

BCC701 – Programação de Computadores I
Universidade Federal de Ouro Preto
Departamento de Ciência da Computação

www.decom.ufop.br/bcc701
2012/01



Matrizes.

Material Didático Unificado.

Agenda

- Introdução;
- Declaração de Matrizes;
- Algumas operações com matrizes;
- Algumas funções aplicadas a matrizes;
- Exercícios.

Introdução;

Declaração de matrizes;

Algumas operações com matrizes;

Algumas funções aplicadas a matrizes;

Exercícios.

INTRODUÇÃO

Conjunto de variáveis

- Ao estudar **vetores** observamos que, em determinadas situações, é necessário utilizar muitas variáveis com um propósito comum. Relembrando exemplos:
 - Para armazenar três notas de um aluno:
 - `Nota1 = input('Digite a nota 1: ');`
 - `Nota2 = input('Digite a nota 2: ');`
 - `Nota3 = input('Digite a nota 3: ');`
 - Ler e imprimir cinco números:
 - `for i = 1 : 5`
 - `Num = input('Digite um numero: ');`
 - `printf('Numero digitado: %g', num);`
 - `end`

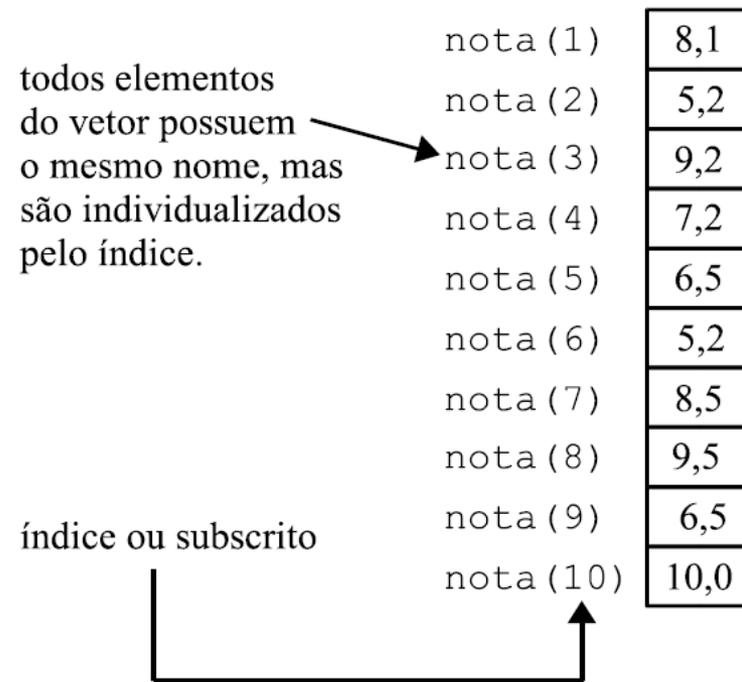
Relembrando Vetor

- Nestes casos, todas as variáveis representam um conjunto de valores, possuem um objetivo em comum e são do mesmo tipo de dados;
- Um vetor representa conjuntos ordenados de valores homogêneos (do mesmo tipo), que podem ser números, *strings* e booleanos;
 - A palavra ordenado é empregada no sentido dos valores estarem localizados em posições ordenadas de memória, e não no sentido de estarem respeitando uma relação (<, <=, >, ou >=).

Relembrando Vetor

- Os itens contidos em um vetor são chamados de **elementos**;
- A posição do elemento no vetor é chamado de **índice** ou subscrito, e é usado para individualizar um elemento do vetor;
- O vetor `nota = [8.1 5.2 9.2 7.2 6.5 5.2 8.5 9.5 6.5 10.0]`,

pode ser representado na memória como uma sequência de variáveis distintas, com o mesmo nome, mas diferenciadas pelo índice:



O tipo de dados **Matriz**

- Agora imagine a seguinte situação:
 - Desejo armazenar **3 notas** para **5 alunos**;
 - Para isto eu preciso de **3 vetores** ou de **5 vetores**?

O tipo de dados Matriz

- Agora imagine a seguinte situação:
 - Desejo armazenar **3 notas** para **5 alunos**;
 - Para isto eu preciso de **3 vetores** ou de **5 vetores**?
 - **Nenhum dos dois:** posso utilizar uma matriz em que **cada linha representa um aluno** e **cada coluna representa uma nota**:

	Nota 1	Nota 2	Nota 3
Aluno 1	8.1	9.2	6.0
⋮	5.2	6.8	9.5
⋮	6.0	6.1	6.2
⋮	3.5	5.2	8.3
Aluno 5	2.4	1.5	5.3

O tipo de dados Matriz

- Matrizes são variáveis que contêm uma quantidade potencialmente grande de valores;
- Assim como nos vetores, elementos da matriz são acessados através de índices;
- Uma matriz bidimensional **A**, com dimensão **m x n** (ou seja, de **m** linhas e **n** colunas:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- **OBS:** Um vetor corresponde a uma matriz **m x 1** (no caso de um **vetor coluna**), ou uma matriz **1 x n** (no caso de um **vetor linha**).

O tipo de dados **Matriz**

- Além das matrizes serem muito úteis para o armazenamento e manipulação de um grande volume de dados, elas também são muito utilizadas em diversas áreas:
 - Para se resolver sistemas de equações lineares;
 - Translação, rotação, escala de objetos em computação gráfica;
 - Para resolver problemas de circuitos elétricos e linhas de transmissão de energia elétrica;
 - Algoritmos para determinar rotas entre dois pontos;
 - E muito mais;
- É no tratamento de matrizes que o Scilab mostra grande superioridade sobre linguagens como C, Fortran ou Java;

Exemplos de uso de Matriz

- Para se ter uma pequena ideia do poder das matrizes, vejamos alguns exemplos simples do nosso cotidiano que envolvem a multiplicação de matrizes:

1. Uma lanchonete prepara três tipos de salgados utilizando diferentes tipos de ingredientes, conforme as tabelas abaixo. Qual o preço de custo de cada salgado?

