Apostila de MATLAB

Luis Marcelo de Mattos Zeri

INPE

dezembro/2001

Apresentação

No documento final do I-EPGMET (Encontro dos Alunos de Pós-Graduação em Meteorologia), realizado em 2000, expressa-se claramente que os alunos gostariam de ter cursos introdutórios dos principais softwares e linguagens de programação utilizados pela comunidade de Meteorologia do INPE. Desde então, essa demanda permaneceu presente, mas adormecida. Recentemente, em setembro, com a retomada das discussões sobre a necessidade de cursos, propôs-se a elaboração de apostilas simples, básicas, voltadas para o usuário que nada ou muito pouco conhece do software ou da linguagem de programação. Dessa proposta nasceu o **Projeto Apostila**.

O Projeto Apostila é composto de 5 grupos. Cada grupo foi responsável pela elaboração de uma apostila sobre um software ou uma linguagem de programação. Os grupos são: Rosane Chaves e Daniel Andres (Grads); Rita Andreoli e João Carlos Carvalho (Fortran); Luis Marcelo de Mattos Zeri (Matlab); Pablo Fernandez e Emanuel Giarolla (Latex); e Hélio Camargo e Marcos Oyama (Unix). Os grupos, durante 2 meses, trabalharam intensamente (sem se descuidar das suas dissertações e teses) para produzir as apostilas. A apostila que você está recebendo é fruto do esforço de um dos grupos.

Gostaríamos de agradecer a todos os colegas que revisaram a versão "0" das apostilas: Alexandra Amaro de Lima, Antônio Marcos Mendonça, Edna Sousa, Elizabeth Reuters, Everson dal Piva, José Francisco de Oliveira Júnior, Marcos Yoshida, Maurício Bolzan, Patrícia Moreno Simões, Paulo Marcelo Tasinaffo, Raimundo Moura, Rodrigo de Souza e Wantuir Aparecido de Freitas. As sugestões, críticas e os comentários apresentados foram de grande valia. Muito obrigado!

Gostaríamos, também, de agradecer a Anísio Moliterno por disponibilizar a área pública da fractal para os arquivos de exemplos das apostilas.

As apostilas não têm o objetivo de competir com os cursos que são oferecidos pelo CPTEC ou INPE, mas complementar. **A idéia é que o usuário, após estudar a apostila, possa caminhar sozinho, consultando manuais ou os colegas; ou seja, torne-se independente.** A apostila é uma ponte, não o fim. Recomenda-se aos leitores da apostila que façam os cursos oferecidos pelo CPTEC ou INPE: sempre temos algo a aprender! Além disso, no futuro, as apostilas podem servir de base para cursos ministrados por alunos-instrutores.

Espera-se que, no futuro, outros Projetos Apostila sejam realizados, melhorando e atualizando as apostilas existentes. Além disso, outras apostilas (p.ex. Fortran 90), poderão ser elaboradas.

Boa leitura!

Instalação dos exemplos

Peça a alguém do Suporte (ou algum colega que conheça um pouco de UNIX) para instalar os exemplos na sua área. As instruções são as seguintes:

1. Transfira (via ftp, como usuário anônimo) o arquivo **`instala_apostila`** da área pública da **fractal** (**`/pub/software/apostila`**) para o `home` do usuário.

(em negrito está o que você deve digitar; θ é a tecla Enter)

```
nevasca:/home/fulano>ftp fractal θ
Name (fractal:fulano):anonymous θ
Guest login ok...
Password:fulano@cptec.inpe.br θ          #ATENÇÃO: não irá aparecer na tela!
Guest login ok...
ftp>cd pub/software/apostila θ
ftp>asc θ
ftp>get instala_apostila θ
ftp>quit θ
nevasca:/home/fulano>
```

2. Abra a permissão de execução de `instala`.

```
nevasca:/home/fulano>chmod u+x instala_apostila θ
```

3. Execute `instala_apostila` (e entre com as informações pedidas durante a instalação).

```
nevasca:/home/fulano>instala_apostila θ
```

Explicação dos exemplos

- Usuários do LMO:

O MATLAB está instalado na Fenix e roda em diversas máquinas além dela, como Caos, Neblina, Nevasca, Terra, Urano e Vortice.

- Todos os usuários

Dos 3 exemplos que acompanham a apostila apenas o segundo exige intervenção do usuário na sua execução. Os detalhes para tal procedimento estão no arquivo chamado **leiame.txt**, na mesma pasta desse exemplo. Abaixo é mostrado o conteúdo do arquivo **leiame.txt**:

> Esse programa pede que seja entrado com o nome de um arquivo contendo dados. Esse arquivo acompanha o programa e chama-se d25413h.dat. Basta digitar o seu nome quando solicitado pelo programa.

Sumário

1. Introdução

O MATLAB é tanto uma linguagem de programação quanto um ambiente de computação técnica que possibilita a análise e visualização de dados, cálculos matemáticos e desenvolvimento de algoritmos, entre outras aplicações. Sua primeira versão foi escrita no final da década de 70 nas Universidades do Novo México e Stanford visando fornecer suporte a cursos de teoria matricial, álgebra linear e análise numérica.

2. Iniciando o MATLAB

O espaço de trabalho do MATLAB é iniciado ao se digitar no *prompt* do terminal o comando *matlab*. Feito isso é iniciado o ambiente de trabalho. Nesse ambiente é possível usar comandos de gerenciamento de arquivos, como **delete arquivo** e **edit arquivo**, e comandos de movimentação entre diretórios, comuns aos usuários de DOS e shell (UNIX), como **dir**, **ls** e **cd diretório**. Para sair do ambiente basta digitar **exit** ou **quit**.

