

Unidade Acadêmica do Cabo de Santo Agostinho  
(UACSA) - UFRPE

# **Apostila Introdutória de Octave/Matlab**

Fernando Gonçalves de Almeida Neto  
Vítor Heloiz Nascimento

Recife - PE  
2016

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Algumas operações básicas com escalares</b>	<b>4</b>
<b>3</b>	<b>Algumas constantes usadas em Octave/Matlab</b>	<b>5</b>
<b>4</b>	<b>Vetores e Matrizes</b>	<b>6</b>
4.1	Definindo vetores . . . . .	6
4.2	Definindo matrizes . . . . .	7
4.3	Acesso aos elementos de vetores e matrizes . . . . .	7
4.4	Transposição de vetores e matrizes . . . . .	9
4.5	Solução de sistemas de equações lineares . . . . .	11
4.6	Cálculo de determinante . . . . .	11
4.7	Construção de matrizes e vetores . . . . .	12
4.8	Multiplicação por escalar . . . . .	13
4.9	Operações entre matrizes e vetores . . . . .	14
4.10	Operações elemento-a-elemento em vetores e matrizes . . . . .	15
4.11	Produto escalar . . . . .	17
4.12	Descobrimdo as dimensões de vetores e matrizes . . . . .	17
<b>5</b>	<b>Matrizes especiais</b>	<b>19</b>
<b>6</b>	<b>Mais alguns recursos úteis para programação</b>	<b>21</b>
6.1	Ponto-e-vírgula depois de comandos . . . . .	21
6.2	A variável <code>ans</code> . . . . .	21
6.3	Comentários no programa (%) . . . . .	21
6.4	Encontrando as raízes de polinômios . . . . .	22
6.5	Obtendo o polinômio a partir de suas raízes . . . . .	22
6.6	A função <code>help</code> . . . . .	22
6.7	“Limpando” os comando da janela de comando . . . . .	23
6.8	Listando as variáveis usadas: <code>who</code> e <code>whos</code> . . . . .	24
6.9	Apagando variáveis da memória . . . . .	25
<b>7</b>	<b>Funções usadas em estatística</b>	<b>26</b>
7.1	Cálculo de máximo e de mínimo . . . . .	26
7.2	Cálculo de média . . . . .	26
7.3	Cálculo de desvio padrão e de variância . . . . .	27

<b>8</b>	<b>Gráficos</b>	<b>28</b>
8.1	Função <code>plot</code> . . . . .	28
8.2	Colocando título, legenda e informações nos eixos de figuras . . . . .	30
8.3	Outras formas de fazer gráficos . . . . .	33
<b>9</b>	<b>Estruturas de controle de fluxo de programa</b>	<b>35</b>
9.1	<i>Loops</i> : <code>for</code> e <code>while</code> . . . . .	35
9.1.1	Elementos relacionais . . . . .	35
9.2	Usando estruturas condicionais do tipo <code>if</code> , <code>elseif</code> e <code>else</code> . . . . .	36
<b>10</b>	<b>Criando <i>scripts</i> e funções</b>	<b>37</b>
<b>11</b>	<b>Criando funções</b>	<b>39</b>
<b>12</b>	<b>Exemplos adicionais</b>	<b>41</b>
12.1	Probabilidade de dar cara no lançamento de uma moeda . . . . .	41
12.2	Probabilidades das faces de um dado honesto . . . . .	42
12.3	Método dos Mínimos Quadrados . . . . .	43
<b>13</b>	<b>Referências Adicionais</b>	<b>45</b>

# 1 Introdução

Esta apostila tem o objetivo de apresentar alguns recursos básicos disponibilizados pelos *softwares* Octave e Matlab. O primeiro desses *softwares* é uma ferramenta de livre distribuição e é utilizado em processamento numérico intensivo. O segundo, muito utilizado na indústria para desenvolvimento e implementação de produtos, é um *software* proprietário, usado para a mesma finalidade. Embora ambos apresentem grande similaridade com relação à linguagem que utilizam e sejam praticamente compatíveis entre si, existem diversas *toolboxes* para o Matlab que não possuem correspondentes para o Octave. Contudo, para o escopo dessa primeira apresentação, essas diferenças não serão relevantes e os conceitos apresentados serão aplicáveis às duas ferramentas de simulação.

Esta apostila começa apresentando a definição e utilização de variáveis, que podem ser escalares, vetores ou matrizes em geral, e a realização de operações entre esses entes. Em seguida, são apresentados estruturas de controle de fluxo de programa (ex.: `for`, `while`, `if ...`), para depois apresentar o conceito e a forma de elaboração de *scripts* e funções. Para facilitar a organização do texto, os exemplos, retirados diretamente da linha de comando do Matlab/Octave, serão apresentados com fonte *typewriter*. Além disso, para facilitar a compreensão, as variáveis usadas para representar matrizes serão sempre grafadas com letra maiúscula, enquanto vetores e escalares serão representados com letra minúscula.

A apostila apresentada nesta breve introdução é a terceira versão de um material desenvolvido inicialmente com o Prof. Vítor Heloiz Nascimento, professor da Escola Politécnica da Universidade de São Paulo, em 2011. A apostila foi inicialmente proposta para auxiliar o desenvolvimento de exercícios computacionais nas disciplinas Introdução a Processos Estocásticos e, posteriormente, Sistemas e Sinais, ofertadas aos alunos dos cursos de Engenharia de Telecomunicações e de Sistemas Eletrônicos.

Esta apostila não visa esgotar as possibilidades de utilização dessas ferramentas de simulação matemática: espera-se que essa curta introdução à linguagem Octave/Matlab sirva como um incentivo para que os alunos busquem mais referências e percebam como tais ferramentas podem facilitar o estudo e o desenvolvimento de projetos em Engenharia.

## 2 Algumas operações básicas com escalares

Para as funções básicas com escalares, a linguagem Octave/Matlab usa comandos semelhantes aos adotados em calculadoras gráficas e científicas. Por esse motivo, o uso dessas funções é bastante intuitivo para indivíduos habituados com essas calculadoras. A Tabela 1 apresenta algumas dessas funções.

Tabela 1: Exemplos de operações básicas em escalares

Comando	Função desempenhada
$a + b$	Soma de dois escalares
$a - b$	Subtração de dois escalares
$a * b$	Multiplicação de dois escalares
$a/b$	Divisão de dois escalares ( $\frac{a}{b}$ )
$a \setminus b$	Divisão de dois escalares ( $\frac{b}{a}$ )
$a \wedge b$	Potenciação de dois escalares ( $a^b$ )
$\exp(a)$	Exponencial ( $e^a$ )
$\text{sqrt}(a)$	Raiz quadrada ( $\sqrt{a}$ )
$\sin(a)$	seno de $a$ ( $a$ em radianos)
$\cos(a)$	cosseno de $a$ ( $a$ em radianos)
$\text{sind}(a)$	seno de $a$ ( $a$ em graus)
$\text{cosd}(a)$	cosseno de $a$ ( $a$ em graus)
$\log(a)$	Logarítmo de $a$ na base $e$
$\log10(a)$	Logarítmo de $a$ na base 10

### 3 Algumas constantes usadas em Octave/Matlab

Em Octave/Matlab, existem algumas variáveis previamente definidas que funcionam como constantes. Nesse caso, em qualquer função ou *script*, basta usá-las sem a necessidade de atribuição de valores. A Tabela 2 apresenta algumas dessas constantes e o que elas representam.

Tabela 2: Constantes e seus valores

Comando	Função desempenhada
pi	$\pi$ (3.141592...)
i (ou j)	Unidade imaginária (Se forem atribuídos valores para i e j, eles são encarados como variáveis comuns. Se não houver atribuição, eles valem $\sqrt{-1}$ ).
inf	Infinito (na verdade, um número muito grande que o programa assume como sendo infinito)
NaN	<i>Not-a-number</i> (Corresponde ao resultado de uma operação que possui resultado numérico indefinido, como 0/0)

## 4 Vetores e Matrizes

Vetores e matrizes são grandezas que aparecem constantemente no desenvolvimento de códigos para cálculo intensivo. Sua manipulação em linguagem Octave/Matlab segue de perto a abordagem geralmente utilizada em Álgebra Linear.

