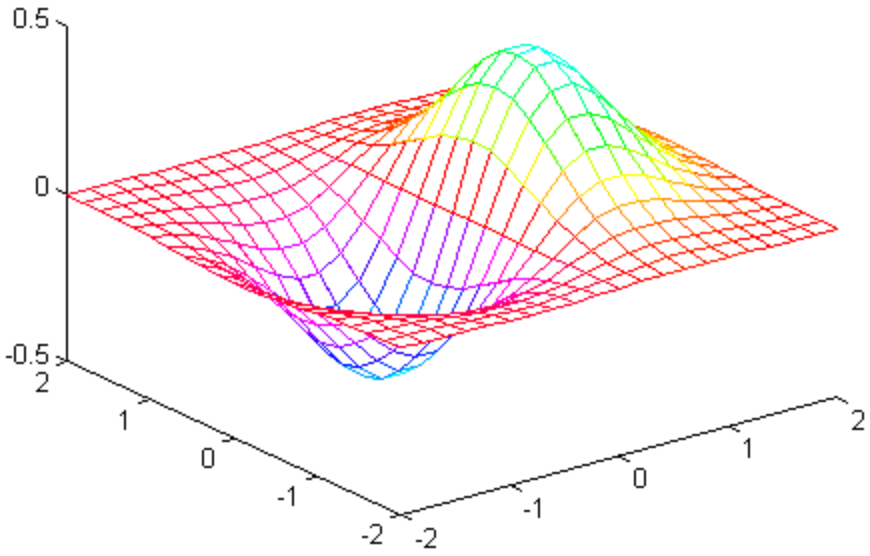
Centro Universitário de Belo Horizonte

Curso de Engenharia de Telecomunicações

Curso de Extensão

MATLAB para Engenharia



Adriana Maria Tonini

Daniela Naufel Schettino

1ª edição – Agosto de 2002

Belo Horizonte – Minas Gerais

Apostila de MATLAB

1) Introdução

Esta apostila surgiu da necessidade de capacitar rapidamente alunos de graduação a utilizarem o MATLAB como suporte computacional em estudos nas áreas da engenharia, e, principalmente para os alunos do Curso de Engenharia de Telecomunicações, cuja necessidade de conhecer e saber trabalhar com o MATLAB é extremamente importante nas disciplinas que compõem a grade curricular do curso.

O MATLAB é um "software" de alto desempenho destinado a fazer cálculos com matrizes (MATrix LABoratory), podendo funcionar como uma calculadora ou como uma linguagem de programação científica (FORTRAN, Pascal, C, etc.). Entretanto, os comandos do MATLAB são mais próximos da forma como escrevemos expressões algébricas, tornando mais simples o seu uso. Atualmente, o MATLAB é definido como um sistema interativo e uma linguagem de programação para computação técnica e científica em geral, integrando a capacidade de fazer cálculos, visualização gráfica e programação (Tonini e Couto, 1999).

- **Uso típico do MATLAB**:

- cálculos matemáticos;
- desenvolvimento de algoritmos;
- modelagem, simulação e confecção de protótipos;
- análise, simulação e confecção de dados;
- gráficos científicos e de engenharia;
- desenvolvimento de aplicações, incluindo a elaboração de interfaces gráficas com o usuário.

2) Cálculos Científicos

2.1- Área de Trabalho

- **Carregando o MATLAB**

No gerenciador de programas do Windows, um duplo clique no ícone MATLAB carrega o aplicativo. Uma vez inicializado o MATLAB, aparecerá na tela uma janela de comandos e o "prompt" padrão (EDU>> ou >>) é exibido na tela. A partir deste ponto, o MATLAB espera um comando (instruções) do usuário. Todo comando deve ser finalizado teclando-se **Enter**.

- **Editor de Linhas de Comando**

As teclas com setas podem ser usadas para se encontrar comandos dados anteriormente, para execução novamente ou sua reedição. Por exemplo, suponha que você entre com

```
» sen(0)
```

Ao apertar a tecla **Enter** , o MATLAB responde com uma mensagem de erro:

```
??? Undefined function or variable sen.
```

Isto acontece porque para se determinar o seno de um ângulo é necessário digitar em inglês o comando `sin`. Ao invés de rescrever a linha inteira, simplesmente pressione a tecla "seta para cima". O comando errado retorna, e você pode, então, mover o cursor para trás usando a tecla "seta para esquerda" ou o ponto de inserção com o "mouse" ao lugar apropriado para inserir a letra `i`:

```
» sin(0)

ans =

     0
```

Note que o MATLAB chamou o resultado de `ans` (*answer*=resposta). Além das teclas com setas, pode-se usar outras teclas para reeditar a linha de comando. A seguir é dada uma breve descrição destas teclas:

↑	retorna a linha anterior
↓	retorna a linha posterior
←	move um espaço para a esquerda
→	move um espaço para a direita
Ctrl ←	move uma palavra para a esquerda
Ctrl →	move uma palavra para a direita
Home	move para o começo da linha
End	move para o final da linha
Del	apaga um caracter a direita
Backspace	apaga um caracter a esquerda

Tabela 1 – Teclas de edição

2.2- Operações Básicas e Expressões Lógicas

O MATLAB oferece as seguintes operações aritméticas básicas:

Operação	Símbolo	Exemplos
Adição, a+b	+	5 + 6
Subtração, a-b	-	19 - 4.7
Multiplicação, a.b	*	5.02 * 7.1
Divisão, a÷b	/ ou \	45/5 ou 5\45
Potência, a^b	^	3^4

Tabela 2 – Operações aritméticas

A ordem nas expressões segue a ordem matemática - potência, seguida da multiplicação e da divisão, que por sua vez são seguidas pelas operações de adição e subtração. Parêntesis podem ser usados para alterar esta ordem. Neste caso, os parêntesis mais internos são avaliados antes dos mais externos.

Uma expressão se diz lógica se os operadores são lógicos e os operandos são relações e/ou variáveis do tipo lógico. Os operadores relacionais realizam comparações entre valores do mesmo tipo. Os operadores relacionais utilizados pelo MATLAB são:

Operador Relacional	Descrição
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
==	igual a
~=	diferente de

Tabela 3 – Operações relacionais

Note que (=) é usado para atribuição de um valor a uma variável, enquanto que (==) é usado para comparação de igualdade. No MATLAB os operadores relacionais podem ser usados para comparar vetores de mesmo tamanho ou escalares. O resultado de uma relação ou de uma expressão lógica é verdadeiro ou falso; contudo, no MATLAB o resultado é numérico, sendo que 1 significa verdadeiro e 0 significa falso. Por exemplo:

```
 » 5>8
ans =
     0
 » 5==5
ans =
     1
```

Os operadores lógicos permitem a combinação ou negação das relações lógicas. Os operadores lógicos do MATLAB são:

Operador lógico	Descrição	Uso
&	E	Conjunção
|	ou	Disjunção
~	Não	Negação

Tabela 4 – Operações lógicas

2.3- Constantes e Variáveis

O MATLAB faz cálculos simples e científicos como uma calculadora. Para tal, os comandos devem ser digitados diretamente no *prompt* (>>) do MATLAB, já que este se trata de um software interativo. Por exemplo:

```
 >> 3*25 + 5*12
ans =
   135
```

Observe que no MATLAB a multiplicação tem precedência sobre a adição.

Uma constante numérica no MATLAB é formada por uma sequência de dígitos que pode estar ou não precedida de um sinal positivo (+) ou negativo (-) e pode conter um ponto decimal (.). Esta sequência pode terminar ou não por uma das letras `e`, `E`, `d` ou `D`, seguida de outra sequência de dígitos precedida ou não de um sinal de (+) ou de (-). Esta segunda sequência é a potência de 10 pela qual a primeira sequência deve ser multiplicada. Por exemplo,

```
» 1.23e-1
```

significa 0,123.

O formato em que uma constante numérica é mostrada no MATLAB segue, como opção *default*, os seguintes critérios: se um resultado é inteiro, o MATLAB mostra o número como inteiro; quando o resultado é real, o MATLAB mostra o número com 4 dígitos à direita do ponto decimal; se os dígitos do resultado estiverem fora desta faixa, o MATLAB mostra o resultado usando a notação científica. Este default pode, entretanto, ser modificado utilizando-se o **Numeric Format** do item **Options** na barra de menus. Usando-se a constante numérica (33,5), considere a tabela 5 a título de exemplo dos formatos numéricos do MATLAB:

Comando	Formato	Comentário
format short	33.5000	4 dígitos decimais (formato default)
format long	33.50000000000000	16 dígitos
format short e	3.3500e+001	5 dígitos mais expoente
format long e	3.350000000000000e+001	16 dígitos mais expoente
format hex	4040c00000000000	Hexadecimal
format bank	33.50	2 dígitos decimais
format +	+	positivo, negativo ou zero
format rat	67/2	Racional

Tabela 5 – Formatos Numéricos

Alternativamente, você pode usar **variáveis** para armazenar informação. Por exemplo:

```
 >> q1=3, p1=25, q2=5, p2=12
q1 =
     3
p1 =
    25
q2 =
     5
p2 =
    12
 >> total=q1*p1+q2*p2
total =
   135
```

Primeiro, criamos quatro variáveis, `q1`, `p1`, `q2` e `p2`, atribuindo a elas os seus valores respectivos. Observe que o sinal de igual (=) aqui significa atribuição. O que estiver à direita do sinal de igual é "colocado" na variável que estiver à esquerda. Finalmente, criamos uma variável chamada `total` que recebeu o total da compra.

Os nomes das variáveis devem consistir de uma única palavra, conforme as regras expressas na tabela 6:

Regras de construção das variáveis	Comentários/Exemplos
Variáveis com letras minúsculas e maiúsculas são diferentes, mesmo que consistam das mesmas letras.	`Total`, `total`, `TOTAL` e `ToTaL` são variáveis diferentes.
As variáveis podem consistir de até 19 caracteres	`Sdtf65erkjh3448bafg`

As variáveis devem começar com uma letra e pode ser seguida de letras, números ou subscrito (_).	`Var_2`
	`X34`
	`a_b_c`

Tabela 6 – Regras para construção de variáveis

As variáveis podem ser redefinidas a qualquer momento, bastando para isso atribuir-lhes um novo valor.

