FGV EMAp
ESCOLA DE MATEMÁTICA APLICADA

Aula Prática 1

Álgebra Linear Numérica

Gerardo Mikael Do Carmo Pereira

Professor: Antônio Carlos Saraiva Branco

RIO DE JANEIRO
2024

Conteúdo

1 Introdução

O presente trabalho refere-se à Aula Prática 1, na qual aplicaremos os conceitos aprendidos nas primeiras aulas do curso de Álgebra Linear Numérica. Inicialmente, implementaremos um algoritmo de eliminação Gaussiana simplificado para resolver sistemas de equações lineares quadradas Ax=b, assumindo que A seja invertível.

Ao longo dos exercícios, aprimoraremos progressivamente o código. Nosso objetivo final é desenvolver uma função eficiente que realize a eliminação Gaussiana e retorne a decomposição LU (e posteriormente PLU) da matriz A.

2 Implementação

2.1 <Tarefa 1> Teste da Função GaussianElimination1

Nos é fornecido previamente uma versão simplificada do código, onde precisamos levar algumas considerações para a função funcionar corretamente, vamos fazer alguns testes com essa função e verificar suas capacidades e limitações.

Testes com matrizes com solução conhecida:

Testaremos com a matriz simples:

$$A = \begin{bmatrix} 3 & 4 \\ 6 & 10 \end{bmatrix}, b = \begin{bmatrix} 8 \\ 2 \end{bmatrix}$$

cuja solução de $Ax = b$ é:

$$x = \begin{bmatrix} 12 \\ -7 \end{bmatrix}$$

E tem decomposição LU:

$$LU = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 0 & 2 \end{bmatrix}$$

```
--> A = [3 4; 6 10]
 A  =

   3.   4.
   6.   10.

--> b = [8; 2]
 b  =

   8.
   2.

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

   12.
  -7.
 C  =

   3.   4.
   2.   2.
```

Figura 1: Resultado obtido no console do Scilab

Como previsto obtivemos o vetor x com valores corretos, e também obtivemos a matriz C que contém os valores da diagonal e acima como os valores correspondentes na nossa matriz U, e os valores abaixo da diagonal como os valores correspondentes na matriz L, cuja diagonal é conhecida pois sempre é diagonal de 1's.

Testaremos uma segunda vez com a matriz 3x3:

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$$

cuja solução de $Ax = b$ é:

$$x = \begin{bmatrix} 0 \\ 2 \\ -1 \end{bmatrix}$$

E tem decomposição LU:

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

```
--> A = [2 1 1; 4 3 3; 8 7 9]
 A  =

   2.    1.    1.
   4.    3.    3.
   8.    7.    9.

--> b = [1 ; 3 ; 5]
 b  =

   1.
   3.
   5.

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

   0.
   2.
  -1.
 C  =

   2.    1.    1.
   2.    1.    1.
   4.    3.    2.
```

Figura 2: Resultado obtido no console do Scilab

Como previsto, novamente obtivemos o vetor x com valores corretos, e a matriz C correta.

2.2 <Tarefa 2> Fora do escopo da função GaussianElimination1

A tarefa dois consiste em testar a função dada para a seguinte matriz:

$$A1 = \begin{bmatrix} 1 & -2 & 5 & 0 \\ 2 & -4 & 1 & 3 \\ -1 & 1 & 0 & 2 \\ 0 & 3 & 3 & 1 \end{bmatrix}$$

com o vetor:

$$b1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
--> 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
```

Figura 3: Resultado impreciso obtido no console do Scilab

Ao usar essa função com a matriz A1 e o vetor b1, houve um problema, usamos os valores zero como pivôs para fazer a eliminação o que implicou em divisão por zero. Temos então como saida uma matriz C cheia de valores infinitos e indefinidos.

2.3 <Tarefa 3> Modificação da função para evitar divisão por zero (GaussianElimination2)

O objetivo da terceira tarefa é modificar a GaussianElimination1 de forma a eliminar a limitação de certas situações que implicam em divisão por zero. Faremos então como pedido no enunciado a troca de linhas caso o pivô observado tenha valor zero.

A modificação consiste em usar a função do scilab find(A, 1), que retornará o índice da primeira linha após o pivô que tem valor não zero. Sabendo o índice saberemos onde realizar as trocas. Faremos então o teste novamente na matriz que nos deu problemas (A1) com a antiga versão da função:

```
--> 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_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.
```

Figura 4: Resultado obtido no console com a Matriz A1, o vetor b1 e a função GaussianElimination2

A função teve sucesso em obter a solução x e fazer a Gaussiana. Seguindo o roteiro da atividade testaremos então a função para a seguinte matriz com o seguinte vetor:

$$A2 = \begin{bmatrix} 0 & 10^{-20} & 1 \\ 10^{-20} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}, b2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

```
--> 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
```

Figura 5: Resultado obtido no console com a Matriz A2, o vetor b2 e a função GaussianElimination2

Entendo o propósito do teste e o resultado não foi o esperado. Quando os valores na matriz são muito pequenos, especialmente quando estão próximos de zero, eles podem causar problemas numéricos durante a eliminação gaussiana, levando a resultados imprecisos ou não definidos, como NaN (Not a Number) ou infinito (Inf). Neste caso ainda funcionou normalmente, então farei outro teste mais extremo.

$$A2 = \begin{bmatrix} 0 & 10^{-200} & 1 \\ 10^{-200} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}, b2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

```
--> A2 = [0 10^(-200) 1; 10^(-200) 1 1; 1 2 1]
 A2  =

   0.          1.00D-200   1.
   1.00D-200   1.          1.
   1.          2.          1.