3. Definição de variáveis

No MATLAB é possível definir variáveis que serão usadas na sessão de trabalho, como no exemplo abaixo:

```
» a=10
a=
   10
```

Pressionando a tecla ENTER, o MATLAB confirma na tela o valor inserido a menos que seja colocado um sinal de ponto e vírgula (;) após o fim da linha:

```
» b=20;
»
```

Essa opção é interessante para a definição de variáveis extensas, evitando que todos os seus valores sejam exibidos na tela.

Uma variável complexa é definida especificando-se suas partes real e a imaginária:

```
»c=1 + 2i
c =
  1.0000 + 2.0000i
»
```

Definidas as variáveis é possível então efetuar operações matemáticas com elas. As operações básicas são listadas abaixo

Operação	Símbolo
adição, a+b	+
subtração, a-b	-
multiplicação, a·b	*
Divisão, a÷b	/
potenciação	^

e alguns exemplos são dados a seguir:

```
» a+b
ans=
     30
```

onde **ans** é a variável padrão usada para resultados. Pode-se direcionar a saída para outra variável da seguinte forma:

```
» d=a+b
d=
   30
```

Algumas regras devem ser obedecidas na definição de variáveis:

- as variáveis são sensíveis à maiúsculas e minúsculas;
- devem conter no máximo 31 caracteres;
- os nomes **devem** começar com letras e caracteres acentuados não são permitidos.

Alguns exemplos de nomes para variáveis são: **total_anual2000**, **media25**, **MEDIA25**, **X1251330**. Note que, segundo as regras acima, o segundo e o terceiro exemplo referem-se a variáveis diferentes.

Algumas variáveis já são pré-definidas, como **pi** (π), **i** e **j** ($\sqrt{-1}$), **realmin** e **realmax** (menor e maior número real utilizável pelo MATLAB, respectivamente).

Pode-se visualizar todas as variáveis já definidas através do comando **whos**:

```
» whos
Name     Size       Bytes  Class

a        1x1        8      double array
ans      1x1        8      double array
b        1x1        8      double array
c        1x1        16     double array (complex)
d        1x1        8      double array

Grand total is 5 elements using 48 bytes

»
```

Nessa listagem vê-se o nome (*name*), dimensão (*size*) e tamanho (medido em *bytes*) de cada variável definida anteriormente. A última coluna informa o tipo da variável (*class*). Para apagar uma variável usa-se o comando **clear nome_da_variável**. O comando **clear all** remove todas as variáveis do espaço de trabalho.

4. Vetores e Matrizes

4.1 Vetores

Um vetor no MATLAB é uma matriz linha (vetor linha) ou uma matriz coluna (vetor coluna). Um vetor linha pode ser definido separando os valores entre colchetes por espaços *ou* vírgulas. A separação dos valores por ; produz um vetor coluna.

```
»pares=[2 4 6 8 10];
»primos=[2; 3; 5; 7; 11];
»
```

Pode-se então invocar essas variáveis a qualquer momento:

```
» pares
pares =
       2    4    6    8   10
```

Os valores de um vetor linha ou coluna podem ser acessados usando-se índices que identificam a posição do elemento no vetor, como nos exemplos abaixo:

```
» pares(3)
ans =
      6
»x=pares(4)
x=
      8
»ans+x
ans=
      14
```

Intervalos de valores podem ser selecionados dentro de um vetor. No exemplo abaixo é definido um novo vetor contendo apenas os elementos da segunda à quinta linha do vetor **primos**:

```
» impares=primos(2:5)
impares=
         3
         5
         7
         11
```

O comando **whos** agora produz o seguinte resultado:

```
» whos
Name      Size       Bytes  Class

ans       1x1        8      double array
impares   4x1        32     double array
pares     1x5        40     double array
primos    5x1        40     double array
x         1x1        8      double array

Grand total is 16 elements using 128 bytes
»clear all
```

```
»whos
»
```

Acima vemos as dimensões dos vetores definidos e também que o vetor **impares** tem apenas 4 linhas devido ao intervalo especificado na sua definição. Após a visualização todas as variáveis foram apagadas com o comando **clear all**, o que resultou numa saída vazia para comando **whos** digitado em seguida.

Os vetores definidos acima tiveram seus elementos definidos um-a-um. No entanto vetores podem ser construídos especificando-se valores iniciais, finais e incrementos:

```
» a=1:5, b=1:2:9
a=
    1    2    3    4    5
b=
    1    3    5    7    9
»
```

(No exemplo acima a definição dos dois vetores foi encadeada numa única linha, separando as duas definições por vírgula. Diferentes comandos podem ser definidos de uma só vez, desde que separados por vírgula ou ponto-e-vírgula. O espaço entre a vírgula e o próximo comando é opcional.) Foi definido um vetor **a** variando de 1 a 5 (de um em um) e outro indo de 1 a 9, mas de dois em dois.

O comando **linspace(i,f,n)** fornece um vetor linha de ***n*** elementos entres os valores *i* e *f*. O vetor **a** do exemplo anterior pode ser redefinido da seguinte forma:

```
» a=linspace(1,5,5)
a=
    1    2    3    4    5
»
```

O comando **linspace** sem o último argumento fornece um vetor de 100 elementos como padrão. O tamanho de um vetor pode ser verificado com o comando **length**, como no exemplo abaixo:

```
»b=linspace(1,5);
»length(b)
ans=
     100
»
```

De maneira similar o comando **logspace(i,j,n)** fornece n elementos entre 10^i e 10^j. A ausência do terceiro argumento, **n**, fornece 50 elementos entre 10^i e 10^j.