Alguns nomes são usados para variáveis predefinidas, ou seja, são variáveis especiais do MATLAB. Estas são:

Variáveis especiais	Significado
`ans`	Variável usada para exibir os resultados
`pi`	Número 3,14159
`eps`	Menor número tal que, quando adicionado a 1, cria um número maior que 1 no computador.
`flops`	Armazena o número de operações em ponto flutuante realizadas.
`inf`	Significa infinito
`NAN` ou `nan`	Significa não é um número, por exemplo, 0/0.
`i` e `j`	Unidade imaginária [$\sqrt{-1}$].

nargin	Número de argumentos de entrada de uma função
nargout	Número de argumentos de saída de uma função
realmin	Menor número que o computador pode armazenar
realmax	Maior número que o computador pode armazenar

Tabela 7 – Variáveis do Matlab

Comentário e pontuações

Símbolo	Função
,	Separar comandos dados em uma mesma linha.
;	Separar comandos dados em uma mesma linha. Se o último caractere da declaração é um **ponto e vírgula**, a impressão na tela é suprimida, mas a tarefa é realizada.
%	Todo e qualquer caracter depois do símbolo de porcentagem é tomado como **comentário.**
...	Pode-se continuar uma certa expressão na próxima linha usando um espaço em branco e **três pontos**,"...", ao final das linhas incompletas.

Tabela 8 – Comentário e pontuações

Exemplo:

```
 » q1=3, p1=25, ...
q2=5; p2=12;  %Exemplo de uso da vírgula, ponto e vírgula e
três pontos

q1 =
     3
p1 =
    25
```

Os espaços em branco entre os operadores (aritméticos, lógicos, relacionais) e as variáveis (ou constantes) são opcionais. O mesmo para vale para a vírgula, o ponto e vírgula e o símbolo de porcentagem. No entanto, o espaço em branco entre a última variável (ou constante) de uma linha e os três pontos é obrigatório (veja exemplo anterior).

Variáveis literais

Uma variável pode conter uma cadeia de caracteres ao invés de um número. Estes caracteres são manipulados como vetores linha (assunto que será tratado mais adiante). A cadeia de caracteres deve estar limitada por apóstrofos (‘cadeia de caracteres’) para ser atribuída a uma variável literal. Por exemplo:

```
» a='MATLAB'
a =
MATLAB
```

2.4- Obtendo Informações da Área de Trabalho

Os exemplos de declarações mostrados nos itens acima criaram variáveis que são armazenadas na *Área de Trabalho* do MATLAB. Executando

```
>> who
```

obtêm-se uma lista das variáveis armazenadas na Área de Trabalho:

```
Your variables are:
ans       p2        q2
p1        q1        total
```

Que mostra as seis variáveis geradas em nossos exemplos anteriores, incluindo `ans`. Uma informação mais detalhada mostrando a dimensão de cada uma das variáveis correntes é obtido com `whos` que para nosso exemplo produz:

```
 » whos
Name        Size      Elements    Bytes    Density   Complex
ans      1 by 1             1        8       Full      No
p1       1 by 1             1        8       Full      No
p2       1 by 1             1        8       Full      No
q1       1 by 1             1        8       Full      No
q2       1 by 1             1        8       Full      No
total    1 by 1             1        8       Full      No

Grand total is 6 elements using 48 bytes
```

Em qualquer momento, podemos ver o valor que está contido em uma variável, simplesmente digitando no prompt o seu nome.

```
 >> total
total =
   135
```

As variáveis no espaço de trabalho podem ser removidas incondicionalmente usando o comando `clear`. Por exemplo:

```
 » clear p2     %remove a variávelp2
 » clear        %remove todas as variáveis do espaço de
trabalho
```

O comando `save` é usado para gravar as variáveis do espaço de trabalho em um arquivo (.mat) em disco. O comando `load` é usado para recuperar os dados gravados em um arquivo pelo comando `save` e colocá-los no espaço de trabalho. Maiores informações a respeito da sintaxe destes comandos pode ser obtida através do comando `help`, a ser tratado posteriormente.

O comando `clc` limpa a janela de comandos e coloca o cursor na posição inicial.

2.5- Funções Matemáticas

O MATLAB tem uma série de funções científicas pré-definidas. A palavra função no MATLAB tem um significado diferente daquele que tem na Matemática. Aqui, função é um comando, que pode ter alguns argumentos de entrada e alguns de saída. Algumas dessas funções são intrínsecas, ou seja, não podem ser alteradas pelo usuário. Outras funções estão disponíveis em uma biblioteca externa distribuídas com o programa original (MATLAB TOOLBOX), que são na realidade arquivos com a extensão ".m" criados a partir das funções intrínsecas. A biblioteca externa (MATLAB TOOLBOX) pode ser constantemente atualizada à medida que novas aplicações são desenvolvidas. As funções do MATLAB, intrínsecas ou arquivos ".m", podem ser utilizadas apenas no ambiente MATLAB.

As categorias gerais de funções matemáticas disponíveis no MATLAB incluem:

· Matemática elementar;
· Funções especiais;
· Matrizes elementares e especiais;
· Decomposição e fatorização de matrizes;
· Análise de dados;
· Polinômios;
· Solução de equações diferenciais;
· Equações não-lineares e otimização;
· Integração numérica;
· Processamento de sinais.

A maioria das funções pode ser usada da mesma forma que seria escrita matematicamente. Por exemplo:

```
 >> x=sqrt(2)/2
x =
    0.7071
 >> y=acos(x)
y =
    0.7854
 >> y_graus=y*180/pi
y_graus =
   45.0000
```

Estes comandos calculam o arco cujo cosseno é $\sqrt{2}/2$, inicialmente em radianos, depois em graus. Abaixo segue uma lista de funções científicas disponíveis:

`abs(x)`	valor absoluto de x.
`acos(x)`	arco cujo coseno é x
`asin(x)`	arco cujo seno é x.
`atan(x)`	arco cuja tangente é x.
`conj(x)`	conjugado complexo
`cos(x)`	coseno de x.
`cosh(x)`	coseno hiperbólico de x.
`exp(x)`	exponencial e^x.
`floor(x)`	arredondamento em direção ao -∞

`gcd(x,y)`	máximo divisor comum de x e y.
`lcm(x,y)`	mínimo múltiplo comum de x e y.
`log(x)`	logaritmo de x na base *e*.
`log10(x)`	logaritmo de x na base 10.
`rem(x,y)`	resto da divisão de x por y.
`round(x)`	arredondamento para o inteiro mais próximo
`sign(x)`	função *signum*
`sin(x)`	seno de x.
`sinh(x)`	seno hiperbólico de x.
`sqrt(x)`	raiz quadrada de x.
`tan(x)`	tangente de x.
`tanh(x)`	tangente hiperbólica de x.

Tabela 9 – Algumas funções matemáticas

Comandos de auxílio

No MATLAB, pode-se obter ajuda sobre qualquer comando ou função. Isto pode ser feito basicamente de três formas: interativamente através do menu de barras, através do comando `help` ou do comando `lookfor`.

Digitando-se simplesmente o comando `help`,

```
>> help
```

o MATLAB mostra uma listagem de todos os pacotes disponíveis. Ajuda sobre um pacote específico ou sobre um comando ou função específica é obtida com o comando `help` **<**`tópico`**>**, onde tópico pode ser o nome de um pacote, de um comando ou função. Por exemplo:

```
» help sign
SIGN Signum function.
     For each element of X, SIGN(X) returns 1 if the
     element is greater than zero, 0 if it equals zero and
     -1 if it is less than zero.  For complex X,
     SIGN(X) = X ./ ABS(X).
```

O comando `help` é a maneira mais simples de se obter auxílio no caso do usuário conhecer o tópico em que ele quer assistência. Note que no exemplo mostrado a função `SIGN` está escrita em letras maiúsculas somente para destacar. Deve-se lembrar que todos os comandos do MATLAB devem ser escritos em letras minúsculas. Portanto, para utilizar esta função deve-se digitar:

```
» sign (x)
```

O Comando `lookfor` provê assistência pela procura através de todas as primeiras linhas dos tópicos de auxílio do MATLAB e retornando aquelas que contenham a palavra-chave especificada. O interessante deste comando é que a palavra chave não precisa ser um comando do MATLAB. Sua sintaxe é `lookfor` `<palavra-chave>`, onde palavra-chave é a cadeia de caracteres que será procurada

nos comandos do MATLAB. Por exemplo, para se obter informações sobre funções para se resolver integral:

```
» lookfor integral
ELLIPKE Complete elliptic integral.
EXPINT Exponential integral function.
DBLQUAD Numerically evaluate double integral.
INNERLP Used with DBLQUAD to evaluate inner loop of
integral.
QUAD   Numerically evaluate integral, low order method.
QUAD8  Numerically evaluate integral, higher order method.
COSINT Cosine integral function.
SININT Sine integral function.
ASSEMA Assembles area integral contributions in a PDE
problem.
COSINT Cosine integral function.
FOURIER Fourier integral transform.
IFOURIER Inverse Fourier integral transform.
SININT Sine integral function.
BLKPIDCON The output of the block is the sum of
proportional, integral and
```

Apesar da palavra `integral` não ser um comando do MATLAB, ela foi encontrada na descrição de 14 comandos. Tendo esta informação, o comando `help` pode ser usado para exibir informações a respeito de um comando específico, como por exemplo:

```
» help quad
```

2.6- Números Complexos

Algumas linguagens de programação requerem um tratamento especial para números complexos, o que não é o caso do MATLAB. Números complexos são permitidos em todas as operações e funções no MATLAB. Os números complexos são introduzidos usando-se as funções especiais `i` e `j`. Eles podem ser representados de várias maneiras. Por exemplo:

```
 » z1=3+4*i
z1 =
   3.0000 + 4.0000i
 » z2=3+4j
z2 =
   3.0000 + 4.0000i
 » z1+z2
ans =
   6.0000 + 8.0000i
```

- **Identidade de Euler:** relaciona a forma polar de um número complexo com a sua forma retangular.

$M\angle\theta \equiv M.e^{j\theta} = a+bj$, onde: $M=\sqrt{a^2+b^2}$

$\theta=\tan^{-1}(b/a)$

$a=M.\cos\theta$

$b=M.\text{sen}\theta$

No MATLAB, a conversão entre as formas polar e retangular de um número complexo utiliza as seguintes funções:

- `real`: parte real de um número complexo
- `imag`: parte imaginária de um número complexo
- `abs`: calcula o valor absoluto ou módulo de um número complexo
- `angle`: calcula o ângulo de um número complexo

Exemplo:

```
» x=1-4i
x =
   1.0000 - 4.0000i
 » a=real(x)
a =
     1
 » b=imag(x)
b =
    -4
 » M=abs(x)
M =
    4.1231
 » theta=angle(x)*180/pi
theta =
  -75.9638
```

2.7- Expressões Simbólicas

No MATLAB, é possível manipularmos expressões que além de números e variáveis numéricas, contêm também variáveis simbólicas. Por exemplo:

```
 >> syms x
 >> simplify((sin(x))^2+(cos(x))^2)
ans =
1
```

Estes comandos mandam o MATLAB simplificar a expressão $\text{sen}^2x+\cos^2x$. Primeiro precisamos dizer ao MATLAB que `x` é uma variável simbólica, depois pedimos para simplificar a expressão que envolve `x`. Neste caso usamos uma **função** chamada `simplify`. Neste caso, a função `simplify` tem como argumento de entrada uma expressão simbólica e de saída também.