### 4.1 Definindo vetores

Em linguagem Octave/Matlab, define-se um vetor linha como uma sequência de valores entre colchetes, separados por espaços ou por vírgulas. Por exemplo, suponha um vetor linha **a** com três elementos, correspondentes aos números 1, 5 e 7, nessa ordem. Em linguagem Matlab/Octave isso pode ser escrito como

```
>> a = [1 2 3]
```

```
a =
```

```
1    2    3
```

```
ou
```

```
>> a = [1, 2, 3]
```

```
a =
```

```
1    2    3
```

Nesse caso, a variável **a** é definida como um vetor de dimensão  $1 \times 3$ . Se, por outro lado, fosse desejado escrever um vetor coluna **b** cujos elementos fossem 4, 8 e 12, bastaria escrever entre colchetes os elementos separados por ponto-e-vírgula (;), ou seja,

```
>> b = [4; 8; 12]
```

```
b =
```

```
4
8
12
```

## 4.2 Definindo matrizes

De forma semelhante ao apresentado para vetores, matrizes são definidas escrevendo uma lista de elementos entre colchetes. Nesse caso, os elementos de cada linha são separados por espaços ou vírgulas, enquanto as diferentes linhas são separadas por ponto-e-vírgula. Ex. :

```
>> A = [2 4 5 7; 8 3 0 1; 0 9 6 7]
```

```
A =
```

```
     2     4     5     7
     8     3     0     1
     0     9     6     7
```

## 4.3 Acesso aos elementos de vetores e matrizes

Para acessar algum elemento específico em um vetor ou matriz, é necessário informar sua localização em termos de linha e de coluna ocupada. Considere, por exemplo, a matriz **A** do exemplo do item 4.2. Para acessar o elemento da segunda linha, quarta coluna, basta escrever **A(2,4)**, obtendo

```
>> A(2,4)
```

```
ans =
```

```
     1
```

Portando, basta escrever entre parênteses o número da linha seguido pelo número da coluna, separados por vírgula. Vale lembrar que em Matlab/Octave **a contagem dos elementos das linhas e colunas começa em 1, de forma que não existem os elementos  $A(0,0)$ ,  $A(0,\cdot)$  ou  $A(\cdot,0)$ . Se o usuário tentar acessar erradamente essas posições, o programa acusará erro e o fluxo do programa será interrompido.**

Suponha agora que o usuário deseje acessar todas os elementos da segunda linha de **A**. Nesse caso, basta fazer **A(2,:)**:

```
>> A(2,:)
```

```
ans =
```

```
     8     3     0     1
```

Quando se usa `:` (dois pontos) no lugar da posição da coluna, o programa interpreta os dois pontos como sendo “os elementos de todas as colunas de **A**, localizados na linha 2”, fornecendo o resultado anterior. O mesmo raciocínio pode ser estendido para as linhas de **A**:

```
>> A(:,3)
```

```
ans =
```

```
5
0
6
```

Suponha que o usuário deseje acessar apenas os elementos das linhas 2 e 3 de **A**, nas colunas 1 e 3. Isso pode ser escrito como `A(2:3, 1:2:4)`.

```
>> A(2:1:3, 1:2:4)
```

```
ans =
```

```
8    0
0    6
```

Quando foi usado `2 : 1 : 3`, o programa interpretou o comando como “todos os números entre 2 e 3 (incluindo 2), considerados usando um passo 1”. Dessa forma, as linhas 2 e 3 são consideradas. De forma equivalente, o comando `1 : 2 : 4` significa “todos os números entre 1 e 4 (incluindo 1), considerados segundo um passo 2”. Isso implica que a coluna 1 e a coluna 3 (=1+2) são consideradas. A próxima coluna que seria considerada, se existisse, seria a coluna 5 (=3+2). O resultado obtido com `A(2:1:3, 1:2:4)` equivale à intersecção dessas duas situações, ou seja

$$A(2 : 1 : 3, 1 : 2 : 4) = \begin{bmatrix} A(2,1) & A(2,3) \\ A(3,1) & A(3,3) \end{bmatrix}.$$

De fato, usar `2:3` ou `2:1:3` são formas equivalentes, já para o Matlab/Octave o passo 1 já fica implícito. Uma outra possibilidade é usar um passo negativo, por exemplo, `4 : -2 : 1`. Nesse caso, os números são acessados de forma decrescente e apenas os elementos 4 e 2 são usados, e nessa ordem. Por exemplo, na matriz **A**, fazer `A(2:3,4:-2:1)` equivale a pegar

$$A(2 : 3, 4 : -2 : 1) = \begin{bmatrix} A(2,4) & A(2,2) \\ A(3,4) & A(3,2) \end{bmatrix}.$$

Exemplo:

```
>> A(2:3, 4:-2:1)
```

```
ans =
```

```
    1    3  
    7    9
```

Essas formas para acessar apenas alguns elementos de uma matriz são extremamente úteis em diversas situações, pois reduzem o número de passagens para a obtenção dos valores desejados. Essas técnicas também são aplicáveis em vetores, com a diferença de que um dos números do argumento da variável do vetor passa a ser igual a 1 ou então é simplesmente ignorado.

Ex. :

```
>> c = [2 7 9 13 7 34]
```

```
c =
```

```
    2    7    9   13    7   34
```

```
>> c(1,3:1:5)
```

```
ans =
```

```
    9   13    7
```

```
>> c(3:1:5)
```

```
ans =
```

```
    9   13    7
```

#### 4.4 Transposição de vetores e matrizes

A transposição de matrizes e vetores é uma necessidade recorrente em cálculo matricial. Em Octave/Matlab, a transposição é indicada por um apóstrofe (') colocado após a vetor/matriz que se deseja transpor. Exs.:

```
>> B = [2 3; 8 1]
```

```
B =
```

```
     2     3
     8     1
```

```
>> B'
```

```
ans =
```

```
     2     8
     3     1
```

```
>> d = [1 1 2 3 5 8 13]
```

```
d =
```

```
     1     1     2     3     5
```

```
>> d'
```

```
ans =
```

```
     1
     1
     2
     3
     5
```

Nota: Se o vetor/matriz for composto por elementos complexos, a transposição fornece seu Hermitiano (transposto do complexo-conjugado). Ex.:

```
>> H = [i 1-4i 2i; 4+3i -9i 1-i]
```

```
H =
```

```
     0 + 1.0000i   1.0000 - 4.0000i   0 + 2.0000i
 4.0000 + 3.0000i         0 - 9.0000i   1.0000 - 1.0000i
```

```
>> H'

ans =

    0 - 1.0000i    4.0000 - 3.0000i
    1.0000 + 4.0000i         0 + 9.0000i
    0 - 2.0000i    1.0000 + 1.0000i
```

## 4.5 Solução de sistemas de equações lineares

Para resolver um sistema linear do tipo  $Ax = b$  em Matlab/Octave, usa-se a barra invertida (`\`), fazendo `A\b`. Ex.:

```
>> A

A =

    1    2
    4   -3
```

```
>> b
```

```
b =

    1
    0
```

```
>> A\b
```

```
ans =

    0.2727
    0.3636
```

## 4.6 Cálculo de determinante

Para calcular o determinante de uma matriz quadrada **A**, basta usar a função `det(A)`. Se a matriz não for quadrada (ou seja, o número de linhas não for igual ao número de colunas), o fluxo do programa é interrompido e é indicado erro na janela de comando. Ex.:

```
>> A = [1 1 2; 3 5 8; 13 21 33]
```

```
A =
```

```
     1     1     2
     3     5     8
    13    21    33
```

```
>> det(A)
```

```
ans =
```

```
-2.0000
```