Um vetor pode ser transposto usando-se o operador de transposição **'**, o apóstrofo. Para vetores complexos é realizada a transposição do complexo conjugado. Para a operação apenas de transposição usa-se a transposição pontuada, ou seja, o operador de transposição **'** precedido de um ponto. Abaixo é mostrada a transposição do vetor **a** definido anteriormente:

```
» a=a.'
a =
   1
```

```
   2
   3
   4
   5
```

4.2 Matrizes

O conhecimento das técnicas de manipulação de matrizes é fundamental pois dados importados para o espaço de trabalho do MATLAB são tratados como matrizes. A importação de dados será vista adiante.

Matrizes são definidas no MATLAB de forma análoga à definição dos vetores linhas ou colunas: usam-se vírgulas ou espaços para separar os elementos de uma linha e pontos-e-vírgulas para separar as linhas:

```
» m=[8 1 6;3 5 7;4 9 2]
m =
   8   1   6
   3   5   7
   4   9   2
»
```

Como no caso dos vetores, os elementos, linhas e colunas de uma matriz podem ser acessadas usando-se índices que definem a posição do elemento dentro da matriz. Assim, o elemento da segunda linha na terceira coluna pode ser acessado da seguinte maneira:

```
» m(2,3)
ans =
      7
»
```

Para a seleção de linhas ou colunas inteiras usa-se o operador dois pontos **:**. O comando **m(:,2)** significa "todas as linhas da coluna 2"; **m(3,:)** retorna a linha 3, com todos os seus elementos. À partir da matriz **m** é possível definir outra:

```
» n=m(1:2,2:3)
n =
    1   6
    5   7
»
```

O operador dois pontos foi usado duas vezes: uma para delimitar o intervalo de linhas desejado (da linha 1 a 2) e outra para especificar o intervalo de colunas (da coluna 2 a 3).

Quando é necessário remover alguma linha ou coluna de uma matriz atribui-se a esta linha ou coluna a matriz nula [ ]. Dessa forma a matriz remanescente será composta apenas das linhas (ou colunas) remanescentes:

```
» m(:,3)=[]
m =
   8   1
   3   5
   4   9
```

No exemplo acima foi atribuída à coluna 3 a matriz nula, resultando numa nova matriz **m**. É possível também montar grandes matrizes através de outras pequenas, como:

```
» a=[1 1;1 1];b=[2 2;2 2];c=[a b;b a]
c =
   1   1   2   2
   1   1   2   2
   2   2   1   1
   2   2   1   1
```

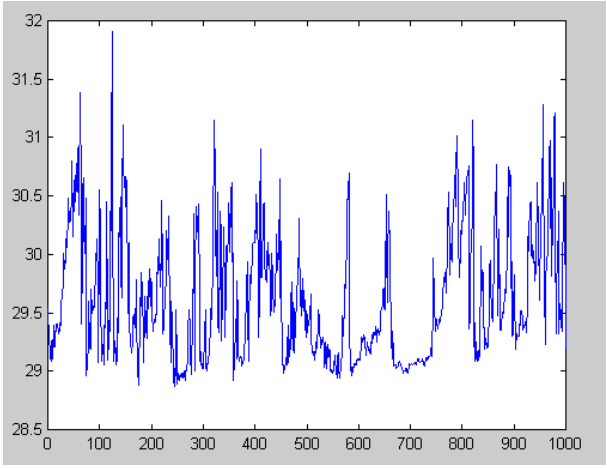
4.3 Importando dados

Dados importados para o espaço de trabalho do MATLAB são tratados como matrizes. O comando usado para carregar um arquivo contendo dados é o **load nome_do_arquivo.** Esse comando é utilizado para dados no formato ASCII. Arquivos contendo dados binários não serão tratados nessa apostila. Para maiores informações consulte a ajuda do MATLAB sobre o comando **fread** (digite **help fread**). Abaixo é visto um exemplo.

```
»cd \dados
»dir
.       ..      temp.dat
»load temp.dat
»whos
Name     Size        Bytes      Class
temp     3600x1      28800      double array

Grand total is 3600 elements using 28800 bytes
»temp(1:10)
ans =
  29.5346
  29.2707
  29.2381
  29.2211
  29.1069
  29.1025
  29.0861
  29.2600
  29.2710
  29.1076
»
```

No exemplo acima, após ter entrado e visualizado o conteúdo da pasta dados, o arquivo *temp.dat* é carregado. O comando **whos** mostra as dimensões da variável associada ao arquivo: 3600 linhas e 1 coluna. Após isso as dez primeiras linhas do arquivo carregado são exibidas. Note que a variável resultante do carregamento do arquivo possui o mesmo nome deste, exceto pela extensão. O gráfico da variável **temp** pode então ser feito.

```
»plot(temp(1:1000))
```



O comando **plot** utilizado será descrito em detalhes em outra seção. Destaca-se aqui que apenas as 1000 primeiras linhas da variável foram impressas no gráfico.