Uma vez definido que a variável `x` é uma variável simbólica, podemos definir expressões que envolvem esta variável. Por exemplo, dadas duas funções $f(x) = 2x^2+3x-5$ e $g(x) = x^2-x+7$, podemos fazer uma série de operações algébricas envolvendo estas funções. Por exemplo:

```
 >> f=2*x^2+3*x-5; g=x^2-x+7;
 >> f+g
ans =
3*x^2+2*x+2
 >> f-g
ans =
x^2+4*x-12
 >> f*g
ans =
(2*x^2+3*x-5)*(x^2-x+7)
 >> expand(ans)
ans =
2*x^4+x^3+6*x^2+26*x-35
 >> f/g
ans =
(2*x^2+3*x-5)/(x^2-x+7)
 >> expand(ans)
ans =
2/(x^2-x+7)*x^2+3/(x^2-x+7)*x-5/(x^2-x+7)
```

O MATLAB pode realizar operações mais avançadas sobre expressões simbólicas. A função `compose` calcula a composição das funções $f(x)$ e $g(x)$ em $f(g(x))$, a função `finverse` encontra a inversa funcional de uma expressão e a função `subs` substitui uma variável por um número (ou por outra variável) em uma expressão. Por exemplo:

```
 >> f=1/(1-x^2); g=sin(x);
 >> compose(f,g)
ans = 1/(1-sin(x)^2)
 >> compose(g,f)
ans = sin(1/(1-x^2))
 >> finverse(g)
ans = asin(x)
 >> subs(f,x,2)
ans = -1/3
```

O MATLAB pode resolver equações. Por exemplo, para resolver a equação
$ax^2+bx+c=0$
algebricamente, podemos usar os comandos:

```
 >> syms a b c x
 >> solve(a*x^2+b*x+c)
ans =
[1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

O MATLAB pode exibir este resultado de uma forma mais fácil de enxergar usando a função `pretty`.

```
>> pretty(ans)
          [            2          1/2]
          [    -b + (b  - 4 a c)    ]
          [1/2 --------------------]
          [              a          ]
          [                         ]
          [            2          1/2]
          [    -b - (b  - 4 a c)    ]
          [1/2 --------------------]
          [              a          ]
```

Abaixo segue um resumo das funções para manipulação de expressões algébricas:
`diff(f)` - calcula a derivada de f.
`compose(f,g)` - determina a composta $f(g(x))$.
`expand(expr)` - expande uma expressão expr.
`finverse(expr)` - determina a inversa funcional da expressão expr.
`pretty(expr)` - exibe a expressão expr numa forma mais bonita.
`simple(expr)` - procura encontrar uma forma mais simples de escrever uma expressão expr.
`simplify(expr)` - simplifica a expressão expr.
`solve(expr)` - acha a(s) solução(es) da equação expr = 0.
`subs(expr,x,a)` - substitui na expressão expr a variável x por a.
`syms x y z a b` - define as variáveis simbólicas x, y, z, a e b.

Existem várias outras funções para manipulação de expressões algébricas. Você pode obter informações sobre elas digitando `help symbolic`. Uma função interessante que mostra as capacidades do MATLAB em tratar com funções matemáticas é `funtool` que é uma calculadora para funções.

3) VETORES E MATRIZES

O MATLAB permite a manipulação de linhas, colunas, elementos individuais e partes de matrizes.

Na tabela 10, tem-se um resumo das diversas formas de se construir um vetor no MATLAB.

X=primero : último	Cria um vetor **x** começando com o valor **primeiro**, incrementando-se de 1(um) em 1(um) até atingir o valor **último** ou o valor mais próximo possível de **último**
X=primeiro:incremento:último	Cria um vetor **x** começando com o valor **primeiro**, incrementando-se do valor **incremento** até atingir o valor **último** ou o valor mais próximo possível de último
X=linspace(primeiro, último, n)	Cria um vetor **x** começando com o valor **primeiro** e terminado no valor **último**, contendo **n** elementos linearmente espaçados.
X=logspace(primeiro, último, n)	Cria um vetor x começando com o valor $10^{\textbf{primeiro}}$ e terminando no valor $10^{\textbf{último}}$, contendo **n** elementos logaritmicamente espaçados
X=[2 2*pi sqrt(2) 2-3j]	Cria um vetor **x** contendo os elementos especificados

Tabela 10 – Construção de Vetores

Exemplo 1:

```
>> x = 1 : 5
```

gera um vetor linha contendo os números de 1 a 5 com incremento unitário. Produzindo

```
X =
        1    2    3    4    5

>> x=1:10.5
x=
     1    2    3    4    5    6    7    8    9    10
```

Exemplo 2:

```
>> z = 6 : -1 : 1
Z =
        6    5    4    3    2    1
```

Exemplo 3:

Pode-se, também, gerar vetores usando a função **linspace**. Por exemplo,

```
>> k = linspace (0, 1, 6)
K =
        0       0.2000  0.4000  0.6000  0.8000  1.0000
```

gera um vetor linearmente espaçado de 0 a 1, contendo 6 elementos.

```
>> x=linspace(1,10.5,5)
x=
     1.0000    3.3750    5.7500     8.1250       10.5000
```

Exemplo 4:

```
>> x=logspace(0,2,5)
x=
     1.0000    3.1623    10.0000     31.6228       100.00
```

Exemplo 5:

```
>> x=[8   6  8.10  5-6j]
x=
     8.0000    6.0000    8.1000    5.0000-6.0000i
```

Nos exemplos acima os vetores possuem uma linha e várias colunas (vetores linha). Da mesma forma podem existir vetores coluna (uma coluna e várias linhas). Para se criar um vetor coluna elemento por elemento estes devem estar separados por (**;**). Por exemplo:

```
>> v=[1.5;-3.2;9]
v =
    1.5000
   -3.2000
    9.0000
```

Esses vetores coluna podem também ser criados a partir dos comandos utilizados anteriormente para criar os vetores linha, acompanhados do símbolo (**'**), que é o **operador de transposição**. Exemplo:

```
>> y=(1:0.5:3)'
y =
    1.0000
    1.5000
    2.0000
    2.5000
    3.0000

>> z=[0 -2.3 4 sqrt(33)]'
z =
         0
   -2.3000
    4.0000
    5.7446
```

ENDEREÇAMENTO DE VETORES

No MatLab, cada um dos elementos de um vetor podem ser acessados através de seu ***índice*** que identifica cada uma das colunas. Por exemplo :

```
>> x=1:10
x=
     1    2    3    4    5    6    7    8    9    10

>> x(3)   % Acessa o terceiro elemento de x
ans =
     3

>> x(5)   % Acessa o quinto elemento de x
ans =
     5
```

Esses elementos de um vetor também podem ser acessados em blocos. Por exemplo:

```
>> c=linspace(10,40,7)
c =
    10    15    20    25    30    35    40

>> c(3:5)      % terceiro a quinto elemento de c
ans =
    20    25    30

>>c(5:-2:1)    % quinto, terceiro e primeiro elementos de c
ans =
    30    20    10
```

O endereçamento indireto também é possível, permitindo referenciar os elementos em qualquer ordem:

```
>> c([4 1])  %quarto e primeiro elementos
ans =
    25    10
```

No caso de vetores coluna, os comandos acima funcionam de maneira similar. Por exemplo:

```
>> d=c'
d =
    10
    15
    20
    25
    30
    35
    40
```

```
>> d([4 1])  %quarto e primeiro elementos
ans =
    25
    10

>> d(5:-2:1)
ans =
    30
    20
    10
```

Operações entre vetores

As operações básicas entre vetores só são definidas quando estes tiverem o mesmo tamanho e orientação (linha ou coluna). Estas operações são:

Seja **a=[a_1 a_2 ... a_n]** , **b=[b_1 b_2 ... b_n]** e **c** um escalar		
operação	**expressão**	**resultado**
adição escalar	a+c	[a_1+c a_2+c ... a_n+c]
adição vetorial	a+b	[a_1+b_1 a_2+b_2 ... a_n+b_n]
multiplicação escalar	a*c	[a_1*c a_2*c ... a_n+c]
multiplicação vetorial	a.*b	[a_1*b_1 a_2*b_2 ... a_n*b_n]
divisão	a./b	[a_1/b_1 a_2/b_2 ... a_n/b_n]
potenciação	a.^c	[a_1^c a_2^c ... a_n^c]
	c.^a	[c^a_1 c^a_2 ... c^a_n]
	a.^b	[a_1^b_1 a_2^b_2 ... a_n^b_n]

MATRIZES:

O MATLAB trabalha essencialmente com um tipo de objeto, uma matriz numérica retangular (1x1; 2x2; 3x3; **i** (linha) x **j** (coluna); etc).

Os elementos de cada linha da matriz são separados por espaços em branco ou vírgulas e as colunas separadas por ponto e vírgula, colocando-se colchetes em volta do grupo de elementos que formam a matriz. Por exemplo, entre com a expressão

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

Pressionando <enter> o MATLAB mostra o resultado

```
  A =
 1   2   3
 4   5   6
 7   8   9
```

As linhas das matrizes também podem ser definidas através dos comandos utilizados anteriormente para se definir vetores linha. Por exemplo:

```
>> A=[1:3;linspace(4,9,3);0:.5:1]
A =
          1.00          2.00          3.00
          4.00          6.50          9.00
             0          0.50          1.00
```

Os elementos de uma matriz (ou de um vetor) também podem ser definidos por operações ou funções matemáticas. Por exemplo:

```
>> B=[15 7;sqrt(36) cos(pi/3);12/7 2.5^2]
B =
   15.0000    7.0000
    6.0000    0.5000
    1.7143    6.2500
```

OPERAÇÕES COM MATRIZES

As operações com matrizes no MATLAB são as seguintes:

- Transposta;
- Adição;
- Subtração;
- Multiplicação;
- Divisão à direita;
- Divisão à esquerda;
- Exponenciação;

3.1 Transposta

O caracter apóstrofo, " ' " , indica a transposta de uma matriz. Considere os exemplos a seguir:

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

>> x = [-1 0 2]'
X =
       -1
        0
        2
```

3.2 Adição e Subtração

A adição e subtração de matrizes são indicadas, respectivamente, por "+" e "-". As operações são definidas somente se as matrizes tiverem as mesmas dimensões. Por exemplo, a soma com as matrizes mostradas acima, **A + x**, não é correta porque **A** é 3x3 e **x** é 3x1. Porém,

```
>> C = A + B
```

é aceitável, e o resultado da soma é

```
C =
        2     6     10
        6     10    14
        10    14    0
```