#### 4.7 Construção de matrizes e vetores a partir de outras matrizes e vetores

Um recurso muito útil para a manipulação de matrizes e vetores é a possibilidade de concatená-los para gerar novos elementos. Nesse caso, basta colocar entre colchetes os vetores e matrizes, atentando para que exista coerência entre o número de linhas e colunas da matriz/vetor gerado após a concatenação. No exemplo seguinte, deseja-se concatenar os vetores  $c$  ( $1 \times 4$ ),  $d$  ( $3 \times 1$ ) e a matriz  $A$  ( $3 \times 3$ ) para obter uma matriz  $B$  ( $4 \times 4$ ) da forma

$$B = \begin{bmatrix} & c & & \\ A & & d & \end{bmatrix}.$$

Para isso, basta fazer  $B = [c; A \ d]$ , como apresentado no exemplo.

```
>> c = [1 2 3 4]
```

```
c =
```

```
     1     2     3     4
```

```
>> d = [10; 20; 30]
```

```
d =
```

```
    10
    20
    30
```

30

```
>> A = [100 200 300; 400 500 600; 700 800 900]
```

```
A =
```

```
    100    200    300
    400    500    600
    700    800    900
```

```
>> D = [c; A d]
```

```
D =
```

```
     1     2     3     4
    100    200    300    10
    400    500    600    20
    700    800    900    30
```

Para criar vetores através da concatenação de vetores menores, o procedimento é semelhante.

## 4.8 Multiplicação por escalar

A multiplicação de um vetor ou de uma matriz por um escalar simplesmente implica na multiplicação de cada um dos elementos por um escalar. Ex.:

```
>> a
```

```
a =
```

```
     1     2     3
```

```
>> 3*a
```

```
ans =
```

```
     3     6     9
```

## 4.9 Operações entre matrizes e vetores

As operações básicas entre matrizes e vetores fazem uso dos mesmos operadores básicos apresentado na Tabela 1, +, -, \*, e  $\wedge$ . Vale lembrar que a soma e a subtração de matrizes ou vetores só é definida para entes com o mesmo número de linhas e colunas. A multiplicação, por sua vez, só é definida quando o primeiro termo da operação possui um número de colunas igual ao número de linhas do segundo termo. Ex.:

```
>> a = [1 2 3]

a =

     1     2     3

>> b = [4 -3 -1]

b =

     4    -3    -1

>> a+b

ans =

     5    -1     2

>> a-b

ans =

    -3     5     4

>> a*b' % Esta operação é o produto escalar

ans =

    -5
```

```
>> a'*b % Esta operação é o produto externo
```

```
ans =
```

```
    4    -3    -1  
    8    -6    -2  
   12    -9    -3
```

A potenciação é definida apenas para matrizes quadradas. Ex.:

```
>> A = [-1.5 2 3.3; 4 6.2 1; 0 1 0]
```

```
A =
```

```
-1.5000    2.0000    3.3000  
 4.0000    6.2000    1.0000  
 0         1.0000         0
```

```
>> A^2
```

```
ans =
```

```
10.2500    12.7000   -2.9500  
18.8000    47.4400    19.4000  
 4.0000     6.2000     1.0000
```

```
>> A^3
```

```
ans =
```

```
35.4250    96.2900    46.5250  
161.5600   351.1280   109.4800  
18.8000    47.4400    19.4000
```

## 4.10 Operações elemento-a-elemento em vetores e matrizes

Em Octave/Matlab, é possível manipular os elementos de vetores e matrizes termo-a-termo. Imagine, por exemplo que se possui duas matrizes  $A$  e  $B$  de dimensões  $N \times N$  e que se deseja multiplicar o elemento  $A(1, 1)$  pelo elemento

$B(1, 1)$ , o elemento  $A(1, 2)$  por  $B(1, 2)$  e assim por diante, de forma a obter a matriz  $C$ ,

$$C = \begin{bmatrix} A(1, 1) * B(1, 1) & A(1, 2) * B(1, 2) & \cdots & A(1, N) * B(1, N) \\ A(2, 1) * B(2, 1) & A(2, 2) * B(2, 2) & \cdots & A(2, N) * B(2, N) \\ \vdots & \vdots & \ddots & \vdots \\ A(N, 1) * B(N, 1) & A(N, 2) * B(N, 2) & \cdots & A(N, N) * B(N, N) \end{bmatrix}.$$

Com a multiplicação típica de matrizes, usando o operador  $*$ , não é possível obter  $C$ . Nesse caso, pode-se usar o operador  $.*$ , que realiza a operação de multiplicação termo-a-termo. Ex.:

```
>> A = [1 2 -3; 4 2 12]
```

```
A =
```

```
    1     2    -3
    4     2    12
```

```
>> B = [ 3 1 7; -9 0 8]
```

```
B =
```

```
    3     1     7
   -9     0     8
```

```
>> C = A.*B
```

```
C =
```

```
    3     2   -21
  -36     0    96
```

De modo geral, colocando o ponto antes do operador (por exemplo,  $.*$ ,  $./$  e  $.\$  sendo que neste último caso,  $a.\backslash b$  resulta nos elementos de  $b$  divididos termo a termo pelos elementos de  $a$ ), obtém-se uma operação elemento-a-elemento entre matrizes.

Outras funções (como seno, cosseno, módulo, etc), se usadas em uma matriz  $A$  ( $M \times N$ ), fornecem como resultado uma matriz em que cada elemento corresponde ao resultado da função no respectivo elemento da matriz

original, ou seja,

$$f(A) = \begin{bmatrix} f(A(1,1)) & \dots & f(A(1,N)) \\ \vdots & \ddots & \vdots \\ f(A(M,N)) & \dots & f(A(M,N)) \end{bmatrix},$$

onde  $f(\cdot)$  pode ser um seno, um cosseno ou qualquer outra função.

#### 4.11 Produto escalar

O produto escalar de dois vetores pode ser realizado de duas maneiras: por meio da função `dot(a,b)`, que possui como argumentos os vetores que constituem o produto escalar, ou por meio de uma multiplicação vetorial. Ex.:

```
>> a = [1 2 3]

a =

     1     2     3

>> b = [5 -1 3]

b =

     5    -1     3

>> dot(a,b)

ans =

    12

>> a*b'

ans =

    12
```

#### 4.12 Descobrimo as dimensões de vetores e matrizes

Quando se manipula vetores e matrizes, muitas vezes é necessário usar ou checar suas dimensões. Nesse caso, a função `size(\cdot)` pode ser bem útil, já

que ela fornece o número de linhas e de colunas de matrizes e vetores. Exs.:

```
>> A
```

```
A =
```

```
    1    2   -3
    4    2   12
```

```
>> [nlinhas ncolunas] = size(A)
```

```
nlinhas =
```

```
    2
```

```
ncolunas =
```

```
    3
```

Uma outra possibilidade é a função `length(·)`, usada somente para vetores, que fornece o número de elementos. Ex.:

```
>> b
```

```
b =
```

```
    5   -1    3
```

```
>> nelementos = length(b)
```

```
nelementos =
```

```
    3
```

## 5 Matrizes especiais

Nesta seção, algumas funções usadas na geração de matrizes especiais são apresentadas. Essas funções são extremamente úteis para definir matrizes sem que haja a necessidade de introduzir elemento por elemento. As funções e suas características são apresentadas na Tabela 3.

Tabela 3: Funções que definem matrizes com estruturas especiais

Comando	Função desempenhada
<code>A = ones(m, n)</code>	A é definida como uma matriz de dimensão $m \times n$ , em que todos os elementos são iguais a 1.
<code>A = zeros(m, n)</code>	A é definida como uma matriz de dimensão $m \times n$ , em que todos os elementos são iguais a 0.
<code>A = eye(m, n)</code>	A é definida como uma matriz de dimensão $m \times n$ , em que todos os elementos da diagonal principal são iguais a 1 e os demais são iguais a 0. No caso $m = n$ , temos a matriz identidade de dimensão $n$ ( $I$ ), daí o nome ("eye" é como se pronuncia "I" em inglês).
<code>A = rand(m, n)</code>	A é definida como uma matriz de dimensão $m \times n$ , em que os elementos são aleatoriamente distribuídos no intervalo $(0, 1)$ , de maneira uniforme.
<code>A = randn(m, n)</code>	A é definida como uma matriz de dimensão $m \times n$ , em que os elementos são distribuídos segundo uma distribuição normal padrão.