	Ovos	Farinha	Açúcar	Carne
Pastéis	3	6	1	3
Empadas	4	4	2	2
Quibes	1	1	1	6

Ingredientes	Preço (R\$)
Ovos	0,20
Farinha	0,30
Açúcar	0,50
Carne	0,80

Exemplos de uso de Matriz

- **Solução:**

3	6	1	3
4	4	2	2
1	1	1	6

 X

0,20
0,30
0,50
0,80

 =

5,30
4,60
5,80

- **Custos:**

- Pastéis: R\$ 5,30;
- Empadas: R\$ 4,60;
- Quibes: R\$ 5,80.

Exemplos de uso de Matriz

2. Uma fábrica de automóveis deseja produzir uma certa quantidade de carros de dois modelos (X e Y) em três diferentes versões, utilizando três tipos de peças. Quantas peças serão necessárias para executar o plano de produção representado nas tabelas abaixo?

	Carro X	Carro Y
Peça A	4	3
Peça B	3	5
Peça C	6	2

	Standard	Luxo	Super Luxo
Carro X	2	4	3
Carro Y	3	2	5

Exemplos de uso de Matriz

- Solução:

$$\begin{array}{|c|c|} \hline 4 & 3 \\ \hline 3 & 5 \\ \hline 6 & 2 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline 2 & 4 & 3 \\ \hline 3 & 2 & 5 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 17 & 22 & 27 \\ \hline 21 & 22 & 34 \\ \hline 18 & 28 & 28 \\ \hline \end{array}$$

- Assim, a quantidades de peças será:
 - Peça A: $17 + 22 + 27 = \mathbf{66}$;
 - Peça B: $21 + 22 + 34 = \mathbf{77}$;
 - Peça C: $18 + 28 + 28 = \mathbf{74}$;
- Calcule quantas peças cada versão demandará no total.

Exemplos de uso de Matriz

- Na resolução de sistemas de equações lineares:
 - Dado um sistema linear do tipo: $\mathbf{A} * \mathbf{X} = \mathbf{B}$;
 - A solução é obtida resolvendo: $\mathbf{X} = \mathbf{A}^{-1} * \mathbf{B}$;
 - Exemplo:

$$\begin{array}{l}
 3x + y + 2z = 13 \\
 x + y - 8z = -1 \\
 -x + 2y + 5z = 13
 \end{array}
 \longrightarrow
 \begin{array}{c}
 \begin{bmatrix} 3 & 1 & 2 \\ 1 & 1 & -8 \\ -1 & 2 & 5 \end{bmatrix}
 \begin{bmatrix} x \\ y \\ z \end{bmatrix}
 =
 \begin{bmatrix} 13 \\ -1 \\ 13 \end{bmatrix} \\
 \mathbf{A}_{33} \qquad \qquad \mathbf{X}_{31} \qquad \mathbf{B}_{31}
 \end{array}$$

Exemplos de uso de Matriz

- Na resolução de sistemas de equações lineares:

- Exemplo:

$$\begin{array}{l}
 3x + y + 2z = 13 \\
 x + y - 8z = -1 \\
 -x + 2y + 5z = 13
 \end{array}
 \longrightarrow
 \begin{array}{c}
 \begin{bmatrix} 3 & 1 & 2 \\ 1 & 1 & -8 \\ -1 & 2 & 5 \end{bmatrix}
 \begin{bmatrix} x \\ y \\ z \end{bmatrix}
 =
 \begin{bmatrix} 13 \\ -1 \\ 13 \end{bmatrix} \\
 A_{33} \qquad X_{31} \qquad B_{31}
 \end{array}$$

--> $A = [3, 1, 2; 1, 1, -8; -1, 2, 5];$

--> $B = [13; -1; 13];$

--> $X = \text{inv}(A) * B$

$X =$ 2.
5.
1.

Assim, chega-se à solução:

$x = 2, y = 5, z = 1.$

Introdução;

Declaração de matrizes;

Algumas operações com matrizes;

Algumas funções aplicadas a matrizes;

Exercícios.

DECLARAÇÃO DE MATRIZES

Tópicos

- Definindo todos os elementos;
- Definindo a partir de outras matrizes;
- Matriz de 1's;
- Matriz de 0's;
- Matriz identidade;
- Modificando o formato de uma matriz conhecida;
- Preenchendo com valores randômicos.

Definindo todos os elementos

- Utiliza-se colchetes para delimitar todos os elementos;
- Cada elemento de uma linha é separado por espaço ou vírgula;
- Cada linha é separada por um ponto-e-vírgula;
- Exemplo:

--> $M = [1, 2, 3; 4, 5, 6; 7, 8, 9]$

M =

1. 2. 3.

4. 5. 6.

7. 8. 9.

-->

A partir de matrizes

- A definição pode ser feita a partir de matrizes já existentes;
- Exemplos:

--> $A = [1 \ 2; 3 \ 4]$

$A =$
 1. 2.
 3. 4.

--> $B = [5 \ 6; 7 \ 8]$

$B =$
 5. 6.
 7. 8.

--> $C = [A \ B]$

$C =$
 1. 2. 5. 6.
 3. 4. 7. 8.

--> $D = [A; B]$

$D =$
 1. 2.
 3. 4.
 5. 6.
 7. 8.

-->

Matriz de 1's

- Todos os elementos assumirão valor inicial **1**:

Matriz = ones(<linhas>, <colunas>)

- **Matriz**: nome da variável do tipo matriz;
- **ones**: função que retorna uma **matriz** com valores 1;
- **<linhas>**: número de linhas;
- **<colunas>**: número de colunas.

Matriz de 1's

- Exemplos:

- > M1 = ones(2, 5)

M1 = 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1.

-->

- > M2 = ones(5, 2)

M1 = 1. 1.
 1. 1.
 1. 1.
 1. 1.
 1. 1.

-->

Matriz de 0's

- Todos os elementos assumirão valor inicial **0**:

Matriz = zeros(<linhas>, <colunas>)

- **Matriz**: nome da variável do tipo matriz;
- **zeros**: função que retorna uma **matriz** com valores 0;
- **<linhas>**: número de linhas;
- **<colunas>**: número de colunas.