4.4 Operações matemáticas com matrizes[1]

Operações matemáticas simples (adição, subtração, divisão e multiplicação) envolvendo escalares e matrizes seguem a mesma interpretação natural.

```
» m=[8 1 6;3 5 7;4 9 2];
» 3*m
ans =
   24    3   18
    9   15   21
   12   27    6

»m+100
ans =
  108  101  106
  103  105  107
  104  109  102
»
```

Nas operações entre matrizes devem ser respeitadas as regras usuais da matemática quanto ao número de linhas e colunas que duas matrizes devem ter para serem somadas, multiplicadas, etc. No entanto existem operações especiais. Sendo $A=[a_1\ a_2\ ...\ a_n]$ e $B=[b_1\ b_2\ ...\ b_n]$ duas matrizes, então:

- $A./B = [a_1/b_1\ \ a_2/b_2\ \ ...\ \ a_n/b_n]$;
- $A.*B = [a_1 \cdot b_1\ \ a_2 \cdot b_2\ \ ...\ \ a_n \cdot b_n]$;
- $A.\hat{}B = [a_1\hat{}b_1\ \ a_2\hat{}b_2\ \ ...\ \ a_n\hat{}b_n]$;

A multiplicação de um vetor de **m** linhas e **1** coluna (**m X** 1) por um de **1** linha e **m** colunas (**1 X m**) resulta numa matriz de **m** linhas e **m** colunas. Já a operação de multiplicação pontuada entre dois vetores de mesma dimensão, $A_{\mathbf{m\ X\ 1}}$ X $B_{\mathbf{m\ X\ 1}}$, por exemplo, produz um terceiro vetor de mesma dimensão cujos elementos são resultado da multiplicação elemento-a-elemento entre os dois vetores iniciais.

[1] Para evitar redundâncias, vetores (linha ou coluna) serão tratados como matrizes.

```
» a=1:100;b=2*a;plot(sin(a)*cos(b))

??? Error using ==> *
Inner matrix dimensions must agree.
»
```

Acima foram definidos dois vetores de mesma dimensão e em seguida um comando de gráfico foi dado sobre a operação entre esses dois vetores. A mensagem de erro exibida pelo MATLAB refere-se ao fato das dimensões para a multiplicação matricial não coincidirem (**1 x 100 X 1 x 100**). Para o cálculo do vetor de elementos sen(a1)*cos(b1), sen(a2)*cos(b2), ..., sen(a100)*cos(b100) é necessário o uso da multiplicação pontuada. O comando **plot(sin(a).*cos(b))** produz então o gráfico desejado (note o ponto antes do sinal de multiplicação). Nas operações pontuadas as dimensões dos fatores devem ser iguais.

Abaixo são listadas algumas funções matemáticas elementares.

Funções Elementares	
abs(x)	Valor absoluto ou módulo de um número complexo
acos(x)	Arco co-seno
acosh(x)	Arco co-seno hiperbólico
angle(x)	Ângulo de um número complexo
asin(x)	Arco seno
asinh(x)	Arco seno hiperbólico
atan(x)	Arco tangente
atanh(x)	Arco tangente hiperbólico
cos(x)	Co-seno
cosh(x)	Co-seno hiperbólico
cross(a,b)	Produto vetorial dos vetores a e b
exp(x)	Exponencial
format	*format long* mostra 15 dígitos significativos
inv(x)	Matriz inversa da matriz x
log(x)	Logaritmo natural
log10(x)	Logaritmo na base 10
max(x)	Maior elemento em x
mean(x)	Média de x
min(x)	Menor elemento em x
sin(x)	Seno
sinh(x)	Seno hiperbólico
sqrt(x)	Raiz quadrada
std(x)	Desvio padrão
sum(x)	Soma dos elementos de x
tan(x)	Tangente
tanh(x)	Tangente hiperbólica

Algumas dessas operações podem ser aplicadas a vetores ou matrizes. Para a função de desvio padrão **std(x)**, por exemplo, se o argumento **x** for um vetor o resultado será o desvio padrão desse vetor. Se **x** for uma matriz, será calculado o desvio para cada coluna dessa matriz. O mesmo ocorre com as funções **sum**, **max** e **mean**, entre outras.

Utilizando a função **sum** é possível, por exemplo, calcular o produto escalar de dois vetores.

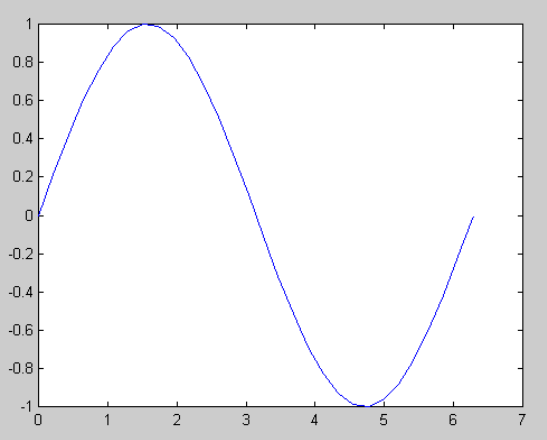
```
» a=[1 2 3];b=[3 2 4];c=sum(a.*b)
  c=
```

```
19
```

5. Gráficos

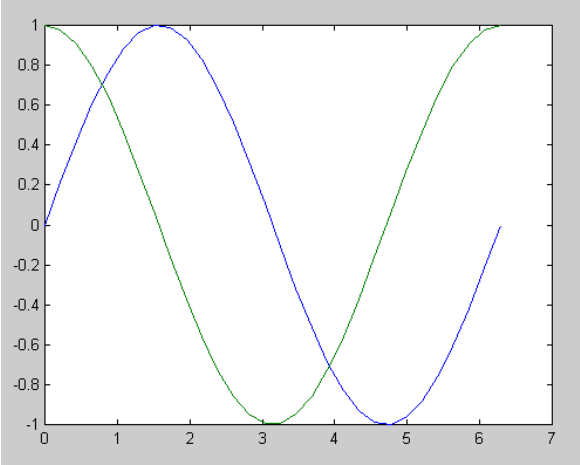
O comando utilizado para fazer gráficos bidimensionais no MATLAB é o **plot**. Esse comando tem a seguinte forma: **plot(x1, y1, S1, x2, y2, S2, ...)**, onde **(xn,yn)** são conjuntos de dados e **Sn** são seqüências de caracteres que especificam cor, símbolos utilizados e estilos de linha.

```
» x=linspace(0,2*pi,30); y=sin(x); plot(x,y)
```