Exemplos:

```
>> A = eye(4,3)
```

```
A =
```

```
1    0    0
0    1    0
0    0    1
0    0    0
```

```
>> A = zeros(2,3)
```

```
A =
```

```
    0    0    0
    0    0    0
```

```
>> A = rand(3,3)
```

```
A =
```

```
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

## 6 Mais alguns recursos úteis para programação

Nesta seção, são detalhadas alguns recursos e funções que podem facilitar o desenvolvimento de códigos em linguagem Octave/Matlab.

### 6.1 Ponto-e-vírgula depois de comandos

Quando não se deseja que o Matlab/Octave imprima na tela os valores de uma variável ou de uma função, usa-se ponto-e-vírgula (;) depois do comando. Este recurso é muito útil, principalmente quando se trabalha com *scripts* (vide Seção 10) e funções criadas pelo usuário (vide seção 11). Nesses casos, reduz-se o tempo de processamento, já que o fluxo de execução de programa não é interrompido para a apresentação dos valores das variáveis na tela.

### 6.2 A variável ans

Sempre que alguma operação é realizada e o resultado não é atribuído à nenhuma variável, o Matlab atribui o resultado da operação para a variável **ans**. Vale lembrar que **ans** guarda somente o resultado da última operação realizada e que não foi atribuída a nenhuma variável. Ex.:

```
>> b = [6 7.4 2];  
>> a = [2 4 6];  
>> b*a'
```

```
ans =
```

```
53.6000
```

```
>> sqrt(ans)
```

```
ans =
```

```
7.3212
```

### 6.3 Comentários no programa (%)

Para acrescentar comentários em *scripts* (vide Seção 10) e funções (vide seção 11), usa-se o símbolo de porcentagem (%), colocado antes do texto que se deseja comentar. Esse recurso funciona tanto no Matlab quanto no Octave. Além desse símbolo, no Octave também é possível usar o símbolo sustenido (#) para introduzir comentários, o que não funciona no Matlab.

## 6.4 Encontrando as raízes de polinômios

A função `roots(a)` recebe como argumento um vetor contendo os coeficientes de um polinômio de grau qualquer e retorna os valores de suas raízes. Nesse caso, os coeficientes devem ser escritos em ordem decrescente de grau. Como exemplo, assuma que seja necessário encontrar as raízes do polinômio  $x^2 - 5x + 6$ . Escrevendo o vetor de coeficientes `c=[1 -5 6]` e usando a função `roots`, obtém-se

```
>> roots(c)

ans =

    3.0000
    2.0000
```

## 6.5 Obtendo o polinômio a partir de suas raízes

De maneira complementar à função `roots`, a função `poly(b)` encontra os coeficientes de um polinômio a partir de suas raízes, escritas como um vetor `b`. Ex.:

```
>> b = [3 2]

b =

    3    2

>> poly(b)

ans =

    1   -5    6
```

## 6.6 `help`: a função mais importante e mais útil do Matlab/Octave

A função `help` certamente é o recurso mais útil e mais usado quando se faz uso do Matlab ou do Octave. Por meio dessa função, é possível descobrir e entender quais são os argumentos e quais são as variáveis de retorno de funções pré-definidas. Para acessar as informações relacionadas a uma função, basta digitar na janela de comandos a palavra `help` seguida pelo nome da função

desejada. Como consequência, toda a informação disponível sobre a função solicitada é apresentada na tela.

Como exemplo, suponha que um usuário de Matlab deseja consultar informações sobre a função `abs`. Digitando `help abs` na janela de comando, o seguinte texto é apresentado na tela:

```
>> help abs
ABS      Absolute value.
        ABS(X) is the absolute value of the elements of X. When
        X is complex, ABS(X) is the complex modulus (magnitude) of
        the elements of X.

        See also sign, angle, unwrap, hypot.

        Overloaded methods:
            frd/abs
            codistributed/abs
            iddata/abs

        Reference page in Help browser
            doc abs
```

Dessa forma, descobre-se que a função `abs` fornece o valor absoluto dos elementos do argumento `X`, se os elementos forem números reais, e a magnitude dos elementos de `X`, se eles forem números complexos. Além dessas informações, são apresentados referências para outras funções que possuem alguma relação com `abs`.

Uma outra maneira de usar `help` é digitá-lo diretamente na janela de comando e pressionar *enter*. Nesse caso, uma lista de temas é apresentada e a partir de outro comando `help` pode-se chegar a listas de comandos. Um exemplo disso é o comando `help elfun`, que lista funções elementares.

## 6.7 “Limpendo” os comando da janela de comando

Após diversas operações na janela de comando, as operações e resultados se acumulam de forma “empilhada” na tela, o que pode se tornar incômodo para o usuário. Nessa situação, pode-se usar o comando `clc`, que “limpa” a tela, apagando todos os registros visuais passados. Contudo, vale lembrar que as variáveis definidas até então não são apagadas da memória.

## 6.8 Listando as variáveis usadas: `who` e `whos`

Quando se usa a janela de comando do Matlab/Octave, não fica visível para o usuário quais as variáveis já foram usadas e estão armazenadas na memória. Para ter acesso a uma lista das variáveis, podem ser usados dois diferentes comandos: `who` e `whos`.

Digitando `who` na linha de comando, todas as variáveis armazenadas são apresentadas na tela. O comando `whos` funciona de forma semelhante, mas também disponibiliza outras informações sobre as variáveis, como as dimensões (revelando se a variável é um escalar, vetor ou matriz, por exemplo), o número de *bytes* ocupados para o armazenamento e se a variável é armazenada como um *double* ou um *float*. Exemplo:

```
>> a

a =

     1     2     3

>> b

b =

     1     0     0
     0     1     0
     0     0     1

>> who

Your variables are:

a b

>> whos

  Name      Size      Bytes  Class  Attributes

  a         1x3         24  double

  b         3x3         72  double
```

## 6.9 Apagando variáveis da memória

Para apagar uma variável da memória, basta digitar na janela de comando `clear`, seguido da variável que será excluída. Se apenas `clear` for digitado, todas as variáveis serão apagadas. Além dessas, existem outras opções para o comando `clear`, que podem ser consultadas com o auxílio do comando `help`.

Por exemplo, suponha que o usuário estava usando a variável `i` e que em seu conteúdo exista o valor 5. Usando `clear`, a variável `i` retorna para sua condição de unidade imaginária ( $i = \sqrt{-1}$ ). Ex.:

```
>> i

i =

    5

>> clear i
>> i

ans =

    0 + 1.0000i
```

## 7 Funções usadas em estatística

O Matlab/Octave possui diversas funções ligadas à manipulação de dados estatísticos. Nesta seção deseja-se apresentar as funções mais básicas disponíveis, apresentando alguns exemplos.

### 7.1 Cálculo de máximo e de mínimo

As funções `max` e `min`, quando usadas em um vetor, retornam o valor máximo e mínimo encontrados, respectivamente. Quando usada em uma matriz, a função `max` (`min`) fornece um vetor cujos elementos correspondem ao valor máximo (mínimo) de cada coluna da matriz. Existem outras sintaxes possíveis, que podem ser compreendidas por meio do comando `help max` (`min`). Exs.:

```
>> C

C =

     3     2    -21
    -36     0     96

>> max(C)

ans =

     3     2     96

>> max(ans)

ans =

     96
```

### 7.2 Cálculo de média

Para calcular a média de um vetor, basta usar função `mean`, tendo como argumento o vetor que se deseja avaliar. Se for aplicada em uma matriz, semelhante ao apresentado para as funções `max` e `min`, a função retorna um

vetor em que cada elemento corresponde à média dos elementos de cada coluna da matriz. Ex. :