Matriz de 0's

- Exemplos:

- > M1 = zeros(2, 5)

$$M1 = \begin{bmatrix} 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. \end{bmatrix}$$

-->

- > M2 = zeros (5, 2)

$$M1 = \begin{bmatrix} 0. & 0. \\ 0. & 0. \\ 0. & 0. \\ 0. & 0. \\ 0. & 0. \end{bmatrix}$$

-->

Matriz identidade

- Todos os elementos da diagonal principal assumirão valor inicial **1**, e os demais elementos assumirão **0**:

Matriz = eye(<linhas>, <colunas>)

Matriz = eye(<matriz parâmetro>)

- **Matriz**: nome da variável do tipo matriz;
- **<linhas>**: número de linhas;
- **<colunas>**: número de colunas;
- **<matriz parâmetro>**: matriz que definirá as dimensões da matriz resultante.

Matriz identidade

- Exemplos:

--> Id1 = eye(4,3)

$$\text{Id1} = \begin{bmatrix} 1. & 0. & 0. \\ 0. & 1. & 0. \\ 0. & 0. & 1. \\ 0. & 0. & 0. \end{bmatrix}$$

--> Id2 = eye(A)

$$\text{Id2} = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

--> Id3 = eye(10) // 10 é uma matriz com um elemento ($a_{11} = 10$)

$$\text{Id3} = \begin{bmatrix} 1. \end{bmatrix}$$

-->

Modificando o formato

- Pode-se declarar uma matriz modificando o formato de uma matriz conhecida:

Matriz = matrix(<matriz parâmetro>, <linhas>, <colunas>)

- **Matriz**: nome da variável do tipo matriz;
- **<matriz parâmetro>**: matriz que definirá os elementos da matriz resultante;
- **<linhas>**: número de linhas da matriz resultante;
- **<colunas>**: número de colunas da matriz resultante.

Modificando o formato

- Exemplos:

```
--> Mpar=[1 2 3;4 5 6]
```

```
Mpar =    1.    2.    3.  
         4.    5.    6.
```

```
--> Mres1 = matrix(Mpar, 1, 6)
```

```
Mres1 =    1.    4.    2.    5.    3.    6.
```

```
--> Mres2 = matrix(Mpar, 3, 2)
```

```
Mres2 =    1.    5.  
         4.    3.  
         2.    6.
```

```
-->
```

Valores randômicos

- Pode-se declarar uma matriz com valores randômicos (gerados aleatoriamente):

Matriz = rand(<linhas>, <colunas>)

- **Matriz**: nome da variável do tipo matriz;
- **<linhas>**: número de linhas da matriz resultante;
- **<colunas>**: número de colunas da matriz resultante;

- Gera valores entre 0 e 1;
- A cada chamada são gerados valores diferentes.

Valores randômicos

- Exemplos:

--> Mr1 = rand(2,3)

```
Mr1 =      0.2113249   0.0002211   0.6653811
          0.7560439   0.3303271   0.6283918
```

-->Mr2 = rand(2,3)

```
Mr2 =      0.8497452   0.8782165   0.5608486
          0.6857310   0.0683740   0.6623569
```

--> Mr3 = `int(rand(2,3) * 10)` // Matriz com valores inteiros entre 0 e 10

```
Mr3 =      7.   5.   2.
          1.   2.   2.
```

-->

Introdução;

Declaração de matrizes;

Algumas operações com matrizes;

Algumas funções aplicadas a matrizes;

Exercícios.

ALGUMAS OPERAÇÕES COM MATRIZES

Tópicos

- Acesso aos elementos;
- Transposição de matrizes;
- Aritmética matricial:
 - Adição e subtração de matrizes;
 - Multiplicação por um escalar;
 - Multiplicação entre matrizes;
 - Divisão por um escalar;
 - Divisão entre matrizes;
 - Exponenciação;
 - Expressões relacionais;
 - Mais sobre operações binárias.

Acesso aos elementos

- Para acessar um elemento específico:

Matriz(<índice de linha>, <índice de coluna>)

- Exemplo:

--> $M = [1, 2, 3; 4, 5, 6];$

--> $E1 = M(2, 3)$

$E1 = 6.$

--> $E2 = M(1, 2)$

$E2 = 2.$

-->

- Pode ser usado para modificar o valor: $M(1, 3) = 300$, modifica o valor da **linha 1 e coluna 3** de **3** para **300**.
 - OBS.: Utilizando este recurso é possível definir uma matriz definindo o valor de cada um dos seus elementos individualmente.

Acesso aos elementos

- Para acessar múltiplos elementos:

Matriz(<faixa para linhas>, <faixa para colunas>)

- Permite manipular vetores e matrizes;
- Exemplo 1:

x =	23.	30.	29.	50.	91.	28.	68.
	23.	93.	56.	43.	4.	12.	15.
	21.	21.	48.	26.	48.	77.	69.
	88.	31.	33.	63.	26.	21.	84.
	65.	36.	59.	40.	41.	11.	40.

--> $y = x(2:4, 3:5)$

y =	56.	43.	4.
	48.	26.	48.
	33.	63.	26.

Acesso aos elementos

- Para acessar múltiplos elementos:

Matriz(<faixa para linhas>, <faixa para colunas>)

- Permite manipular vetores e matrizes;
- Exemplo 2:

x = 23. 30. 29. 50. 91. 28. 68.

23. 93. 56. 43. 4. 12. 15.

21. 21. 48. 26. 48. 77. 69.

88. 31. 33. 63. 26. 21. 84.

65. 36. 59. 40. 41. 11. 40.

--> y = x(2:2, :)

y = 23. 93. 56. 43. 4. 12. 15.

21. 21. 48. 26. 48. 77. 69.

Acesso aos elementos

- Para acessar múltiplos elementos:

Matriz(<faixa para linhas>, <faixa para colunas>)

- Permite manipular vetores e matrizes;
- Exemplo 3:

x =	23.	30.	29.	50.	91.	28.	68.
	23.	93.	56.	43.	4.	12.	15.
	21.	21.	48.	26.	48.	77.	69.
	88.	31.	33.	63.	26.	21.	84.
	65.	36.	59.	40.	41.	11.	40.

--> $y = x(:, 3)$

y =	29.
	56.
	:
	:

Transposição de matrizes

- **Operador apóstrofo ('):** Matriz'

- Transforma linhas em colunas e colunas em linhas;
- Exemplo:

$x =$

23.	30.	29.	50.	91.	28.	68.
23.	93.	56.	43.	4.	12.	15.
21.	21.	48.	26.	48.	77.	69.
88.	31.	33.	63.	26.	21.	84.
65.	36.	59.	40.	41.	11.	40.