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

   Nan
   Nan
   Nan
 C  =

   1.00D-200   1.          1.
   0.          1.00D-200   1.
   1.00D+200  -Inf         Inf
```

Figura 6: Resultado obtido no console com a Matriz A2 modificada, o vetor b2 e a função GaussianElimination2

Com um valor extremamente pequeno os problemas numéricos se tornam mais aparentes, o scilab não conseguiu trabalhar a gaussiana e não conseguiu encontrar a solução x.

2.4 <Tarefa 4> Evitando trabalhar com valores pequenos (GaussianElimination3)

Para evitar o problema apresentado na tarefa 3 ao invés de fazer a troca de linhas com a primeira linha não nula, em caso de zero no pivô escolheremos a linha que tem o maior valor abaixo do pivô analisado. Usaremos a função max(maxvalue, maxrow) do scilab para selecionar a linha com maior valor dessa forma teremos uma limitação a menos em nossa função.

Refaremos então o teste com:

$$A2 = \begin{bmatrix} 0 & 10^{-200} & 1 \\ 10^{-200} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}, b2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

```
--> A2
 A2  =

   0.          1.00D-200   1.
   1.00D-200   1.          1.
   1.          2.          1.

--> b2
 b2  =

   1.
   0.
   0.

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

   1.
  -1.
   1.
 C  =

   1.          2.          1.
   1.00D-200   1.          1.
   0.          1.00D-200   1.
```

Figura 7: Resultado obtido no console com a Matriz A2 modificada, o vetor b2 e a função GaussianElimination3

A função funcionou como esperado e a solução x foi encontrada sem problemas devido as modificações feitas. Seguindo a tarefa testaremos agora com:

$$A3 = \begin{bmatrix} 10^{-20} & 10^{-20} & 1 \\ 10^{-20} & 1 & 1 \\ 1 & 2 & 1 \end{bmatrix}, b3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

```
--> 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
```

Figura 8: Resultado obtido no console com a Matriz A3 modificada, o vetor b3 e a função GaussianElimination3

Mais uma vez a função retorna a solução e a decomposição LU da matriz A3 sem nenhum problema, a função está cada vez mais eficiente.

2.5 <Tarefa 5> Sempre usando o maior valor em módulo como pivô e retornando a matriz permutação (GaussianElimination4)

Nesta última versão da função faremos a troca de pivôs pelo maior valor em môdulo na coluna do pivô analisado independente de ser zero ou não, afim de evitar trabalhar com números próximos de zero como vimos na tarefa 4. Além disso retornaremos a matriz de permutação P, assim agora decompomos a matriz A como. $A = PLU$. Testaremos com a matriz A3 e b3:

```
--> A3
A3  =

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

--> b3
b3  =

  1.
  0.
  0.

--> [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.
```

Figura 9: Resultado obtido no console com a Matriz A3, o vetor b3 e a função GaussianElimination4

Com esta versão final do algorítmo de Eliminação Gaussiana temos uma função é projetada para lidar com matrizes de tamanho razoável, mas também considera casos em que os elementos da matriz são muito pequenos ou tem valor zero. Isso é crucial para garantir que a eliminação gaussiana seja estável numericamente, mesmo em situações onde os valores dos elementos da matriz estão próximos de zero as trocas de linhas nos garantem no geral uma solução confiável de sistemas lineares.

A função também retorna P que registra as permutações de linhas que foram feitas na matriz de coeficientes A. Isso é útil para acompanhar as mudanças na ordem das equações originais do sistema linear durante o processo de eliminação gaussiana e para restaurar a ordem da matriz original.

2.6 <Tarefa 6> Resolução com PLU

A última parte da tarefa consiste em usar a decomposição PLU para resolver múltiplos sistemas lineares. A decomposição LU é um processo computacionalmente intensivo, mas uma vez que ela é calculada, pode ser reutilizada para resolver múltiplos sistemas lineares. Isso economiza tempo computacional. Dessa forma resolveremos os últimos exemplos da seguinte forma. Faremos a gaussiana uma vez para obter P e C (contendo L e U). Após isso usaremos PLU para resolver múltiplos sistemas.

Exemplo 1 - Matriz A1 e Matriz B1 (contendo varios vetores cada um com uma solução x diferente):

```
--> A1
 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 ]
 B1  =

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

--> [x, C, P] = Gaussian_Elimination_4(A1, b1)
 x  =

  -0.3247863
  -0.1709402
   0.1965812
  -0.0769231
 C  =

   2.   -4.          1.          3.
   0.    3.          3.          1.
   0.5   0.          4.5        -1.5
  -0.5  -0.3333333   0.3333333   4.3333333
 P  =

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

--> X = Resolve_com_PLU(P, C, B1)
 X  =

  -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
```

Figura 10: Resultado obtido no console com a Matriz A1, A Matriz B1 e a função Resolve com PLU

X que é retornado tem cada coluna como uma solução de um sistema linear.

Exemplo 2 - Matriz A2 e Matriz B2:

```
--> A2
 A2  =

   0.          1.00D-200   1.
   1.00D-200   1.          1.
   1.          2.          1.

--> [x, C, P] = Gaussian_Elimination_4(A2, b2)
 x  =

   1.
  -1.
   1.
 C  =

   1.          2.          1.
   1.00D-200   1.          1.
   0.          1.00D-200   1.
 P  =

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

--> X = Resolve_com_PLU(P, C, B2)
 X  =

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

Figura 11: Resultado obtido no console com a Matriz A2, A Matriz B2 e a função Resolve com PLU