```
>> b = [1 3 5 -7 9 0 11]
```

```
b =
```

```
1     3     5    -7     9     0    11
```

```
>> mean(b)
```

```
ans =
```

```
3.1429
```

### 7.3 Cálculo de desvio padrão e de variância

Para calcular o desvio padrão e a variância de um vetor de dados, basta usar as funções `std` e `var`. Da forma como são apresentados nos exemplos seguintes, essas funções seguem as equações

$$std(x) = \left( \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{1/2}$$

e

$$var(x) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2,$$

onde  $\bar{x}$  é a média dos  $n$  elementos de  $x$ . Existem mais possibilidades que podem ser verificadas com o auxílio de `help`. Exemplos:

```
>> b = [1     3     5     -7     9     0     11];  
>> std(b)
```

```
ans =
```

```
6.0119
```

```
>> var(b)
```

```
ans =
```

```
36.1429
```

## 8 Gráficos

Nesta seção são apresentadas funções usadas para a geração de gráficos via Matlab/Octave. Apenas são apresentados os recursos mais básicos, com exemplos bem simplificados.

### 8.1 Função plot

A função `plot` é a mais comumente usada para criar figuras 2D em Matlab/Octave. Em sua forma mais simples de uso, ela recebe como argumentos um vetor de dados correspondente aos valores da abcissa e um vetor de dados com as informações do eixo das ordenadas. Exemplo.:

```
>> a = 0:0.5:6;  
>> b = sin(a);  
>> c = 1:1:length(a);  
>> plot(c, b)
```

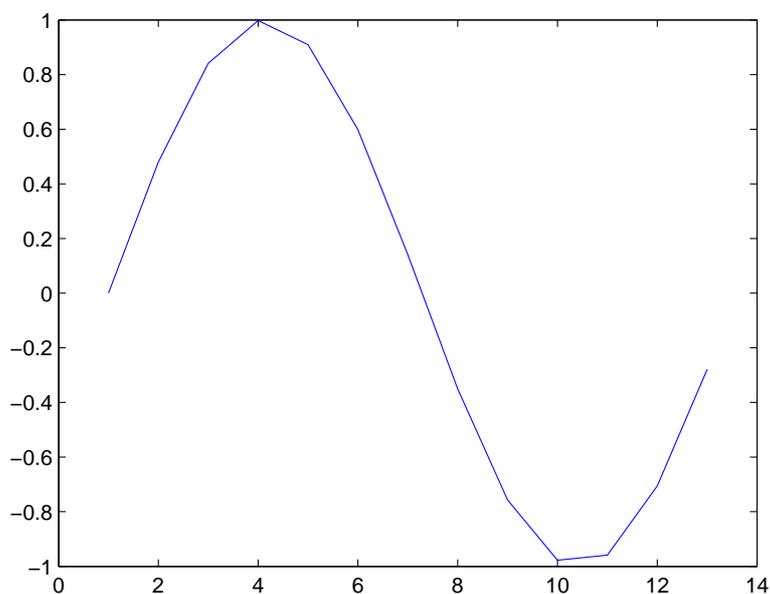


Figura 1: Figura obtida para o exemplo usando comando `plot`

De fato, se não for informado o vetor do eixo das abcissas, a função `plot` ainda realiza a plotagem do gráfico, assumindo que os valores do eixo x começam em 1 e estão espaçados de uma unidade entre si.

A função `plot` também pode ser usada para plotar mais do que uma figura. Nesse caso, é necessário informar (nessa ordem), os dados da primeira abscissa `x`, da primeira ordenada `y`, da outra abscissa `x` e do segundo `y`.

Exemplo:

```
>> x = 0:20;  
>> y1 = sin(2*pi*x/20);  
>> y2 = cos(2*pi*x/20);  
>> plot(x, y1, x, y2)
```

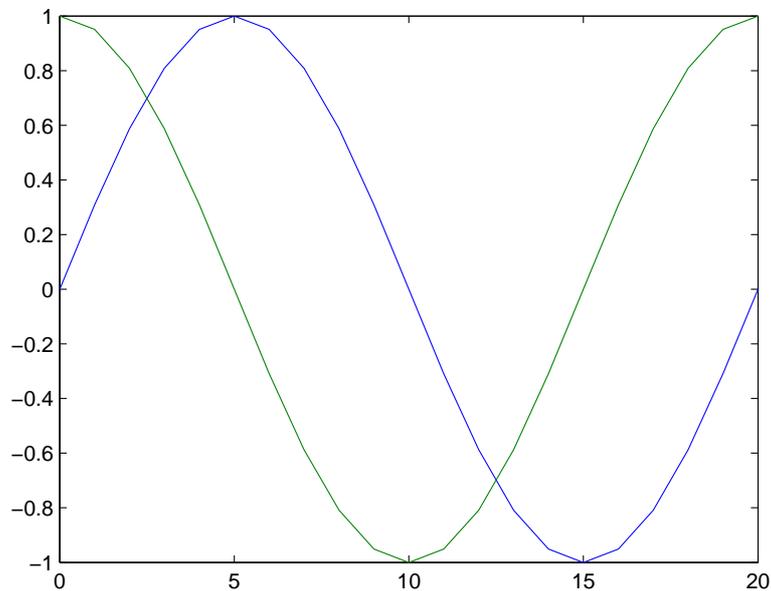


Figura 2: Figura obtida para o exemplo usando comando `plot` para plotar duas curvas diferentes

A função `plot` possui diversos termos opcionais, que permitem a escolha da espessura da linha, a cor, o uso de marcadores e outros recursos que podem ser explorados com o auxílio de `help plot`. Nesta breve introdução, iremos explorar apenas opções de mudança de cor da linha, de espessura e o uso de marcadores. Para essas propriedades, a sintaxe de `plot` corresponde a

```
plot(abcissa, ordenada, 'pont-cor-marc', 'LineWidth', largura-linha)
```

em que:

1. `abcissa` representa o vetor com os dados da abcissa
2. `ordenada` representa o vetor com os dados da ordenada
3. `'pont-cor-marc'` detalha a forma como devem ser colocadas as informações do formato da linha (tracejada (- -), contínua (-), pontilhada (:), etc), a cor (vermelho (r), preto (k), amarelo (y), azul (b), verde (g), etc) e o tipo de marcador (quadrado (s), circular (o), asterisco (\*), triângulo para cima (^), triângulo para baixo (v), etc.). Essas informações devem ser **colocadas juntas, entre apóstrofes e sem hífen**.
4. `'LineWidth'`, `largura-linha` é o comando para determinar a espessura da linha. A primeira parte (`'LineWidth'`) deve ser sempre mantida e `largura-linha` deve ser substituída por um valor numérico.

Exemplo:

```
>> x = 1:13;
>> y = sin(2*pi*x/13);
>> plot(x,y,':bs','LineWidth',3)
```

De forma semelhante, usando o comando

```
>> plot(x,y,'-g^','LineWidth',3)}
```

a cor é alterada para verde, a linha passa a ser contínua e os marcadores são substituídos por triângulos (vide Figura 4). Vale lembrar que, no Matlab, a figura também pode ser editada posteriormente para modificar todos esses atributos.

## 8.2 Colocando título, legenda e informações nos eixos de figuras

Para colocar título em gráficos, basta usar o comando

```
>> title('Aqui vai o título')
```

Com isso, o texto entre os apóstrofes é colocado sobre a figura.

A legenda também costuma ser usada durante a plotagem de diversas curvas em uma mesma figura, com o intuito de evitar ambiguidades na identificação. Para adicionar legenda a uma figura, basta usar