A adição e subtração também são definidas se um dos operadores é um escalar, ou seja, uma matriz l x l. Neste caso, o escalar é adicionado ou subtraído de todos os elementos do outro operador. Por exemplo:

```
>> y = x - 1
```

resulta em

```
Y =
        -2
        -1
        1
```

3.3 Multiplicação

A multiplicação de matrizes é indicada por "*". A multiplicação **x*y** é definida somente se a segunda dimensão de x for igual à primeira dimensão de **y.** A multiplicação

```
>> x'* y
```

é aceitável, e resulta em

```
Ans =
        4
```

É evidente que o resultado da multiplicação **y'*x** será o mesmo. Existem dois outros produtos que são transpostos um do outro.

```
>> x*y'
Ans =
        2     1     -1
        0     0     0
        -4  -2    2

>> y*x'
Ans =
        2     0     -4
        1     0     -2
        -1    0     2
```

O produto de uma matriz por um vetor é um caso especial do produto entre matrizes. Por exemplo **A** e **X,**

```
>> b = A*x
```

que resulta em

```
B =
        5
        8
        -7
```

Naturalmente, um escalar pode multiplicar ou ser multiplicado por qualquer matriz.

```
   >> pi*x
Ans =
        -3.1416
        0
        6.2832
```

Além da multiplicação matricial e escalar, podemos ter a multiplicação por elemento de matrizes de mesma dimensão. Esse tipo de operação é feita utilizando-se um ponto (**.**) antes do operador de multiplicação (*). Ou seja, se **A** e **B** são matrizes definidas por **A=[a_{11} a_{12} ... a_{1n} ; a_{21} a_{22} ... a_{2n} ; ... ; a_{m1} a_{m2} ... a_{mn}]** e **B=[b_{11} b_{12} ... b_{1n} ; b_{21} b_{22} ... b_{2n} ; ... ; b_{m1} b_{m2} ... b_{mn}]**, então **A.*B =a_{ij}*b_{ij}**. Por exemplo:

```
>> A.*B
ans =
     1     8    21
     8    25    48
    21    48     0
```

3.4 Divisão

Existem dois símbolos para divisão de matrizes no MATLAB "\" e "/". Se A é uma matriz quadrada não singular, então **A\B** e **B/A** correspondem respectivamente à multiplicação à esquerda e à direita da matriz **B** pela inversa da matriz **A**, ou **inv(A)*B** e **B*inv(A),** mas o resultado é obtido diretamente. Em geral,

- X = A\B é a solução de A*X = B
- X = B/A é a solução de X*A = B

Por exemplo, como o vetor **b** foi definido como **A*x,** a declaração

```
   >> z = A\b
```

resulta em

Z =

```
        -1
        0
        2
```

A divisão por elemento entre matrizes é definida de maneira similar à multiplicação por elemento, ou seja, **A./B= a_{ij}/b_{ij}** e **A.\B= a_{ij}\b_{ij}** , onde **A** e **B** têm mesma dimensão.

3.5 Exponenciação

A expressão **A^p** eleva **A** à **p**-ésima potência e é definida se **A** é matriz quadrada e **p** um escalar. Se **p** é um inteiro maior do que um, a exponenciação é computada como múltiplas multiplicações. Por exemplo,

```
>> A^3
```

Ans =

279	*360*	*306*
684	*873*	*684*
738	*900*	*441*

A exponenciação por elemento entre matrizes é definida de maneira similar à multiplicação por elemento, ou seja, $\mathbf{A.\hat{}B} = a_{ij}^{b_{ij}}$, onde **A** e **B** têm mesma dimensão. De maneira similar, a potenciação por elemento entre uma matriz e um escalar apresenta as seguintes formas: $\mathbf{A.\hat{}c} = a_{ij}^{c}$ **e** $\mathbf{c.\hat{}A} = c^{a_{ij}}$

Elementos das Matrizes

Um elemento individual da matriz pode ser indicado incluindo os seus subscritos entre parênteses. Por exemplo, dada a matriz A:

```
A =
        1       2       3
        4       5       6
        7       8       9
```

a declaração

```
>> A(3,3) = A(1,3) + A(3,l)
```

resulta em

```
A =
        1       2       3
        4       5       6
        7       8       10
```

```
>> A(1:3,2)
Ans =
        2
        5
        8
```

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

é uma submatriz 3x2, que consiste das três linhas e das últimas duas colunas de A.

```
Ans =
        2         3
        5         6
        8         10
```

Utilizando os dois pontos no lugar de um subscrito denota-se todos elementos da linha ou coluna. Por exemplo,

```
>> A(1:2,:)
Ans =
        1     2     3
        4     5     6
```

é uma submatriz 2x3 que consiste da primeira e segunda linhas e todas colunas da matriz **A**.

- **Funções**: o MATLAB possui algumas funções que se aplicam a matrizes como, por exemplo, as funções `size` (fornece o número de linhas e colunas de uma matriz) e `length` (fornece o maior valor entre o número de linhas e colunas). O MATLAB tem também funções que se aplicam individualmente à cada coluna da matriz produzindo um vetor linha com os elementos correspondentes ao resultado de cada coluna. Se a função for aplicada à transposta de da matriz, os resultados serão relativos a cada linha da matriz. Se o argumento da função for um vetor, o resultado será um escalar. algumas dessas funções são:

função	descrição
`sum`	soma dos elementos
`prod`	produto dos elementos
`mean`	média aritmética
`std`	desvio padrão
`max`	maior elemento
`min`	menor elemento
`sort`	ordena em ordem crescente

- **Submatrizes.**

Sendo B uma matriz 5x5 unitária, podemos defini-la através da seguinte função:

```
>> B = ones (5)
B =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

Sendo C uma matriz de zeros 3x4, podemos defini-la como:

```
>> C=zeros(3,4)
C =
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Para que o MATLAB gere uma matriz de números aleatórios entre 0 e 1, utilizamos a função `rand` (veja também a função `randn`, utilizando o comando `help`). Exemplo:

```
>> D=rand(2,3)
D =
    0.2190    0.6789    0.9347
    0.0470    0.6793    0.3835
```

4) GRÁFICOS NO MATLAB

A construção de gráficos no MATLAB é mais uma das facilidades do sistema. Através de comandos simples pode-se obter gráficos bidimensionais ou tridimensionais com qualquer tipo de escala e coordenada.

4.1 Gráficos Bidimensionais

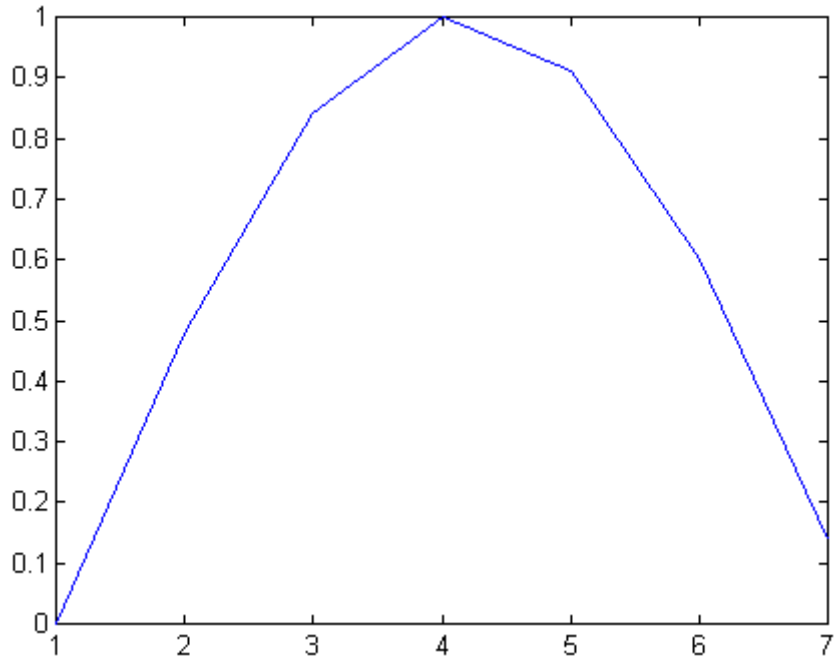
Estes são os comandos para *plotar* gráficos bidimensionais:

plot	Plotar linear.
loglog	Plotar em escala loglog.
semilogx	Plotar em semilog.
semilogy	Plotar em semilog.
fill	Desenhar polígono 2D.
polar	Plotar em coordenada polar.
bar	Gráfico de barras.
stem	Seqüência discreta.
stairs	Plotar em degrau.
errorbar	Plotar erro.
hist	Plotar histograma.
rose	Plotar histograma em ângulo.
compass	Plotar em forma de bússola.
feather	Plotar em forma de pena.
fplot	Plotar função.
comet	Plotar com trajetória de cometa.

Tabela 11 – comandos para gráficos bidimensionais

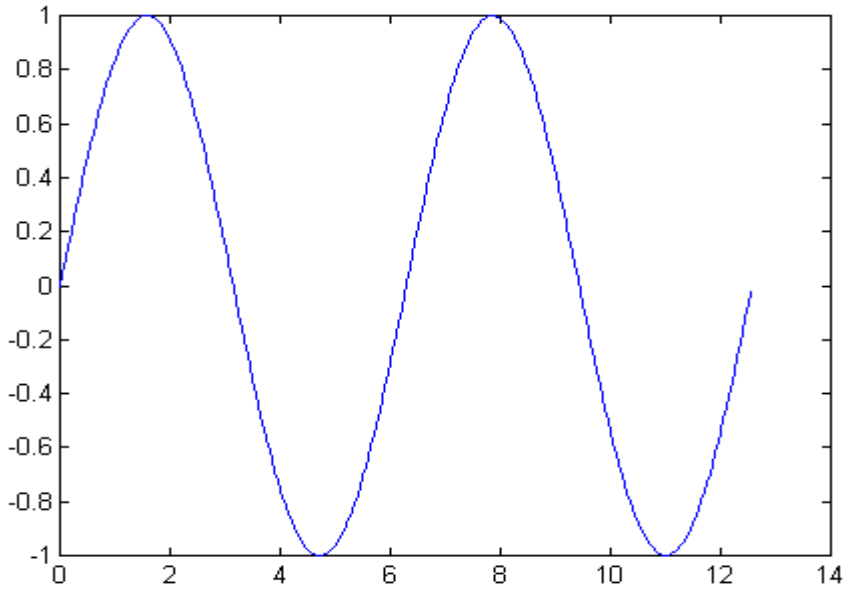
Se **Y** é um vetor, **plot(Y)** produz um gráfico linear dos elementos de **Y** versos o índice dos elementos de **Y.** Por exemplo, para plotar os números [0.0, 0.48, 0.84, 1.0, 0.91, 0.6, 0,14], entre com o vetor e execute o comando **plot:**

```
>> Y = [0.0, 0.48, 0.84, 1.0, 0.91, 0.6, 0,14];
>> plot(Y)
```