--> $y = x'$

$y =$

23.	23.	21.	88.	65.
30.	93.	21.	31.	36.
29.	56.	48.	33.	59.
50.	43.	26.	63.	40.
91.	4.	48.	26.	41.
28.	12.	77.	21.	11.
68.	15.	69.	84.	40.

Aritmética matricial

- Como todas as variáveis Scilab são matrizes, as operações aritméticas usuais (+, -, *, /, ^) são entendidas pelo Scilab como operações matriciais;
 - Assim, $a*b$ designa o produto matricial da matriz a pela matriz b ;
- As operações escalares usam os mesmos símbolos aritméticos, porém precedidos por um "." (ponto) como, por exemplo, $.*$ e $.^$;
- Exemplos a seguir.

Adição e subtração de matrizes

- Operadores **+** e **-** aplicados a duas matrizes de mesmas dimensões ou a uma matriz e um valor escalar;
- Exemplos com duas matrizes:

```
x = 1.  2.  3.
      4.  5.  6.
y = 10. 20. 30.
      40. 50. 60.
```

```
--> x + y
```

```
ans = 11.  22.  33.
       44.  55.  66.
```

```
--> y - x
```

```
ans = 9.  18.  27.
       36.  45.  54.
```

Como estas operações são sempre realizadas elemento a elemento, não são necessários os operadores **.+** e **.-**. Sendo assim, eles não existem no Scilab.

Adição e subtração de matrizes

- Exemplos de matrizes e valores escalares:

$$x = \begin{bmatrix} 1. & 2. & 3. \\ 4. & 5. & 6. \end{bmatrix}$$

$$\text{--> } x + 2$$

$$\text{ans} = \begin{bmatrix} 3. & 4. & 5. \\ 6. & 7. & 8. \end{bmatrix}$$

$$\text{--> } 2 - x$$

$$\text{ans} = \begin{bmatrix} 3. & 4. & 5. \\ 6. & 7. & 8. \end{bmatrix}$$

Multiplicação por um escalar

- Uma matriz pode ser multiplicada por um valor escalar;
- Neste caso, os operadores `*` e `.*` obterão o mesmo resultado;
- Exemplos:

```
x = 1. 2. 3.
     4. 5. 6.
```

```
--> x * 2
ans = 2. 4. 6.
      8. 10. 12.
```

```
--> x .* 2
ans = 2. 4. 6.
      8. 10. 12.
```

A inversão dos termos não alteram o produto. Assim, $2 * x$ e $2 .* x$, também obterão o mesmo resultado.

Multiplicação entre matrizes

- A “**multiplicação pontuada**”, operador **.***, realiza a multiplicação elemento por elemento entre duas matrizes;
- Esta operação exige que as duas matrizes tenham as mesmas dimensões;
- O Scilab emite uma mensagem de erro na tentativa de multiplicar duas matrizes de dimensões incompatíveis;

- Exemplos:

```

X =  1.    2.
     3.    4.
Y = 10.   20.
     30.   40.
--> X .* Y
ans =   10.   40.
       90.  160.
  
```

$$R_{11} = 1 * 10 = 10$$

$$R_{12} = 2 * 20 = 40$$

$$R_{21} = 3 * 30 = 90$$

$$R_{22} = 4 * 40 = 160$$

Multiplicação entre matrizes

- Pela álgebra linear, a multiplicação da matriz $X_{m \times n}$ pela matriz $Y_{n \times p}$ resultará em uma matriz $R_{m \times p}$, onde $R_{ij} = \sum_{k=1}^n X_{ik} * Y_{kj}$;
- Esta operação é conhecida por **produto matricial**;
- O Scilab emite uma mensagem de erro na tentativa de multiplicar duas matrizes de dimensões incompatíveis;
- Exemplos:

```

X =  1.    2.
     3.    4.
Y = 10.   20.
     30.   40.
--> X * Y
ans =    70.   100.
        150.   220.
  
```

$$R_{11} = 1 * 10 + 2 * 30 = 70$$

$$R_{12} = 1 * 20 + 2 * 40 = 100$$

$$R_{21} = 3 * 10 + 4 * 30 = 150$$

$$R_{22} = 3 * 20 + 4 * 40 = 220$$

Divisão por um escalar

- Uma matriz pode ser dividida por um valor escalar;
- Neste caso, os operadores `/` e `./` obterão o mesmo resultado;
- Exemplos:

```
x = 10. 20. 30.
     40. 50. 60.
```

```
--> x / 2
```

```
ans = 5.    10.    15.
       20.    25.    30.
```

```
--> x ./ 2
```

```
ans = 5.    10.    15.
       20.    25.    30.
```

Divisão entre matrizes

- A “**divisão pontuada**”, operadores **`./`** e **`.\`**, realiza a divisão elemento por elemento entre duas matrizes;
- Esta operação exige que as duas matrizes tenham as mesmas dimensões;
- O Scilab emite uma mensagem de erro na tentativa de dividir duas matrizes de dimensões incompatíveis;

- Exemplos:

X = 1. 2.

 3. 4.

Y = 10. 20.

 30. 40.

--> X ./ Y

ans = 0.1 0.1

 0.1 0.1

Cada elemento de **X** é dividido pelo elemento de **Y**.

--> X .\ Y

ans = 10. 10.

 10. 10.

Cada elemento de **Y** é dividido pelo elemento de **X**.

