

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

**[www.decom.ufop.br/bcc701](http://www.decom.ufop.br/bcc701)**  
**2012/01**



# Vetores.

Material Didático Unificado.

# Agenda

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

**Introdução;**

Declaração de vetores;

Algumas operações com vetores;

Algumas funções aplicadas a vetores;

Exercícios.

# INTRODUÇÃO

# Conjunto de variáveis

- Em determinadas situações é necessário utilizar muitas variáveis, por exemplo:
  - 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`

# Conjunto de variáveis

- Em determinadas situações é necessário utilizar muitas variáveis, por exemplo:

- 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: ');

E se de repente for necessário manipular 10 notas de alunos?

- Ler e imprimir cinco números:

- for i = 1 : 5  
    Num = input('Digite um numero: ');  
    printf('Numero digitado: %g', Num);  
end

E se de repente for necessário manipular os números digitados depois do laço for?

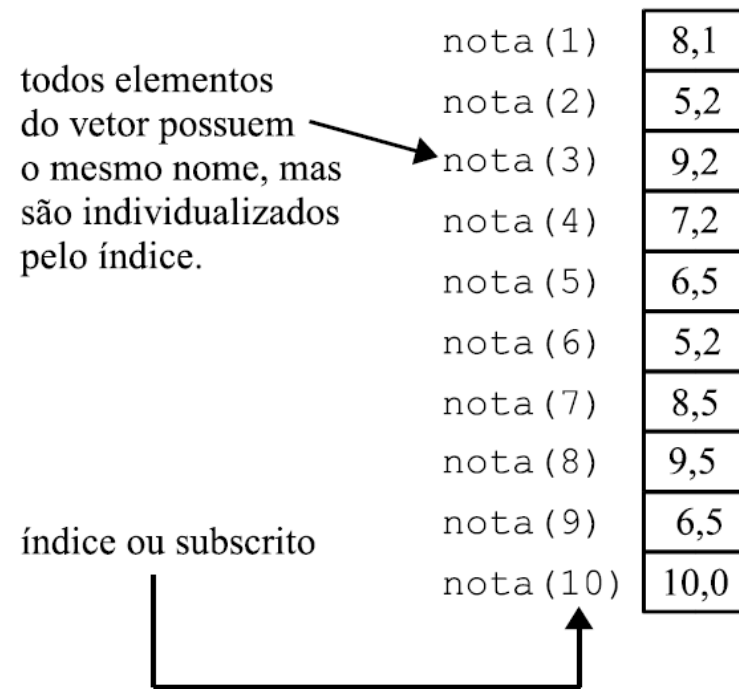
# O tipo de dados **Vetor**

- Nestes casos, todas as variáveis representam um conjunto de valores, possuem um objetivo em comum e são do mesmo tipo de dados;
- Uma estrutura de dados muito utilizada para armazenar e manipular este tipo de conjunto de variáveis é o **Vetor**;
- 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 >=).

# O tipo de dados 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:



Introdução;

**Declaração de vetores;**

Algumas operações com vetores;

Algumas funções aplicadas a vetores;

Exercícios.

# DECLARAÇÃO DE VETORES



# Tópicos

- Definindo todos os elementos;
- Definindo elementos por faixas;
- Vetor de 1's;
- Vetor de 0's.

# Definindo todos os elementos

- Utiliza-se colchetes para delimitar todos os elementos;
- Para definir um “**vetor linha**”, utiliza-se espaço ou vírgula para separar os elementos:

- Exemplos:

V1 = [1 2 3 4 5];

V2 = [5,4,3,2,1];

- Resultados (para V1 e V2 respectivamente):

1. 2. 3. 4. 5.

-->

5. 4. 3. 2. 1.

-->

# Definindo todos os elementos

- Para definir um “**vetor coluna**”, utiliza-se ponto-e-vírgula para separar os elementos:

- Exemplo:

$V3 = [1;2;3;4;5];$

- Resultado:

1.

2.

3.

4.

5.

-->

# Definindo elementos por faixas

- Semelhante à definição dos valores de um laço for:

**Vetor = <valor inicial> : <incremento> : <valor final>**

- Exemplos:

V4 = 1:5; // o incremento de 1 é opcional

V5 = 5:-1:1;

- Resultados (para V4 e V5 respectivamente):

1. 2. 3. 4. 5.

-->

5. 4. 3. 2. 1.

-->

# Vetor de 1's

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

**Vetor = ones(<linhas>, <colunas>)**

- **Vetor**: nome da variável do tipo vetor;
- **ones**: função que retorna uma **matriz**\* com valores 1;
- **<linhas>**: número de linhas;
- **<colunas>**: número de colunas;
  
- \* **Matriz** é objeto de estudos do próximo tópico abordado na disciplina;
- Para construir um **vetor**, o número de linhas OU o número de colunas deve ser igual a **um**.