Pode-se traçar dois gráficos simultaneamente:

```
» z=cos(x); plot(x,y,x,z)
```

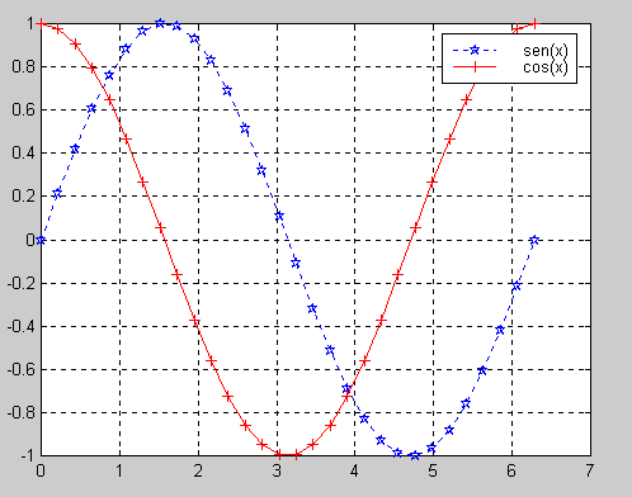


Note que o gráfico anterior foi apagado e substituído pelo novo. Abaixo será mostrado um comando a respeito disso. A seqüência de caracteres que contém a formatação da linha, dos símbolos e das cores usadas nos gráficos pode conter os seguintes símbolos:

Símbolo	Cor	Símbolo	Marcador	Símbolo	Tipo de Linha
b	azul	.	ponto	-	linha contínua
g	verde	o	círculo	:	linha pontilhada
r	vermelho	x	x	-.	traços e pontos
c	ciano	+	+	--	linha tracejada
m	magenta	*	estrela		
y	amarelo	s	quadrado		
k	black	d	losango		
w	branco	v	triângulo p/ baixo		
		^	triângulo p/ cima		
		<	triângulo p/ esquerda		
		>	triângulo p/ direita		
		p	pentagrama		
		h	hexagrama		

Abaixo vê-se um exemplo de utilização dos caracteres

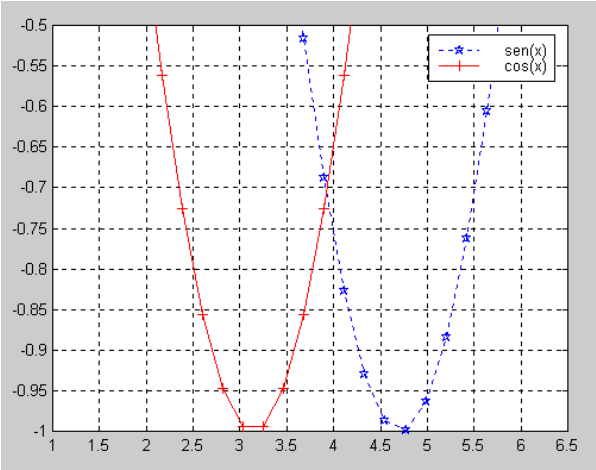
```
» plot(x,y,'b:p',x,z,'r+-') , grid on, legend('sen(x)','cos(x)')
```



No gráfico acima foram utilizadas duas outras opções: **grid on** e **legend**. O primeiro adiciona linhas de grade ao gráfico e o segundo cria uma caixa de legenda no canto superior direito do gráfico. A legenda pode ser movida com o *mouse* após o gráfico ser desenhado.

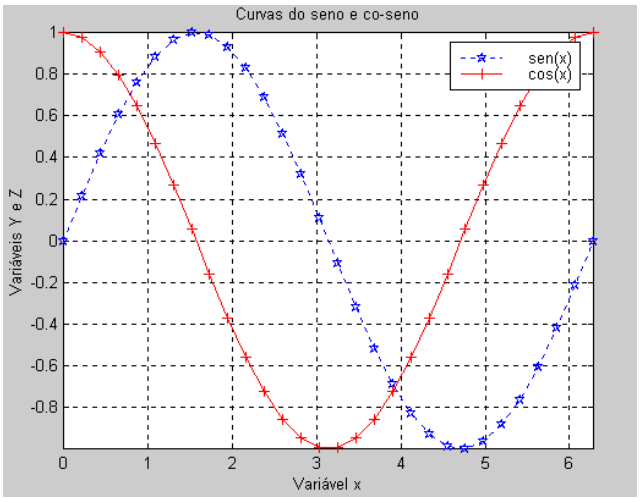
Os valores máximos e mínimos dos eixos são controlados com o comando **axis ([xmin xmax ymin ymax])**. No gráfico acima esse comando teria o seguinte efeito:

```
» axis([1 6.5 -1 -0.5]);
```



Os limites normais podem ser restaurados com o comando **axis** seguido da opção **auto** (**axis auto**). Essa opção, que é o padrão usado nos gráficos, adiciona uma "folga" aos limites máximos dos eixos x e y. Para que os limites dos eixos sejam os limites máximos dos dados usa-se a opção **tight**:

```
» axis tight
» title('Curvas do seno e co-seno'),xlabel('Variável x'),ylabel('Variáveis Y e Z')
```