Divisão entre matrizes

- A utilização dos operadores $/$ e \backslash , por sua vez, não correspondem propriamente à operações de divisão;
- Seja \mathbf{A} matriz quadrada e *não singular*¹ e \mathbf{B} de dimensões compatíveis em cada caso. Então:
 - $\mathbf{X} = \mathbf{A} \backslash \mathbf{B} = \mathbf{A}^{-1} \mathbf{B} = \text{inv}(\mathbf{A}) * \mathbf{B}$ (solução de $\mathbf{A} * \mathbf{X} = \mathbf{B}$)²
 - $\mathbf{X} = \mathbf{B} / \mathbf{A} = \mathbf{B} \mathbf{A}^{-1} = \mathbf{B} * \text{inv}(\mathbf{A})$ (solução de $\mathbf{X} * \mathbf{A} = \mathbf{B}$)
- Se \mathbf{A} não for quadrada, \mathbf{X} é obtido como solução de:
 - $\mathbf{A} * \mathbf{X} = \mathbf{B}$ ou $\mathbf{X} * \mathbf{A} = \mathbf{B}$

¹ Uma matriz quadrada é dita não singular quando não admite uma inversa. Propriedades:

- Uma matriz é singular se e somente se seu determinante é nulo.
- Uma matriz é singular se e somente se existir um vetor \mathbf{x} não nulo tal que $\mathbf{Ax} = \mathbf{0}$;
- Se uma matriz \mathbf{A} é singular, então $\mathbf{Ax} = \mathbf{b}$ não possui solução, ou possui infinitas soluções;
- Uma matriz é singular se, e somente se, ela é um *divisor de zero*.

² Importante para a solução de sistemas de equações lineares.

Divisão entre matrizes

- Solução de sistemas de equações lineares:
 - Seja o sistema:

$$\begin{cases} x_1 - x_2 + 2x_3 = 5 \\ x_1 - x_2 - 6x_3 = 0 \\ 4x_1 \quad \quad + x_3 = 5 \end{cases}$$

- Escrito na forma matricial:

$$\begin{bmatrix} 1 & -1 & 2 \\ 1 & -1 & -6 \\ 4 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 5 \end{bmatrix}$$

- Sua solução em Scilab é:

--> A = [1 -1 2; 1 -1 -6; 4 0 1];

--> b = [5;0;5];

--> A\b

```
ans =      1.09375  ← valor de x1
      -2.65625  ← valor de x2
      0.625     ← valor de x3
```

Exponenciação

- A exponenciação é encarada como a multiplicação sucessiva de uma matriz por ela mesma;
- O produto escalar (ex.: $x^3 = x * x * x$) só faz sentido quando x é uma matriz quadrada;

- Exemplo:

```
X =  1.    2.  
     3.    4.
```

```
--> X ^ 2
```

```
ans =      7.    10.  
          15.    22.
```

Exponenciação

- Já a “**exponenciação pontuada**” (ex.: $x.^3 = x.^x.^x$) realiza a multiplicação elemento a elemento de matrizes de dimensões arbitrárias;
- Exemplo:

$$X = \begin{bmatrix} 1. & 2. \\ 3. & 4. \end{bmatrix}$$

$$\rightarrow X.^2$$

$$\text{ans} = \begin{bmatrix} 1. & 4. \\ 9. & 16. \end{bmatrix}$$

Expressões relacionais

- O resultado de uma expressão relacional envolvendo matrizes resulta em uma matriz de valores booleanos resultantes da aplicação da expressão elemento a elemento;

- Exemplos:

--> $a = \begin{bmatrix} 3 & 7 \\ 8 & 2 \end{bmatrix}$;

--> $b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$;

--> $a > 5$

ans =	F	T
	T	F

--> $a > b$

ans =	F	T
	T	F

Expressões relacionais

- Uma expressão relacional envolvendo matrizes pode ser empregada em um comando condicional **if**;
- Neste caso, a cláusula **then** será executada apenas quando **todos** os elementos da matriz booleana resultante forem verdadeiros (%t);

- Exemplo:

```
--> a = [3 9; 12 1];
```

```
--> x = 0; y = 0;
```

```
--> if a > 5 then x = 10000; end;
```

```
--> if a > 0 then y = 10000; end;
```

```
--> [x y]
```

```
ans =      0.      10000.
```

Expressões relacionais

- Outras operações também podem ser realizadas, como a atribuição, em que apenas os elementos que satisfazem à condição serão afetados;

- Exemplo:

--> $a = [3 \ 9; 12 \ 1];$

--> $a(a > 5) = -1;$

ans =	3.	-1.
	-1.	1.

Mais sobre operações binárias

- Para mais informações, procure pelos operadores do scilab:
 - Soma (*plus*: +):
 - http://help.scilab.org/docs/5.3.3/pt_BR/plus.html
 - Subtração (*minus*: -):
 - http://help.scilab.org/docs/5.3.3/pt_BR/minus.html
 - Multiplicação (*star*: *):
 - http://help.scilab.org/docs/5.3.3/pt_BR/star.html
 - Divisão (*slash*: \ e *backslash*: /):
 - http://help.scilab.org/docs/5.3.3/pt_BR/slash.html
 - http://help.scilab.org/docs/5.3.3/pt_BR/backslash.html

Introdução;
Declaração de matrizes;
Algumas operações com matrizes;
Algumas funções aplicadas a matrizes;
Exercícios.

ALGUMAS FUNÇÕES APLICADAS A MATRIZES

Tópicos

- Dimensões de uma matriz;
- Matriz inversa;
- Determinante;
- Somatório;
- Somatório cumulativo;
- Produtório;
- Produtório cumulativo;
- Elementos únicos;
- União;
- Interseção;
- Diferença;
- Busca (pesquisa);
- Ordenação;
- Plotando gráficos.

Dimensões de uma matriz

numElementos = length(<Matriz>)

- Retorna o número elementos da matriz (ou seja, número de linhas vezes o número de colunas);
- Exemplo:
 - > A = [1 2 3; 4 5 6];
 - > ne = length(A)
 - ans = 6.

Dimensões de uma matriz

[numLinhas, numColunas] = size(<Matriz>)

- Retorna o número de linhas e o número de colunas da matriz;
- Exemplos:
 - > A = [1 2 3; 4 5 6];
 - > [nl,nc] = size(A)
 - nc = 3.
 - nl = 2.
 - > k = 0;
 - > [L,C] = size(k)
 - C = 1.
 - L = 1.

Dimensões de uma matriz

[numLinhas, numColunas] = size(<Matriz>)

- O Scilab é tão orientado para matrizes que todas as variáveis Scilab são matrizes;
 - As variáveis simples com que temos trabalhado são, na verdade, matrizes com uma única linha e uma única coluna;
- Uma matriz “cresce” quando atribuímos valores a elementos com índices superiores aos índices já referenciado:

```
--> x = 7;           // matriz 1 x 1
--> x(2 , 3) = 13    // x se transforma em uma matriz 2 x 3
x =  7.      0.      0.
     0.      0.      13.
```

Matriz inversa

[resultado] = inv(<Matriz>)

- Retorna a inversa da matriz;
- Exemplo:

A = 4. 7. 6.
2. 2. 1.
1. 1. 6.