# Vetor de 1's

- Exemplos:
  - **Vetor linha com cinco colunas:**
    - `c = ones(1, 5)`  
`c = 1. 1. 1. 1. 1.`  
-->
  - **Vetor coluna com cinco linhas:**
    - `c = ones(5, 1)`  
`c = 1.`  
`1.`  
`1.`  
`1.`  
`1.`  
-->

# Vetor de 0's

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

**Vetor = zeros(<linhas>, <colunas>)**

- **Vetor**: nome da variável do tipo vetor;
- **zeros**: função que retorna uma **matriz**\* com valores 0;
- **<linhas>**: número de linhas;
- **<colunas>**: número de colunas;
  
- \* **Matriz** é objeto de estudos do próximo tópico abordado na disciplina;
- Para construir um **vetor**, o número de linhas OU o número de colunas deve ser igual a **um**.

# Vetor de 0's

- Exemplos:
  - **Vetor linha com cinco colunas:**
    - `c = zeros(1, 5)`  
`c = 0. 0. 0. 0. 0.`  
-->
  - **Vetor coluna com cinco linhas:**
    - `c = zeros (5, 1)`  
`c = 0.`  
`0.`  
`0.`  
`0.`  
`0.`  
-->



Introdução;

Declaração de vetores;

**Algumas operações com vetores;**

Algumas funções aplicadas a vetores;

Exercícios.

# ALGUMAS OPERAÇÕES COM VETORES

# Tópicos

- Acesso aos elementos;
- Transposição de vetores;
- Operações binárias:
  - Soma com escalar e de vetores;
  - Subtração com escalar e de vetores;
  - Multiplicação com escalar e de vetores;
  - Produto interno;
  - Divisão por escalar e de vetores (à esquerda e à direita).

# Acesso aos elementos

- Para acessar um elemento específico:

**Vetor(<índice>)**

- Exemplo:

```
clc;
```

```
V = [10, 20, 30, 40, 50];
```

```
disp(V(3));
```

- Resultado:

```
30.
```

```
-->
```

- Pode ser aplicado tanto a vetor de coluna quanto de linha:
  - Para **vetor linha**, <índice> corresponde ao **número da coluna**;
  - Para **vetor coluna**, <índice> corresponde ao **número da linha**;
- Pode ser usado para modificar o valor: **V(3) = 300**, modifica o valor do **índice 3** de **30** para **300**.

# Acesso aos elementos

- Com o acesso a elementos específicos é possível definir o vetor, no exemplo anterior:

```
clc;
```

```
V(1) = 10; V(2) = 20; V(3) = 30; V(4) = 40; V(5) = 50;
```

```
disp(V);
```

- Resultado:

```
10.
```

```
20.
```

```
30.
```

```
40.
```

```
50.
```

```
-->
```

- Note que o resultado é um vetor coluna.

[ 20 ]

# Transposição de vetores

- **Operador apóstrofo ('):** Vetor'
  - Transforma um vetor coluna em um vetor linha, e vice-versa:

```
clc;  
V = [10, 20, 30, 40, 50];  
V = V';  
disp(V);
```
  - Resultado:

```
10.  
20.  
30.  
40.  
50.  
-->
```
  - Também poderia ser feito:  $V = [10:10:50]'$ , para obter o mesmo resultado anterior;

# Operações binárias

- **Soma com escalar:**  $V + \langle \text{valor} \rangle$  OU  $\langle \text{valor} \rangle + V$ :
  - Exemplo:

```
clc;  
V1 = [1 2 3 4 5];  
V2 = V1 + 2; // Ou V2 = 2 + V1; (causará o mesmo efeito)  
disp(V2);
```
  - Resultado:

```
3. 4. 5. 6. 7.  
-->
```

# Operações binárias

- **Soma de vetores:  $V1 + V2$ :**
  - $V1$  e  $V2$  devem ser da mesma dimensão;

- Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5];
```

```
V2 = [5 4 3 2 1];
```

```
Soma = V1 + V2;
```

```
disp(Soma);
```

- Resultado:

```
6. 6. 6. 6. 6.
```

```
-->
```

# Operações binárias

- **Subtração com escalar:** Vetor - <valor> OU <valor> - Vetor:

- Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5];
```

```
V2 = V1 - 1;
```

```
V3 = 1 - V1;
```

```
disp(V2);
```

```
disp(V3);
```

- Resultado:

```
0. 1. 2. 3. 4.
```

```
0. -1. -2. -3. -4.
```

```
-->
```



# Operações binárias

- **Subtração de vetores:  $V1 - V2$ :**
  - $V1$  e  $V2$  devem ser da mesma dimensão;

- Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5];
```

```
V2 = [5 4 3 2 1];
```

```
Subtracao = V1 - V2;
```

```
disp(Subtracao);
```

- Resultado:

```
- 4. - 2. 0. 2. 4.
```

```
-->
```

# Operações binárias

- **Multiplicação por escalar:**  $V * \langle \text{valor} \rangle$  OU  $\langle \text{valor} \rangle * V$ :

- Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5];
```

```
V2 = V1 * 2; // Ou V2 = 2 * V1; (causará o mesmo efeito)
```

```
disp(V2);
```

