decomposição de A, basta atribuir a esse argumento a matriz identidade).

Em seguida, ele salva as dimensões da matriz B nas variáveis n e m e extrai da matriz C as matrizes L e U que compõem a decomposição de A. Após isso, criamos uma nova matriz $D = PB$ para podermos utilizar a decomposição de A apropriadamente. Isso se dá da seguinte forma:

$$AX = B$$
$$PAX = PB$$
$$LUX = D$$

Agora, fazemos $UX = Y$ e resolvemos $LY = D$. Como L é triangular inferior, podemos fazer isso simplesmente por substituição, de cima para baixo. Feito isso, basta resolvermos $UX = Y$ e teremos a solução do problema. Da mesma forma, como U é triangular superior, basta usarmos a substituição, agora de baixo para cima. Com isso, obtemos nosso X desejado.

Finalmente, confira abaixo os resultados dos 3 testes pedidos pelo exercício.

```
"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

"A1*X1"

 2.          4.  -1.          5.
 4.441D-16   1.   5.551D-17   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.
```

Solução de $A1X1 = B1$

```
"X2"

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

"A2*X2"

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

"B2"

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

Solução de $A2X2 = B2$

```
"X3"

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

"A3*X3"

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

"B3"

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

Solução de $A3X3 = B3$