--> IA = inv(A)

IA = - 0.3333333	1.0909091	0.1515152
0.3333333	- 0.5454545	- 0.2424242
0.	- 0.0909091	0.1818182

Matriz inversa

$$[\text{resultado}] = \text{inv}(\langle \text{Matriz} \rangle)$$

- Espera-se que $A * IA$ e $IA * A$ resultem na matriz unidade;
- Exemplo:

--> A * IA

ans =	1.	0.	- 4.441D-16 ¹
	1.110D-16 ¹	1.	- 1.110D-16 ¹
	5.551D-17 ¹	0.	1.

--> IA * A

ans =	1.	8.327D-17 ¹	0.
	0.	1.	0.
	0.	0.	1.

¹ Erros de aproximação

Determinante

$$[e, m] = \det(\langle \text{Matriz} \rangle)$$

- Retorna o determinante de uma matriz quadrada em dois componentes:
 - **m**: um número real ou complexo que representa a mantissa de base 10 do determinante;
 - **e**: um número inteiro que representa o expoente de base 10 do determinante;
- Ou seja, o determinante da matriz quadrada é dado por:

$$\text{Determinante} = m * 10^e$$

Somatório

$$[\text{resultado}] = \text{sum}(\langle \text{Matriz} \rangle, \langle \text{orientação} \rangle)$$

- Retorna o somatório dos elementos da matriz;
- **<orientação>** define como será realizado o somatório:
 - “*”: o resultado será um valor escalar representando o somatório de todos os elementos da matriz;
 - “r”: o resultado será um **vetor linha** de valores escalares que representam os somatórios das colunas da matriz;
 - “c”: o resultado será um **vetor coluna** de valores escalares que representam os somatórios das linhas matriz;

Somatório

[resultado] = sum(<Matriz>, <orientação>)

- Exemplos:

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

A = 1. 2.
 3. 4.

--> **sum(A, '*')**

ans = 10.

--> **sum(A, 'r')**

ans = 4. 6.

--> **sum(A, 'c')**

ans = 3.
 7.

Somatório cumulativo

[resultado] = cumsum(<Matriz>, <orientação>)

- Retorna o somatório dos elementos da matriz, de forma acumulativa a cada linha/coluna;
- **<orientação>** define como será realizado o somatório:
 - **“*”**: o resultado será uma matriz com valores escalares representando o somatório de todos os elementos da matriz anteriores à posição da matriz resultante;
 - **“r”**: o resultado será uma matriz de valores escalares que representam os somatórios cumulativos das colunas da matriz;
 - **“c”**: o resultado será uma matriz de valores escalares que representam os somatórios cumulativos das linhas matriz;

Somatório cumulativo

[resultado] = cumsum(<Matriz>, <orientação>)

- Exemplo 1:

--> A = [1 2 3; 4 5 6; 7 8 9]

A = 1. 2. 3.

4. 5. 6.

7. 8. 9.

--> **cumsum(A, '*')**

ans = 1. 14. 30.

5. 19. 36.

12. 27. 45.

Somatório cumulativo

[resultado] = cumsum(<Matriz>, <orientação>)

- Exemplo 2:

--> A = [1 2 3; 4 5 6; 7 8 9]

A = 1. 2. 3.

4. 5. 6.

7. 8. 9.

--> cumsum(A, 'r')

ans = 1. 2. 3.

5. 7. 9.

12. 15. 18.

Somatório cumulativo

[resultado] = cumsum(<Matriz>, <orientação>)

- Exemplo 3:

--> A = [1 2 3; 4 5 6; 7 8 9]

A = 1. 2. 3.

4. 5. 6.

7. 8. 9.

--> cumsum(A, 'c')

ans = 1. 3. 6.

4. 9. 15.

7. 15. 24.

Produtório

$$[\text{resultado}] = \text{prod}(\langle \text{Matrix} \rangle, \langle \text{orientação} \rangle)$$

- Retorna o produtório dos elementos da matriz;
- Tem funcionamento similar ao somatório, mas realiza a operação de multiplicação em lugar da soma;
- **<orientação>** define como será realizado o produtório:
 - “*”: o resultado será um valor escalar representando o produtório de todos os elementos da matriz;
 - “r”: o resultado será um **vetor linha** de valores escalares que representam os produtórios das colunas da matriz;
 - “c”: o resultado será um **vetor coluna** de valores escalares que representam os produtórios das linhas matriz.

Produtório

[resultado] = prod(<Matriz>, <orientação>)

- Exemplos:

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

A = 1. 2.
 3. 4.

--> prod(A, '*')

ans = 24.

--> prod(A, 'r')

ans = 3. 8.

--> prod(A, 'c')

ans = 2.
 12.

Produtório cumulativo

[resultado] = cumprod(<Matriz>, <orientação>)

- Retorna o produtório dos elementos da matriz, de forma acumulativa a cada linha/coluna;
- **<orientação>** define como será realizado o produtório:
 - **“*”**: o resultado será uma matriz com valores escalares representando o produtório de todos os elementos da matriz anteriores à posição da matriz resultante;
 - **“r”**: o resultado será uma matriz de valores escalares que representam os produtórios cumulativos das colunas da matriz;
 - **“c”**: o resultado será uma matriz de valores escalares que representam os produtórios cumulativos das linhas matriz;

Produtório cumulativo

[resultado] = cumprod(<Matriz>, <orientação>)

- Exemplo 1:

--> A = [1 2 3; 4 5 6; 7 8 9]

A = 1. 2. 3.

4. 5. 6.

7. 8. 9.

--> cumprod(A, '*')

ans = 1. 56. 6720.

4. 280. 40320.

28. 2240. 362880.

Produtório cumulativo

[resultado] = cumprod(<Matriz>, <orientação>)

- Exemplo 2:

--> A = [1 2 3; 4 5 6; 7 8 9]

A = 1. 2. 3.

4. 5. 6.

7. 8. 9.

--> cumprod(A, 'r')

ans = 1. 2. 3.

4. 10. 18.