```
>> legend('legenda da primeira curva', 'legenda da segunda curva')
```

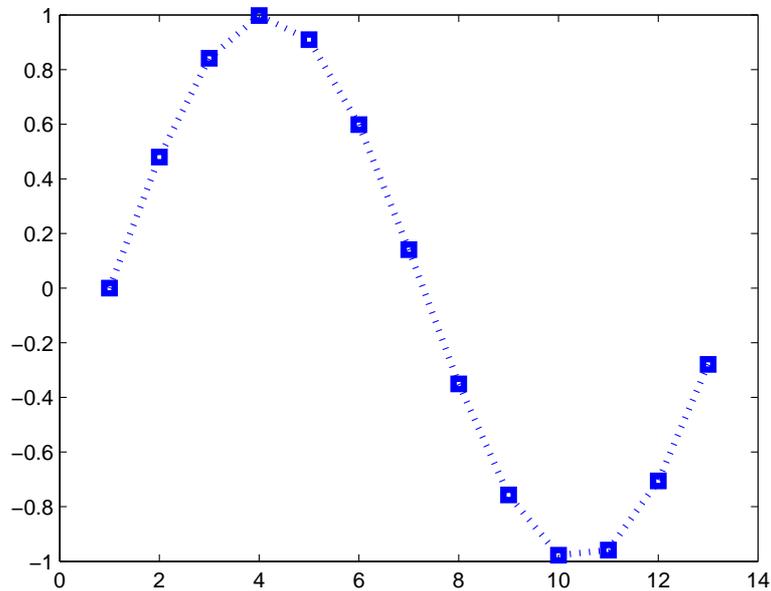


Figura 3: Figura obtida para o exemplo usando comando `plot` com os opcionais linha pontilhada e azul, de espessura 3 e com marcadores quadrados

Dessa forma, a legenda é acrescentada como uma caixa de texto. Também é possível usar  $\alpha$ ,  $\beta$ ,  $\pi$ , por exemplo, para usar letras gregas em legendas e eixos.

Um outro recurso típico em gráficos é o uso de texto para identificação das grandezas nos eixos. Para acrescentar informações aos eixos x e y de uma figura, basta usar os comandos

```
>> xlabel('Aqui vai o texto do eixo das abcissas')
>> ylabel('Aqui vai o texto do eixo das ordenadas')
```

O exemplo seguinte apresenta a aplicação desses conceitos.

```
>> y1 = sin(0:pi/10:2*pi);
>> y2 = cos(0:pi/10:2*pi);
>> x = 0:20;
>> plot(x, y1, 'r', x, y2, 'b')
>> title('Seno e cosseno')
>> legend('Seno', 'Cosseno')
>> xlabel('Tempo')
>> ylabel('Amplitude')
```

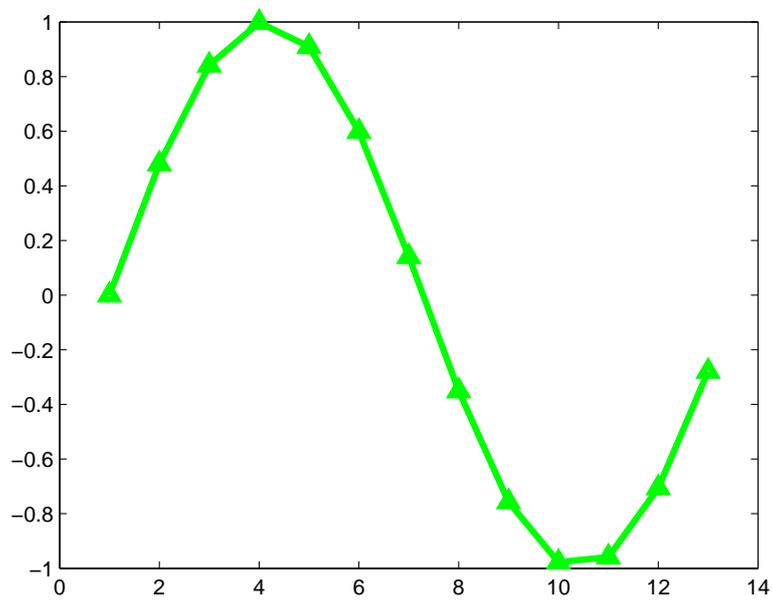


Figura 4: Figura obtida para o exemplo usando comando `plot` com os opcionais linha contínua e verde, de espessura 3 e com marcadores triangulares

Assim como outras funções, `title`, `legend` e `xlabel` apresentam alguns outros itens opcionais, como alteração do tamanho de letra e de posição na imagem. Para obter maiores informações sobre esses itens, pode-se consultar `help`.

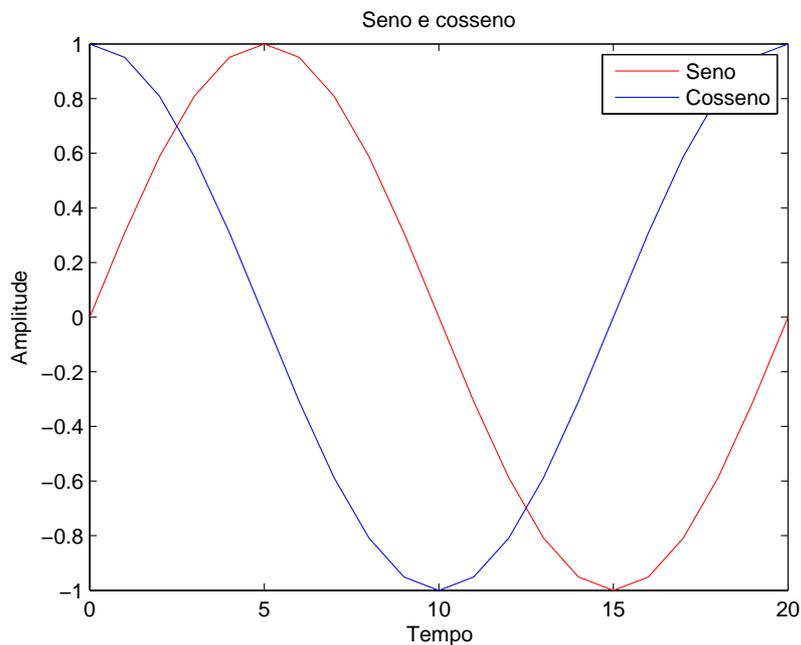


Figura 5: Figura mostrando título, eixos e legenda

### 8.3 Outras formas de fazer gráficos

Além de `plot`, existem muitas outras formas de criar figuras. Ainda pensando em 2D, alguns exemplos são:

1. `semilogx` e `semilogy`: criam figuras com um dos eixos em escala logarítmica.
2. `loglog`: os dois eixos são apresentados em escala logarítmica
3. `stem`: plota sequências de dados discretos
4. `hist`: cria um histograma

Além disso, ainda existe a possibilidade de criar figuras com várias sub-plotagens por janela, usando `subplot`. Exemplo:

```
>> x1 = 1:20;
>> x2 = 1:33;
>> y1 = sin(2*pi/20*x1);
>> y2 = cos(2*pi/20*x2);
>> subplot(2,1,1);
>> plot(x1, y1, 'r');
```

```
>> subplot(2,1,2);  
>> plot(x2, y2, 'b');
```

Nesse caso, o comando `subplot` foi usado para inserir 2 plotagens diferentes. A sintaxe é simples: o primeiro número indica o número de “linhas” de figuras que se deseja plotar (2, neste exemplo), enquanto que o segundo número indica o número de “colunas” (1, no exemplo). O terceiro número indica qual a figura está sendo plotada. Seguindo números de 1 até N, onde N é o número de figuras que serão plotadas, as imagens são plotadas de forma a completar uma linha por vez, começando pela linha superior no extremo esquerdo.

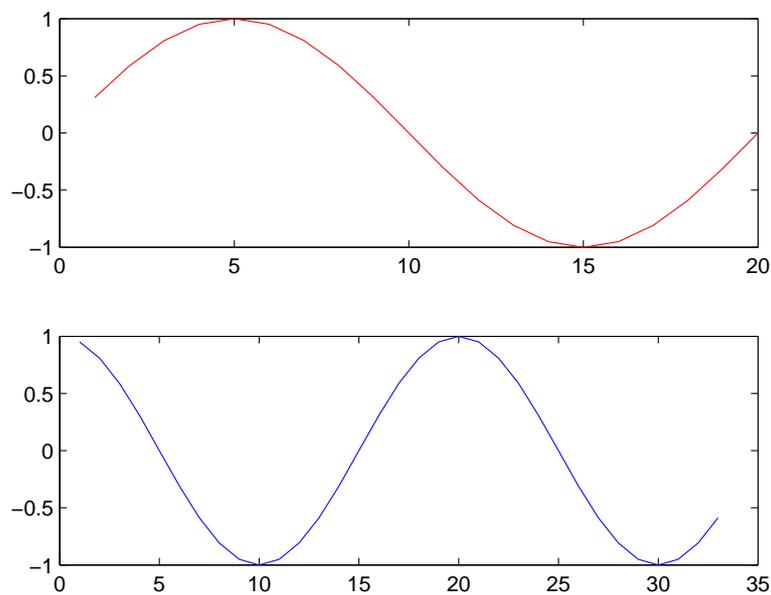


Figura 6: subplot com duas figuras

## 9 Estruturas de controle de fluxo de programa

Durante a execução de um programa, frequentemente é necessário realizar operações repetitivas ou percorrer elementos de vetores e matrizes, alterando seus valores com base em algum tipo de regra, que precisa ser checada em algum momento. Nesta seção, algumas das estruturas que desempenham essas funções são apresentadas e exemplificadas.

### 9.1 *Loops*: for e while

Em Matlab/Octave, as estruturas básicas para se implementar um *loop* são `for` e `while`. A sintaxe básica de cada uma delas é dada por