Se **X** e **Y** são vetores com dimensões iguais, o comando **plot(X,Y)** produz um gráfico bidimensional dos elementos de **X** versos os elementos de **Y**, por exemplo

```
>> t = 0:0.05:4*pi;
>> y = sin(t);
>> plot(t,y)
```



O MATLAB pode também plotar múltiplas linhas e apenas um gráfico. Existem duas maneiras, a primeira é usado apenas dois argumentos, como em **plot(X,Y)**, onde **X** e/ou **Y** são matrizes. Então:

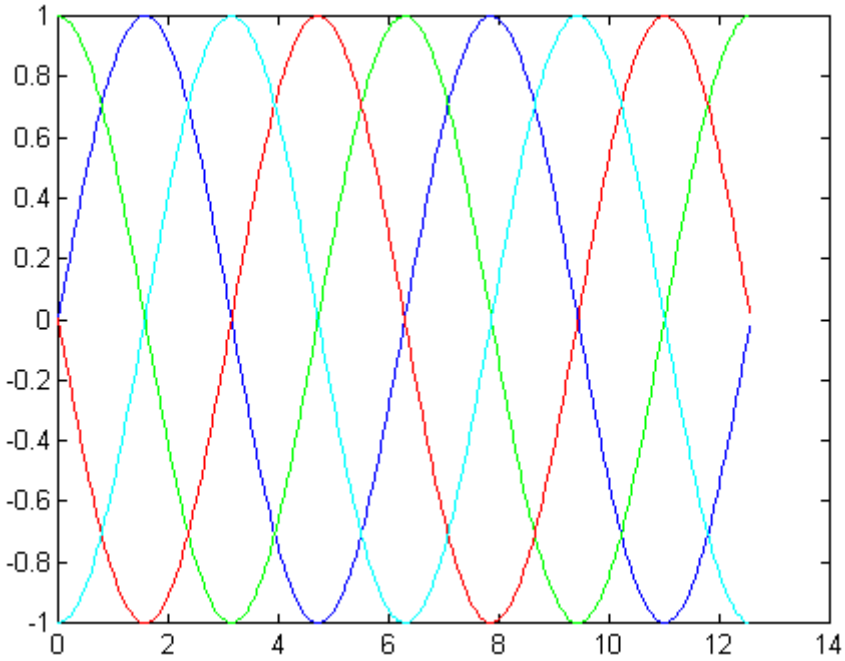
- Se **Y** é uma matriz e **X** um vetor, **plot(X,Y)** plota sucessivamente as linhas ou colunas de **Y** versos o vetor **X**.

- Se **X** é uma matriz e **Y** é um vetor, **plot(X,Y)** plota sucessivamente as linhas ou colunas de **X** versos o vetor **Y**.

- Se **X** e Y são matrizes com mesma dimensão, **plot(X,Y)** plota sucessivamente as colunas de **X** versos as colunas de **Y**.

- Se **Y** é uma matriz, **plot(Y)** plota sucessivamente as colunas de **Y** versos o índice de cada elemento da linha de **Y**.

A segunda, e mais fácil, maneira de plotar gráficos com múltiplas linhas é usando o comando **plot** com múltiplos argumentos. Por exemplo:

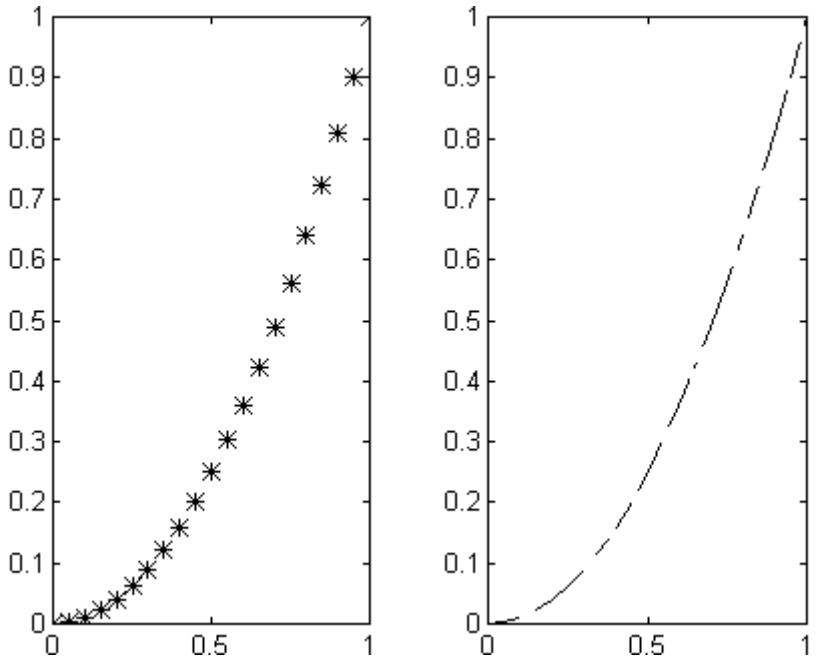


>> plot(t, sin(t), t, cos(t), t, sin(t + pi), t, cos(t + pi))

4.2 Estilos de Linha e Símbolo

Os tipos de linhas, símbolos e cores usados para plotar gráficos podem ser controlados se os padrões não são satisfatórios. Por exemplo,

>> X = 0:0.05:1;
>> subplot(l2l), plot(X,X.^2,'k')*
>> subplot(l22), plot(X,X.^2,'k --')



Outros tipos de linhas, pontos e cores também podem ser usados:

TIPO DE LINHA	
_	_____
--	----------
-.	-.-.-.-.-.-.-.-.-.-.-.
.	.....

TIPO DE PONTO	
.	.....
*	* * * * * * * *
o	o o oo o o o o o o
+	++++++++++
x	xx x x x x x x

CORES	
y	amarelo
m	lilás
c	azul claro
r	vermelho
g	verde
b	azul escuro
w	branco
k	preto

4.3 Números Complexos

Quando os argumentos para plotar são complexos, a parte imaginária é ignorada, exceto quando é dado simplesmente um argumento complexo. Para este caso especial é plotada a parte real versos a parte imaginária. Então, **plot(Z)**, quando **Z** é um vetor complexo, é equivalente a **plot(real(Z),imag(Z))**.

4.4 Escala Logarítmica, Coordenada Polar e Gráfico de Barras

O uso de **loglog**, **semilogx**, **semilogy** e **polar** é idêntico ao uso de **plot**. Estes comandos são usados para plotar gráficos em diferentes coordenadas e escalas:

- **polar(Theta,R)** plota em coordenadas polares o ângulo **THETA**, em radianos, versos o raio **R**;
- **loglog** plota usando a escala $\log_{10}$x$\log_{10}$;
- **semilogx** plota usando a escala semi-logarítmica. O eixo x é $\log_{10}$ e o eixo y é linear;
- **semilogy** plota usando a escala semi-logarítmica. O eixo x é linear e o eixo y é $\log_{10}$;

O comando **bar(X)** mostra um gráfico de barras dos elementos do vetor **X**, e não aceita múltiplos argumentos.

4.5 Plotando Gráficos Tridimensionais e Contornos

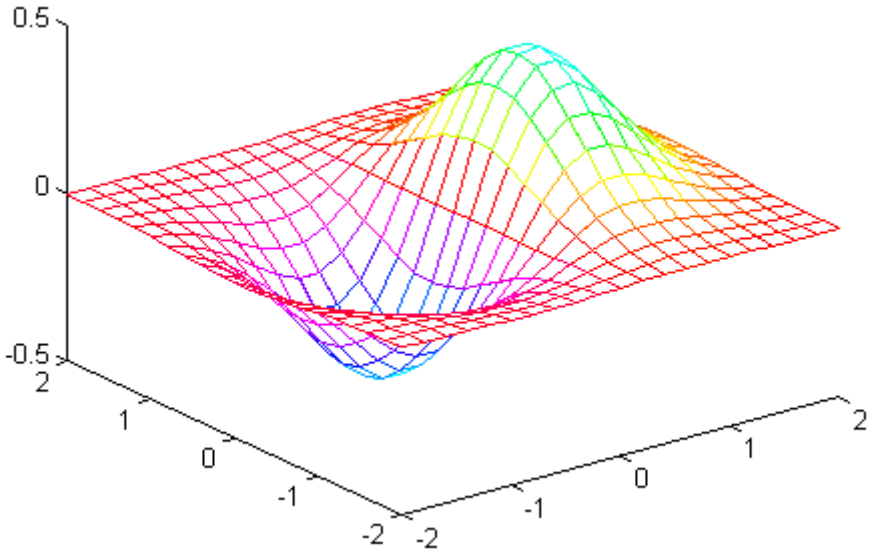
Estes são alguns comandos para plotar gráficos tridimensionais e contornos.

Plot3	Plotar em espaço 3D.
fill3	Desenhar polígono 3D.
comet3	Plotar em 3D com trajetória de cometa.
contour	Plotar contorno 2D.
contour3	Plotar contorno 3D.
clabel	Plotar contorno com valores.
quiver	Plotar gradiente.
mesh	Plotar malha 3D.
meshc	Combinação mesh/contour.
surf	Plotar superfície 3D.
surfc	Combinação surf/contour.
surfil	Plotar superfície 3D com iluminação.
slice	Plot visualização volumétrica.
cylinder	Gerar cilindro.
sphere	Gerar esfera.