- Resultado:

```
2. 4. 6. 8. 10.
```

```
-->
```

# Operações binárias

- **Multiplicação de vetores:  $V1 .* V2$ :**
  - $V1$  e  $V2$  devem ser da mesma dimensão;

- Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5];
```

```
V2 = [5 4 3 2 1];
```

```
Mult = V1 .* V2;
```

```
disp(Mult);
```

- Resultado:

```
5. 8. 9. 8. 5.
```

```
-->
```

# Operações binárias

- **Produto interno:  $V1 * V2$ :**
  - $V1$  é um vetor linha e  $V2$  é um vetor coluna;
  - O número de colunas de  $V1$  deve ser igual ao número de linhas de  $V2$ ;
  - Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5]; // Vetor linha
```

```
V2 = [5;4;3;2;1]; // Vetor coluna
```

```
Mult = V1 * V2;
```

```
disp(Mult);
```

- Resultado:

```
35.
```

```
-->
```

# Operações binárias

- **Divisão por escalar:**  $V / \langle \text{valor} \rangle$  OU  $\langle \text{valor} \rangle \setminus V$ :
  - Exemplo:

```
clc;  
V1 = [1 2 3 4 5];  
V2 = V1 / 2; // Ou V2 = 2 \ V1; (causará o mesmo efeito)  
disp(V2);
```
  - Resultado:

```
0.5  1.  1.5  2.  2.5  
-->
```

# Operações binárias

- **Divisão de vetores à direita:  $V1 ./ V2$ :**
  - $V1$  e  $V2$  devem ser da mesma dimensão;

- Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5];
```

```
V2 = [5 4 3 2 1];
```

```
Div = V1 ./ V2;
```

```
disp(Div);
```

- Resultado:

```
0.2  0.5  1.  2.  5.
```

```
-->
```

# Operações binárias

- **Divisão de vetores à esquerda:  $V1 \ .\ V2$ :**

- V1 e V2 devem ser da mesma dimensão;

- Exemplo:

```
clc;
```

```
V1 = [1 2 3 4 5];
```

```
V2 = [5 4 3 2 1];
```

```
Div = V1 .\ V2;
```

```
disp(Div);
```

- Resultado:

```
5.  2.  1.  0.5  0.2
```

```
-->
```

# Operações binárias

- Para mais informações, procure pelos operadores do scilab:
  - Soma (*plus*: +):
    - [http://help.scilab.org/docs/5.3.3/pt\\_BR/plus.html](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](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](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/slash.html)
    - [http://help.scilab.org/docs/5.3.3/pt\\_BR/backslash.html](http://help.scilab.org/docs/5.3.3/pt_BR/backslash.html)



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

# ALGUMAS FUNÇÕES APLICADAS A VETORES

# Tópicos

- Dimensão de vetores;
- Somatório;
- Somatório cumulativo;
- Produtório;
- Produtório cumulativo;
- Elementos únicos;
- União;
- Interseção;
- Busca (pesquisa);
- Ordenação;
- Plotando gráficos.

# Dimensão de vetores

**[resultado] = length(<Vetor>)**

- Retorna a quantidade de elementos do vetor, muito útil para construir laços de repetição para percorrer os elementos do vetor:

```
clc;
```

```
V = [10, 20, 30, 40, 50];
```

```
n = length(V);
```

```
disp(n);
```

- Resultado:

```
5.
```

```
-->
```

# Somatório

**[resultado] = sum(<Vetor>)**

- Retorna o somatório de todos os elementos do vetor:

```
clc;
```

```
V = [10, 20, 30, 40, 50];
```

```
somatorio = sum(V);
```

```
disp(somatorio);
```

- Resultado:

```
150.
```

```
-->
```

- Perceba que o resultado é um valor numérico.

# Somatório cumulativo

**[resultado] = cumsum(<Vetor>)**

- Retorna o somatório de todos os elementos do vetor, de forma acumulativa a cada linha/coluna:

```
clc;
```

```
V = [10, 20, 30, 40, 50];
```

```
somatorio = cumsum(V);
```

```
disp(somatorio);
```

- Resultado:

```
10. 30. 60. 100. 150.
```

```
-->
```

- Perceba que o resultado é um vetor.

# Produtório

**[resultado] = prod(<Vetor>)**

- Retorna o produtório de todos os elementos do vetor:

```
clc;
```

```
V = [10, 20, 30, 40, 50];
```

```
produtorio = prod(V);
```

```
disp(produtorio);
```

- Resultado:

```
12000000.
```

```
-->
```

- Perceba que o resultado é um valor numérico.

# Produtório cumulativo

**[resultado] = cumprod(<Vetor>)**

- Retorna o produtório de todos os elementos do vetor, de forma acumulativa a cada linha/coluna:

```
clc;
```

```
V = [10, 20, 30, 40, 50];
```

```
produtorio = cumprod(V);
```

```
disp(produtorio);
```

- Resultado:

```
10. 200. 6000. 240000. 12000000.
```

```
-->
```

- Perceba que o resultado é um vetor.

# Elementos únicos

**[resultado [, k]] = unique(<Vetor>)**

- Retorna um vetor ordenado contendo os elementos únicos de um vetor, adicionalmente retorna um vetor com os índices dos elementos no vetor de entrada:

```
clc;
```

```
V= [60, 30, 40, 50, 20, 20, 30, 10, 70, 80];
```

```
[unicos, indices] = unique(V);
```

```
disp(unicos);
```

```
disp(indices);
```

- Resultado:

```
10. 20. 30. 40. 50. 60. 70. 80.
```

```
8. 5. 2. 3. 7. 1. 9. 10.
```

```
-->
```

← Elementos únicos de V

← Índices dos elementos em V



# União

**[resultado [, kA, kB]] = union(<Vetor A>, <Vetor B>)**

- Retorna um vetor ordenado contendo a união entre os elementos de dois vetores, adicionalmente retorna vetores com os índices dos elementos em cada vetor de entrada:

```
clc;
```

```
V1 = [60, 30, 40, 50, 20];
```

```
V2 = [20, 30, 10, 70, 80];
```

```
[uniao, indicesA, indicesB] = union(V1, V2);
```

```
disp(uniao);
```

```
disp(indicesA);
```

```
disp(indicesB);
```

- Resultado:

```
10. 20. 30. 40. 50. 60. 70. 80.
```

```
5. 2. 3. 4. 1.
```

```
3. 4. 5.
```

```
-->
```

← Elementos únicos de  $V1 \cup V2$

← Índices dos elementos em V1

← Índices dos elementos em V2

# Interseção

**[resultado [, kA, kB]] = intersect(<Vetor A>, <Vetor B>)**

- Retorna um vetor ordenado contendo os elementos em comum de dois vetores, adicionalmente retorna vetores com os índices dos elementos em cada vetor de entrada:

```
clc;
```

```
V1 = [60, 30, 40, 50, 20];
```

```
V2 = [20, 30, 10, 70, 80];
```

```
[intersecao, indicesA, indicesB] = intersect(V1, V2);
```

```
disp(intersecao);
```

```
disp(indicesA);
```

```
disp(indicesB);
```

- Resultado:

```
20. 30.
```

```
5. 2.
```

```
1. 2.
```

```
-->
```

← Elementos de  $V1 \cap V2$

← Índices dos elementos em V1

← Índices dos elementos em V2

# Busca (pesquisa)

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

- Retorna um vetor ordenado contendo os índices dos elementos de um vetor que atendem à **condição** de entrada (o número de índices é limitado a **nmax**, o valor -1 (padrão) indica “todos”):

```
clc;
```

```
V = [60, 30, 40, 50, 20, 20, 30, 10, 70, 80];
```

```
encontrados1 = find(V > 50);
```

```
encontrados2 = find(V == 30);
```

```
encontrados3 = find(V == 30 | V == 20);
```

```
disp(encontrados1);
```

```
disp(encontrados2);
```

```
disp(encontrados3);
```

- Resultado:

1. 9. 10.     ← Elementos de V maiores de 50

2. 7.        ← Elementos de V iguais a 30

2. 5. 6. 7. ← Elementos de V iguais a 30 OU iguais a 20

-->

# Ordenação

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

- Retorna um vetor ordenado contendo os elementos de um vetor de entrada, adicionalmente retorna um vetor com os índices dos elementos no vetor de entrada;
  - Utiliza o algoritmo “**quick sort**”;
- **tipo**: usado para definir o tipo de ordenação, no caso de vetores, recomenda-se utilizar sempre o valor ‘**g**’ (valor padrão), que significa ordenar todos os elementos;
- **direção** : usado para definir a direção de ordenação:
  - ‘**i**’: para ordem crescente;
  - ‘**d**’: para ordem decrescente (padrão);

# Ordenação

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

- Exemplo:

```
clc;  
V = [60, 30, 40, 50, 20, 20, 30, 10, 70, 80];  
[ordenado1, indice1] = gsort(V);  
[ordenado2, indice2] = gsort(V, 'g', 'i');  
disp(ordenado1);           // Ordenação padrão  
disp(indice1);            // Índices dos elementos da ordenação padrão  
disp(ordenado2);         // Ordenação de forma crescente  
disp(indice2);           // Índices dos elementos da ordenação crescente
```

- Resultado:

```
80. 70. 60. 50. 40. 30. 30. 20. 20. 10.  
10. 9. 1. 4. 3. 2. 7. 5. 6. 8.  
10. 20. 20. 30. 30. 40. 50. 60. 70. 80.  
8. 5. 6. 2. 7. 3. 4. 1. 9. 10.  
-->
```

# Plotando gráficos

**plot2d(<Vetor X>, <Vetor Y>)**

- Plota um gráfico no plano cartesiano unindo segmentos de reta formados pelas coordenadas X e Y dos vetores passados como parâmetro:
  - <Vetor X> possui as coordenadas do eixo X;
  - <Vetor Y> possui as coordenadas do eixo Y;
  - Os dois vetores devem possuir a mesma dimensão (**n**);
  - Cada posição corresponde a uma coordenada:  $(x(1), y(1))$ ,  $(x(2), y(2))$ , ...,  $(x(n), y(n))$ .

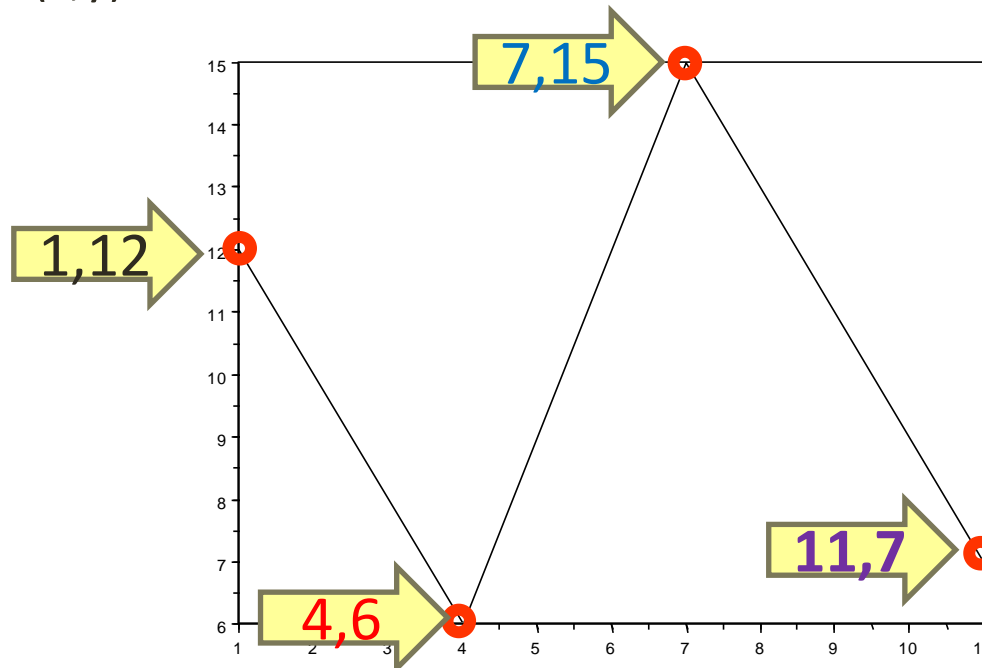
# Plotando gráficos

`plot2d(<Vetor X>, <Vetor Y>)`

- Exemplo 1:

--> `x = [1 4 7 11]`; `y = [12 6 15 7]`;

--> `plot2d(x,y)`



# Plotando gráficos

`plot2d(<Vetor X>, <Vetor Y>)`

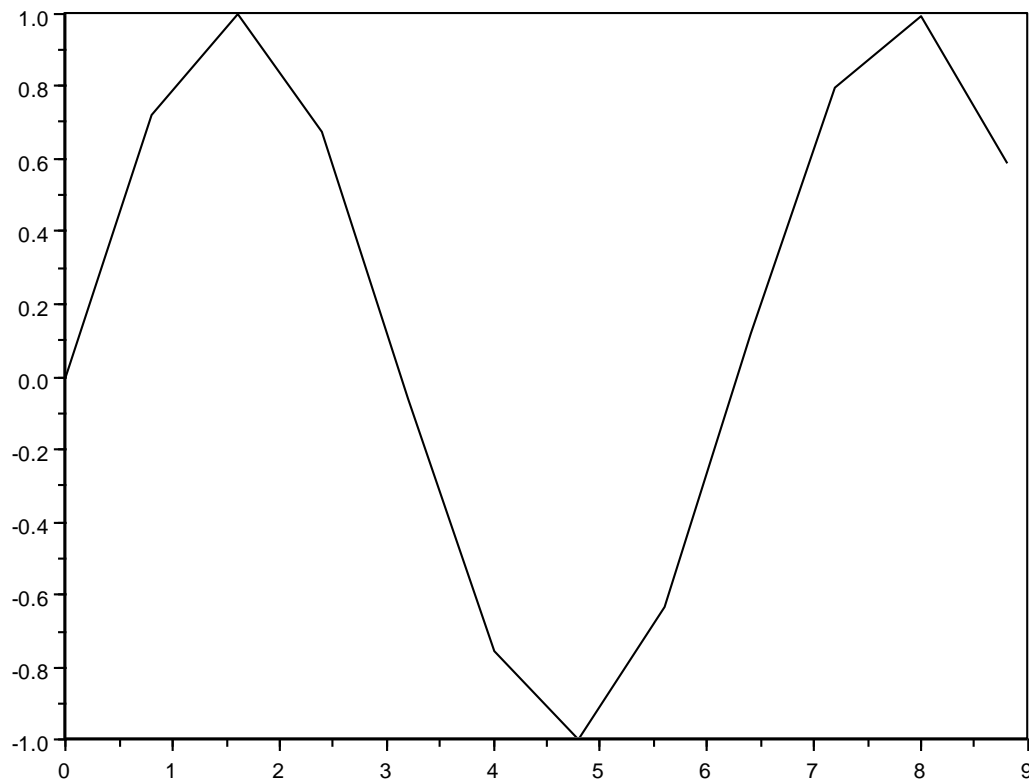
- Exemplo 2:

```
-->x = 0:0.8:3*%pi;
```

```
-->y = sin(x);
```

```
--> clf();
```

```
--> plot2d(x,y)
```





# Plotando gráficos

`plot2d(<Vetor X>, <Vetor Y>)`

- Exemplo 2:

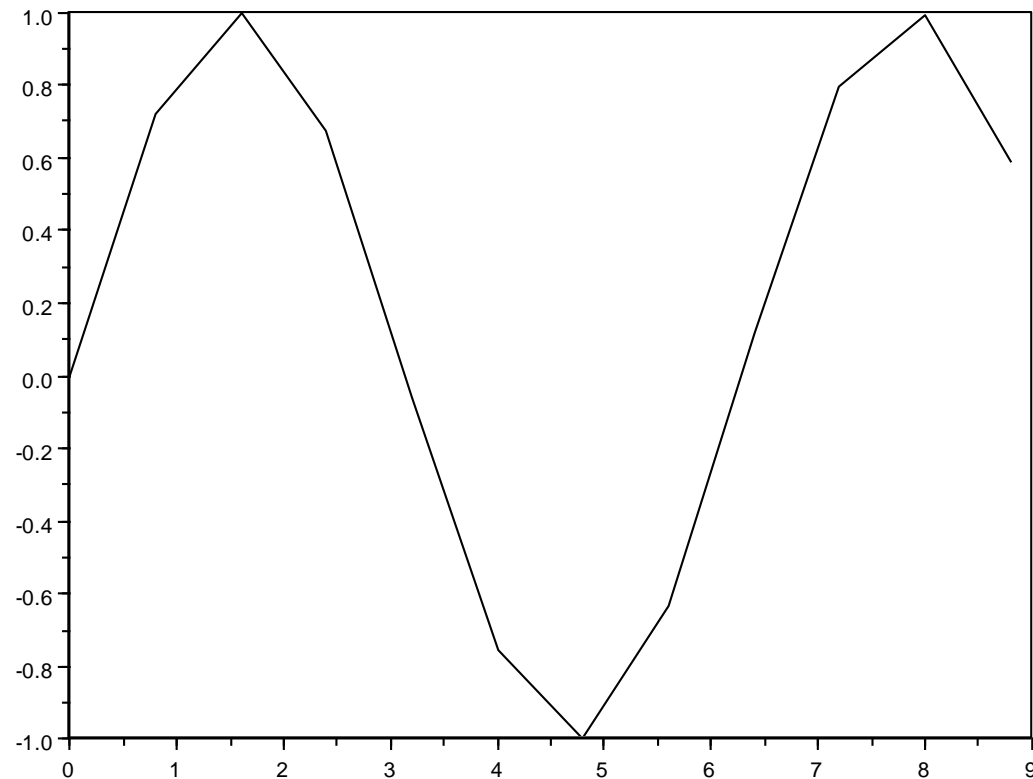
--> `x = 0:0.8:3*%pi;`

--> **`y = sin(x);`**

--> `clf();`

--> `plot2d(`

A função **seno** é aplicada a cada elemento do vetor (x), originando outro vetor (y) com os resultados individuais.



# Plotando gráficos

`plot2d(<Vetor X>, <Vetor Y>)`

- Exemplo 2:

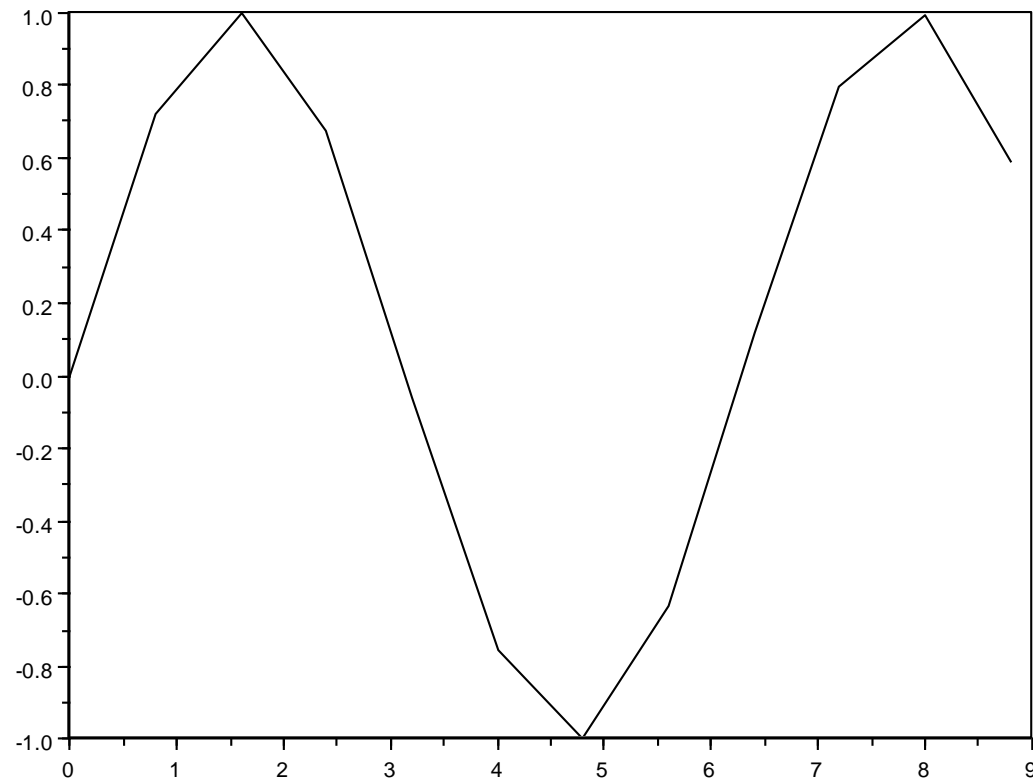
--> `x = 0:0.8:3*%pi;`

--> `y = sin(x);`

--> `clf();`

--> `plot2d(x, y)`

Por padrão, os gráficos plotados são sobrepostos. Para apagar os gráficos plotados anteriormente, utilize a função `clf()`.



# Plotando gráficos

`plot2d(<Vetor X>, <Vetor Y>)`

- Exemplo 2:

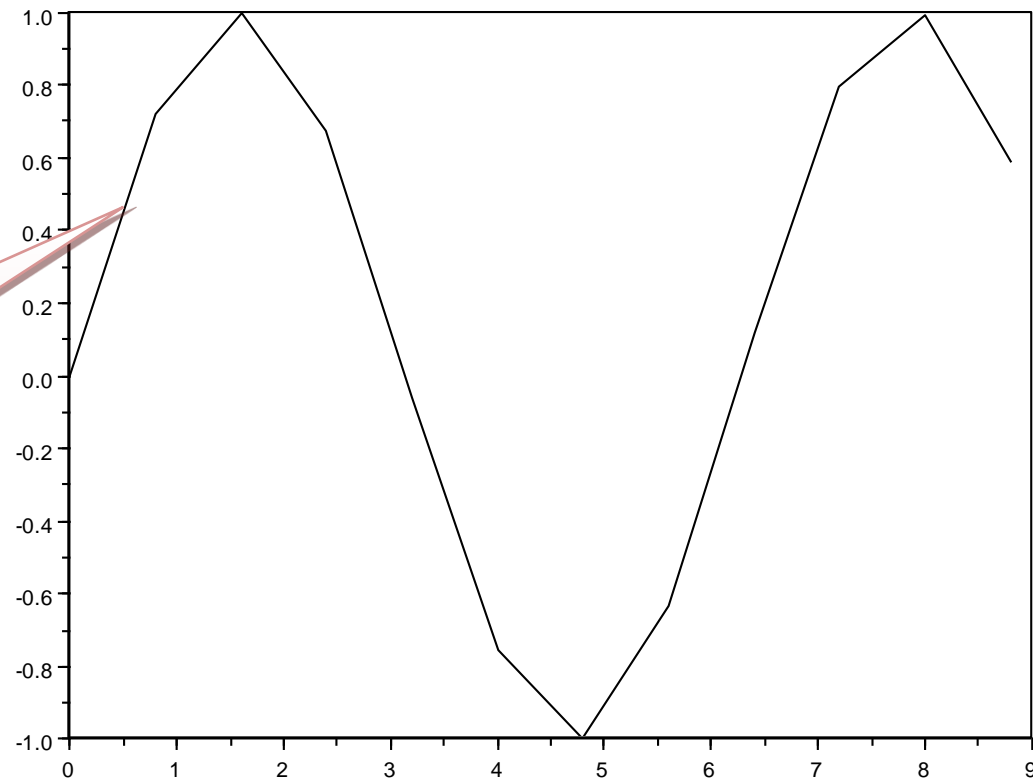
--> `x = 0:0.8:3*%pi;`

--> `y = sin(x);`

--> `clf();`

--> `plot2d(x,y)`

Observe no gráfico que o espaçamento de **0.8**, a cada coordenada x, originou um gráfico da função seno "ruim".



# Plotando gráficos

`plot2d(<Vetor X>, <Vetor Y>)`

- Exemplo 3:

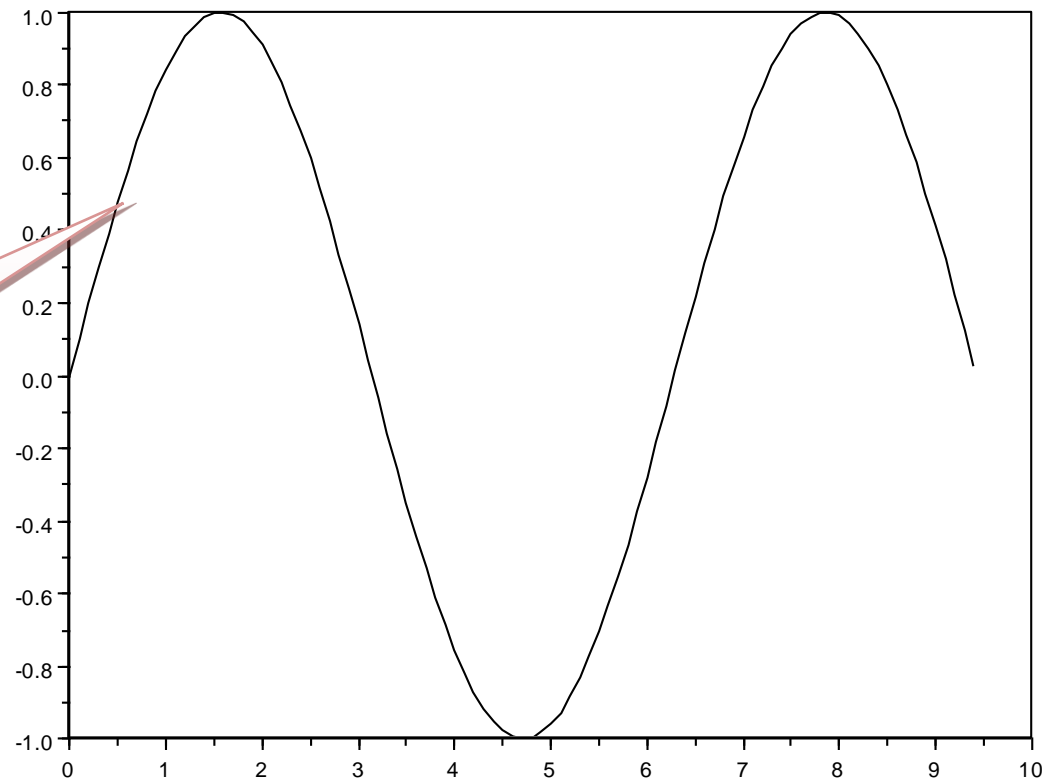
--> `x = 0:0.1:3*%pi;`

--> `y = sin(x);`

--> `clf();`

--> `plot2d(x,y)`

Observe agora que, com um espaçamento de **0.1**, o gráfico da função seno originado ficou bem melhor.



# 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

# Exercícios propostos

1. Faça um programa que preencha um vetor de 10 elementos através de entradas do usuário. Após a definição dos elementos do vetor, calcule a média dos valores.
2. Faça um programa que preencha dois vetores de 10 elementos através de entradas do usuário. Após a definição dos dois vetores, construa um terceiro vetor onde cada elemento corresponde ao dobro da soma entre os elementos correspondentes dos outros dois vetores. Imprima o conteúdo do vetor calculado.
3. Faça um programa que preencha dois vetores de 10 elementos através de entradas do usuário. Após a definição dos dois vetores, construa um terceiro vetor onde cada elemento de *índice ímpar* receba o valor correspondente do *primeiro vetor* e cada elemento de *índice par* receba o valor correspondente do *segundo vetor*. Imprima o conteúdo do vetor calculado.

# Exercícios propostos

4. Escreva um programa que preencha um vetor com entradas do usuário. Considere que o usuário definirá apenas valores numéricos positivos, e que, ao desejar encerrar a definição dos elementos ele digite um valor negativo. Após a entrada de todos os elementos do vetor, calcule e imprima o seu somatório, **sem a utilização da função sum**.
5. Escreva um programa semelhante ao anterior, que retorne e imprima o produtório cumulativo, **sem a utilização da função cumprod**.
6. Escreva um programa semelhante aos anteriores, mas que retorne e imprima um vetor contendo apenas os elementos únicos, **sem a utilização da função unique**. *Dicas: Com o vetor preenchido, percorra seus elementos inserindo os elementos únicos em um novo vetor. Um elemento único é aquele que ainda não se encontra no novo vetor. Para descobrir se um elemento já está inserido no novo vetor, utilize a função **find**.*



# Lista 4 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: Variáveis Homogêneas - Matrizes.

**FIM!**

**DÚVIDAS?**