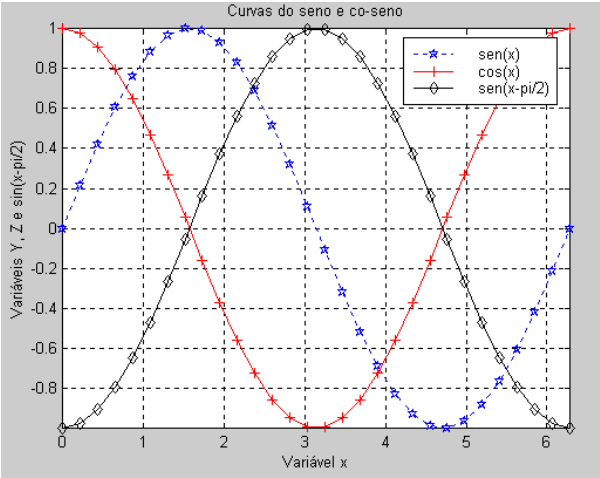


Após o primeiro comando percebe-se que o limite máximo em *x* passou a ser o máximo valor dos dados nesse eixo. Outros comandos foram usados nesse gráfico: **title**, **xlabel** e **ylabel** para especificar títulos ao gráfico, ao eixo *x* e ao eixo *y*, respectivamente.

O MATLAB oferece também um comando de *zoom*. Essa opção é ativada digitando-se **zoom on** na tela de comandos. Feito isso, a ampliação é feita ao clique do *mouse* sobre o gráfico (botão esquerdo do *mouse*). O clique com o botão direito (ou o clique duplo com o esquerdo) faz o gráfico retornar ao nível de *zoom* anterior. Outra possibilidade é "desenhar" a janela na qual se deseja que o *zoom* seja feito. Para isso mantém-se o botão esquerdo do *mouse* pressionado enquanto a setinha do *mouse* é arrastada sobre o gráfico. Obtendo-se o tamanho de janela desejado o *zoom* é executado assim que o botão é solto. A opção de *zoom* é desligada com o comando **zoom off**.

Um novo comando de gráfico, por padrão, tem a sua saída direcionada para a figura já aberta, limpando-a (apagando os gráficos já desenhados) e desenhando o novo. Para que os gráficos anteriores sejam mantidos faz-se uso do comando **hold on**.

```
» hold on, plot(x,sin(x-pi/2),'kd-'),legend('sen(x)','cos(x)','sen(x-pi/2)')
» ylabel('Variáveis Y, Z e sin(x-pi/2)')
```

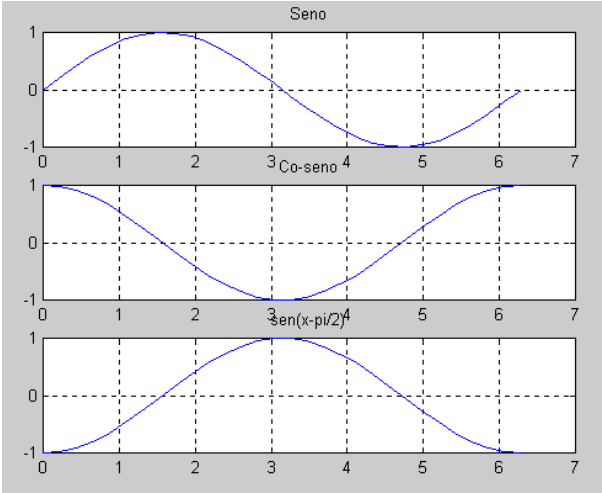


No gráfico acima os comandos **legend** e **ylabel** foram re-aplicados para que fizessem referência à nova curva.

Para que a figura atual seja mantida e a saída dos comandos gráficos seja direcionada para outra janela, utiliza-se o comando **figure**. Tendo várias figuras abertas, a saída dos comandos gráficos é especificada fazendo-se: **figure(n)**, onde **n** é o número da figura para a qual os comandos digitados devem ser direcionados. No final desse capítulo serão vistos exemplos da utilização desse comando.

É possível que vários gráficos sejam desenhados na mesma janela, mas com conjunto de eixos separados, através do comando **subplot(abc)**. O parâmetro **a** é o número de linhas desejado; **b** especifica o número de colunas e **c** é o índice que indica a posição nessa "matriz" para onde a saída do gráfico será direcionada, a contar da esquerda para a direita e de cima para baixo.

```
» figure,subplot(311),plot(x,y),grid on,title('Seno')
» subplot(312),plot(x,z),grid on,title('Co-seno')
» subplot(313),plot(x,sin(x-pi/2)),grid on,title('sen(x-pi/2)')
```



Gráficos com outras escalas podem ser criados através dos comandos **loglog**, **semilogx**, **semilogy** e **plotyy**. Existem também outros tipos de gráficos no MATLAB. São eles: **area**, **pie**, **pareto**, **polar**, **bar**, **stairs**, **hist**, **stem**, **errorbar**, **compass**, **feather**, **rose**, **fill** e **plotmatrix**. Para maiores informações sobre cada um deles e também sobre gráficos com outras escalas consulte a ajuda via comando **help**: **help hist**, **help pie**, **help loglog**, **help semilogx**, etc. Todos esses gráficos são bidimensionais, mas o MATLAB também faz gráficos tridimensionais. Para isso consulte a ajuda referente aos comandos **plot3, contour**, **pcolor** e **colormap** entre outros.

Feito um gráfico pode-se salvar a figura em diversos formatos ou imprimi-la. O comando usado para essa finalidade é o **print**. Digitando-se **help print** vê-se todos os formatos e opções disponíveis, como eps, epsc, tiff, jpg, etc. Vejamos um exemplo:

```
» figure(1), print –depsc curvas1.epsc
» figure(2), print –dtiff curvas2.tiff
```

Na primeira linha de comandos acima o foco dos comandos é direcionado para a figura 1 através do comando **figure(1)**. Após isso ela é salva no formato *epsc* (eps Color) e **–d** é o operador que deve preceder a abreviação para o formato escolhido. Logo após a opção de formato é especificado o nome do arquivo a ser criado. Para a segunda figura o formato escolhido foi o *tiff*.