28. 80. 162.

Produtório cumulativo

[resultado] = cumprod(<Matriz>, <orientação>)

- Exemplo 3:

--> A = [1 2 3; 4 5 6; 7 8 9]

A = 1. 2. 3.

4. 5. 6.

7. 8. 9.

--> cumprod(A, 'c')

ans = 1. 2. 6.

4. 20. 120.

7. 56. 504.

Elementos únicos

[resultado[, k]] = unique(<Matriz>, <orientação>)

- Retorna uma matriz contendo as linhas/colunas únicas da matriz em ordenação crescente, adicionalmente retorna um vetor com os índices das linhas/colunas remanescentes (k);
- **<orientação>** define como será realizado o produtório:
 - “r”: o resultado será uma matriz contendo apenas as linhas únicas da <Matriz>;
 - “c”: o resultado será uma matriz contendo apenas as colunas únicas da <Matriz>.

Elementos únicos

[resultado[, k]] = unique(<Matriz>, <orientação>)

- Exemplo 1:

```
--> A = [1  2  3  10  10; ...
         4  5  6  10  10; ...
         1  2  3  10  10; ...
         4  5  6  10  10; ...
         7  8  9  10  10]
```

```
--> [r, k] = unique(A, 'r');
```

```
--> r
```

```
r =  1.  2.  3.  10.  10.
     4.  5.  6.  10.  10.
     7.  8.  9.  10.  10.
```

```
--> k
```

```
k =  1.
     2.
     5.
```

Elementos únicos

[resultado[, k]] = unique(<Matriz>, <orientação>)

- Exemplo 2:

```
--> A = [1  2  3  10  10; ...  
         4  5  6  10  10; ...  
         1  2  3  10  10; ...  
         4  5  6  10  10; ...  
         7  8  9  10  10]
```

```
--> [r, k] = unique(A, 'c');
```

```
--> r
```

```
r =  1.  2.  3.  10.  
     4.  5.  6.  10.  
     1.  2.  3.  10.  
     4.  5.  6.  10.  
     7.  8.  9.  10.
```

```
--> k
```

```
k =  1.  2.  3.  4.
```

União

[resultado[, k]] = unique(<Matriz>, <orientação>)

- Retorna uma matriz contendo as linhas/colunas únicas da matriz em ordenação crescente, adicionalmente retorna um vetor com os índices das linhas/colunas remanescentes (k);
- **<orientação>** define como será realizado o produtório:
 - “r”: o resultado será uma matriz contendo apenas as linhas únicas da <Matriz>;
 - “c”: o resultado será uma matriz contendo apenas as colunas únicas da <Matriz>.

Interseção

[resultado[, kA, kB]] = intersect(<MatrizA>, <MatrizB>, <orientação>)

- Retorna uma matriz contendo as linhas/colunas em comum entre duas matrizes, adicionalmente retorna dois vetores com os índices das linhas/colunas em comum de cada matriz;
- **<orientação>** define como será realizada a comparação:
 - “r”: o resultado será uma matriz contendo apenas as linhas em comum, as duas matrizes precisam ter o mesmo número de colunas;
 - “c”: o resultado será uma matriz contendo apenas as colunas em comum, as duas matrizes precisam ter o mesmo número de linhas.

Interseção

$[\text{resultado}, \text{kA}, \text{kB}] = \text{intersect}(\langle \text{MatrizA} \rangle, \langle \text{MatrizB} \rangle, \langle \text{orientação} \rangle)$

- Exemplo:

--> A = [0, 0, 1, 1, 1; ...
 0, 1, 1, 1, 1; ...
 2, 0, 1, 1, 1; ...
 0, 2, 2, 2, 2; ...
 2, 0, 1, 1, 1; ...
 0, 0, 1, 1, 2]

--> B = [1, 0, 1; ...
 1, 0, 2; ...
 1, 2, 3; ...
 2, 0, 4; ...
 1, 2, 5; ...
 1, 0, 6]

--> [R, kA, kB] = intersect(A, B, 'c')

KB = 2. 1.

kA = 1. 3.

R = 0. 1.

0. 1.

2. 1.

0. 2.

2. 1.

0. 1.

Busca (pesquisa)

[índices] = find(<condição>[, <nmax>])

- Retorna um vetor ordenado contendo os índices de elementos de uma matriz que atendem à **condição** de entrada (o número de índices é limitado a **nmax**, o valor -1 (padrão) indica “todos”);
- Os índices são contabilizados continuamente seguindo as colunas, conforme pode ser visto no resultado do exemplo:

Busca (pesquisa)

[índices] = find(<condição>[, <nmax>])

- Exemplo:

--> A = [0,0,1,1,1; 0,1,1,1,1; 2,0,1,1,1; 0,2,2,2,2; 2,0,1,1,1; 0,0,1,1,2]

A =	0.	0.	1.	1.	1.
	0.	1.	1.	1.	1.
	2.	0.	1.	1.	1.
	0.	2.	2.	2.	2.
	2.	0.	1.	1.	1.
	0.	0.	1.	1.	2.

--> find(A > 1)

ans = 3. 5. 10. 16. 22. 28. 30.

Ordenação

[resultado, indices] = gsort(<Matriz>[, tipo[, direção]])

- Retorna uma matriz ordenada contendo os elementos de uma matriz de entrada, adicionalmente retorna uma matriz com os índices dos elementos na matriz de entrada;
 - Utiliza o algoritmo “quick sort”;
- **tipo**: usado para definir o tipo de ordenação:
 - ‘r’: ordena cada coluna de acordo com o valor de suas linhas;
 - ‘c’: ordena cada linha de acordo com o valor de suas colunas;
 - ‘g’: ordena todos os elementos (padrão);
 - ‘lr’: ordem lexicográfica das linhas;
 - ‘lc’: ordem lexicográfica das colunas;
- **direção**: usado para definir a direção de ordenação:
 - ‘i’: para ordem crescente;
 - ‘d’: para ordem decrescente (padrão);

Ordenação

[resultado, indices] = gsort(<Matriz>[, tipo[, direção]])

- Exemplo 1:

```
--> A = [ 1,  2,  2; ...  
         1,  2,  1; ...  
         1,  1,  2; ...  
         1,  1,  1]
```

```
--> [B, Bi] = gsort(A, 'g', 'i')
```

```
Bi =  1.  7.  5.  
      2.  8.  6.  
      3. 10.  9.  
      4. 12. 11.
```

```
B =  1.  1.  2.  
     1.  1.  2.  
     1.  1.  2.  
     1.  1.  2.
```

Mantém a dimensão da matriz, mas ordena todos os elementos.