```
for i = 1:N
    ...
    Aqui vão as operações
    ...
end
```

e

```
while condicao_satisfeita
    ...
    Aqui vão as operações
    ...
end
```

Para a função `for`, o *loop* é mantido enquanto a variável `i` não assumir o valor de `N`. Por outro lado, a função `while` mantém o *loop* enquanto a condição de saída não é satisfeita.

Nota: Embora a sintaxe de `for` tenha sido apresentada com `i = 1:N`, os valores de início, de passo e de final da contagem podem ser quaisquer, desde que façam sentido (ou seja, `i=1:2:-20` não faz sentido, já que não é possível atingir `-20` somando `2` a cada iteração em `i`).

#### 9.1.1 Elementos relacionais

Para implementar condições de saída do *loop* na função `while`, usam-se elementos relacionais (o que também é usado em funções condicionais usando `if` – vide 9.2). A Tabela 4 a seguir apresenta alguns desses elementos e suas funções.

Tabela 4: Exemplos de operações básicas em escalares

Comando	Função desempenhada
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
==	Testa se as variáveis têm conteúdos iguais
~=	Testa se as variáveis são diferentes
&&	Operação E lógica
	Operação OU lógico

## 9.2 Usando estruturas condicionais do tipo `if`, `elseif` e `else`

As estruturas `if`, `elseif` e `else` são usadas para testar condições pré-definidas e executar apenas determinados comandos, caso as condições sejam cumpridas. A sintaxe básica desse tipo de estrutura corresponde a

```

if condicao1
    ...
    Aqui vão os comandos
    ...
elseif condicao2
    ...
    Aqui vão os comandos
    ...
else
    ...
    Aqui vão os comandos
    ...
end

```

Nesse caso, a `condicao1` do comando `if` é testada e, se for atendida, os comandos que ele guarda são realizados e os comandos seguintes são ignorados. Se a `condicao1` for falsa, a `condicao2` do `elseif` é testada. Se `condicao2` for verdadeira, os comandos dentro do `elseif` são realizados. Caso `condicao2` seja falsa, o programa executa as instruções contidas no `else`. Esse tipo de estrutura pode conter tantos `elseif` quantos forem necessários.

## 10 Criando *scripts* e funções

Quando é necessário fazer muitas operações que apresentam dependência da ordem com que devem ser realizadas, digitar as instruções na janela de comando pode se tornar uma tarefa bastante complicada. Essa é uma tarefa que está sujeita a erros, o que pode implicar na necessidade de digitar todos os comandos anteriores à alguma falha novamente. Além disso, se for necessário usar o mesmo conjunto de comandos em outra ocasião, todas as instruções precisam ser digitadas novamente, já que não existe um arquivo armazenando essa sequência de comandos.

Para evitar esses problemas, pode-se criar um *script*. Um *script* nada mais é do que um conjunto de comandos que segue uma ordem pré-especificada e que pode ser armazenado como um arquivo, para ser reutilizado depois. Quando o *script* é executado, os comandos são executados de forma ordenada e as variáveis ficam disponíveis na memória do computador, tal como se os comandos tivessem sido digitados um a um na linha de comando. Em Matlab/Octave, esses arquivos são salvos em *m-files*, que são arquivos com extensão *.m*. Para poder executar esses comandos, basta navegar pela janela de comando até a pasta em que está o *script* desejado, digitar o nome do arquivo e pressionar *Enter*. Uma outra opção é colocar na janela de comando todo o caminho até o local onde está armazenado o *script*, ou colocar o diretório onde está o comando no *path*.

Como exemplo, um pequeno *script* usado para calcular a soma de senos de diferentes frequências, é apresentado a seguir.

```
% script que calcula a soma de alguns senos defasados
% e plota a figura

x = 0:pi/10:6*pi; % calcula as amostras usadas para gerar os senos

% seno de frequência 2 rad/seg e defasagem nula
sen1 = 3*sin(2*x);

% seno de frequência 5 rad/seg e defasagem pi/2
sen2 = sin(5*x + pi/2);

% seno de frequência 1 rad/seg e defasagem pi/11
sen3 = 5*sin(x + pi/11);

soma_de_senos = sen1 + sen2 + sen3; % soma dos senos
```

```
% A plotagem é implementada a seguir
plot(x, soma_de_senos, '--k', 'LineWidth', 3);
title('Soma de senos');
xlabel('Tempo');
ylabel('Amplitude');
```

Como resultado desse *script*, obtém-se a Figura 7.

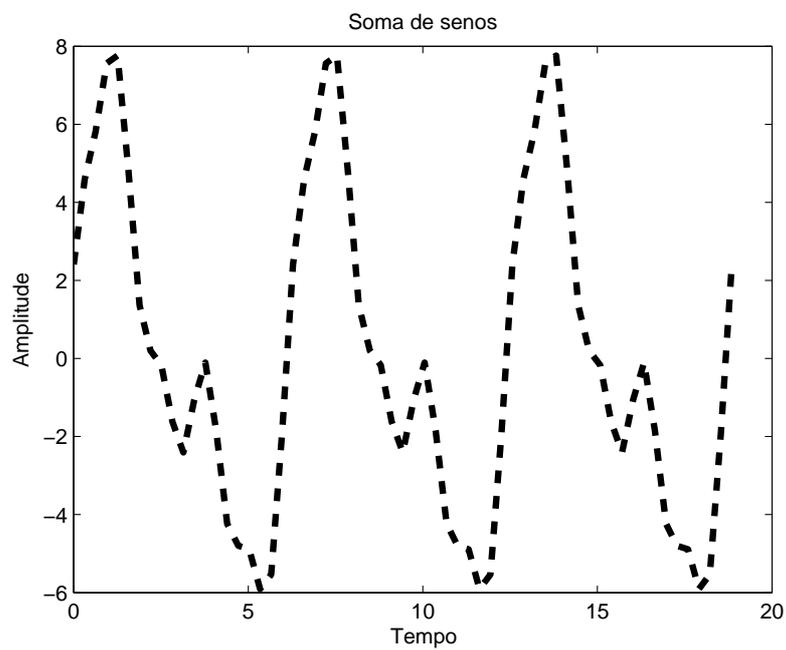


Figura 7: Figura obtida com o *script* do exemplo

## 11 Criando funções

Frequentemente é necessário criar funções específicas para aplicar em algum código do usuário. Em geral, funções são definidas em termos dos parâmetros que recebem e dos parâmetros que retornam, após processar os parâmetros iniciais. Algumas funções podem não possuir parâmetros de entrada ou de saída (e às vezes ambos...). Qualquer que seja o tipo, podem ser chamadas por *scripts* ou diretamente da janela de comando. Também é comum funções mais complexas conterem outras funções em seu conjunto de instruções. A grande vantagem de usar funções é a redução de partes repetitivas no código, o que reduz a possibilidade de erros. Além disso, funções podem ser aproveitadas em outros códigos, reduzindo o trabalho do programador.

Em linguagem Octave/Matlab, a sintaxe de uma função é dada por