6. Escrevendo programas no MATLAB

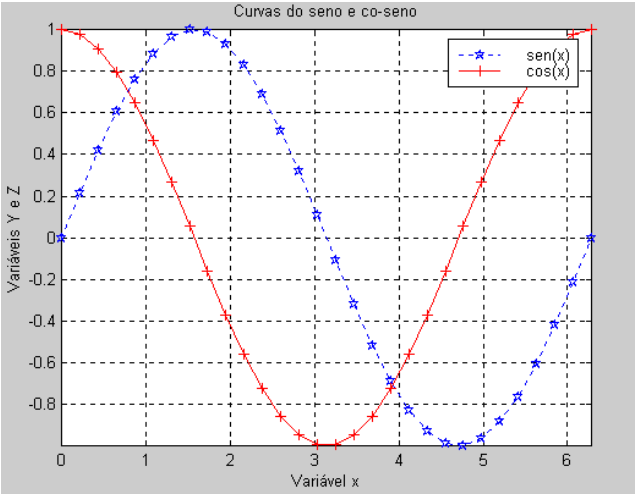
Seqüências de comandos podem ser armazenadas em arquivos e executadas no espaço de trabalho do MATLAB. Para isso basta que o arquivo seja salvo com a extensão "m". Os comandos do exemplo acima poderiam ser guardados num arquivo:

```
x=linspace(0,2*pi,30);
y=sin(x);
z=cos(x);

plot(x,y,'b:p',x,z,'r+-')

grid on
legend('sen(x)','cos(x)')
axis tight
title('Curvas do seno e co-seno')
```

```
xlabel('Variável x')
ylabel('Variáveis Y e Z')
```

Usando-se qualquer editor de texto, esses comandos são guardados num arquivo chamado **teste.m**, por exemplo, na pasta de trabalho chamada dados.

```
» cd \dados
» dir
.       ..      temp.dat teste.m
» teste
»
```



Programas eventualmente podem necessitar de informações fornecidas na tela pelo usuário. Da mesma forma pode ser interessante que informações sejam exibidas durante a execução de um programa. Para a primeira finalidade destaca-se o comando **input**. No programa feito acima poderia ser colocada a seguinte linha:

```
npontos=input('Quantos pontos deve ter a variável x ?')
x=linspace(0,2*pi,npontos);
y=sin(x);
z=cos(x);
...
```

A variável que irá receber o valor na tela foi colocada no comando **linspace**. Durante a execução o usuário seria avisado da seguinte forma:

```
» teste
Quantos pontos deve ter a variável x ?50
npontos =
    50
»
```

Após a pergunta foi entrado o valor 50 e então esse valor foi exibido na tela e o gráfico foi feito. Note que a ausência do ponto-e-vírgula no final da linha do comando **input** fez com que o valor de **npontos** fosse exibido na tela.

O comando **input** pode ser configurado para receber uma seqüência de caracteres como entrada. Essa seqüência pode ser, por exemplo, o nome do arquivo a ser carregado e processado. Veja as linhas abaixo que executam essa tarefa:

```
arquivo=input('Digite o nome do arquivo: ','s');
load(arquivo)
```

A opção **'s'** faz com que a entrada seja interpretada como uma seqüência de caracteres. O comando **load** deve ser usado nessa forma (**load(nome_do_arquivo)**) quando o seu argumento for uma seqüência de caracteres. Veja no exemplo:

```
arquivo=input('Digite o nome do arquivo: ','s');
load(arquivo)
eval(['T=' arquivo(1:length(arquivo)-4) ';'])
```

Para o caso do nome do arquivo ser **temp.dat** a variável **arquivo** conteria 8 caracteres, incluindo aí o ponto que separa o nome da extensão. Após o carregamento do arquivo uma variável é criada pelo comando **load** com o mesmo nome do arquivo menos a extensão dele. Nesse caso essa variável seria chamada **temp**. Até esse ponto o programa em questão "não sabe" que existe essa variável **temp**, é preciso defini-la no programa. A função **eval** permite que o nome do arquivo contido na variável chamada **arquivo**, menos a sua extensão, seja usado nessa definição.

Os colchetes foram usados para concatenar horizontalmente várias seqüências de caracteres. A primeira delas é **T=**, a segunda é **arquivo(1: length(arquivo)-4)**, que tem como resultado *temp* e a terceira é **;**. Com esse comando foi dada a seguinte ordem ao MATLAB: "crie uma variável chamada T que seja igual a variável cujo nome se encontra na variável **arquivo** menos os quatro últimos caracteres, de maneira a evitar a extensão e o ponto, ou seja, faça isso: **T=temp;**". Verificamos o resultado disso na listagem abaixo:

```
» whos
Name     Size        Bytes     Class

T        3600x1      28800     double array
arquivo  1x8         16        char array
temp     3600x1      28800     double array

Grand total is 7208 elements using 57616 bytes
»
```

Sem o uso da função **eval**, ou seja, se fizéssemos apenas **T= arquivo(1: length(arquivo)-4);** criaríamos uma nova variável chamada T, mas que conteria apenas um nome, uma seqüência de caracteres, que no caso seria a palavra *temp*.