Ordenação

`[resultado, indices] = gsort(<Matriz>[, tipo[, direção]])`

- Exemplo 2:

```
--> A = [ 1,  2,  2; ...
         1,  2,  1; ...
         1,  1,  2; ...
         1,  1,  1]
```

```
--> [B, Bi] = gsort(A, 'r', 'i')
```

```
Bi =  1.  3.  2.
      2.  4.  4.
      3.  1.  1.
      4.  2.  3.

B =  1.  1.  1.
     1.  1.  1.
     1.  2.  2.
     1.  2.  2.
```

Alteram a ordem dos **elementos**, não das colunas/linhas.

```
--> [B, Bi] = gsort(A, 'c', 'i')
```

```
Bi =  1.  2.  3.
      1.  3.  2.
      1.  2.  3.
      1.  2.  3.

B =  1.  2.  2.
     1.  1.  2.
     1.  1.  2.
     1.  1.  1.
```

Ordenação

[resultado, indices] = gsort(<Matriz>[, tipo[, direção]])

- Exemplo 3:

```
--> A = [ 1,  2,  2; ...
         1,  2,  1; ...
         1,  1,  2; ...
         1,  1,  1]
```

```
--> [B, Bi] = gsort(A, 'lr', 'i')
```

```
Bi =  4.
      3.
      2.
      1.

B =  1.  1.  1.
     1.  1.  2.
     1.  2.  1.
     1.  2.  2.
```

Alteram a ordem das **colunas/linhas**, não dos elementos.

```
--> [B, Bi] = gsort(A, 'lc', 'i')
```

```
Bi =  1.  2.  3.

B =  1.  2.  2.
     1.  1.  2.
     1.  2.  1.
     1.  1.  1.
```

Plotando gráficos

plot2d(<Vetor X>, <Matriz Y's>)

- As funções de plotagem de gráficos aplicadas a vetores também podem ser usadas com matrizes;
- Neste caso, serão traçadas várias curvas em um único gráfico;
- Resultado semelhante pode ser obtido com a utilização de uma sequência de funções **plot2d()** com vetores, sem a utilização da função **clf()**.

Plotando gráficos

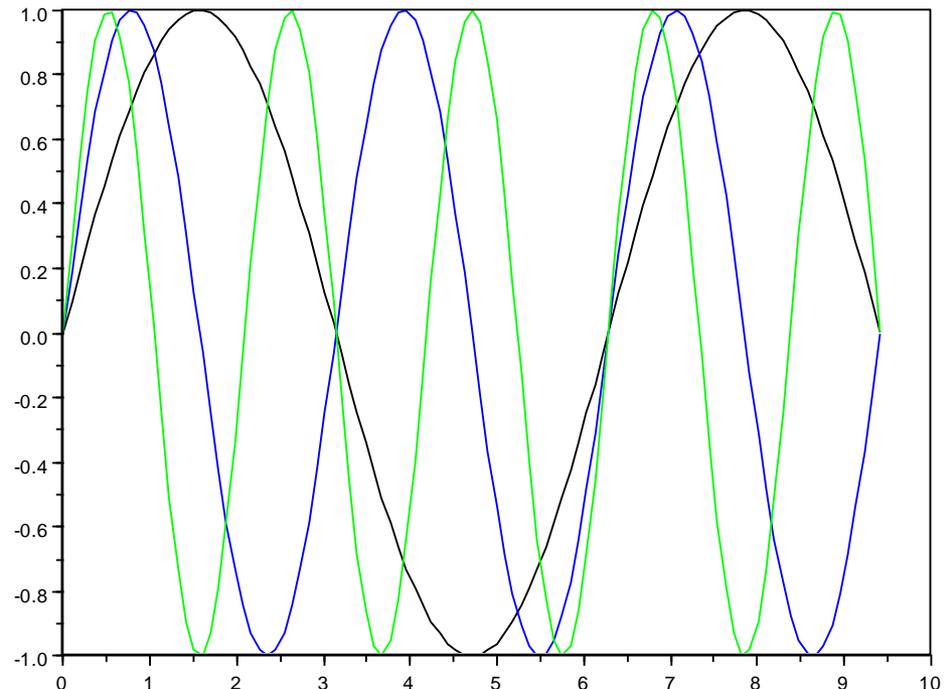
`plot2d(<Vetor X>, <Matriz Y's>)`

- Exemplo:

```
--> X = (0:0.1:3*%pi)';
```

```
--> plot2d(X, [sin(X) sin(2*X) sin(3*X)])
```

X é um vetor coluna contendo as coordenadas do eixo x e as funções compõem uma matriz onde cada coluna representa as coordenadas do eixo y e de sua função correspondente.



Plotando gráficos

- Existem variações da função **plot2D**, consulte o *help on-line do Scilab* (<http://help.scilab.org/>) para mais informações;
- Alguns exemplos:
 - `plot()`;
 - `plot2d1()`;
 - `plot2d2()`;
 - `plot2d3()`;
 - `plot2d4()`;

Introdução;
Declaração de vetores;
Algumas operações com vetores;
Algumas funções aplicadas a vetores;
Exercícios.

EXERCÍCIOS

Lista 5 do prof. David

- Resolução dos exercícios da lista conforme distribuição predefinida;
- A resolução da lista deve ser feita sem a utilização de funções como somatório, produtório, etc. O objetivo é fortalecer o aprendizado da programação de computadores e da lógica aplicada à resolução de problemas computacionais;
- **Dica de estudo complementar:** identifique os exercícios da lista de exercícios que poderiam ser resolvidos com o uso destas funções e implemente suas soluções desta maneira. O objetivo é consolidar o conhecimento das funções avançadas da linguagem, para resolver problemas do seu cotidiano de forma mais rápida e eficiente.

Próxima aula prática: resolução de exercícios com o Scilab.

Próxima aula teórica: Funções.

FIM!

DÚVIDAS?