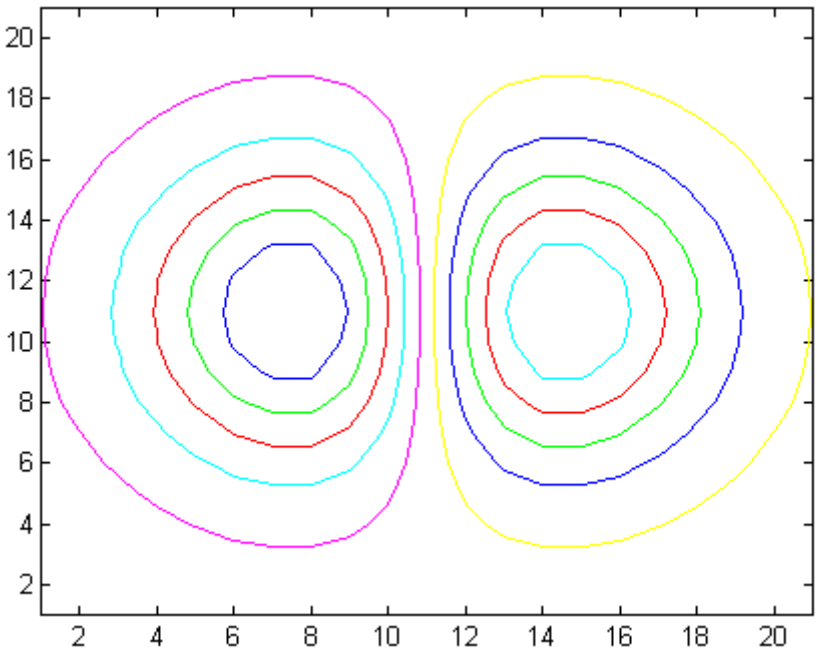
Tabela 12 - comandos para gráficos tridimensionais

O comando **mesh(X,Y,Z)** cria uma perspectiva tridimensional plotando os elementos da matriz **Z** em relação ao plano definindo pelas matrizes **X** e **Y.** Por exemplo,

```
>> [X,Y] = meshdom(-2:.2:2, -2:.2:2);
>> Z = X.* exp(-X..^2 - Y.^2);
>> mesh(X,Y,Z)
```



e o comando **contour(Z,10)** mostra a projeção da superfície acima no plano xy com 10 iso-linhas:



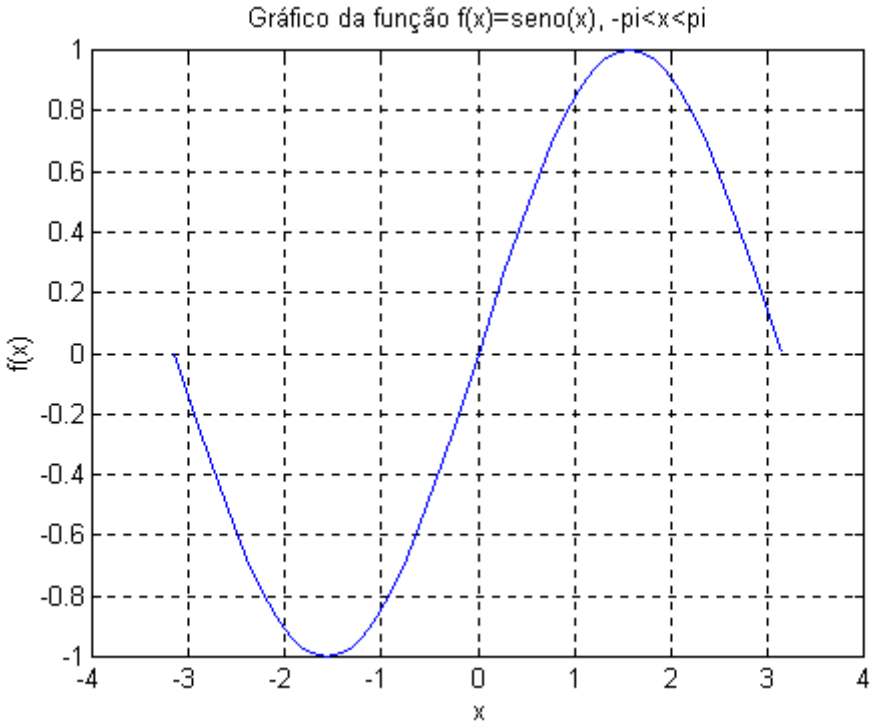
4.6 Anotações no Gráfico

O MATLAB possui comandos de fácil utilização para adicionar informações em um gráfico:

title	Título do gráfico.
xlabel	Título do eixo-X.
ylabel	Título do eixo-Y.
zlabel	Título do eixo-Z.
text	Inserir anotação no gráfico.
gtext	Inserir anotação com o "mouse".
grid	Linhas de grade.

Tabela 13 – Anotações em gráficos

```
>> fplot('sin', [-pi pi])
>> title('Gráfico da função f(x)=seno(x), -pi<x<pi')
>> xlabel('x')
>> ylabel('f(x)')
>> grid
```



5) Programação

5.1- Arquivos .m

Para resolver problemas simples, é cômodo e eficiente utilizar o MATLAB como se fosse uma calculadora, entrando-se com os comandos diretamente no prompt. Ou seja, cada linha de comando é introduzida na Janela de Comandos e processada imediatamente. Entretanto, à medida que o número de comandos aumenta, ou quando se deseja mudar o valor de uma ou mais variáveis e executar novamente os comandos, o melhor é utilizar o MATLAB como uma linguagem de programação, ou seja, utilizar o MATLAB para executar seqüências de comandos armazenadas em arquivos de roteiro (*script*). Esses arquivos que contêm as declarações do MATLAB são chamados arquivos "**.m**" (ou *M-files*), como, por exemplo, **exemplo1.m**. Esses *M-files* são os **programas** fontes do MATLAB e consistem de seqüências de comandos normais do MATLAB, possibilitando incluir outros arquivos "**.m**" escritos no formato texto (ASCII).

Para escrever um programa (ou arquivo **.m**) no MATLAB, escolha **File** na barra de menu. Dentro do menu **File** escolha **New** e selecione **M-file**. Abre-se, então, um editor de textos, onde pode-se escrever os comandos do MATLAB. Para editar um arquivo já existente, selecione a opção **Open M-File**, a partir do menu **File**. Os arquivos podem, também, ser editados fora do MATLAB utilizando qualquer editor de texto.

Escreva, por exemplo, o programa abaixo :

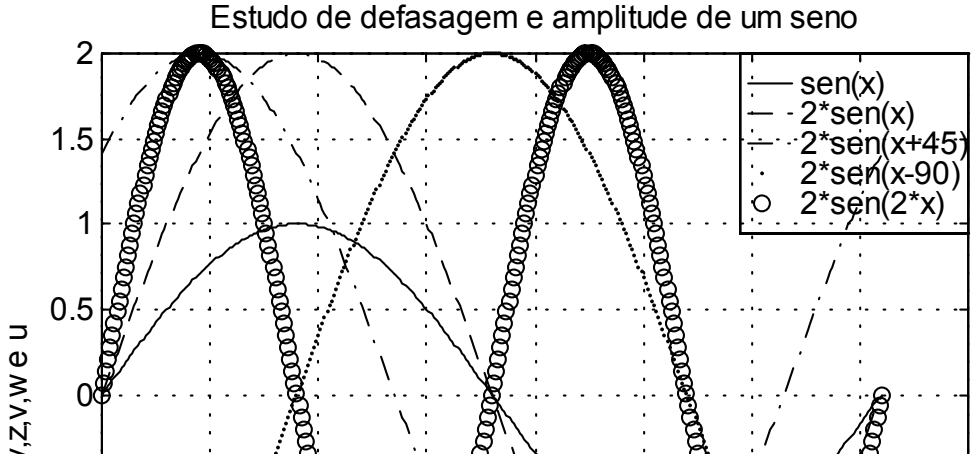
```
%=================================================
% Exemplo de programação no MATLAB
% Este exemplo plota uma função seno nas sequintes
% condições:
% sen(x)
% 2*sen(x)
% 2*sen(x+45)
% 2*sen(x-90)
% 2*sen(2*x)
%=================================================
%
x=0:360;
%
% Seno com amplitude A=1 e defasagem phi=0 graus
A=1;
phi=0;
y=A*sin(2*pi*x/360+2*pi*phi/360);
% Seno com amplitude A=2 e defasagem phi=0 graus
A=2;
z=A*sin(2*pi*x/360+2*pi*phi/360);
% Seno com amplitude A=2 e defasagem phi=45 graus
phi=45;
v=A*sin(2*pi*x/360+2*pi*phi/360);
% Seno com amplitude A= 2 e defasagem phi=-90 graus
phi=-90;
```

```
w=A*sin(2*pi*x/360+2*pi*phi/360);
% Seno com amplitude A= 2 e defasagem phi=0 graus
phi=0;
u=A*sin(2*pi*2*x/360+2*pi*phi/360);
% Plotagem do resultado
plot(x,y,'k-',x,z,'k--',x,v,'k-.',x,w,'k.',x,u, 'ko')
grid
xlabel('Valores de x em graus')
ylabel('y,z,v,w e u')
title('Estudo de defasagem e amplitude de um seno')
legend('sen(x)','2*sen(x)','2*sen(x+45)','2*sen(x-90)',
'2*sen(2*x)')
```

Uma vez escrito o programa, entre no menu **File** da janela do editor de textos e escolha a opção **Save as...** Nesta opção do menu, salve o programa como *prog1.m* no seu diretório de trabalho. Em seguida, volte à janela de comandos do MATLAB e use o comando *cd* ou a opção **Set Path...** do menu **File** para ir ao diretório onde o programa *prog1.m* foi salvo. Em seguida, digite o nome do arquivo (sem a extensão **.m**) para executar o programa:

```
>>prog1 .
```

O gráfico obtido é mostrado na figura abaixo.



5.2- Funções M-files :

Por causa da grande utilidade dos *M-files*, o MATLAB possui diversas funções que tornam os *M-files* ainda mais interessantes. Estas funções estão listadas na tabela 14:

echo on	É usado para que os comandos do *M-file* sejam mostrados na janela de comandos durante a execução.
echo off	É usado para suprimir a exibição dos comandos feita através do echo on.
input	Permite entrada de dados durante a execução do programa via teclado.
pause	Faz uma pausa na execução do programa até que uma tecla qualquer seja pressionada.
pause(n)	Faz uma pausa de **n** segundos na execução do programa.
disp(ans)	Visualiza os resultados sem mostrar os nomes das variáveis.
waitforbottonpress	Faz uma pausa até que uma tecla do "mouse" ou do teclado seja pressionada.
keyboard	Passa o controle temporariamente para o teclado ("Type **return** to quit")

Tabela 14 – Funções *M-files*

Como exemplo, considere os seguintes programas (*M-file*) :

```
% Exemplo 1 de utilização da função M-file "input"
%=========================================
%Plota uma função y=ax^2 + bx + c no intervalo -5<x<5
clear
aux='s';
while aux= = 's',
    a=input('a =');
    b=input('b =');
    c=input('c =');
    x=-5:0.1:5;
    y=a*x.^2+b*x+c;
    plot(y)
    figure(1)
    pause
    clc
    close
    aux=input('Plotar outro ? (s/n) = => ','s');
end
```