```
function [saida1,..., saidaM] = nome(entrada1, ..., entradaN)
```

Na definição da função, *entrada1, ..., entradaN* são os parâmetros de entrada, *saida1, ..., saidaM* são os parâmetros de saída e *nome* é o nome dado a função.

Funções são armazenadas em arquivos do tipo **.m**. **O arquivo de armazenamento e a função devem obrigatoriamente possuir o mesmo nome**, caso contrário, quando a função for chamada, ocorrerá um erro.

Para usar uma função na janela de comando ou em um *script*, podemos usar a sintaxe

```
[saida1,..., saidaM] = nome(entrada1, ..., entradaN)
```

ou

```
nome(entrada1, ..., entradaN)
```

No primeiro caso, a saída é armazenada nas variáveis *saida1, ..., saidaM*, enquanto que na segunda chamada, o resultado é apresentado diretamente na janela de comando.

O exemplo seguinte mostra uma função usada para calcular a primeira derivada de um polinômio de ordem qualquer. A entrada da função `deriva_pol` é um vetor com os coeficientes do polinômio (`coef`) e a saída são os coeficientes do polinômio derivado (`deriv`).

```

% Esta função calcula a derivada de primeira ordem de
% um polinômio de grau qualquer.

% A entrada da função (coef) é um vetor que contém
% os coeficientes multiplicativos do polinômio,
% apresentados da ordem mais elevada para a menor ordem.

% A saída (deriv) contém os coeficientes
% multiplicativos do polinômio derivado, da ordem mais
% elevada para a mais baixa.

function [deriv] = deriva_pol(coef)

ncoef = length(coef); % número de coeficientes do polinômio

if ncoef == 1
    deriv = 0;
else
    for i = 1:ncoef - 1
        deriv(i) = coef(i)*(ncoef - i);
    end
end
end

```

Os comentários colocados antes da função são geralmente explicações de como a função funciona. Se for usado o comando `help deriva_pol`, o texto desses comentários será apresentado na janela de comando.

## 12 Exemplos adicionais

Neste t3pico s3o apresentadas algumas fun33es desenvolvidas em linguagem Matlab.

### 12.1 Probabilidade de dar cara no lan33amento de uma moeda

Nessa pequena fun333o apresentada a seguir, o par33metro de entrada 3 x, que corresponde a um vetor de  $N$  elementos, composto por valores entre 0 e 1, sorteados aleatoriamente por meio da fun333o `x=rand(1,N)`. Para simular os resultados dos lan33amentos de uma moeda honesta, a fun333o separa os resultados em cara (se  $x < 0.5$ ) e coroa ( $x \geq 0.5$ ). O n3mero de vezes que sai cara 3 armazenado e depois dividido pelo numero de vezes que a moeda foi lan33ada ( $N$ ). O resultado 3 apresentado na vari33vel de s33ida `P`. Al3m disso, um histograma mostrando o n3mero de vezes que sai cara ou coroa tamb3m 3 apresentado na tela.

```
function [P] = conta_caras(x)
caras = 0;
coroas = 0;

for i = 1:length(x)
    if x(i) < 0.5
        x(i) = 0; % cara
    else
        x(i) = 1; % coroa
    end
end

hist(x);
title('Caras e coroas');

for i = 1: length(x)
    if x(i) == 0
        caras = caras + 1;
    else
        coroas = coroas + 1;
    end
end
```

```
P = caras/length(x);
```

## 12.2 Probabilidades das faces de um dado honesto

De forma semelhante ao apresentado no exemplo anterior, esse programa recebe um vetor de dados entre 0 e 1 (gerado de forma aleatória, por meio de `x=rand(1,N)`), e devolve em `resultado` as probabilidades de saída associadas a cada face. Para isso, são definidos seis intervalos:

1. Se  $x < \frac{1}{6}$ , a face de valor 1 é sorteada;
2. Se  $\frac{1}{6} < x \leq \frac{2}{6}$ , a face de valor 2 é sorteada;
3. Se  $\frac{2}{6} < x \leq \frac{3}{6}$ , a face de valor 3 é sorteada;
4. Se  $\frac{3}{6} < x \leq \frac{4}{6}$ , a face de valor 4 é sorteada;
5. Se  $\frac{4}{6} < x \leq \frac{5}{6}$ , a face de valor 5 é sorteada;
6. Se  $x \geq \frac{5}{6}$ , a face de valor 6 é sorteada.

Usando essa abordagem, as ocorrências de cada face são contadas e apresentadas em uma figura de barras.

```
function [resultado] = dado(x)

resultado = zeros(1, 6);

for i = 1:length(x)
    if x(i) < 1/6
        resultado(1) = resultado(1) + 1;
        x(i) = 1/6;
    elseif x(i) < 2/6
        resultado(2) = resultado(2) + 1;
        x(i) = 2/6;
    elseif x(i) < 3/6
        resultado(3) = resultado(3) + 1;
        x(i) = 3/6;
    elseif x(i) < 4/6
        resultado(4) = resultado(4) + 1;
        x(i) = 4/6;
    elseif x(i) < 5/6
        resultado(5) = resultado(5) + 1;
```

```

        x(i) = 5/6;
    else
        resultado(6) = resultado(6) + 1;
        x(i) = 1;
    end
end

figure
bar([1:6], resultado);
title('Número de ocorrência de cada face');

resultado = resultado./length(x);

```

### 12.3 Método dos Mínimos Quadrados

A função a seguir implementa o Método dos Mínimos Quadrados. A função recebe como parâmetros de entrada dois vetores  $x$  e  $y$ , de  $N$  elementos cada, que contém as coordenadas no eixo das abcissas e das ordenadas de diversos pontos, respectivamente. A saída da função é um vetor de dois elementos  $p$ , que contém o coeficiente angular e o ponto em que a reta de aproximação corta o eixo das ordenadas. Uma figura com os pontos dos vetores  $x$  e  $y$  e a reta obtida pelo Método dos Mínimos Quadrados também é apresentada.

```

function [p] = minimos_quadrados_for(x, y)

N = length(x);

media_x = 0;
media_y = 0;
media_xy = 0;
media_x_2 = 0;

for i = 1:N
    media_x = media_x + x(i);
    media_y = media_y + y(i);
    media_xy = media_xy + x(i)*y(i);
    media_x_2 = media_x_2 + x(i)^2;
end

media_x = media_x/N;
media_y = media_y/N;

```

```
media_xy = media_xy/N;
media_x_2 = media_x_2/N;

p(1) = (media_xy - media_x*media_y)/(media_x_2 - media_x^2);
p(2) = (media_y*media_x_2 - media_x*media_xy)/(media_x_2 - media_x^2);

figure
plot(x, y, '*r', x, p(1)*x + p(2))
```

## 13 Referências Adicionais

Existem diversos livros e tutoriais disponíveis na internet para Matlab e Octave, apresentando desde conceitos básicos – como os aqui abordados – até ferramentas de *toolboxes* (como as *toolboxes* de processamento de sinais, de redes neurais, de lógica *fuzzy* e de design de filtros). A seguir, alguns livros e sites são sugeridos e listados como fontes auxiliares para o aprendizado dessas ferramentas.

- Matlab Guide - Desmond J. Higham e Nicholas J. Higham - Second edition, Siam
- A Matlab Guide for Beginners and Experienced Users - Brian R. Hunt, Ronald L. Lipsman e Jontahan M. Rosenberg - Second edition, Cambridge
- Matlab for Engineers - Adrian Biran - 2. ed. England: Addison Wesley (Disponível na biblioteca da UACSA)
- <http://www.mathworks.com/>
- <http://www.gnu.org/software/octave/> (Para *download* e informações sobre o Octave)
- <http://www.castilho.prof.ufu.br/cn/Octave.pdf> (Mais uma apostila básica de Octave)