Como o conteúdo do arquivo agora foi associado a uma nova variável, a variável associada ao nome do arquivo (*temp*) pode ser apagada, para liberar espaço na memória. Para isso usa-se novamente o comando **eval**, adicionando-se uma linha ao programa anterior:

```
arquivo=input('Digite o nome do arquivo: ','s');
load(arquivo)
eval(['T=' arquivo(1: length(arquivo)-4) ';'])
eval(['clear ' arquivo(1: length(arquivo)-4)])
```

```
» whos
Name     Size        Bytes      Class

T        3600x1      28800      double array
arquivo  1x8         16         char array

Grand total is 3608 elements using 28816 bytes
»
```

Note que a variável **arquivo** ainda permanece no espaço de trabalho. Para apagá-la usa-se apenas o comando **clear arquivo**.

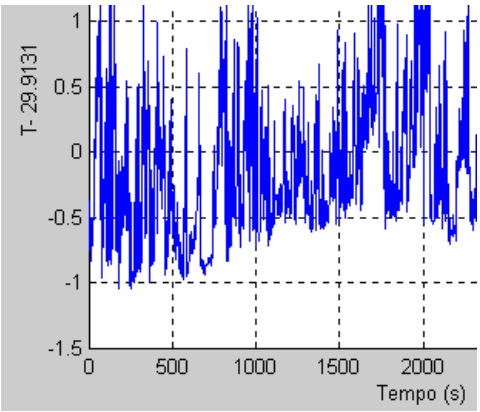
O programa acima poderia então continuar com comandos matemáticos e/ou gráficos.

```
arquivo=input('Digite o nome do arquivo: ','s');
load(arquivo)
eval(['T=' arquivo(1: length(arquivo)-4) ';'])
eval(['clear ' arquivo(1: length(arquivo)-4)])
media=mean(T);
plot(T-media),grid on,zoom on,xlabel('Tempo (s)'),ylabel(['T- ' num2str(media)])
title('Desvio da média para temperatura')
disp(media)
```

O último comando do programa acima tem quase o mesmo efeito da ausência do ponto-e-vírgula na definição de uma variável: ele exibe na tela o valor da variável mas não exibe o seu nome.

```
»teste2
Digite o nome do arquivo: temp.dat
   29.9131
»
```

Na definição do título do eixo y foi usado um novo comando, o **num2str**. Esse comando converte um número em uma seqüência de caracteres que pode ser usada num comando de título, por exemplo. Os colchetes foram novamente usados para concatenar as seqüências '**T** ' e a resultante do comando **num2str**. Pode-se também fazer o contrário, converter uma seqüência de caracteres em um número, usando-se para isso o comando **str2num**. Abaixo vê-se o resultado do comando **num2str** no título do eixo y.



6.1 Controle de Fluxo num Programa

O controle de fluxo num programa escrito no MATLAB é bem similar àquele usado em outras linguagens de programação. Ele faz com que uma série de comandos seja executada repetidas vezes. Para isso utiliza-se o comando **for**.

```
» for n=1:10
    x(n)=sin(n*pi/10);
  end
»
```

Existe também outra forma de condicionar a execução de alguns comandos. Isso pode ser feito com o comando **while**:

```
» EPS=1;
» while (1+EPS)>1
    EPS=EPS/2;
  end
»
```

Enquanto a soma (1+EPS) for maior do que 1 o comando EPS=EPS/2 é executado. Quando isso deixar de ser verdadeiro o ciclo termina. Esse programa só é possível porque quando EPS chega a 10^{-16} ele sai do limite de precisão finita do MATLAB e é então considerado zero.

Por fim, pode-se usar a estrutura **if-else-end**, que apresenta a seguinte forma:

```
» if expressão
    comandos executados se expressão for Verdadeira
  else
    comandos executados se Falso
  end
»
```

6.2 Criação de funções pelo usuário

A criação de funções pelo usuário também pode ser feita no MATLAB. Seja o exemplo de criação de uma função para o cálculo das raízes de uma equação de segundo grau. O arquivo que contém os comandos dessa função deve ter o mesmo nome da função e ter extensão ".m". Dessa forma pode-se criar a função **raizes** salvando os seguintes comandos num arquivo chamado raizes.m:

```
function [x1,x2]=raizes(a,b,c)
delta=sqrt(b^2 – 4*a*c);
x1=(-b + delta)/2*a;
x2=(-b - delta)/2*a;
```

Essa função é usada então fornecendo-se os parâmetros a, b e c:

```
» a=1;b=1;c=-1;[x1,x2]=raizes(a,b,c)

x1 =
      0.618033988749895
x2 =
      -1.61803398874989
```

```
»
```

Criada a função, é possível usar os resultados dela em outros programas criados no MATLAB.

7. Ajuda on line no MATLAB e referências

O MATLAB oferece uma ajuda on-line no ambiente de trabalho. Ao longo do texto dessa apostila já foi sugerido que essa ajuda fosse consultada para maiores detalhes e opções para alguns comandos usados.

Pode-se também procurar por alguma palavra contida nas descrições das diversas funções do MATLAB usando-se o comando **lookfor palavra**.

Com o comando **helpwin** é aberta uma janela que possibilita a navegação entre as funções e suas descrições. Outra fonte de informações e ajuda é obtida com comando **helpdesk**, que abre um arquivo em HTML no navegador da internet, contendo informações mais detalhadas.

Na internet o endereço http://www.mathworks.com/products/matlab/ oferece documentação (formato PDF), lista de funções e literatura técnica a respeito.

Na preparação dessa apostila foi usado o seguinte livro:

Hanselman, D; Littlefield, B. **MATLAB 5 – Versão do Estudante – Guia do Usuário**, MAKRON *Books*, 413 pp, São Paulo, 1999.