Repare, além do uso do comando **input**, o uso do caracter % (comentário) no texto, do comando **clear** (apaga todos os dados da memória), **, pause** (provoca uma pausa na execução do arquivo até que qualquer tecla seja digitada), **clc** (limpa a Janela de Comando), **figure(1)** (mostra a Janela Gráfica número 1) e **close** (fecha todas as Janelas Gráficas).

```
% Exemplo 2 de utilização da função M-file "input"
%=========================================
% Programa para traçar a curva :
%
% y=A.sin(x+phi),
%
%sendo que os valores de x[rad], A e phi[graus] devem ser
%entrados via teclado durante a execução do programa
%
x=input('Entre com o vetor x [rad]> ');
A=input('Entre com o valor de A> ');
phi=input('Entre com o valor de phi [graus]> ');
%
y=A*sin(x+pi*phi/180);
plot(x,y,'r');
grid on
title('Exemplo de utilização da função "input"')
xlabel('x em rad/s')
ylabel('y=A.sin(x+phi)')
```

```
% Exemplo 3 de utilização da função M-file "input"
% Programa decsomat.m
%=========================================
% Programa para gerar uma matriz com elementos aleatórios
% entre -10 e 10 e decompô-la na soma de três matrizes :
% uma triangular inferior, uma diagonal e outra triangular
% superior
%-----------------------------------------------------

n = input('Ordem da matriz : ');
A = fix(20*(rand(n) -0.5 * ones(n)));
D = diag(diag(A));
L = tril(A) - D;
U = triu(A) - D;
A, L, D, U

% Fim do programa
%-------------------------------------------------------------
```

```
>> decsomat
Ordem da matriz : 3
A =
-5 3 0
-9 8 6
```

```
3  -2  -9

L =
0   0   0
-9   0   0
3  -2   0

D =
-5   0   0
0   8   0
0   0  -9

U =
0   3   0
0   0   6
0   0   0
```

5.3- Gerenciamento de arquivos

O MATLAB possui uma série de comandos para gerenciamento de arquivos, tais como listar os nomes de arquivos, vizualizar, deletar, etc. Na tabela abaixo tem-se um resumo dos principais comandos :

cd	Mostra o diretório de trabalho atual ou corrente
p=cd	Retorna para a variável **p** o diretório de trabalho corrente
cd temp	Muda para o diretório **temp**
cd ..	Muda para o diretório um nível acima
chdir	O mesmo que cd
chdir path	O mesmo que cd temp
delete test	deleta o arquivo **test.m**
dir	Lista todos os arquivos do diretório de trabalho presente
ls	Faz o mesmo que o comando **dir**
matlabroot	Retorna o caminho do diretório onde se encontra o programa MATLAB executável
path	Visualiza todos os caminhos de diretório do MATLAB
pwd	O mesmo que o comando **cd**
type test	Visualiza o arquivo *M-file* **test.m** na janela de comandos
what	Retorna uma lista de todos os *M-files* do diretório corrente
which test	Visualiza o caminho do diretório do arquivo **test.m**

Tabela 15 : Comandos para Gerenciamento de Arquivos

5.4- Controles de Fluxo

5.4.1- Estruturas Condicionais

Uma estrutura condicional permite a escolha do grupo de comandos a serem executados quando uma dada condição for satisfeita ou não, possibilitando desta forma alterar o fluxo natural de comandos. Esta condição é representada por uma expressão lógica.

5.4.1.1 Estrutura **if-end**
A estrutura condicional mais simples do MA TLAB é:

```
if <condição>
   <comandos>
end
```

Se o resultado da expressão lógica <condição> for 1 (verdadeiro) en tão a lista de <comandos> será executada. Se o resultado for 0 (falso) os <comandos > não serão executados. Por exemplo, considere o arquivo **estcond1.m** cujo conteúdo é:

```
a = input('Entre com o valor de a : ');
if a >= 0
     b = sqrt(a)
end
```

Para executá-lo basta fornecer o seu nome na área de trabalho

```
>> estcond1
Entre com o valor de a : 2
b = 1.4142
```

Neste exemplo, a raiz quadrada de a será atribuída a b somente se o valor de a for maior ou igual a 0 .

Considere o arquivo banana.m:

```
custo=5;
bananas=10;
if bananas>5
     custo=0.1*custo;
end
custo

>>banana
custo =
0.5000
```

No exemplo acima, a expressão *bananas* > 5 é verdadeira, assim o comando :
custo=0.1* custo

Exemplo 2 :
custo=5;
bananas=5;

```
if bananas>5
custo=0.1*custo;
end
custo

>>banana
custo =
5
```

Neste exemplo, a expressão *bananas* > 5 é falsa, assim o comando :
custo=0.1* custo
não foi executado. Assim o custo continua igual a 5.

5.4.1.2 Estrutura **if-else-end**

No caso de haver duas alternativas, uma outra estrutura condicional deve ser usada:

```
if <condição>
   <comandos 1>
else
   <comandos 0>
end
```

Se o resultado da expressão lógica <condição > for 1 (verdadeiro) então a lista <comandos 1> será executada. Se <condição> for 0 (falso) então será a lista <comandos 0> a ser executada. Por exemplo, o programa do arquivo **estcond2.m**

```
a = input('Entre com o valor de a : ');
if a > 0
     b = log(a)
else
     b = exp(a)
end
```

quando executado resultará

```
>> estcond2
Entre com o valor de a : 5
b = 1.6094
```

Se a for positivo, então o logaritmo natural de a será atribuído a b e se a for negativo ou nulo, então b será igual ao exponencial de a .

Exemplo: Plote uma função retangular utilizando-se a estrutura *if-else-end.*

```
%
x=linspace(0,2*pi,100); % Criou-se 100 amostras
entre 0 e 2*pi
%
for n=1:100
```

```
if x(n)<=pi
f(n)=1; %Faz f(t)=1 para 0<t<=pi,i.e.,
%as primeiras 50 amostras de
%f(t) são iguais a 1
else
f(n)= -1; % Faz f(t)=-1 para pi<t<=2*pi,
% i.e., as últimas 50 amostras de
% f(t) são iguais a 1
end
end
plot(x,f, 'ko'); grid on
title('Função retangular')
xlabel('t em radianos')
ylabel('f(t)')
```

5.4.1.3 Estrutura **if-elseif-end**

Quando houver mais de duas alternativas, a estrutura **if-else-end** do MATLAB torna-se

```
if <condição 1>
     <comandos 1>
elseif <condição 2>
     <comandos 2>
elseif <condição 3>
     <comandos 3>
     .
     .
     .
else
     <comandos 0>
end
```

A lista <comandos 1> será executada se <condição 1> for igual a 1 (verdadeiro), já a lista <comandos 2> será executada se <condição 2> for 1 e assim para as outras condições. Se nenhuma das condições for 1 então <comandos 0> será executada. Quando a primeira <condição> for satisfeita e os <comandos> executados, a estrutura if-elseif-end será abandonada, ou seja, o controle do processamento será transferido para o comando imediatamente após o end . Por exemplo, **`estcond3.m`**

```
a = input('Entre com o valor de a : ');
if a <= -10
     b = exp(a)
elseif a < 0
     b = 1/a
elseif a <= 10
     b = a^2
elseif a < 100
     b = sqrt(a)
else
     b = 10
end
```

quando executado resultará

```
>> estcond3
Entre com o valor de a : 4
b = 16
```

Deste modo foi executado o primeiro comando para o qual a condição `a<=10` foi satisfeita, ou seja, apesar da condição `a<100` ser também verdadeiro, o comando referente a ela não foi executado. Assim, na estrutura `if-elseif-end` é executada somente uma única lista de comandos.

5.4.2- Estruturas de repetição

A estrutura de repetição faz com que uma sequência de comandos seja executada repetidamente até que uma dada condição de interrupção seja satisfeita. O MATLAB possui duas estruturas de repetição: as estruturas **for-end** e a **while-end**

5.4.2.1. Estrutura **for-end**

A estrutura **for-end** permite que um grupo de comandos seja repetido um número específico de vezes. Sua sintaxe é:

```
for <variável>=<arranjo>
     <comandos>
end
```

onde <**variável**> é a variável-de-controle que assume todos os valores contidos no vetor linha <**arranjo**> . Assim, o número de repetições da lista <comandos > é igual ao número de elementos no vetor <**arranjo**>. A variável-de-controle não pode ser redefinida dentro da estrutura **for-end** .

O laço **for** é o controlador de fluxo mais simples e usado na programação MATLAB. Analisando a expressão:

```
for i=1:5
     X(i)=i^2
end
```

pode-se notar que o laço **for** é dividido em três partes:

- A primeira parte (**i=1**) é realizada uma vez, antes do laço ser inicializado.
- A segunda parte é o teste ou condição que controla o laço, (**i<=5**).
- Esta condição é avaliada; se verdadeira, o corpo do laço (**X(i)=i^2**) é executado.

A terceira parte acontece quando a condição se torna falsa e o laço termina.
O comando **end** é usado como limite inferior do corpo do laço.

Vamos considerar um exemplo, executando o programa estrep1.m abaixo:

```
n = input('Valor de n : ');
s = 0;
n2 = n^2;
```

```
for i = 1:2:2*n-1
     s = s + i;
end,
n2, s

>> estrep1
Valor de n : 5
n2 = 25
s = 25
```

Este exemplo mostra que a soma dos n primeiros números ímpares é igual ao quadrado de n, pois para n=5 a variável-de-controle i assume os valores 1 3 5 7 9. Deve ser observado o uso do (;) para suprimir a exibição de resultados intermediários no cálculo de s .

Exercícios :
1) Crie o vetor x=[0 36 72 108 144 180 216 252 288 324] através do comando **for** (exercicio1.m).
Solução:

```
x(1)=0;
for n=2:10
     x(n)=x(n-1)+36;
end
x
```

```
>>exercicio1
x =
0 36 72 108 144 180 216 252 288 324
```

2) Plote 360 pontos de um período da função y=sen(2*pi*x/360), usando o loop *for* (exercicio2.m).
Solução:

```
for x=1:360
y(x)=sin(2*pi*x/360);
end
plot(y)
```

Isto é, a primeira instrução diz : para n igual a 2 até 10, execute todas os comandos até a instrução de *end*. No primeiro ciclo do *for*, n=2, no segundo n=3 e assim por diante, até n=10. Depois do ciclo para n=10, o loop *for* termina e os comandos após a instrução *end* são executados, como é o caso da apresentação dos resultados em x.
OBS: no Matlab, é mais eficiente construir vetores como feito no capítulo 3 do que com a utilização do comando *for*.

Para mostrar que as estruturas **for-end** podem estar encadeadas, considere, por exemplo, os programas abaixo:

```
%estrep2.m
n = input('Ordem do quadrado magico : ');
A = magic(n);
Soma_Linhas = zeros(n,1) ;
```

```
Soma_Colunas = zeros(1,n);
for i = 1:n
     for j = 1:n
          Soma_Linhas(i) = Soma_Linhas(i) + A(i,j);
          Soma_Colunas(j) = Soma_Colunas(j) + A(i,j);
     end
end
A, Soma_Linhas , Soma_Colunas

>> estrep2
Ordem do quadrado magico : 4
A = 16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

Soma_Linhas = 34 34 34 34

Soma_Colunas = 34 34 34 34
```

Cumpre observar que o MA TLAB possui comandos para determinar estes somatórios de um modo mais simples, através do comando sum que fornece a soma das colunas de uma matriz.

É comum construções em que conjuntos de laços **for** são usados principalmente com matrizes:

```
%estrep3.m
     for i=1:8
          for j=1:8
               A(i,j)=i+j;
               B(i,j)=i-j;
          end
     end
     C=A+B;
     A, B, C

>>estrep3

A =
     2     3     4     5     6     7     8     9
     3     4     5     6     7     8     9    10
     4     5     6     7     8     9    10    11
     5     6     7     8     9    10    11    12
     6     7     8     9    10    11    12    13
     7     8     9    10    11    12    13    14
     8     9    10    11    12    13    14    15
     9    10    11    12    13    14    15    16

B =
     0    -1    -2    -3    -4    -5    -6    -7
     1     0    -1    -2    -3    -4    -5    -6
     2     1     0    -1    -2    -3    -4    -5
     3     2     1     0    -1    -2    -3    -4
     4     3     2     1     0    -1    -2    -3
```

```
     5     4     3     2     1     0    -1    -2
     6     5     4     3     2     1     0    -1
     7     6     5     4     3     2     1     0

C =

     2     2     2     2     2     2     2     2
     4     4     4     4     4     4     4     4
     6     6     6     6     6     6     6     6
     8     8     8     8     8     8     8     8
    10    10    10    10    10    10    10    10
    12    12    12    12    12    12    12    12
    14    14    14    14    14    14    14    14
    16    16    16    16    16    16    16    16
```

5.4.2.2 Estrutura **while-end**

A estrutura while-end, ao contrário da **for-end**, repete um grupo de comandos um número indefinido de vezes. Sua sintaxe é

```
while <condição>
      <comandos>
end
```

Enquanto a expressão lógica <condição> for verdadeira a lista <comandos> será repetida.

No laço **while** apenas a condição é testada. Por exemplo, na expressão

```
a = 1; b = 15;
while a<b,
clc
a = a+1
b = b-1
pause(1)
end
disp('fim do loop')
```

a condição **a<b** é testada. Se ela for verdadeira o corpo do laço, será executado. Então a condição é testada novamente, e se verdadeira o corpo será executado novamente. Quando o teste se tornar falso o laço terminará, e a execução continuará no comando que segue o laço após o **end**.

Ao contrário do loop *for*, que executa um conjunto de comandos um número fixo de vezes, o loop *while* executa um conjunto de comandos um número indefinido de vezes. Os comandos entre as instruções *while* e *end* são executadas enquanto todos os elementos na *expressão* forem verdadeiras.

Exercício: Construa o vetor y = [64 32 16 4 2 1] , usando o loop while
Solução:

```
num=128;
n=0;
while num>1
      num=num/2;
```

```
      n=n+1;
      y(n)=num;
end
y
```

Por exemplo, em precisao.m

```
n = 0;
Epsilon= 1;
while 1 + Epsilon > 1
      n = n + 1;
      Epsilon = Epsilon / 2;
end
n, Epsilon, eps

>> precisao
n = 53
Epsilon = 1.1102e-16
eps = 2.2204e-16
```

Epsilon é a chamada precisão da máquina, ou seja, o maior número que somado a 1 é igual a 1. Comparada com a variável especial eps do MATLAB
>> 1+eps-1
ans = 2.2204e-16
>> 1+Epsilon-1
ans = 0

Note que quando eps é somado a 1 resulta em um número maior do que 1. O mesmo não ocorre com Epsilon, porque qualquer valor igual ou menor do que ele somado a 1 será simplesmente 1.

5.4.2.3 Comando **break** (estruturas com interrupção no interior)

A estrutura **while-end** permite que um grupo de comandos seja repetido um número indeterminado de vezes. No entanto, a condição de interrupção é testada no início da estrutura. Em várias situações em programação se faz necessário interromper a execução da repetição verificando a condição no interior da estrutura e não no seu início. O comando **break** interrompe a execução de uma estrutura **while-end** ou **for-end** e transfere a execução para o comando imediatamente seguinte ao **end** . Em repetições aninhadas, o **break** interrompe a execução apenas da estrutura mais interna. Uma repetição com condição de interrupção no interior pode ter a forma

while 1
      <comandos 1>
      if <condição>
            break
      end
      <comandos 2>
end

A estrutura **while-end** é executada indefinidamente a princípio pois a condição do **while** é sempre verdadeira. Contudo, quando a <condição> do **if** for satisfeita o comando **break** será executado causando a interrupção da repetição **while-end**. Por exemplo, o programa no arquivo estrep3.m

```
while 1
      a=input('Entre com a, a>0 : ');
      if a <= 0
            break
      end
      disp(rats(a ))
end
```

lista continuamente a representação racional de um número fornecido enquanto este for positivo. Deste modo,

```
>> estrep3
Entre com a, a>0 : pi
355/113
Entre com a, a>0 : sqrt(2)
1393/985
Entre com a, a>0 : -8
```

Considere mais um programa para exemplificar o uso do comando **break**:

```
%Programa para criar e modificar uma matriz A
for i = l:5,
   for j = l:5,
      if i == j
         A(i,j) = 2;
      elseif abs(i-j) == 1
         A(i,j) = -1;
      else
         A(i,j) = 0;
      end
   end
end

clc
x = 's';
for i = l:5,
   if x == 'q',
      break
   end
   j = l;
   while j<=5,
      ['A('num2str(i) ',' num2str(j)') = 'num2str(A(i,j))]
      x = input('Modifica? (s-sim, n-não, p-próxima linha, q-
sair) =>');
      if x == 's',
         A(i,j) = input('Entre com o novo valor de A(i,j) ');
         j=j+l;
         clc
      end
```

```
      if x == 'n',
         j=j+1;
         clc
      end
      if x == 'p',
         clc
         break
      end
      if x == 'q',
         clc
         break
      end
   end
end
```

5.3- Subprograma `function`

Um outro tipo de arquivo de roteiro é usado para o próprio usuário criar novas funções para o MatLab. Na realidade, várias funções do MatLab são arquivos **.m**. Uma função é criada no MatLab como um arquivo **.m**, porém começando sempre com o seguinte cabeçalho:

`function` **[***variáveis de saída***]** = *Nome_da_Função* **(***variáveis de entrada***)**

Todas as variáveis temporárias usadas na função são variáveis locais e, com isso, após a execução da função, elas são removidas do espaço de trabalho. Como exemplo, veja como o MatLab implementa a função `trace`:

```
function t = trace(a)
%TRACE  Sum of diagonal elements.
%    TRACE(A) is the sum of the diagonal elements of A, which is
%    also the sum of the eigenvalues of A.

%   Copyright (c) 1984-98 by The MathWorks, Inc.
t = sum(diag(a));
```

As linhas de comentário (prefixadas por %) de uma função, quando introduzidas imediatamente após o cabeçalho da função, definem o *help on-line* da própria função.

Veja agora um exemplo de uma função escrita pelo usuário e como ela é utilizada por um programa do Matlab:

```
function azr=rmz(a)
%a1=rmz(a) removes the leading zero elements of a vector
%until a possible scalar variable remains

azr=a;
while (azr(1)==0)&(length(azr)>1),
   azr(1)=[];
end
```

```
%Programa que utiliza uma função criada pelo usuário

a=[0 0 0 1 0 3 6 0];
a1=rmz(a);
a
a1
```

Considere também a seguinte exemplo:

- Abra um arquivo, salvando-o com nome de prog_funcao.m
- Digite os seguintes comandos neste arquivo

```
% prog_funcao.m
% CRIANDO UMA SUBROTINA
v = 1:1:10;
m = media(v);
s = sprintf('\n A média é: %4.2f', m);
disp(s);
% final do programa prog_funcao.m
```

- Agora crie o seguinte arquivo, com o nome de media.m

```
function x = media(u)
% function x = media(u) calcula a média do vetor u, colocando o
resultado em x
x = sum(u)/length(u);
% final da subrotina media.m
```

- Na linha de comando do Matlab, digite:

```
>> prog_funcao
>> echo on
>> prog_funcao
>> echo off
```

6) Exercícios de Fixação

```
% TRABALHANDO COM NÚMEROS COMPLEXOS
a = [1 2;3 4] + i*[5 6;7 8]
realz = real(z)
imagz = imag(z)
modz = abs(z)
fasez = angle (z)


% MULTIPLICAÇÃO DE POLINÔMIOS
% x3 = (x^2 + 3x + 2).(x^2 - 2x + 1)
x3 = conv([1 3 2],[1 -2 1]) % Como ele faz isto?
% Determinação das raízes de um polinômio
roots([1 3 2])
roots([1 -2 1])
roots(x3)


% RECURSOS GRÁFICOS
y = [0 2 5 4 1 0];
plot(y)
help pi
t = 0:.4:4*pi
y = sin(t)
z = cos(t);
plot(t, y, '.', t, z "-.")
title('Funções')
xlabel("t")
ylabel("Seno e Cosseno")
text(3, 0.5, 'Seno')
% Após o próximo comando, selecione a posição que deseja colocar
o texto 'Cosseno' com
% o mouse
gtext('Cosseno')


% PROGRAMANDO COM O MATLAB
% Abra um arquivo a partir do Matlab (File, New, M-File)
% Digite os seguintes comandos e grave o arquivo com o nome
% testel.m, no diretório de usuários (alunos).

n = 3 ;
m = 3;
for i = 1: m
   for j= 1 : n
      a(i, j) = i + j;
   end;
end
disp('Matriz A')
disp(a)
%final do programa testel.m
```

```
% CRIANDO UM PROGRAMA EXEMPLO DE GRÁFICO 3D
% Digite os seguintes comandos em um outro arquivo .m
clear
n = 30;
m = 30;
for i = 1:m
   for j = 1:n
      a(i,j) = sqrt(i+j);
   end
end
b = [a+0.5 a'-0.5;(a.^2)/5 ((a'-0.1).^2)/2];
mesh(b)
```

Bibliografia

Reginaldo de Jesus Santos, "Introdução ao Matlab," Departamento de Matemática, ICEX, UFMG
http://www.mat.ufmg.br/~regi

Frederico Ferreira Campos Filho, "Apostila de Matlab," Departamento de Ciência da Computação, ICEX, UFMG

Grupo PET, "Curso de MATLAB," Engenharia Elétrica - UFMS
http://www.del.ufms.br/tutoriais/matlab/apresentacao.htm

Apostila "MATLAB - Versão Estudante"

Adriana M. Tonini e Bráulio R.G.M. Couto, "Ensinando Geometria Analítica com uso do MATLAB," Departamento de Ciências Exatas e Tecnologia do Centro Universitário de Belo Horizonte / DECET - UniBH.